

Das
Praxisbuch

Maximilian Vollendorf
Frank Bongers

jQuery

jQuery

Das Praxisbuch

jQuery

- ▶ Grundlagen, Einsatz, Praxisbeispiele
- ▶ Plug-ins nutzen und erstellen, jQuery UI
- ▶ Navigationen, Tooltips, Sprites, Bildergalerien, Formulare, flexible Tabellen, jQTouch



Beispiele, Praxislösungen und hilfreiche Tools

Galileo Computing



Vollendorf
Bongers



1288



Frank Bongers, Maximilian Vollendorf

jQuery

Das Praxisbuch

Liebe Leserin, lieber Leser,

mit dem JavaScript-Framework jQuery stehen Ihnen zahlreiche Funktionen zur Verfügung, um zeitgemäße Websites in kurzer Zeit umzusetzen. Mit oft nur wenigen Zeilen Code können Sie bereits beeindruckende Wirkungen erzielen und Ihren Webauftritt bereichern. jQuery vereinfacht die Arbeit mit JavaScript, indem es die Benutzung sehr kurzer und einprägsamer Befehle statt der herkömmlichen komplizierteren JavaScript-Befehle ermöglicht. Das macht auch für weniger programmiererfahrene Webdesigner das Einbinden von Effekten, Animationen und Ajax-Techniken leicht und nachvollziehbar.

Mit diesem Buch erhalten Sie eine verständliche und umfassende Anleitung zur Nutzung des Frameworks an die Hand, angefangen von einfachen Webseiten, die nur um einzelne Effekte aufgewertet werden sollen bis hin zu komplexen Anwendungen. Besonderen Wert haben die Autoren auf praxisorientierte Beispiele gelegt. Dort finden Sie Anleitungen zu allen wichtigen Anwendungsgebieten rund um Navigation, Formulare, Bildergalerien, Tabellen und Inhaltsmanipulation. Fortgeschrittenere Leser finden in den letzten Kapiteln Informationen zu Plugins und dem jQuery UI.

Dieses Buch wurde mit großer Sorgfalt lektoriert und produziert. Sollten Sie dennoch Fehler finden oder inhaltliche Anregungen haben, scheuen Sie sich nicht, mit uns Kontakt aufzunehmen. Ihre Fragen und Änderungswünsche sind uns jederzeit willkommen.

Viel Vergnügen beim Lesen!

Wir freuen uns auf den Dialog mit Ihnen.

Stephan Mattescheck

Lektorat Galileo Computing

stephan.mattescheck@galileo-press.de

www.galileocomputing.de

Galileo Press • Rheinwerkallee 4 • 53227 Bonn

Auf einen Blick

1	jQuery kennenlernen	15
2	Den Arbeitsplatz einrichten	25
3	jQuery – der Einstieg	35
4	jQuery – die Übersicht	59
5	jQuery – der Praxiseinsatz	277
6	Unit Tests	485
A	HTML und CSS	495
B	JavaScript und DOM	535
C	Inhalt der DVD-ROM	569

Der Name Galileo Press geht auf den italienischen Mathematiker und Philosophen Galileo Galilei (1564–1642) zurück. Er gilt als Gründungsfigur der neuzeitlichen Wissenschaft und wurde berühmt als Verfechter des modernen, heliozentrischen Weltbilds. Legendär ist sein Ausspruch *Eppur se muove* (Und sie bewegt sich doch). Das Emblem von Galileo Press ist der Jupiter, umkreist von den vier Galileischen Monden. Galilei entdeckte die nach ihm benannten Monde 1610.

Lektorat Stephan Mattescheck

Korrektorat Angelika Glock

Cover Barbara Thoben, Köln

Titelbild Barbara Thoben, Köln

Typografie und Layout Vera Brauner

Herstellung Norbert Englert

Satz Typographie & Computer, Krefeld

Druck und Bindung Bercker Graphischer Betrieb, Kevelaer

Dieses Buch wurde gesetzt aus der Linotype Syntax Serif (9,25/13,25 pt) in FrameMaker.
Gedruckt wurde es auf chlorfrei gebleichtem Offsetpapier.

Gerne stehen wir Ihnen mit Rat und Tat zur Seite:

stephan.mattescheck@galileo-press.de bei Fragen und Anmerkungen zum Inhalt des Buches
service@galileo-press.de für versandkostenfreie Bestellungen und Reklamationen
britta.behrens@galileo-press.de für Rezensionen- und Schulungsexemplare

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-8362-1288-5

© Galileo Press, Bonn 2010

1. Auflage 2010

Das vorliegende Werk ist in all seinen Teilen urheberrechtlich geschützt. Alle Rechte vorbehalten, insbesondere das Recht der Übersetzung, des Vortrags, der Reproduktion, der Vervielfältigung auf fotomechanischem oder anderen Wegen und der Speicherung in elektronischen Medien. Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Programmen verwendet wurde, können weder Verlag noch Autor, Herausgeber oder Übersetzer für mögliche Fehler und deren Folgen eine juristische Verantwortung oder irgendeine Haftung übernehmen. Die in diesem Werk wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Inhalt

Vorwort	11

1 jQuery kennenlernen 15

1.1 Was jQuery alles kann	15
1.2 Ein Framework? Eine Community!	17
1.3 Nicht ohne mein JavaScript	18
1.3.1 Gründe für das Entstehen von Frameworks	19
1.3.2 Nochmals – was ist ein Framework?	20
1.3.3 Aufgaben eines Frameworks	20
1.3.4 Aktuelle Frameworks für JavaScript	21
1.3.5 Frameworks – ein Rückblick	22
1.4 jQuery – viel mit wenig erreichen	23

2 Den Arbeitsplatz einrichten 25

2.1 Rechner und Betriebssystem	25
2.1.1 Windows	26
2.1.2 Mac OS X	26
2.1.3 Linux	26
2.2 Browser	27
2.2.1 Internet Explorer	27
2.2.2 Firefox	27
2.2.3 Opera	28
2.2.4 Safari	28
2.2.5 Google Chrome	28
2.3 Webserver	29
2.4 IDEs und Editoren	30

3 jQuery – der Einstieg 35

3.1 Vergleich: JavaScript mit und ohne jQuery	35
3.1.1 JavaScript ohne jQuery	35
3.2 jQuery einbinden	37
3.2.1 jQuery online und offline nutzen	38
3.2.2 jQuery lokal einbinden	39
3.2.3 jQuery aus dem Google Repository einbinden	39
3.3 Unser Beispiel mit jQuery	41
3.4 Wir haben fertig	45

3.5	Das Mausereignis – Bindung eines Click-Events	46
3.5.1	Zunächst – die »aufdringliche« Variante	47
3.5.2	Etwas weniger aufdringlich bitte!	48
3.5.3	Ein unaufdringlicher Dreizeiler, dank jQuery	49
3.6	Give me more! – Verkettung von jQuery-Methoden	52
3.6.1	Den Elternknoten eines Elements manipulieren	53
3.7	Zusammenfassung	58

4 jQuery – die Übersicht 59

4.1	Im Zentrum – das jQuery-Objekt	59
4.1.1	Drei Arten von jQuery-Methoden	60
4.2	Die Funktion <code>\$()</code> und ihre Signatur	61
4.2.1	DOM-Element oder jQuery-Objekt als Argument	61
4.2.2	HTML-String als Argument	63
4.2.3	Callback-Funktion als Argument	65
4.2.4	CSS-Selektor als Argument	66
4.3	CSS-Selektoren für die primäre Collection	67
4.3.1	Die Basisselektoren	68
4.3.2	Mehrfachklassenselektor	68
4.3.3	Gruppen- und Kontextselektoren	69
4.3.4	Filterausdrücke für Selektoren	72
4.3.5	Inhaltsfilter	75
4.3.6	Sichtbarkeitsfilter	77
4.3.7	Attributfilter	79
4.3.8	Child-Filter	83
4.4	Accessoren – Eigenschaften der Collection	86
4.5	Traversieren – ausgehend von Collections	91
4.5.1	Was versteht man unter »Traversieren«?	92
4.5.2	jQuery-Methoden zur Achsen-Traverse	92
4.5.3	jQuery-Methoden zur erweiterten Achsen-Traverse	96
4.5.4	Filtern von Collections	104
4.5.5	Aufheben einer Filterung	109
4.5.6	Kopieren einer Collection	110
4.6	Events und Event Handling	112
4.6.1	Das Eventobjekt in JavaScript	113
4.6.2	Das jQuery-Eventobjekt	115
4.6.3	Allgemeine Methoden zu Eventbindung	121
4.6.4	Spezielle Eventbindungen (Convenience-Methoden)	126
4.6.5	Erzeugen von Events	129
4.6.6	Shortcut-Methoden für bestimmte Eventtypen	130
4.6.7	Binden sonstiger Events	134

4.7	Inhalte, Attribute, Datenspeicher	135
4.7.1	Lesen, Ändern und Entfernen von Attributen	135
4.7.2	Manipulation von Text- und Elementinhalt	137
4.7.3	Daten in Objekten speichern	141
4.8	Formulare verarbeiten mit jQuery	143
4.8.1	Filterausdrücke für Formularelemente	143
4.8.2	Filter für Zustände von Formularinputs	146
4.8.3	Binden von Events an Formularelemente	151
4.8.4	Serialisierung von Formulardaten	152
4.8.5	Extraktion von Formularfeldwerten	155
4.9	DOM-Manipulation	159
4.9.1	Methoden zum Einfügen von Knoten	161
4.9.2	Entfernen von Knoten	166
4.9.3	Ersetzen von Knoten	168
4.9.4	Wrapping-Methoden	169
4.10	CSS und Styleeigenschaften	173
4.10.1	Methoden für das class-Attribut	173
4.10.2	CSS-Eigenschaften manipulieren	179
4.10.3	Abmessungen von Containern	181
4.10.4	Position von Containern	182
4.11	Scrollen und Scrollposition	186
4.11.1	Animationen mit jQuery	189
4.11.2	Zeigen und Verstecken	190
4.11.3	Slides – Zeigen und Verstecken mit Animation	192
4.11.4	Fades – Zeigen und Verstecken über Opacity	196
4.11.5	Utility-Methoden für Animationen	199
4.11.6	Animation mehrerer CSS-Parameter	202
4.11.7	Die Queue – Warteschlange für Effekte	211
4.11.8	Vollständiges Beispiel zu .queue()	214
4.11.9	Utilities für Queue und Animationen	216
4.12	Ajax & JSON	218
4.12.1	Grundlagen zu Ajax	219
4.12.2	Daten und Datentypen für Ajax	223
4.12.3	jQuery und Ajax	225
4.12.4	Low-level Ajax Utilitits	225
4.12.5	Utilities und Convenience-Methoden	234
4.12.6	Globale Handler-Methoden	245
4.13	Utilities – praktisches Dies und Das	248
4.13.1	Konfliktvermeidung mit anderen Frameworks	249
4.13.2	Browser und Feature Detection	255
4.13.3	Utilities zur Array-Verarbeitung	256
4.13.4	Utility zur Stringbearbeitung	261

4.13.5	Utilities für DOM-Knotenverarbeitung	262
4.13.6	Utilities für Funktionsaufrufe	265
4.13.7	Objektverarbeitung und Erweiterung von jQuery	267
4.13.8	Test-Utilities	272

5 jQuery – der Praxiseinsatz 277

5.1	Schönere Navigationen	278
5.1.1	Die FlyOut-Navigation	279
5.1.2	Die Tabs: Karteireiter	290
5.1.3	Das Akkordeon	294
5.1.4	Die Spaltennavigation	298
5.1.5	Von der Spaltennavigation zum Drill-Menü	307
5.1.6	Das Tree Menu	309
5.1.7	Kleines Helferlein: dynamische Sitemap	313
5.1.8	Zusammenfassung	314
5.2	Von Tooltips bis Sprites	315
5.2.1	Tooltips	315
5.2.2	Links sammeln, im Footer ausgeben	318
5.2.3	Die Kobolde auf meinem Bildschirm	319
5.3	Spiel mit Bildern	326
5.3.1	Galerie I: einfache Slideshow	329
5.3.2	Galerie II: Imagebox	337
5.4	Ausgewählte Plugins	348
5.5	Ajax mit jQuery einsetzen	351
5.5.1	Laden von HTML-Elementen	352
5.5.2	Laden von JSON	364
5.5.3	Laden von JSONP	369
5.5.4	Zusammenfassung	372
5.6	Plugin-Entwicklung	372
5.6.1	Eigene Plugins entwickeln	372
5.6.2	Allgemeines Muster eines jQuery-Plugins	381
5.7	jQuery UI	382
5.7.1	Themeroller	384
5.7.2	Theme manuell bearbeiten	386
5.7.3	Ein kleines Beispiel: Hintergrundfarbe animieren	388
5.7.4	Klassen animieren	390
5.7.5	jQuery UI-Dialog	391
5.7.6	jQuery UI Accordion	394
5.7.7	Exkurs: Easing Equations	396
5.7.8	Zusammenfassung	397
5.8	Formulare beherrschen mit jQuery	398

5.8.1	Formulare validieren	398
5.8.2	Formulare senden mit Ajax	404
5.8.3	Datepicker	407
5.8.4	Nächste Ausbaustufe: Autocomplete	410
5.9	Flexiblere Tabellen mit jQuery	413
5.9.1	Die Zebra-Tabelle	414
5.9.2	Die Tabelle sortieren	417
5.9.3	Paginierung von Tabellen	424
5.9.4	Grid-Plugins	428
5.9.5	Zusammenfassung	433
5.10	Browser und Fenster	433
5.10.1	Cookies	434
5.10.2	History des Browsers	440
5.10.3	Flashfilme einbinden mit jQuery	443
5.10.4	Scrolling	446
5.10.5	Scrollen mit Geschichte	450
5.10.6	Das jQuery.ScrollTo-Plugin	454
5.11	Going mobile mit jQTouch	460
5.11.1	Emulatoren und IDEs für Mobilgeräte	461
5.11.2	Emulatoren und IDEs für iPhone	462
5.11.3	Ins mobile Web mit jQuery und jQTouch	466
5.11.4	Konfiguration von jQTouch	478
5.11.5	Das jQTouch-Objekt referenzieren	481
5.11.6	Zusammenfassung	483
6	Unit Tests	485
6.1	Ein Blick auf QUnit	486
6.2	Assertions	487
6.2.1	Test mit .ok()	488
6.2.2	Test mit .equals()	488
6.2.3	Test mit .same()	489
6.2.4	Module	489
6.3	Alle QUnit-Methoden im Überblick	490
6.3.1	Setup	490
6.3.2	Assertions	491
6.3.3	Asynchrone Tests	491
Anhang		493
A	HTML und CSS	495
A.1	Trennungen – Struktur, Präsentation, Verhalten	495

A.2	(X)HTML – Beschreibung der Struktur	497
A.2.1	Was ist eine Markup-Sprache?	497
A.2.2	Grammatik und Dokumenttyp	497
A.2.3	HTML vs. XHTML	498
A.3	Aufbau von HTML- und XHTML-Dokumenten	499
A.3.1	Aufgaben des Dokumentkopfs	501
A.3.2	Der Dokumentrumpf – strukturierte Information	502
A.3.3	Semantischer Grundaufbau eines Dokuments	508
A.4	CSS – Beschreibung der Präsentation	511
A.4.1	Einbindung von CSS in ein HTML-Dokument	511
A.4.2	Aufbau einer CSS-Anweisung	514
A.4.3	CSS-Selektoren – die wichtigsten Grundformen	515
A.4.4	CSS-Selektoren in der Praxis	518
A.4.5	CSS-Selektoren in der Übersicht	520
A.4.6	Neue Selektoren in CSS 3	521
A.4.7	CSS-Eigenschaften	523
A.4.8	Dokumentflow	524
A.4.9	Positionierung	525
A.4.10	Floats	529
A.4.11	Statische Präsentation dynamisieren	532
B	JavaScript und DOM	535
B.1	JavaScript – Beschreibung des Verhaltens	535
B.1.1	Grundlagen	536
B.1.2	Kontrollstrukturen – Bedingungen und Schleifen	540
B.1.3	Funktionen	542
B.1.4	Der Scope von Variablen	544
B.1.5	Closures	546
B.1.6	Objekte	551
B.1.7	Konstruktorfunktionen für Objekte	553
B.1.8	Funktionen als Objekte	557
B.1.9	»Unobtrusive« JavaScript	559
B.2	Die Synthese – das Document Object Model	563
B.2.1	Das Erstellen des DOM-Baums	563
B.2.2	Das »Schmücken« des DOM-Baums	565
B.2.3	Manipulation von DOM und CSS per JavaScript	566
C	Inhalt der DVD-ROM	569
C.1	Verzeichnis Listings	569
C.2	Verzeichnis jQuery	569
C.3	Verzeichnis Software	569
	Index	571

»The biggest benefit of a library is two-fold: it should remove the user from dealing with browser issues, and it should provide the user with an API that will simplify their development.«
John Resig – jQuery Mastermind

Vorwort

Sie wollen Webauftritte planen, gestalten, entwickeln und dem Benutzer eine aufregende, moderne Oberfläche bieten, die Informationen im Hintergrund lädt, ohne die gesamte Website neu zu laden? Sie wollen Ihrer Site eine dynamische, hierarchisch aufgebaute Navigation hinzufügen, die zuverlässig funktioniert und die einzelne Ebenen per Mausbefehl öffnet? Eine benutzerfreundliche Bildergalerie in Ihre Website integrieren, die auf Befehl eine vergrößerte Ansicht eines Vorschaubilds lädt und einblendet? Daten in eine bereits aufgerufene Webseite laden und als Tabelle anzeigen? Ein Shopsystem aufbauen und die Formulareingaben validieren sowie in Abhängigkeit von Benutzereingaben zusätzliche Formularfelder sperren oder anzeigen? Sie wollen sich für all das aber nicht in kryptische Programmierarbeiten vertiefen müssen?

Dann ist *jQuery* Ihr Freund. *jQuery* hilft Ihnen, fortschrittliche Benutzerschnittstellen zu schaffen, aufregende, animierte, interaktive Websites zu entwerfen. *jQuery* vereinfacht den Entwicklungsprozess, nimmt Ihnen lästige Routinearbeiten beim Programmieren ab und eröffnet Ihnen Möglichkeiten, von denen Sie gar nicht gewusst haben, dass ein Browser sie beherrscht.

jQuery steuert dabei keine neuen JavaScript-Funktionalitäten bei, sondern vereinfacht lediglich den Umgang mit allen, besonders den schwer zu nutzenden JavaScript-Methoden. *jQuery* wird so zum Leitfaden, der den Zugang zu JavaScript beinahe zum Kinderspiel macht.

Darüber hinaus besitzt jeder Browser seine Eigenheiten beim Ausführen von JavaScript. Aber das »weiß« *jQuery*. Und sorgt dafür, dass Sie sich um die Verwirklichung Ihrer Ideen kümmern können, anstatt sich mit den verschiedenen Bugs und Tücken bestimmter JavaScript-Engines herumzuärgern. Ohne Übertreibung möchten wir behaupten, dass Webprogrammierung noch nie so einfach war wie mit einem Framework wie *jQuery*.

An wen richtet sich dieses Buch?

Idealerweise haben Sie Kenntnisse in HTML und CSS sowie JavaScript. Mindestens sollten Sie aber wissen, um was es sich bei diesen Begriffen handelt. Es ist eine besondere Herausforderung für die Autoren dieses Buches, auch diejenigen anzusprechen, die kaum Programmierkenntnisse besitzen. Selbstverständlich sind Leser willkommen, die sich einen Background angeeignet haben und ihre Kenntnisse vertiefen wollen.

Einsteiger, Designer, Neugierige

Sie werden vielleicht ganz einfach und unbefangen in die Möglichkeiten des dynamischen, interaktiven Webdesigns eintauchen wollen. Vielleicht sind Sie ja Designer. Sie werden vielleicht niemals perfekte Webdeveloper werden wollen, aber Sie wollen jQuery als Inspirationsquelle nutzen, um aufregende User Interfaces zu gestalten. In den folgenden Kapiteln werden Sie sich ganz komprimiert Grundkenntnisse in HTML und JavaScript aneignen können.

So sollten auch Einsteiger die Möglichkeit haben, jQuery kennenzulernen, ohne großartig Standardwerke für HTML pauken zu müssen. Einsteigern sei gesagt, Sie werden über einige Begriffe stolpern, die nicht erklärt werden können, weil das den Rahmen des Buches sprengen würde. Aus Erfahrung raten wir Ihnen: Lesen Sie zunächst über diese Begriffe hinweg, oder lesen Sie das entsprechende Kapitel ein zweites oder sogar ein drittes Mal. Oder Sie recherchieren im Internet. Jedenfalls gibt es keinen Grund, sich entmutigen zu lassen – Sie werden hier mit Sicherheit zumindest die grundlegenden Zusammenhänge verstehen lernen.

Entwickler, Webdeveloper

Webdeveloper und Webdesigner, die bisher JavaScript ohne Frameworks eingesetzt haben, werden auf ihre Kosten kommen. Sie werden jQuery in aller Ausführlichkeit kennenlernen, sämtliche Methoden und Eigenschaften und Eigenheiten werden Ihnen präsentiert. Sie können dieses Buch als Nachschlagewerk verwenden oder als Material zum Selbstunterricht einsetzen. Nach der Lektüre dieses Buches werden Sie eigene Plugins entwickeln können und einen kurzen Einblick in die Entwicklung von Testroutinen mittels QUnit erhalten haben.

Entwickler, die ein Web Interface für eine Anwendung entwickeln oder planen sollen, werden genug Material finden, um an ihm weiterzuarbeiten. Interaktionen mit Formularen und die Kommunikation via Ajax mit all seinen Datenformaten mit dem Server werden ebenfalls ein Thema sein.

Was finden Sie in diesem Buch?

Um JavaScript-Einsteigern einen Zugang zu verschaffen, stehen Ihnen am Ende des Buches Anhang A, »HTML und CSS«, und Anhang B, »JavaScript und DOM«, hilfreich zur Seite. Dort finden Sie eine einfache Einführung in HTML, CSS und die JavaScript-Sprachkonstrukte. Erwarten Sie aber kein Komplettpaket: Es werden nur die Sprachelemente behandelt, die Sie für die jQuery-Beispiele dringend benötigen. Falls Sie eine umfassende Einführung in diese Themen erwarten, gibt es dafür andere Werke, die diesen Zweck besser erfüllen. Wenn Sie diese Kenntnisse bereits besitzen, können Sie diesen Bereich getrost ignorieren.

Die *jQuery-Übersicht* verschaffen Sie sich im nahezu gleichnamigen Kapitel. Sie können es als kommentierte API-Referenz verwenden oder als systematische Anleitung mit Hinweisen aus der Praxis.

Wenn Kapitel 4, »jQuery – die Übersicht«, eher einem Wörterbuch entspricht, kommt der Praxisteil dem Vergleich mit einer Grammatik nahe. Hier werden Sie die Sprachkonstrukte anwenden. Während die *jQuery-Übersicht* die Systematik der jQuery-API akribisch widerspiegelt, orientiert sich der Praxisteil an den Anforderungen des Webdesigns. Die *jQuery-Übersicht* teilt sich auf in die Module des jQuery Frameworks, die unterschieden sind in die Bereiche *Core*, *Traversing*, *Manipulation*, *Effects*, *Events*, *Utilities*. Der Praxisteil in Kapitel 5, »jQuery – der Praxiseinsatz«, gliedert sich in Navigationen, Content, Ajax, Plugins, User Interface, Formularen, Tabellen und Browsern, besitzt also eine sehr anschauliche Struktur. Der Praxisteil ist im Stil von Tutorials gehalten.

Es macht Ihnen niemand einen Vorwurf, wenn Sie dieses Buch kreuz und quer lesen, sich Beispiele ansehen und ausprobieren und die Sprachkonstrukte in Kapitel 4 nachschlagen. Im Gegenteil, diese Methode sei hier ausdrücklich empfohlen.

Die digitalen Codes der Beispiele dieses Buches finden Sie auf der dem Buch beiliegenden Begleit-DVD. Fühlen Sie sich frei, sie für Ihre Zwecke zu verwenden. Lediglich das Bildmaterial auf der DVD, mit dem einige Demos erstellt wurden, darf nicht ohne ausdrückliche Genehmigung der Urheber für kommerzielle oder private Zwecke verwendet werden.

Und nun an die Arbeit ...

Wir möchten auch diejenigen Leser animieren, die bisher aufgrund der scheinbar kryptischen Schreibweise einer Programmiersprache davor zurückgeschreckt sind, sich mit jQuery näher zu befassen. So mancher Designer wird neue Mög-

lichkeiten entdecken, um sein Projekt mit interaktiven Features zu bereichern. Mit einfachen, manchmal nur dreizeiligen Codeblöcken lässt sich eine beeindruckende Wirkung erzielen. Lassen Sie sich inspirieren! jQuery wird Ihre tägliche Arbeit bei der Entwicklung neuer Websites umfassend verändern. Und nun möchten wir Sie einfach hereinbitten. Viel Spaß beim Erlernen von jQuery!

Maximilian Vollendorf
Frank Bongers

»John Resig is a JavaScript Evangelist for the Mozilla Corporation and the creator and lead developer of the jQuery JavaScript library. This library's goal is to simplify the process of writing JavaScript code that is compatible with all web browsers. For his work on jQuery he was inducted into the Rochester Institute of Technology's Innovation Hall of Fame on April 30, 2010.« – Wikipedia

1 jQuery kennenlernen

Beginnen sollte man dieses Buch mit einem Blick auf jQuery im Speziellen und JavaScript-Frameworks im Allgemeinen. Auch einen kleinen Rückblick auf die Entstehung von jQuery werden wir werfen sowie einen Eindruck von dessen einfacher Handhabung vermitteln.

1.1 Was jQuery alles kann

Der Begriff *Framework* ist im Vorwort bereits gefallen. Aber was ist eigentlich ein Framework? Ein Framework ist kein fertiges Programm. Es stellt lediglich Erweiterungen einer Programmiersprache bereit, die einem Anwendungsentwickler bestimmte, immer wiederkehrende Aufgaben abnehmen. Ein Beispiel aus der wirklichen Welt: Sie können als Möbelmonteur natürlich fast jeden Schrank der Welt mit nur einem einzigen Schraubenzieher montieren. Es dauert, aber es funktioniert. Haben Sie aber einen Akkuschrauber, übernimmt dieser für Sie die immer wiederkehrenden, lästigen Drehbewegungen. Und mit den mit dem Schrauber ausgelieferten Aufsätzen können Sie sogar neue, noch völlig ungeahnte Aufgaben bewältigen.

Genauso verhält es sich mit einem Framework. Es ist ein »Toolset«, das Ihre Arbeit erleichtert. Wobei ein Framework mehr ist als nur ein loses Set an Werkzeugen: Es ist eine Gesamtheit von kooperierenden Methoden mit einer zusammenhängenden Struktur. Wenn Sie Frameworks verwenden, müssen Sie sich mit dem Inneren dieser Logik nicht mehr auseinandersetzen, sondern nur damit, wie Sie die einzelnen Methoden verwenden und mit welchen Werten sie gefüttert werden wollen. So funktioniert auch jQuery.



Abbildung 1.1 jQuery ist unser Universalwerkzeug zur Webprogrammierung.

► **Elemente finden**

jQuery stellt Werkzeuge zur Navigation durch den (X)HTML-Dokumentenbaum zur Verfügung. Einzelne Elemente oder Gruppen von Elementen zu adressieren wird so einfach wie das Schreiben von CSS-Selektoren – genauer: jQuery verwendet hierfür tatsächlich CSS-Selektoren.

► **Elemente manipulieren**

Sie können mit jQuery sämtliche Bestandteile einer Seite manipulieren, darüber hinaus aber auch neue Elementknoten, neue Attribute, neue CSS-Klassen und -Stile hinzufügen, Sie könnten, wenn Sie das wollten, ein Webdokument vollkommen umkrempeln.

► **Inhalte einfügen**

Mit wenigen Codezeilen lassen sich Daten via Ajax in ein bestehendes Dokument einfügen, ohne es dafür neu laden zu müssen. Jede beliebige HTML-Seite lässt sich hierbei als Datenquelle verwenden – es sei denn, Sie ziehen es vor, stattdessen Daten im XML-Format oder in Form von JSON-Objekten einzusetzen. jQuery lässt Ihnen die Wahl.

► **Bestandteile einer Seite animieren**

Auch zur Erzeugung komplexer Animationen genügt in jQuery ein einfacher Aufruf – Sie können Seitenelemente zeigen oder verstecken, ein- oder ausblenden, aufklappen oder schließen, hervorheben oder sich über die Oberfläche bewegen lassen. Interaktive Dynamik? Kein Thema!

► **Event Handling**

Ein weiteres wichtiges Thema in jQuery ist das Event Handling. Beispiel Mausereignisse: Ohne im HTML-Quelltext Attribute wie `onmouseover` bemühen zu

müssen, können Sie Events an ein dediziertes Element binden und auch wieder von ihm lösen. Unobtrusive? Aber sicher!

► **... und noch mehr**

Genügt Ihnen das alles noch nicht, können Sie schließlich auch eigene Erweiterungen schreiben und an jQuery »andocken«. Aber auch die Erweiterungen anderer Nutzer stehen Ihnen entsprechend zur Verfügung – die sehr aktive Community von jQuery bietet hier eine breite Vielfalt an.

► **Gutes, altes JavaScript – nur besser**

Letztlich können Sie noch immer alle Möglichkeiten nutzen, die bewährtes, herkömmliches, clientseitiges Scripting auch bietet. Die Bibliothek nimmt Ihnen aber viele mühselige Programmieraufgaben ab. Manchmal zeigt Ihnen jQuery dabei ungeahnte Wege, die selbst ältere Browser Kunststücke vollbringen lassen, die eigentlich unmöglich schienen.

► **Benutzeroberfläche**

Mit einem eigenen Aufsatz, dem jQuery UI Framework, können Sie komplexe Benutzerschnittstellen benutzen. Von Drag & Drop-Interaktionen bis hin zu Datepickern und Navigationen reichen die Bausteine, und es werden noch viele Widgets hinzukommen.

1.2 Ein Framework? Eine Community!

Es ist der einfache Zugriff nicht nur auf alle Elemente eines Dokuments, sondern auf viele Scriptfunktionen des Browsers, die Frameworks wie jQuery so faszinierend machen. Zum gegenwärtigen Zeitpunkt ist jQuery das beliebteste und am häufigsten angewendete JavaScript-Framework und hat die Konkurrenz wie Dojo, Ext JS, Prototype und scripta.culo.us weit hinter sich gelassen. Seine Bedeutung kann auch daran gemessen werden, dass es im Rahmen der Websites vieler namhafter Firmen zum Einsatz kommt – zu nennen wären hier u.a. Google, Dell, Intel, Amazon, Oracle oder die Mozilla Foundation, um nur einige zu nennen.

Wenn hier die Rede davon ist, dass sich jQuery wie eine »Schicht« zwischen JavaScript-Programm und Script-Engine des Browsers setzt, hört sich das komplizierter an, als es ist: Es wird einfach eine Scriptdatei in eine HTML-Seite eingebunden, bevor die Scripte des Webentwicklers an die Reihe kommen. Und in diesen Scripten greift der Webentwickler auf die kurzen und knappen jQuery-Befehle zurück, die ein wahres Feuerwerk veranstalten.

Die Beliebtheit von jQuery rührt auch daher, dass seine Programmierschnittstelle so klar und zielgerichtet gehalten ist, dass sie einfach zu verstehen ist. Der Clou ist, dass jQuery auf diesem Weg *erweiterbar* ist, also mit den Anforderungen »mit-

wächst«. Um jQuery hat sich dadurch eine große Community entwickelt, die eine enorme Kreativität an den Tag legt. Fast täglich werden neue innovative jQuery-Erweiterungen veröffentlicht, jQuery wird immer öfter standardmäßig in Webapplikationen integriert, und kaum jemand spricht mehr davon, dass JavaScript »Teufelszeug« sei. Verschämt wurde auch der Begriff *DHTML* fallen gelassen; heute spricht man von *DOM-Scripting*, und selbst komplexe Scripte sind aus dem Weballtag nicht mehr wegzudenken.

1.3 Nicht ohne mein JavaScript

Nachdem es jetzt klar ist, *was* ein Framework grundsätzlich ist, mag es interessant sein, detailliertere Gründe für die Entstehung zu erfahren, also *woher* die JavaScript-Frameworks kommen und *warum* sie existieren. JavaScript stellt bekanntermaßen neben (X)HTML und CSS die »dritte Säule« für die Erstellung aktueller Webseiten dar und sorgt (im Rahmen der Applikationsschicht) für Dynamik und Interaktivität eines Webdokuments.

Die Notwendigkeit, eben diese Dynamik und Interaktionsmöglichkeit zu bieten, ist in den vergangenen Jahren immer mehr in den Vordergrund getreten. Entsprechende Features gelten nicht mehr als exotisch (oder überflüssig), sondern sind Teil der Erwartungshaltung des Nutzers geworden: Kaum eine Website kommt mehr ohne dynamisch gesteuerte Menüs, Slideshows, interaktive Oberflächen oder das Nachladen von Inhalten via Ajax aus.

Ein Webprogrammierer ist also heutzutage mit vielerlei Standardanforderungen und -problemen konfrontiert. Er wird daher nicht jedes Mal das »Rad neu erfinden« wollen, sondern sich eine Bibliothek aus bewährten Standardlösungen zusammenstellen, aus der er bei Bedarf entsprechende Bausteine entnimmt.

Für die Frühzeit des DHTML ergab sich auf diesem Weg eine Vielzahl von Insel-Lösungen, was gut funktionierte, solange kombinierte Lösungen aus der gleichen Quelle kamen. Da meist nur ein Programmierer an einer Website arbeitete und man selten externe Programme einband, waren alle zufrieden.

Allerdings hat sich die Situation auch in dieser Beziehung geändert – in der heutigen Zeit bezieht eine Website ihre JavaScript-Daten aus verschiedenen Quellen, eine Vielzahl von Programmierern ist zu verschiedenen Zeiten beteiligt und stellt Lösungen für Teilprobleme zur Verfügung.

Einerseits treten hierbei Redundanzen auf, andererseits steigt das Potenzial für Inkompatibilitäten. Wie stellt man sicher, dass sich zwischen zwei Teilprogrammen nicht Konflikte ergeben – und sei es aus trivialen Gründen wie gleich be-

nannten Variablen? Alle beteiligten Scripte entsprechend zu koordinieren ist fast unmöglich, zumindest aber aufwendig und nicht sonderlich effektiv.

1.3.1 Gründe für das Entstehen von Frameworks

Die Redundanzen zwischen Teilprogrammen ergeben sich aus den Grundproblemen, denen sich Programmierer im Webbereich ausgesetzt sehen. Erstens ist das Handwerkzeug, das JavaScript standardmäßig zur Bewältigung von Aufgaben im Rahmen des Document Object Modells (DOM) bietet – und fast alle aktuellen Aufgabenstellungen sind mit dem DOM verknüpft –, zumindest unzureichend. Man möchte dieses Handwerkzeug verbessern (was möglich ist), geht hierbei aber verschiedene Wege.

Zweitens bestehen (auch jetzt noch) Unterschiede zwischen den Zielbrowsern. Diese werden größer, je älter die ins Visier genommenen Plattformen sind. Muss man (was niemandem zu wünschen ist) beispielsweise die Funktionstüchtigkeit einer Website auf älteren Browsern wie Internet Explorer 5 oder 6 gewährleisten, ist eine Reihe von Workarounds erforderlich. Da auch hier wieder verschiedene Wege gegangen werden und die Komplexität sich multipliziert, sobald weitere Browser berücksichtigt werden müssen, führt dies zu einem enormen Testaufwand.

Es entstand schon früh der Wunsch, beide Bereiche vom Programmieralltag abzukoppeln und hierfür Standardlösungen in Form von Bibliotheken anzubieten. Ein erster Ansatz war die *DynApi* von Dan Steinman. Auch andere Experten wie Douglas Crockford, Sam Stephenson oder Dean Edwards machten sich Gedanken, wie man JavaScript zusätzliche Funktionalitäten hinzufügen könnte, oder (u.a.) die disparaten Eventmodelle der Browser von außen vereinheitlichen könnte. Weitere Bibliotheken wie *Base Library*¹ (Edwards) oder *Prototype*² (Stephenson) entstanden, die die Grundlage dafür bildeten, was heutzutage als »Framework« bezeichnet wird.

Diese und andere Bibliotheken hatten den Anspruch, Zugriffe auf HTML-Elemente zu vereinfachen oder dynamisch sogenannte *Layer* zu erzeugen, die animiert werden sollten. Der Webentwickler sollte Werkzeuge benutzen können, um mittels einfacher JavaScript-Aufrufe dynamische, effektvolle Anwendungen zu schreiben. Damit sollten sich für die damals verbreiteten Browser wie den Netscape Navigator und den Internet Explorer einfach zu implementierende Navigationen oder dynamische Bildergalerien realisieren lassen.

1 Siehe: <http://dean.edwards.name/base/> oder auch <http://code.google.com/p/base2/>

2 Siehe: <http://www.prototypejs.org/>

1.3.2 Nochmals – was ist ein Framework?

Übersetzen ließe sich der Begriff »Framework« mit »Rahmenwerk« oder schlicht »Rahmen«. Man kann es sich als ein speziell errichtetes Gerüst vorstellen, um damit ein Programm zu erstellen und dient (wie bereits gesagt) als zusätzliche Vermittlungsschicht zwischen jenem Programm und der Außenwelt. Das Framework besteht ebenfalls aus Programmdateien, die in der gleichen Sprache geschrieben sind, wie die »inneren« Programme, also (soweit wir Scriptsprachen betrachten) mit diesen zusammen in der gleichen Laufzeitumgebung betrieben werden. Ein JavaScript-Framework ist daher auch »nur« ein in JavaScript geschriebenes Programm – allerdings ein recht komplexes, wie wir sehen werden. Wie wir ebenfalls sehen werden, ist die Anwendung eines Frameworks dagegen (zum Glück) *nicht* sehr komplex.

1.3.3 Aufgaben eines Frameworks

Zu den Aufgaben eines Frameworks gehört die Überbrückung von Unterschieden zwischen Browsern, wie sie in der Interpretation des DOM und des Event Handlings bestehen. Zusätzlich nivelliert es die Differenzen im Funktionsumfang der JavaScript-Versionen, die sich aus den verschiedenen Implementierungen der Browser ergeben – als Beispiel seien die Array-Methoden aus JavaScript 1.6 genannt, die über ihre Framework-Versionen älteren Browsern nachträglich »aufgefropft« werden können.

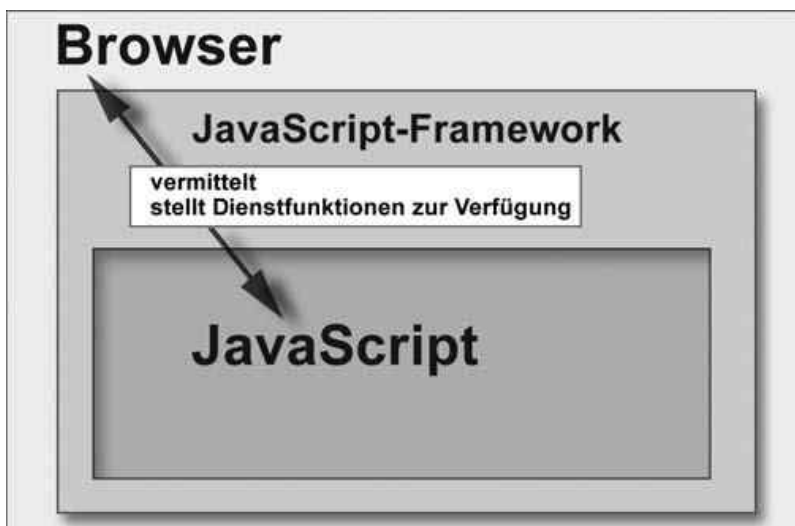


Abbildung 1.2 Framework vermittelt zwischen Browser und JavaScript-Code

Der von Haus aus umständliche Umgang mit dem DOM wird durch Frameworks vereinheitlicht und erleichtert. Von den regulären DOM-Funktionen abgeleitet, werden weitere und simpler zu handhabende Utility-Funktionen zur Verfügung

gestellt, die DOM-orientierte Programmierung stark vereinfachen. Auch für andere häufig wiederkehrende Programmieraufgaben (beispielsweise Ajax-Anwendungen) stellt das Framework Routinen zur Verfügung. Diese Aspekte bilden die Basisfunktionalitäten von jQuery, die in den ersten Kapiteln dieses Buches beschrieben werden.

Der zweite große Aufgabenbereich eines Frameworks besteht in Hilfen zur Generierung interaktiver Oberflächen (sogenannter *Widgets*), die zur Gestaltung von User Interfaces eingesetzt werden. Es kann sich dabei um Formular-Widgets (funktional »aufgebohrte« Formularelemente wie Datepicker oder intelligente Dropdown-Menüs) oder Oberflächen-Widgets (wie Tabsteuerung oder Akkordeonelemente), die die Bedienoberfläche der Webseite bilden. Ein Framework gibt dem Webprogrammierer auf diese Weise Mittel an die Hand, einfach und sicher ein dynamisches Web Interface zu programmieren, das modernen Usererwartungen entspricht. Dieser Part wird bei jQuery durch die Bibliotheken aus jQuery UI (User Interface) abgedeckt, die im Praxisteil des Buches behandelt werden.

1.3.4 Aktuelle Frameworks für JavaScript

Frameworks, gerade solche für JavaScript, gibt es heutzutage quasi »im Dutzend«. Unter diesen kristallisieren sich derzeit einige wenige heraus, die häufiger zum Einsatz kommen. Hierzu zählen das auf *Prototype* aufsetzende *script.aculo.us*, das von Google zur Verfügung gestellte *GWT* (Google Web Toolkit), dessen Konkurrenz *YUI* (Yahoo User Interface Library), der Bolide *Dojo*, das schlanke *jQuery* und schließlich *Ext JS* und *MooTools*.

Allen gemeinsam ist, dass sie als *Open Source* zur Verfügung gestellt werden und (in den meisten Fällen) eine Entwicklercommunity mehr oder weniger koordiniert mit der Weiterentwicklung des Frameworks beschäftigt ist.

Name	Betreuer	Quelle
script.aculo.us	Community	http://script.aculo.us/
GWT	Google	http://code.google.com/webtoolkit/
YUI	Yahoo	http://developer.yahoo.com/yui/
Dojo	Community	http://www.dojotoolkit.org/
jQuery	Community	http://jquery.com/
Ext JS	Community	http://www.extjs.com/
MooTools	Community	http://mootools.net/

Tabelle 1.1 Aktuelle JavaScript-Frameworks

Ihre Berechtigung haben alle diese Frameworks. Eine Bewertung, sofern eine solche überhaupt sinnvoll ist, soll an dieser Stelle nicht vorgenommen werden. Da Sie dieses Buch in Händen halten, haben Sie sich vermutlich ohnehin bereits entschieden (ebenso wir als Autoren dieses Buches) – werfen wir also nun einen Blick auf jQuery im Besonderen, auf seine Entstehung und seine Position unter all den genannten Frameworks.

Bibliothek oder Framework?

Die Meinungen, wann für eine JavaScript-Anwendung wie jQuery der Begriff *Framework* oder der Begriff *Bibliothek* (Library) verwendet werden soll, gehen auseinander. Wir haben uns in diesem Buch für den Begriff *Framework* entschieden. Nach der strengen Definition beinhaltet ein Framework auch Routinen für User-Interface-Gestaltung. Dies trifft zwar nicht auf den jQuery-Core, jedoch für jQuery UI zu.

1.3.5 Frameworks – ein Rückblick

Die frühen JavaScript-Bibliotheken wie Steinmans *DynApi* waren ambitioniert, scheiterten aber kläglich. Die Browser waren einfach zu fehleranfällig, was dazu führte, dass JavaScript alles andere als beliebt war. Von Konzepten wie der Trennung von Struktur, Layout und Verhalten war man weit entfernt, *unobtrusive JavaScript* war ein noch unbekanntes Konzept. JavaScript hatte das Image einer minderen Scriptsprache, mit der vielleicht irgendwelche Informationen in die Statuszeile des Browsers hineingeschrieben werden können oder mit der man dem Nutzer ungeliebte Popups entgegenspringen lassen kann. Mehr nicht. So geschah es, dass die Entwicklung der DynApi im Jahre 2005 eingestellt wurde. Das dürfte auch die Zeit gewesen sein, als die Beliebtheit von JavaScript etwa genau so groß war wie die Beliebtheit von Trojanern.

Etwa zur gleichen Zeit, in der DynApi eingestellt wurde, machte sich John Resig, ein ambitionierter JavaScript-Nerd, Gedanken, wie man mit einer einfach zu lernenden Syntax und mittels JavaScript auf HTML-Elemente zugreifen könnte. Die Arbeiten von Steinman, Crockford und Edwards waren Resig selbstverständlich nicht unbekannt. Crockford und Edwards erreichten ihre Programmierziele durch Erweitern bestehender JavaScript-Objekte und der Nachimplementierung »fehlender« Sprachaspekte beispielsweise durch Prototyping.

Resig nahm sich nicht DynApi zum Vorbild, sondern studierte andere Frameworks wie *Prototype* oder *moo.fx*, die in jener Zeit wie Pilze aus dem Boden schossen. Und es war vor allem Dean Edwards Funktion *cssQuery*, die bereits den Grundgedanken einer CSS-Selektor-Engine aufgriff.

Resig übernahm die Idee, dass es mit CSS-Selektoren bereits eine solche Syntax gäbe, die man nur mit JavaScript kombinieren müsse und so einen Zugriff auf DOM-Elemente ermöglichen könne. CSS ist standardisiert, zudem ist sie bei Webentwicklern bekannt, die Lernkurve bei einem neuen JavaScript-Baukasten würde dementsprechend steil verlaufen. Ein halbes Jahr später, im Januar 2006, wurde eine erste Version des JavaScript-Werkzeugkastens vorgestellt, das durch seine Geschwindigkeit und kleine Dateigröße Aufsehen erregte (die konkurrierenden Frameworks waren um etwa den Faktor fünf größer).

Es sollte ein in JavaScript geschriebenes Toolset sein, mit dem man mittels leicht erlernbarer Anweisungen HTML-Elemente manipulieren, leicht Animationen realisieren und einfach Events wie Mausklicks oder Mouseover-Events an beliebige HTML-Elemente binden kann. Dieses Toolset setzt sich zwischen die Anweisungen, die ein Webentwickler schreibt, und der JavaScript-Engine des Browsers und ermöglicht es so, dass sich der Webentwickler nicht um komplizierte, aber immer wiederkehrende Scriptaufgaben kümmern muss. Dieses Framework sollte den Namen *jQuery* erhalten.

1.4 jQuery – viel mit wenig erreichen

Resig legte Wert auf die *Knappheit des Codes*, der nach Einbinden des Frameworks für den Programmierer zu schreiben bleibt, der dabei aber klar und verständlich bleiben sollte. Er erreichte dies durch Konzentration auf eine Kernfunktion (kürzestmöglich mit `$()` benannt), die sowohl die Selektion von DOM-Elementen übernimmt als auch die Methoden zur Verfügung stellt, diese anschließend zu manipulieren. Diese Codezeile selektiert ein verborgenes Div mit `id="d1"` und blendet es ein:

```
$("div#d1").show();
```

Zusätzlich gibt jede Methode sich selbst als Objekt zurück (auch als *jQuery-Objekt* bezeichnet). Die so erzielte *Verkettbarkeit* (Chaining) verkürzt die Syntax noch weiter. Das folgende Beispiel selektiert (erstens) alle `<div>`-Container des Dokuments, versteckt sie (zweitens), manipuliert (drittens) ihre CSS-Eigenschaften und blendet sie (viertens) anschließend über eine Slide-Animation wieder ein:

```
$("div").hide().css(color,"blue").slideDown();
```

Resig konzipierte sein Framework zudem nicht als abgeschlossene Applikation, sondern sorgte für dessen leichte *Erweiterbarkeit*. Das Prinzip der Plugins und eine hochaktive Community beschert jQuery eine reichhaltige Bibliothek an Zu-

satzfunktionalitäten, von denen die beliebtesten und stabilsten wieder in das Projekt einfließen – beispielsweise im Rahmen der Oberflächenbibliothek jQuery UI.

Daneben spricht für jQuery nach wie vor die Schlankheit des Frameworks selbst und des benötigten Codes. Allerdings muss erwähnt werden, dass der Umfang des einzubindenden JavaScripts abhängig ist von der Anzahl der benötigten Plugins – die oft zitierten 18 KB, die jQuery umfasst, beziehen sich lediglich auf die gepackte Core-Bibliothek.

Zudem offenbaren die verfügbaren Zusatzbibliotheken, sobald man die geprüften Gefilde der anerkannten jQuery UI-Plugins verlässt, durchaus einige Schwächen und Inkompatibilitäten. Resig nimmt dies, wie auch auftretende Redundanzen, zugunsten größerer Offenheit billigend in Kauf und geht davon aus, dass sich »die besseren« Plugins letztendlich durchsetzen werden.

»Because browser bugs are so frustrating and such a burden on top of normal development it should be the responsibility of every web developer to make sure that the browsers they develop for are able to find and fix their bugs. By taking responsibility for the bugs that you find – and to not assume that ,someone else will find it'– will accelerate the rate at which browsers can improve.« – John Resig in seinem Blog

2 Den Arbeitsplatz einrichten

Um mit jQuery zu arbeiten, sollten Sie sich eine Arbeitsumgebung einrichten. Neben einem Rechner und dem Betriebssystem benötigen Sie einen Editor und zum Testen der Ergebnisse einen Browser. Ein einfacher Texteditor reicht für die ersten Schritte aus. Wenn Sie allerdings ernsthaft Websites mit jQuery entwickeln wollen, sollten Sie einen Arbeitsplatz mit einer Entwicklungsumgebung (IDE) oder zumindest einen Editor mit Syntaxhervorhebung, verschiedenen Browsern und einem lokalen Webserver installieren. Wenn Sie noch mehrere Betriebssysteme zum Testen zur Verfügung haben, haben Sie eine perfekte Arbeitsumgebung.

Sie benötigen selbstverständlich eine Version des jQuery Frameworks. Wie sie zu benutzen ist, erfahren Sie in den nächsten Kapiteln.

Link zu jQuery

<http://jquery.com/>

2.1 Rechner und Betriebssystem

Sie sollten auf jeden Fall Hardware einsetzen, auf der die neuesten Browserversionen lauffähig sind. Letzteres bedingt auch halbwegs aktuelle Betriebssysteme: Unter Windows 2000 werden Sie kaum einen Google Chrome oder einen Internet Explorer 8 installieren können. Unter Mac OS X 10.4.x werden Sie ebenfalls keinen Google Chrome installieren können.

2.1.1 Windows

Wenn Sie unter Microsoft Windows arbeiten, haben Sie den Vorteil einer umfangreichen Auswahl an Entwicklungsumgebungen und Browsern, Sie können so ziemlich alle modernen Browser installieren. Selbst den Mac-Browser *Safari* gibt es in einer Windows-Version.

2.1.2 Mac OS X

Wenn Sie auf einem Apple-Rechner arbeiten, können Sie zwar keinen Internet Explorer installieren (die Mac-Version wurde bereits vor Jahren ausgemustert), aber Ihnen stehen mit den verfügbaren Virtualisierungen wie *Parallels Desktop* und *VMware Fusion* alle Möglichkeiten offen, Sie können sogar mehrere Windows-Installationen und auch Linux-Versionen parallel zum laufenden Mac OS X starten, wenn Ihr Arbeitsspeicher groß genug ist.

Das nötige Kleingeld sollten Sie bereithalten, da Sie neben der Lizenz für die Virtualisierungssoftware auch Lizenzen für die Betriebssysteme benötigen. Ohne Virtualisierung können Sie parallel zum Mac OS X eine Windows-Version über die Apple-eigene Lösung *Bootcamp* erstellen. Damit lässt sich aber nur sehr unkomfortabel arbeiten, da Sie zum Wechseln des Betriebssystems stets den Rechner neu starten müssen.

2.1.3 Linux

Sie können auch unter Linux eine Arbeitsumgebung aufbauen, auch auf den Internet Explorer müssen Sie nicht verzichten, hier steht Ihnen die Virtualisierungssoftware *VirtualBox* zur Verfügung.

Auf welchen Plattformen testen?

Sie sollten auf jeden Fall Ihre Arbeitsergebnisse auf mehreren Plattformen testen. Als Zielplattform sind natürlich die Windows-Versionen die wichtigsten, da die Verbreitung dieser Systeme am größten ist. In den letzten Jahren hat aber auch die Mac-Plattform an Bedeutung gewonnen. Es lohnt sich, auch für diese Plattform zu testen. Wohingegen Linux-User meistens diejenigen sind, die Applikationen entwickeln, die Verbreitung beschränkt sich hier also auf eine eingegrenzte Zielgruppe. Wenn es Ihnen jedoch möglich ist, sollten Sie auch für sie Ihre Webapplikationen testen.

Links zu Virtualisierungssoftware für den Mac

- ▶ <http://www.vmware.com/de/products/fusion/>
- ▶ <http://www.parallels.com/de/products/desktop/>
- ▶ <http://www.virtualbox.org>

2.2 Browser

Wichtiger als Tests für verschiedene Betriebssysteme ist es, in den wichtigsten Browsern und Versionen zu testen. Sie sollten unablässig immer wieder Ihre Ergebnisse in den Browsern aufrufen, auch wenn Ihnen jQuery die Aufgabe abnimmt, browserspezifische Implementierungen zu berücksichtigen, der Teufel steckt auch hier im Detail.

2.2.1 Internet Explorer

Der Internet Explorer ist noch immer der am meisten eingesetzte Browser. Die Versionen, mit denen Sie testen sollten, sind die Versionen 6 bis 8. Downloads finden Sie unter <http://www.microsoft.com/downloads/de-de/default.aspx>.

Der Internet Explorer 6

Die Verbreitung dieses Browser beträgt zurzeit, also Mitte 2010, etwa 7 bis 16 %. Die verschiedenen Browserstatistik-Anbieter verwenden unterschiedliche Messmethoden, pauschal kann man diese Frage nicht beantworten, je nach Land, Zielgruppe oder Kriterien wie beruflich/privat variieren die Zahlen, aber immerhin noch 7 bis 16 % aller Nutzer setzen den Internet Explorer 6 ein. So manche Website muss diesen Browser noch unterstützen, so sehr das den Webentwicklern auch aufstoßen mag.

2.2.2 Firefox

Der Mozilla Firefox liegt aktuell (Mitte 2010) in der Version 3.6 vor, die Version 4 steht bereits vor der Tür. Sie soll Ende des Jahres freigegeben werden. Offiziell unterstützt jQuery bereits die Versionen ab 2. Die Verbreitung des Firefox 2 liegt bei ungefähr 0,8 % der Webnutzer. Es hängt von Ihren konkreten Projekten ab, ob Sie diesen Browser noch unterstützen sollten, in den meisten Projekten kann man ihn unberücksichtigt lassen. Es gibt viele nützliche Tools für den Firefox, um das Entwickeln von Websites zu erleichtern.

Tipp: Web Developer und Firebug

Das Web Developer Plugin für den Firefox stellt übrigens hervorragende Werkzeuge zur Analyse von Webdokumenten zur Verfügung, genauso wie das Plugin Firebug, das es ermöglicht, JavaScript- bzw. jQuery-Code zu analysieren und zu debuggen. Andere Browser stellen ebenfalls geeignete Werkzeuge zur Verfügung. Safari besitzt seit der Version 4 bereits von Hause aus Entwicklerwerkzeuge, die in der Version 5 entscheidend verbessert wurden. Der Internet Explorer 8 besitzt ebenso Entwicklertools.

Mehr Informationen zum Firefox und zum Web Developer Plugin:

- ▶ <http://chrispederick.com/work/web-developer/>
- ▶ <https://addons.mozilla.org/de/firefox/>
- ▶ <http://www.mozilla-europe.org/de/firefox/>

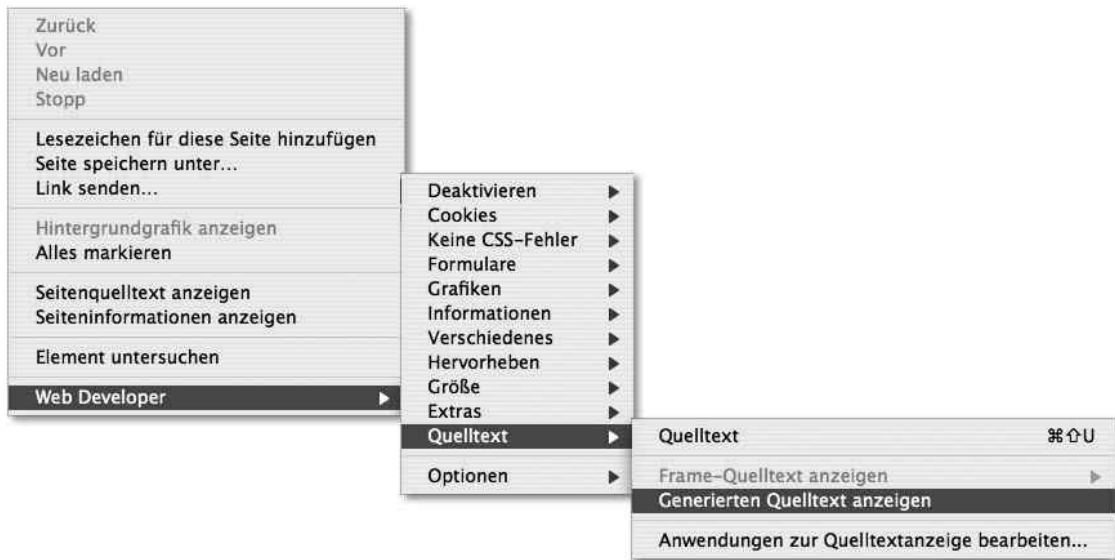


Abbildung 2.1 Das Kontextmenü des Firefox, die Web Developer Tools

2.2.3 Opera

jQuery unterstützt zwar den Opera-Browser ab der Version 9, die Verbreitung dieser Versionen ist allerdings sehr gering, testen sollten Sie auf jeden Fall ab der Version 10.x.

Mehr Informationen: <http://www.opera.com/>

2.2.4 Safari

Der Safari basiert auf der freien Rendering-Engine Webkit. Auch Google Chrome basiert darauf, allerdings besitzen beide Browser unterschiedliche JavaScript-Engines. Mitte 2010 liegt Safari in der Version 5 vor, aber unter Mac-Usern erfreut sich auch die Version 4 noch größter Beliebtheit.

Bezugsquelle: <http://www.apple.com/de/safari/>

2.2.5 Google Chrome

Auch der Shootingstar unter den Browsern wird von jQuery unterstützt, die rasante Entwicklung von Google Chrome hat den Apple-Browser Safari in Sachen Verbreitung bereits überholt. Hier gibt es eine hohe Frequenz an Updates. Mitte

2010 liegt Google Chrome in der stabilen Version 5.x vor. Eine Vorabversion 6.0 ist bereits seit Mitte 2010 im Umlauf.

Adresse: <http://www.google.com/chrome?hl=de>

Eine Auswahl an Browserwatch-Websites

- ▶ http://www.w3schools.com/browsers/browsers_stats.asp
- ▶ <http://www.webhits.de/deutsch/index.shtml?webstats.html>
- ▶ <http://www.w3b.org/tag/browserwatch>

2.3 Webserver

Manche Beispiele im Buch setzen einen Webserver voraus, gerade wenn es um das Thema Ajax geht. Es gibt auch ein paar wenige Beispiele, in denen Sie kleine PHP-Skripte einsetzen werden. Da PHP in einer Serverumgebung läuft, kann es nicht schaden, wenn Sie bei Bedarf lokal auf Ihrem Rechner einen Webserver samt Datenbank und PHP Interpreter installieren.

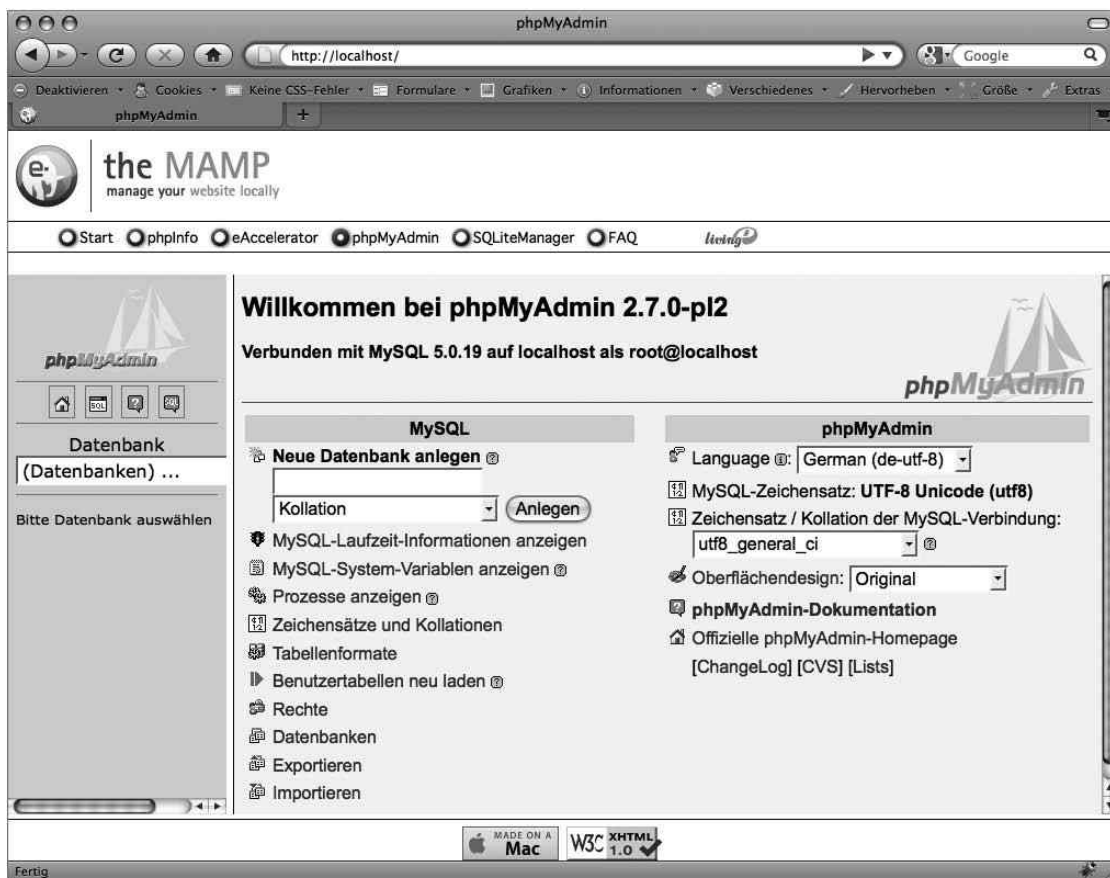


Abbildung 2.2 Webserver und phpMyAdmin

Zum Glück gibt es vorgefertigte Pakete, mit denen Sie alle benötigten Komponenten samt Konfiguration in einem Rutsch installieren können. Für Windows, Mac und Linux verwenden Sie *XAMPP*. Alternativ zu XAMPP für Mac OS X gibt es *MAMP*, das sich eignet, wenn Sie wenige Eingriffsmöglichkeiten benötigen und unkompliziert und sofort loslegen möchten.

Auch die bekannte Datenbankadministration *phpMyAdmin* bringt alle diese Pakete mit. Sie wird automatisch mit installiert; mit ihr verwalten Sie Ihre Datenbanken über ein Web Interface nahezu spielend leicht.

Adressen

- ▶ <http://www.apachefriends.org/de/xampp.html>
- ▶ <http://www.mamp.info/de/index.html>

2.4 IDEs und Editoren

Sie haben nun also die Voraussetzungen geschaffen, um effektiv Webapplikationen zu verstehen und zu testen. Nur schreiben müssen Sie sie noch. Sie wissen, dass HTML-Seiten, CSS- und JavaScript-Dateien reine Textdateien sind. Sie benötigen mindestens einen Texteditor, um HTML-Dokumente und JavaScript-Dateien sinnvoll zu befüllen. Manche Webdeveloper bevorzugen sogar einfache Editoren, da sie schnell starten, rasch reagieren und flink im Umgang sind. Zwei dieser Programme werden im Folgenden vorgestellt. Für Windows können Sie *Notepad++* verwenden (siehe Abbildung 2.3). Das ist ein freier Quellcode-Editor, der Syntaxhervorhebung für diverse Sprachen genauso beherrscht wie Dateivergleich oder eine Autovervollständigung für JavaScript und viele andere Sprachen.

Sie haben mit ihm die Kodierung eines Dokuments im Griff und können Makros ausführen lassen. Für die Zwecke eines einfachen Texteditors eine geradezu komfortable Lösung.

Das Pendant aufseiten des Mac ist *TextWrangler*, der kleine Bruder von *BEdit*, dem Standard-Quellcode-Editor für den Mac. Letzterer ist allerdings nicht unentgeltlich zu bekommen, TextWrangler dagegen ist kostenlos.

Wenn Sie eine anspruchsvollere Lösung bevorzugen, die auch etwas kosten darf, können Sie auf *Adobe Dreamweaver* zurückgreifen. Neben dem WYSIWYG-Modus, in dem Sie HTML-Elemente visuell hin- und herschieben können (Sie sollten allerdings um diesen Modus einen großen Bogen machen), besitzt diese Software auch eine Projektverwaltung, und besonders hervorzuheben ist die Codevervollständigung für jQuery.

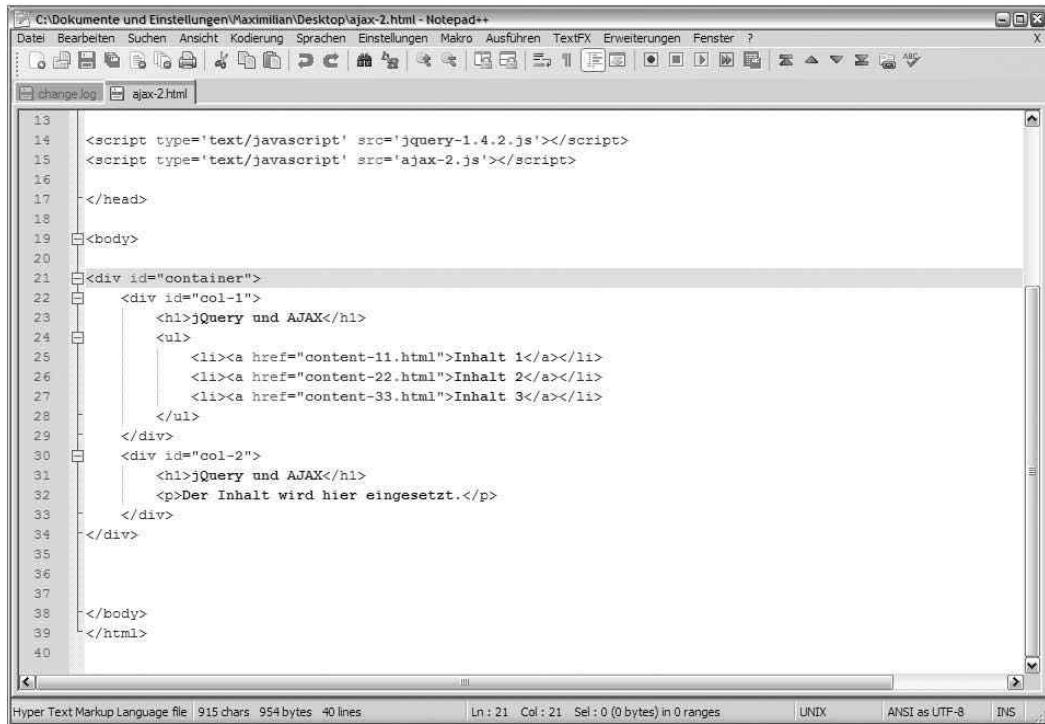


Abbildung 2.3 Editor Notepad++

Dreamweaver unterstützt eine Reihe von Webformaten, Sie können auch eine Webserverumgebung einbinden und Dateien mit dem integrierten FTP-Client auf einen Server kopieren. Dreamweaver gibt es sowohl für Mac als auch für Windows.

Microsoft Visual Studio 2010 dagegen ist ausschließlich für Windows entwickelt. Microsoft nennt seine Codevervollständigung *IntelliSense*, und sie versteht sich vorzüglich mit jQuery. Das Framework wurde vollständig in diese IDE (Interactive Development Environment) integriert.

Wenn Sie eine umfassende Software auch für Microsoft Office-Programmierung, oder die .net-Plattform benötigen, kann das Visual Studio die richtige Wahl für Sie sein. Im Übrigen, mit *Microsoft Expression Studio*, der Entwicklung für Webapplikationen, können Sie zwar Webanwendungen schreiben, eine jQuery-Integration gibt es jedoch nicht.

Zu guter Letzt sei Ihnen *Aptana* ans Herz gelegt, das in seiner aktuellen Version Studio 2 bereits jQuery 1.4.2 mit an Bord hat. Es basiert auf der freien Entwicklungsumgebung Eclipse, und es ist als Standalone-Version wie auch als Plugin für Eclipse verfügbar. Aptana nutzt alle Features der offenen Eclipse-Plattform. Es handelt sich um eine ausgewachsene IDE, und Sie können sie kostenlos verwenden. Auch Aptana besitzt eine Codevervollständigung, die Hersteller nennen sie *Code Assist*.

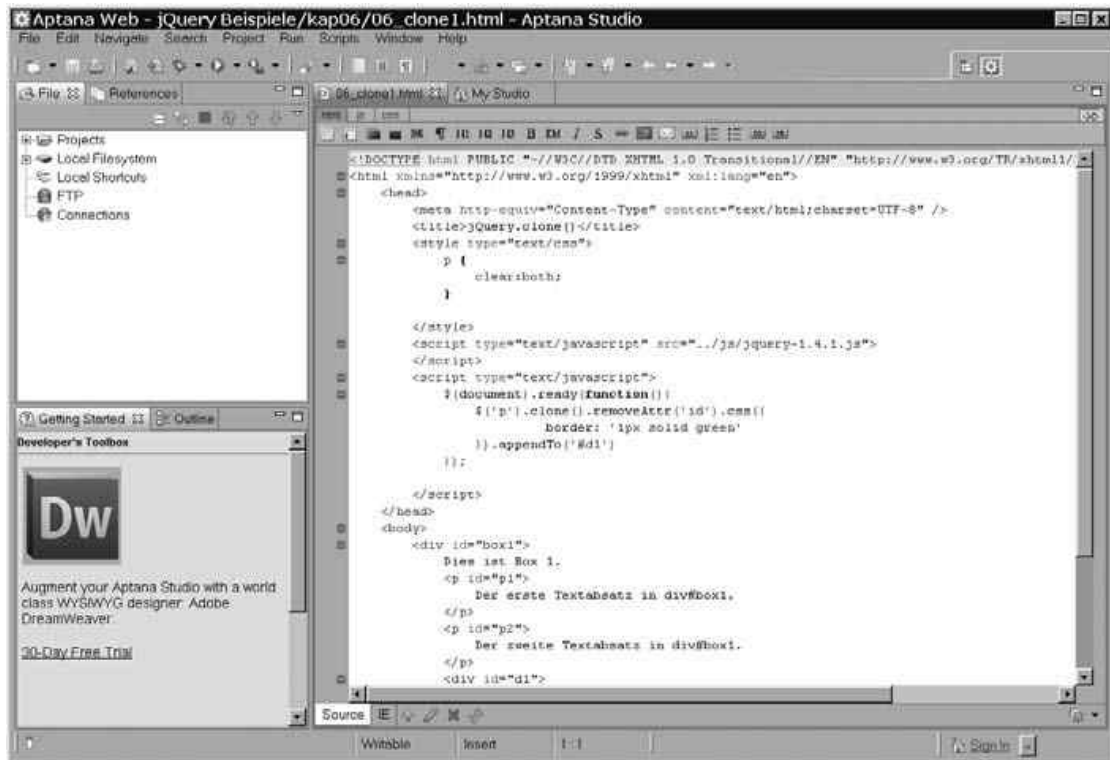


Abbildung 2.4 Aptana Studio 2

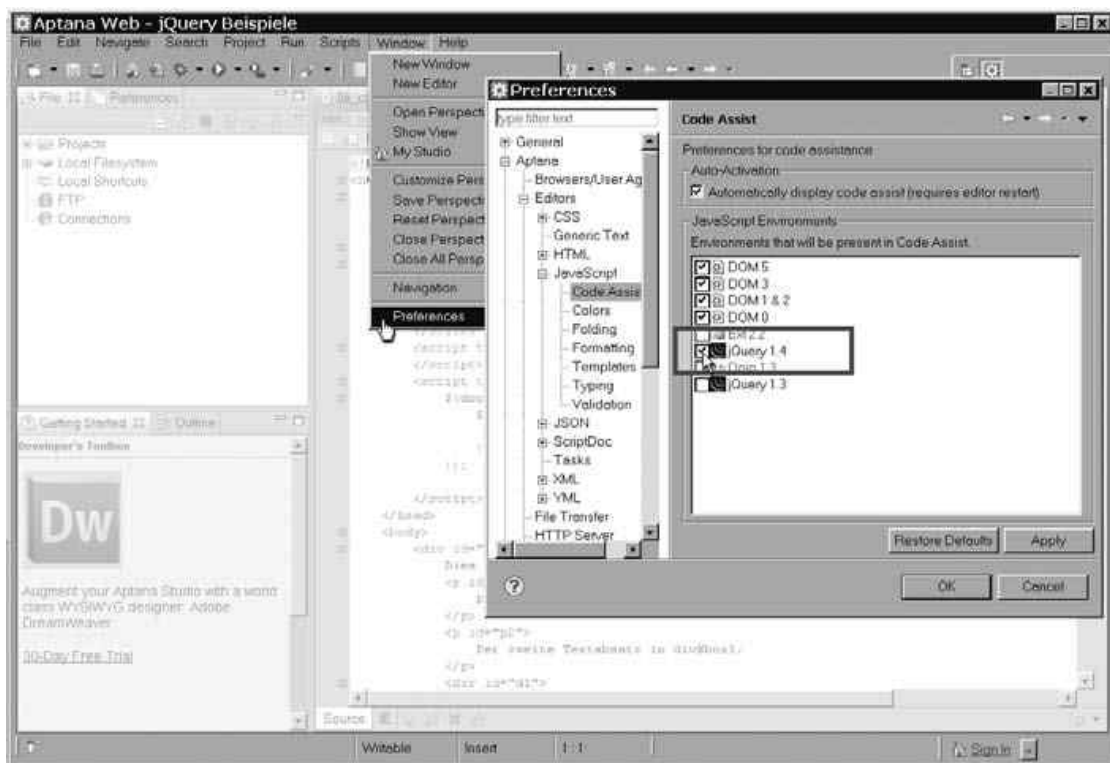


Abbildung 2.5 Aktivieren des jQuery Code Assists in Aptana

Für jQuery ist die Codevervollständigung von Haus aus nicht aktiv – Sie ändern dies einfach über das Menü **WINDOWS** → **PREFERENCES**. Im Preferences-Popup wählen Sie anschließend (im Tree-Menü links) den Pfad **APTANA** → **EDITORS** → **JAVASCRIPT**. Im rechten Bereich sehen Sie eine Liste der verfügbaren Code-Assistenten. Darunter befinden sich auch jQuery 1.3 und 1.4 (siehe Abbildung 2.5). Nach einem Klick auf die jeweilige Version steht der Assistent im Codefenster zur Verfügung (siehe Abbildung 2.6).

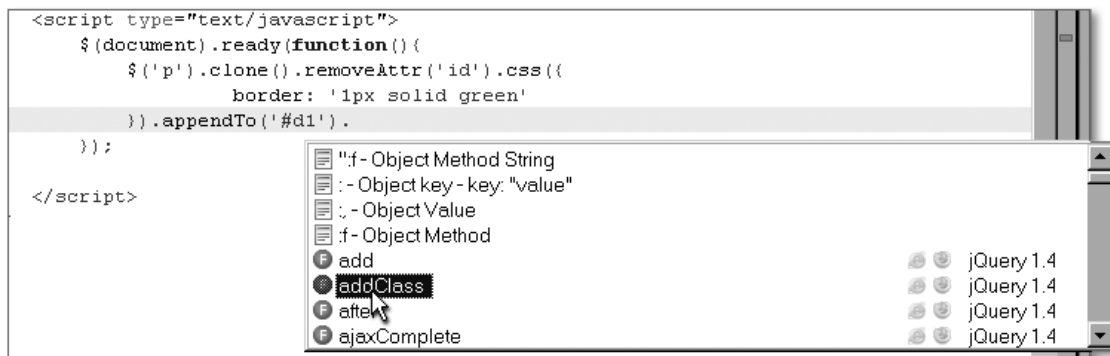


Abbildung 2.6 jQuery Code Assist in Aptana Studio 2

Links zu den Herstellern von IDEs und Editoren

- ▶ TextWrangler: <http://www.barebones.com/products/textwrangler/>
- ▶ Notepad++: <http://notepad-plus-plus.org/de/node/56>
- ▶ Microsoft Visual Studio: <http://www.microsoft.com/germany/visualstudio/>
- ▶ Adobe Dreamweaver: <http://www.adobe.com/de/products/dreamweaver/>
- ▶ Aptana: <http://www.apтана.com/>
- ▶ Eclipse: <http://www.eclipse.org/>

Mit diesen Informationen sind Sie gut gerüstet und können nun in vollen Zügen in die Welt von jQuery eintauchen. Wir wünschen Ihnen viel Spaß mit den Tipps und Tricks, den Fallstricken, den Überraschungen und den Aha-Erlebnissen, die Sie mit jQuery erleben werden. Wenn Sie noch Kenntnisse zu HTML und Konsorten benötigen, blättern Sie bitte zum Ende des Buches, um sich ein kleines »Lunchpaket« mit den nötigsten Ingredienzien der Webentwicklung für die Reise durch die jQuery-Welt einzuverleiben. Ansonsten lesen Sie einfach im nächsten Kapitel weiter.

»Even within the jQuery team, we have people from all backgrounds providing their feedback on the direction of the project. There is one thing that is common across all of jQuery's users, though: We are a community of developers and designers who want JavaScript development to made simple.« – John Resig

3 jQuery – der Einstieg

In diesem Kapitel erfahren Sie einiges über die Grundlagen von jQuery und über die Hintergründe des jQuery-Objekts und seines Einsatzes. Sie lernen, wie Sie mit jQuery Elemente Ihrer HTML-Seite selektieren und anschließend Aktionen ausführen können. Zum Verständnis benötigen Sie zumindest Grundkenntnisse in HTML und JavaScript, sowie – in Zusammenhang mit jQuery beinahe noch wichtiger – ein Grundverständnis von CSS und der Formulierung von CSS-Selektoren. Sollten Sie hier einen Nachschlag benötigen, raten wir Ihnen, zuerst die letzten beiden Kapiteln zu lesen: Anhang A, »HTML und CSS«, und Anhang B, »JavaScript und DOM«. Ansonsten möchten wir auf die einschlägige Fachliteratur verweisen, über die wir Sie, ebenfalls im Anhang, am Ende des Buches informieren. Lassen Sie uns nun also einfach beginnen!

3.1 Vergleich: JavaScript mit und ohne jQuery

Um zu verdeutlichen, wie jQuery dem User tagtäglich viel Arbeit abnimmt und ihm so das Webleben erleichtert, möchten wir hier zunächst mit einem Beispiel anfangen, das – noch ohne die Hilfe von jQuery – die Manipulation eines HTML-Elements vornimmt.

3.1.1 JavaScript ohne jQuery

Angenommen, Sie wollen, abhängig von seinem Textinhalt, einem von mehreren `<p>`-Absätzen die Klasse `green` zuweisen und ihm damit eine neue Schriftfarbe geben. Dann betrachten Sie dazu zunächst einmal den Inhalt des `<script>`-Elements:

```

<html>
<head>
<title>Listing 1</title>
  <style type="text/css">
    .green {
      color:#009933;
    }
  </style>
  <script type="text/javascript">
    window.onload = function() {
      var elements = document.getElementById("box")
        .getElementsByTagName("p");
      for (var i = 0; i < elements.length; i++) {
        if (elements[i]
          .firstChild.data == "Zweiter Absatz") {
          elements[i].className = "green";
        }
      }
    }
  </script>
</head>
<body>
  <div id="box">
    <p>Erster Absatz</p>
    <p>Zweiter Absatz</p>
    <p>Dritter Absatz</p>
  </div>
</body>
</html>

```

Listing 3.1 Schriftfarbe ändern ohne jQuery

Was in diesem Scriptabschnitt passiert, ist Folgendes: Sobald die komplette HTML-Seite geladen ist (`window.onload`), wird eine Kollektion mit allen Absätzen ("p") erstellt, die sich innerhalb des Containers mit dem ID ("box") befinden.

Zum Zeilenumbruch vor dem Punktoperator

Wichtig – Sie dürfen *vor* dem Punktoperator in Objekten praktischerweise einen Zeilenumbruch machen, da der Punkt am Beginn der Folgezeile der Script-Engine deutlich macht, dass die Anweisung weitergeht. Überlange Programmzeilen wie diese können so vermieden werden:

```

var elements = document.getElementById("box")
    .getElementsByTagName("p");

```

Diese Kollektion wird in einer Variablen `elements` gespeichert. Anschließend werden über eine *for-Schleife* und eine *if-Anweisung* alle gefundenen `<p>`-Container dahingehend geprüft, ob sie einen Textknoten mit Inhalt »Zweiter Absatz« enthalten.

Wenn dem so ist, wird dem betreffenden Absatz die Klasse `green` hinzugefügt und damit dessen Schriftfarbe auf »Grün« gesetzt. Sie brauchen an dieser Stelle nicht zu sehr in die Tiefe zu gehen, um sich klarzumachen, dass Sie hier genau überlegen müssen, wie Sie in Ihrem Script mit Bedingungen und Schleifen Ihre Suchergebnisse filtern, bis am Ende das gewünschte Ergebnis im Browser erscheint. Ein Framework kann Ihnen dieses Kopfzerbrechen ersparen. Werden Sie also aktiv!

3.2 jQuery einbinden

Es versteht sich, dass das jQuery Framework, um es zu nutzbar zu machen, zunächst bereitgestellt werden muss: Dem ausführenden Browser müssen die Funktionalitäten zur Verfügung stehen, um die besonderen Eigenschaften und Methoden von jQuery zu interpretieren. Zunächst sollten Sie sich, falls nicht bereits geschehen, eine aktuelle Version von jQuery besorgen – dies können Sie direkt bei der offiziellen jQuery-Website [www.jquery.com](http://jquery.com) tun. Laden Sie von dort sowohl die *minimierte Produktionsversion* (24 KB) als auch die *Developer-Version* (155 KB) von jQuery herunter:

jQuery-Website: <http://jquery.com/>

Downloaden oder doch nicht downloaden?

Sie finden die erforderlichen jQuery-Dateien auch im Order *beispiele/jquery/* auf der Begleit-DVD zu diesem Buch. Alle Beispiele basieren auf Version 1.4.1. Wir gehen auch in Zukunft von einer Abwärtskompatibilität aus. Sollte es eine wesentlich neuere jQuery-Version geben, sind Probleme jedoch nicht auszuschließen. Verwenden Sie dann die Version aus dem Begleitmaterial.

Es gibt nun also, wie bereits gesagt, zwei verschiedene Versionen von jQuery. Einmal die Version *uncompressed*, d.h. die unkomprimierte Datei, und einmal die Version *minified*, also eine verkleinerte Datei.

Sind Sie daran interessiert, einmal einen Blick in das »Getriebe« von jQuery zu werfen? Dann nehmen Sie die unkomprimierte Datei zur Hand. Im Normalfall werden Sie das, selbst als Entwickler, kaum tun wollen, sondern nur die angebotenen Funktionen im Produktiveinsatz nutzen wollen. In diesem Fall verwenden

Sie die kompaktere *minified*-Datei, deren Größe weniger als die Hälfte der unkomprimierten Datei beträgt.

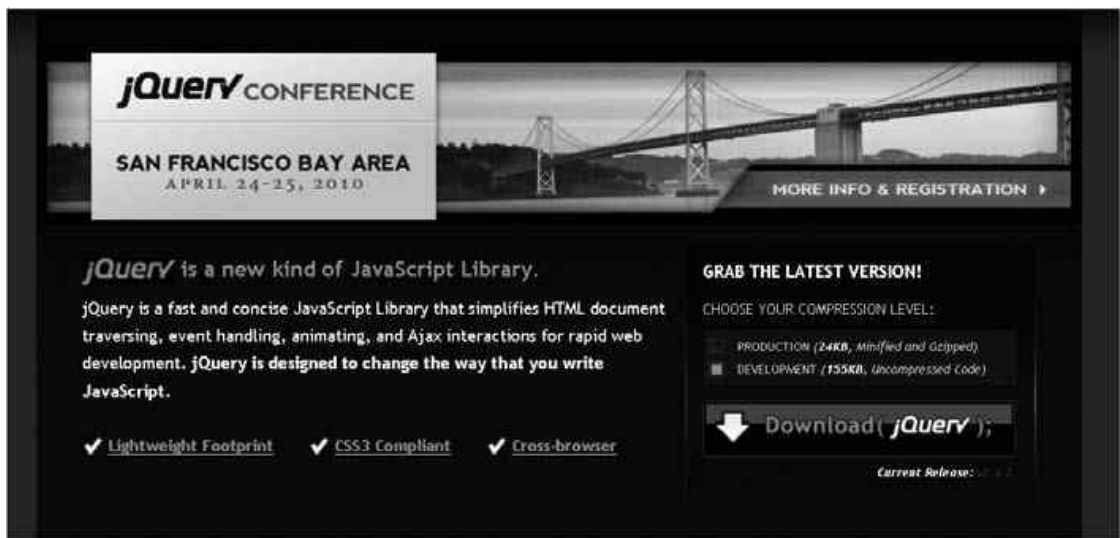


Abbildung 3.1 Download von jquery.com

3.2.1 jQuery online und offline nutzen

Vielleicht zunächst eine kleine Klärung im Vorfeld – benötigen wir zum Test von jQuery eigentlich einen Webserver oder nicht? Frei nach Radio Eriwan könnte man antworten, im Prinzip nicht. Jedoch kann in diesem Fall dann auch nur ein Teil der jQuery-Funktionalität genutzt werden: Einige der Tricks, die das Framework zur Verfügung stellt, basieren auf der Nutzung von Ajax, was in der Regel einen Server voraussetzt, der die HTTP-Anfragen interpretiert, die das Framework dann absetzen wird.

Benötigen wir kein Ajax, kann auch direkt aus dem Dateisystem heraus (sozusagen »offline«) gearbeitet werden: Sie können beispielsweise ein HTML-Dokument unmittelbar aus dem Verzeichnis ins Browserfenster ziehen, um ein Script zu testen. Ansonsten müssten Sie einen lokalen Webserver zur Verfügung haben und die Übungsdateien in dessen Dokumentenverzeichnis ablegen.

Welche Serverinstallation ist geeignet?

Welchen Server Sie in diesem Fall wählen, ist für die Basis, auf der dieses Buch aufsetzt, egal. Wir empfehlen Ihnen das XAMPP-Paket, das Sie in sämtlichen Versionen für alle Plattformen auf der Begleit-DVD zu diesem Buch finden (Mac-User werden auf ihrem Rechner meist schon einen lauffähigen Server vorfinden, den sie der Einfachheit halber auch einsetzen sollten).

XAMPP-Website: www.apachefriends.org

3.2.2 jQuery lokal einbinden

Sie benötigen von jeder HTML-Seite, in der Sie jQuery nutzen möchten, einen Link zur jQuery-Datei. Sie haben die Wahl zwischen der minimierten Version `jquery-1.4.1.min.js` und der unkomprimierten Version `jquery-1.4.1.js`. Für die lokalen Übungen in diesem Buch macht es keinen Unterschied, für welche dieser Versionen Sie sich entscheiden. Legen Sie einfach beide Daten in einem Unterverzeichnis des *htdocs*-Verzeichnisses des lokalen Servers auf Ihrem Rechner ab (wir nennen dieses Verzeichnis *buchbeispiele* – selbstverständlich können Sie auch jeden beliebigen Namen wählen).

Wir empfehlen für die Arbeit mit den Beispielen dieses Buches folgende Ordnerstruktur:

```
buchbeispiele/
  jquery/
    jquery-1.4.1.js
    jquery-1.4.1.min.js
  _uebungen/
  ...
```

Dies erleichtert die kapitelübergreifend konstante Einbindung des Frameworks, dessen Dateien im Ordner *jquery* abgelegt werden (dies ist der Bezeichner, den wir für dieses Buch gewählt haben). Parallel zu diesem Ordner legen Sie einen oder mehrere Ordner mit Ihren Test- und Übungsdateien an. Ressourcen, die von den HTML-Seiten benötigt werden (CSS- und Grafikdateien) platzieren Sie sinnvollerweise ebenfalls in die Übungsordner.

Dies ermöglicht eine stets gleichförmige Formulierung des Links zur jQuery-Datei im Dokumentkopf der Übungsdateien:

```
<script type="text/javascript"
      src="../../jquery/jquery-1.4.1.js"></script>
```

Eine ähnliche Anordnung empfiehlt sich auch für Projekte, in denen Sie jQuery einsetzen (erinnern Sie sich aber daran, für Produktionszwecke auf die minimierte Version der jQuery-Datei zurückzugreifen). Reproduzieren Sie für den Onlinezugriff die gleiche Verzeichnisstruktur auf Ihrem Produktionsserver. Sie können dann die relativen Links zum Framework unverändert beibehalten.

3.2.3 jQuery aus dem Google Repository einbinden

Eine Alternative zur lokalen Einbindung von jQuery bietet sich in der Nutzung des *Google Online Repository* an, das ein frei zur allgemeinen Nutzung zur Verfügung stehendes Framework anbietet. (Gedacht ist dieses Repository für den Live-

betrieb einer Website. Sie können das Prinzip jedoch auch für die Buchbeispiele einsetzen, sofern Ihr Rechner jederzeit über eine Onlineverbindung verfügt.)

Das Content Delivery Network (CDN) von Google hält neben jQuery auch andere JavaScript-Frameworks bereit, wie Prototype, MooTools, Dojo und Ext JS. Im Falle von jQuery 1.4.1 geschieht die Einbindung wie folgt:

```
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/
libs/jquery/1.4.1/jquery.min.js"></script>
```

Man kann auch einen Schritt weiter gehen und jQuery über den `load`-Befehl der *Google JavaScript API* einbinden. Hierfür muss allerdings zunächst eben diese API verlinkt werden. Anschließend steht die Methode `google.load()` zur Verfügung, mit der jQuery ebenfalls bereitgestellt werden kann.

Deren erstes Argument bestimmt das Framework. Achtung – der zweite Parameter, der die gewünschte Version nennt, ist obligatorisch und kann daher nicht einfach weggelassen werden! Mit der folgenden Anordnung wird die minimierte Version von jQuery 1.4.1 aus dem Google CDN geladen:

```
<script type="text/javascript"
      src="http://www.google.com/jsapi"></script>
<script type="text/javascript">
    google.load("jquery", "1.4.2");
</script>
```

Wollen Sie die unkomprimierte Variante inkludieren, fügen Sie noch ein drittes Argument hinzu (achten Sie auf die geschweiften Klammern):

```
<script type="text/javascript">
    google.load("jquery", "1.4.0",{uncompressed:true});
</script>
```

Pro und Kontra

Diese Lösung hat wie so vieles ihre Licht- und Schattenseiten. Als Vorteil könnte man werten, dass man nicht gezwungen ist, jQuery auf dem eigenen Server abzulegen. Die Anbieter werden zudem meist mehrere Versionen des Frameworks (darunter sicherlich die jeweils aktuelle) zur Verfügung stellen. Bei einer stehenden Onlineverbindung steht auch einer Nutzung von Repositories im Rahmen von ansonsten lokalen Tests nichts im Weg.

Ein Nachteil ist, dass die Funktion der eigenen Website von der konstanten Bereitstellung des Frameworks durch eine andere Partei abhängig ist. Bei Yahoo oder Google kann man immerhin davon ausgehen, dass der Service dauerhaft zur Verfügung stehen wird. Nicht vergessen sollte man jedoch, dass die Einbindung

einer extern gehosteten Datei auch ein Protokollieren der Nutzung der eigenen Site durch den Provider des Frameworks ermöglicht (ohne dies unterstellen zu wollen). Ob man sich darauf einlassen möchte, sei dem Einzelnen überlassen. (Wir werden in diesem Buch die konventionelle Einbindung verwenden und jQuery aus dem lokalen Verzeichnis einbinden.)

3.3 Unser Beispiel mit jQuery

Legen Sie die heruntergeladene Framework-Datei wie beschrieben in ein Verzeichnis parallel zu dem, in dem die HTML-Seite gespeichert ist. Bevor Sie nun das Script konstruieren, müssen Sie erst das Framework in die Seite einbinden. Ähnlich einem externen Stylesheet wird die Datei in das Hauptdokument inkludiert.

Im ersten `<script>`-Element geben Sie den Pfad zur Bibliothek an:

```
<script type="text/javascript"
      src="../../jquery/jquery-1.4.min.js"></script>
```

Wo genau soll die Einbindung im Quelltext erfolgen?

Unser Tipp: Inkludieren Sie das jQuery Framework im `<head>`-Verzeichnis stets nach den CSS-Dateien. Achten Sie darauf, dass Ihre eigenen JavaScript-Funktionen bzw. -dateien wiederum stets nach dem Framework eingebunden werden.

Legen Sie nun ein weiteres `<script>`-Element an, und fügen Sie die jQuery-Funktionen ein. Das gesamte Beispiel sieht jetzt folgendermaßen aus (der jQuery-Code ist fett hervorgehoben):

```
<html>
<head>
  <title>Listing 2</title>

  <!-- Zunächst die Styleangaben: -->
  <style type="text/css">
    .green {
      color:#009933;
    }
  </style>

  <!-- Nun das Framework einbinden: -->
  <script type="text/javascript"
        src="../../jquery/jquery-1.4.min.js"></script>
```

```

<!-- ...und jetzt unseren eigenen Code: -->
<script type="text/javascript">
    $(document).ready(function() {
        $("#box p:contains('Zweiter Absatz')")
            .addClass("green");
    });
</script>
</head>
<body>
    <div id="box">
        <p>Erster Absatz</p>
        <p>Zweiter Absatz</p>
        <p>Dritter Absatz</p>
    </div>
</body>
</html>

```

Listing 3.2 Schriftfarbe ändern mit jQuery

Das jQuery-Objekt als Factory-Funktion

Sofort ins Auge springt, dass gerade einmal drei Zeilen Code benötigt werden, um das gesteckte Ziel zu erreichen. Das fundamentale Konstrukt in jQuery ist die Funktion `$()`. Es handelt sich dabei um eine sogenannte *Factory-Funktion* – so bezeichnet, weil sie das erzeugt, was anschließend als Arbeitsgrundlage dient.

Zum Vergleich, die im vorigen Listing eingesetzten DOM-Funktionen `getElementById()` und `getElementsByTagName()` holen »nur« existierende Bestandteile des Dokuments, verändern diese aber nicht. Sie müssen mit den »natürlichen« Eigenschaften der gefundenen Dokumentknoten zurechtkommen.

Anders bei der `$()`-Funktion. Diese Funktion erhält zwar auch ein Argument, das bestimmt, womit die Funktion arbeitet. Hier ist es jedoch das gesamte Dokument – Sie werden aber auch noch andere Fälle kennenlernen. Allerdings begnügt sich die `$()`-Funktion nicht damit, sich das bezeichnete Objekt zu beschaffen, es statet dieses mit weiteren Fähigkeiten aus. Diese neuen Fähigkeiten umgeben das »alte« Objekt wie ein Mantel – man spricht daher von *Wrapping*. Das so erzeugte »neue« Objekt wird (unabhängig davon, welche Art von Objekt als Grundlage fungiert) als *jQuery-Objekt* bezeichnet.

Mit dem Aufruf von `$()` wird also immer ein neues jQuery-Objekt erzeugt und zurückgegeben. Danach steht es mit seinen ganzen Methoden und Eigenschaften für das weitere Scripting zur Verfügung. Eine dieser Methoden nennt sich `.ready()`. Sie wird allerdings speziell dann verwendet, wenn das jQuery-Objekt aus einem `document`-Objekt gebildet wurde (zugegeben, ein Sonderfall).

Jetzt wissen Sie für den Anfang genug – analysieren wir nun den hervorgehobenen Code aus dem vorigen Listing Zeile für Zeile:

Sobald das Dokument fertig geladen wurde ...

```
$(document).ready( ... );
```

... führen Sie die in `ready()` befindliche anonyme Funktion aus ...

```
$(document).ready( function() {  
    ...  
});
```

... die besagt: Bilden Sie aus allen `<p>`-Elementen mit Inhalt 'Zweiter Absatz' innerhalb des `<div>`-Containers mit dem ID `box`, ein jQuery-Objekt:

```
$(document).ready( function() {  
    $("#box p:contains('Zweiter Absatz')")  
    ...  
});
```

... und wenden Sie auf dieses die Funktion `addClass()` an, um den `<p>`-Containern die Klasse `green` hinzuzufügen (wir machen wieder einen Zeilenumbruch vor dem Punktoperator – das ist nicht nur für das Listing im Buch praktisch):

```
$(document).ready( function() {  
    $("#box p:contains('Zweiter Absatz')")  
        .addClass("green");  
});
```

Das war's. Sie brauchen sich nicht mit Schleifen und Bedingungen herumschlagen. Stattdessen navigieren Sie mittels des Selektors `"#box p:contains('Zweiter Absatz')"` direkt zum gewünschten Element, verkapseln es als jQuery-Objekt und verändern es. (Übrigens, der Filterpseudoselektor `:contains()` ist eine Dreingabe von jQuery, der Ihnen in reinem CSS nicht zur Verfügung steht.)

Die gewünschte Veränderung geschieht durch den Aufruf von `addClass()`. Mit dieser Methode können Sie beliebige CSS-Klassen an beliebige HTML-Elemente binden. (Dass die Klasse mit ihren Eigenschaften vorher im Stylesheet definiert werden muss, versteht sich von selbst.)

Warum muss es ein jQuery-Objekt sein?

Die *Verkapselung* in ein jQuery-Objekt ist in diesem Fall der springende Punkt. Den `<p>`-Container bekämen Sie zwar auch anders zu fassen (wie zuvor bereits beschrieben), aber er wäre eben »nur« ein `<p>`-Container. Versuchten Sie, auf ihn *direkt* die Methode `addClass("green")` anzuwenden, hätten Sie Pech. Anders, wenn der `<p>`-Container »jQueryifiziert« wurde (in ein jQuery-Objekt verpackt ist). In diesem Fall stehen ihm unmittelbar alle Methoden von jQuery zur Verfügung.

Das jQuery-Objekt als Knotenliste

Wir haben das jQuery-Objekt als Wrapper für ein Einzelobjekt kennengelernt. Mithilfe dieser Verkapselung stehen Ihnen für dieses Objekt alle jQuery-Tricks zur Verfügung. Ein Einzelobjekt war es aber schlicht deshalb, weil der an `$()` übergebene Ausdruck lediglich ein Objekt als Ergebnis hatte.

Wenn dies aber nicht so eindeutig ist, was dann? Ganz einfach – jQuery macht aus *allen* übergebenen Objekten (nehmen wir mal an, es seien Elementknoten) *gemeinsam* ein jQuery-Objekt. Dieses nimmt dann, wie man sagt, die Eigenschaften einer »Liste« an. Im herkömmlichen JavaScript spricht man von einer *Knotenliste* (node-list). In jQuery läuft dies darauf hinaus, dass für *jedes* Element der Liste alle jQuery-Optionen zur Verfügung stehen.

Legen wir einmal mehrere Elemente im jQuery-Objekt ab, indem wir jetzt *alle* Absätze innerhalb des Containers `#box` selektieren. Hierfür genügt es, den Filter `:contains('Zweiter Absatz')` wegzulassen.¹ Damit ist die Einschränkung auf den zweiten Absatz aufgehoben:

```
$(document).ready(function() {
    $("#box p").addClass("green");
});
```

Das Ergebnis ist wenig spektakulär, vielleicht sogar als »intuitiv vorhersehbar« zu bezeichnen: Alle Textabsätze erhalten grüne Schrift und Hintergrundfarbe.

Obwohl ... das bedeutet doch, dass jQuery *alle* Elemente seiner Liste hernimmt und auf *jedes einzelne* die angehängte Methode anwendet? Genau: jQuery arbeitet eine Knotenliste Item für Item ab, und zwar vollautomatisch. Behalten Sie das im Hinterkopf – Stichwort: implizite Schleife.

Welches Argument erwartet die Funktion `$()`?

An dieser Stelle möchten wir kurz auf die Frage eingehen, *was* für ein Argument die `$()`-Funktion eigentlich erwartet. Wir haben gesagt, dass es sich um CSS-Selektoren handelt, teilweise noch mit jQuery-spezifischen Erweiterungen. Es geht aber auch noch mehr, wie Sie später noch sehen werden. Bleiben wir jedoch zunächst bei CSS:

jQuery unterstützt im Rahmen der `$()`-Funktion nahezu alle CSS-Selektoren, von CSS 1.0 bis CSS 3.0. Dies macht es für Webworker mit CSS-Erfahrung leichter, sich in jQuery einzuarbeiten. Wenn nicht, in Anhang A stellen wir Ihnen die wichtigsten Grundbegriffe vor.

¹ Wir könnten hier sogar noch einfacher `$('p').addClass('green')` schreiben. Schön, nicht?

3.4 Wir haben fertig

Gehen wir kurz auf die `ready()`-Methode im folgenden Ausdruck ein:

```
$(document).ready(fn)
```

Bei `.ready()` handelt es sich um eine grundlegende Methode innerhalb des Eventmoduls von jQuery: Eine Funktion `fn`, die an `.ready()` übergeben wird, führt jQuery aus, sobald das Dokument geladen wurde. In unserem Fall ist es eine anonyme Funktion, die die Dokumentabfrage enthält.

Der Grund für dieses Manöver ist folgender: Wenn Sie die Anweisung `$("#box p").addClass("green")` unmittelbar auswerteten, erhielten Sie keine Treffer – in diesem Moment gäbe es schlicht noch kein Dokument, aus dem das Script den `<p>`-Container holen könnte. Das ist nämlich noch gar nicht geschrieben. Also muss abgewartet werden. JavaScript bietet uns mit `window.onload` zu diesem Zweck bereits von Hause aus einen passenden *Event Handler* an. Ist `fn` die aufzurufende Funktion, schreiben Sie mit seiner Hilfe:

```
window.onload = fn;
```

Den gleichen Dienst leistet der `onload`-Event-Handler im `<body>`-Element. Wo ist der Unterschied zum (zudem komplizierter zu schreibenden) Ansatz von jQuery? Der Unterschied ist das Timing: Mit `window.onload` wird die Funktion `fn` erst aufgerufen, wenn alle Abhängigkeiten des Dokuments aufgelöst sind, beispielsweise die Grafiken geladen wurden. Die Grafiken brauchen Sie aber gar nicht, wenn Sie »nur« an die Elemente wollen!

Der Vorteil von `$(document).ready(fn)` liegt darin, dass das Script bereits dann die *Callback*-Funktion `fn` ausführt, wenn die für Sie relevante HTML-Struktur bereitsteht. Das DOM ist dann zwar nur »weitestgehend geladen«, aber Sie können auch schon früher darauf zugreifen. Ein Problem ist das nicht – das Laden der Bilder können wir bei der Analyse und Manipulation des HTML-Dokuments in den meisten Fällen vernachlässigen.

Sie könnten beliebig oft `$(document).ready(fn)` in einer Seite ausführen lassen. Im Endeffekt mag das aber ein wenig unübersichtlich werden. Hier besteht der Hintergedanke primär darin, Funktionscode aus dem `ready`-Block auszulagern, um diesen möglichst einfach zu halten (sonst könnten die Funktionsblöcke von `tuDies()` und `tuJenes()` auch ebenso im `ready`-Block selbst stehen – der würde dann allerdings möglicherweise ziemlich lang werden):

```
// definieren, was tuDies() machen soll:
function tuDies() {
    $("#div p").click(function(){ ... });
}
```

```

}

// erster ready()-Block:
$(document).ready( function() {
    // und tuDies() aufrufen, wenn's soweit ist:
    tuDies();
});

// definieren, was tuJenes() machen soll:
function tuJenes() {
    $("ul li").click(function(){ ... });
}

// zweiter ready()-Block:
$(document).ready(function() {
    // und tuJenes() aufrufen, wenn's soweit ist:
    tuJenes();
});

```

Wie gesagt, das geht, bietet aber keine Vorteile. Eine Funktion aus dem `ready`-Block auszulagern, ist jedoch immer eine gute Sache. Sie sehen auch, dass Sie dort ebenfalls die jQuery-Schreibweise verwenden dürfen. (Dies ist also nicht etwa auf den `ready`-Block beschränkt, damit dies einmal deutlich gesagt ist.)

Um Übersicht zu schaffen, beschränken Sie sich besser auf nur eine einzige `ready`-Anweisung. Innerhalb der anonymen Callback-Funktion dieser Anweisung (von der es nur eine geben darf!) können Sie »huckepack« beliebig viele Funktionen aufrufen:

```

$(document).ready(function() {
    tuDies();
    tuJenes();
    // etc.
});

```

3.5 Das Mausereignis – Bindung eines Click-Events

Nun macht unser allererstes Beispiel scheinbar nicht allzu viel Sinn, da Sie doch Stileigenschaften allein mittels Stylesheet festlegen könnten,² ohne JavaScript zu Hilfe zu rufen. Geschenkt – verbinden Sie eine solche Manipulation aber mit einem Mausereignis, wird es interessanter: Sie wollen als Nächstes erreichen,

2 Bedenkt man aber, dass Sie den Style anhand des Elementinhalts binden können, ist es vielleicht doch ganz spannend, oder?

dass ein Absatz seine Schriftfarbe ändert, sobald Sie ihn anklicken. Gehen wir kurz durch, wie dies ohne die Hilfe von jQuery erfolgen könnte.

3.5.1 Zunächst – die »aufdringliche« Variante

Warum wir dies mit »aufdringlich« betiteln, wird gleich klarer. Jedenfalls, Aktionen wie der Klick auf ein Element werden gewöhnlich mithilfe von Event Handlern erfasst. Diese werden wie Attribute in den Start-Tag des Elements geschrieben.

Um die `<p>`-Container klickbar zu machen, genügt dann folgender Code:

```
<div id="box">
  <p onclick="farbwechsel(this)">Erster Absatz</p>
  <p onclick="farbwechsel(this)">Zweiter Absatz</p>
  <p onclick="farbwechsel(this)">Dritter Absatz</p>
</div>
```

Was an dieser Stelle kurz erläutert werden soll, ist das Schlüsselwort `this`. Der Ausdruck stellt ein Objekt dar, das an die aufgerufene Funktion `farbwechsel()` übergeben wird. Es bezieht sich jeweils auf das `<p>`-Element, das gerade angeklickt wurde. Es handelt sich hier um das sogenannte *Kontextobjekt* (also das Objekt, an dem aktuell etwas passiert). So weit, so gut – auch die Funktion `farbwechsel()` ist nicht sonderlich kompliziert:

```
// das übergebene this landet in der Variable meinP:
function farbwechsel(meinP) {
  // der geklickte Absatz bekommt die Klasse zugewiesen:
  meinP.className = "green";
}
```

Erst einmal funktioniert alles. Sie sehen, dass Sie auf diese Weise beliebige HTML-Elemente mit einem *Click-Event* versehen können. Allerdings sollten Sie, um die Benutzerführung eindeutig zu halten, bei entsprechend »aktivierten« Elementen die Cursor-Eigenschaft per CSS auf `pointer` setzen, damit der Benutzer erkennt, dass hier eine Interaktion möglich ist.

Dies geschieht in unserem Beispiel in einer Styleregeln für `<p>`-Container. Des Weiteren fügen Sie der Klasse `green` eine Hintergrundfarbe hinzu und setzen die Cursor-Darstellung dort wieder auf »normal« – immerhin braucht der Absatz, sobald er die Klasse hat, ja kein zweites Mal geklickt werden:

```
p {
  cursor:pointer;
}
.green {
  color:#009933;
```



```

    background-color:#E2FFEC;
    cursor:default;
}

```

3.5.2 Etwas weniger aufdringlich bitte!

Was ist nun schlecht daran? Unschön ist, dass hier der Click-Event-Handler samt Funktionsaufruf in das HTML-Dokument geschrieben wurde, obwohl er nicht Teil der Informationsstruktur ist. Er ist außerdem überflüssig, wenn kein JavaScript aktiv ist (weil es abgeschaltet oder vom Client nicht unterstützt ist).

Im Sinne des Ansatzes »Unobtrusive JavaScript« (wir haben dazu im vorangegangenen Kapitel ein paar Worte verloren) bevorzugt man daher, den Event Handler wegzulassen und die Klickfunktionalität per JavaScript nachträglich hinzuzufügen. Das HTML bleibt auf diesem Weg »sauber«.

Um einem Absatz »von außen« den Click-Event zuzuweisen, müssen zunächst alle `<p>`-Container innerhalb des Bereichs `#box` gesammelt werden.

Dies geschieht wie zuvor:

```

var elements = document.getElementById("box")
               .document.getElementsByTagName("p");

```

Über das Array mit den Textabsatzknoten läuft eine Schleife, die den Click-Event bindet. Die Zahl der Knoten wird in einer Variablen `len` abgelegt, damit das Array nicht bei jedem Schleifendurchlauf erneut ausgelesen werden muss:

```

for(var i=0, len=elements.length; i<len; i++) {
    // Fein. Jetzt den Click-Event binden. Aber wie?
}

```

Für die Art und Weise, den Click-Event »ordentlich« zu binden, gibt es verschiedene, untereinander unverträgliche Varianten: die offizielle und die für den Internet Explorer. Eine Fallunterscheidung ist möglich, aber umständlich. Der Rettungsanker findet sich beim guten alten *DOM Level 0* und der dort definierten `onclick`-Eigenschaft. Diese Vorgehensweise ist zwar nicht zeitgemäß, hat aber den Vorteil, dass sie dann doch browserübergreifend funktioniert:

```

window.onload = function() {
    var elements = document.getElementsByTagName("p");
    for(var i=0, len=elements.length; i<len; i++) {
        elements[i].onclick= function() {
            this.className = "green";
        }
    }
}

```

```

    }
}

```

Dies ist in seiner Kompaktheit durchaus vertretbar. Sollte es noch schöner gehen? Oder zumindest – kürzer?

3.5.3 Ein unaufdringlicher Dreizeiler, dank jQuery

In jQuery verwenden Sie nicht eine DOM-Eigenschaft namens `onclick`, sondern eine jQuery-Methode die den (durchaus treffenden) Namen `click()` trägt. Mit ihrer Hilfe kann jedem Absatz innerhalb des Elements mit dem ID `box` ein Click-Event-Handler hinzugefügt werden. Schreiben Sie also das Script einfach einmal entsprechend um. Ob Ihnen wieder drei Zeilen reichen werden? Na sicher!

```

$(document).ready( function() {
    $('#box p').click( function() {
        $(this).addClass("green");
    });
});

```

Dass `$('#box p')` in seiner Kürze dasselbe bewirkt wie der längliche Ausdruck `document.getElementById("box").getElementsByTagName("p")` haben Sie vielleicht bereits verkraftet. Erinnern Sie sich, dass Sie hier auf die `for`-Schleife verzichten können: jQuery wendet ja im Fall einer Knotenliste (Sie wissen, dass es sich nicht um eine »gewöhnliche« Knotenliste handelt) eine Methode auf alle Items der Liste an (als »implizite Schleife«). Praktisch!

Sobald ein `<p>`-Element tatsächlich geklickt wird, tritt die anonyme Funktion im Inneren von `click()` in dessen Namen in Aktion. (Wie Sie bereits zuvor erfahren haben, bezeichnet man diese Funktion als *Callback-Funktion*. Als solche wird sie erst dann ausgeführt, wenn das Klickereignis eintritt.)

Das `this` in ihrem Inneren bezieht sich, wie gehabt, auf den geklickten `<p>`-Container. Da dieses `this` (als herkömmlicher `<p>`-Container) mit `addClass()` nicht anfangen könnte, wrappen Sie es über die `$()`-Funktion wieder in ein jQuery-Objekt.

Unter die Lupe damit:

```

// die anonyme Funktion in click():
$('#box p').click( function() {
    // das Kontextobjekt this wird jqueryifiziert:
    $(this).addClass("green");
});

```

Hier zusammenfassend der Quellcode der gesamten Datei:

```
<html>
<head>
  <title>Listing 3</title>

  <!-- Zunächst die Styleangaben: -->
  <style type="text/css">
    p {
      cursor:pointer;
    }
    .green {
      color:#009933;
      background-color:#E2FFEC;
      cursor:default;
    }
  </style>

  <!-- Nun das Framework einbinden: -->
  <script type="text/javascript"
    src="../../jquery/jquery-1.4.min.js"></script>

  <!-- ...und jetzt unseren eigenen Code: -->
  <script type="text/javascript">
    $(document).ready(function() {
      $("#box p").click(function() {
        $(this).addClass("green");
      });
    });
  </script>
</head>
<body>
  <div id="box">
    <p>Erster Absatz</p>
    <p>Zweiter Absatz</p>
    <p>Dritter Absatz</p>
  </div>
</body>
</html>
```

Listing 3.3 Schriftfarbe ändern mit Click-Event (mit jQuery)

Wollen Sie innerhalb eines klickbaren Textabsatzes die Koordinaten des Klickereignisses erfassen, stoßen Sie auf eine weitere Diskrepanz zwischen den JavaScript-Implementierungen der Browser.

Der Standard verlangt die Übergabe der Informationen über ein Ereignis (u. a. Art und Ort des Ereignisses) an die durch das Ereignis getriggerte Funktion, die die Information auswerten kann. Die Informationsstruktur wird als ein JavaScript-Objekt übergeben, das allgemein als *Eventobjekt* bezeichnet wird. Es wird von der Zielfunktion durch einen Parameter (meist wird für diesen der Bezeichner *e* wie *event* verwendet) in den Argumentklammern »aufgefangen«:

```
meineP[i].onclick = function(e) { ... }, false);
```

Leider implementiert der Internet Explorer dies nicht auf die gleiche Weise. Nicht dass es hier keine Informationen über das stattgefundenere Ereignis gäbe. Allerdings lagern diese in einem Objekt *event*, das dem *window*-Objekt unterstellt ist: *window.event*. Die Funktion erhält keinen Wert übergeben, sondern muss sich diesen vielmehr »holen«.

In standardkonformen Browsern können Sie, um die Klickkoordinaten zu erfassen, schreiben:

```
meineP[i].onclick = function(e) {
    // Zugriff auf das übergebene Eventobjekt e:
    alert('Klick an ' + e.clientX + ' und ' + e.clientY);
};
```

Im Internet Explorer müsste dies wie folgt geschehen (Sie schreiben kurz *event* statt *window.event* und lassen dafür den Übergabeparameter in den Funktionsklammern weg – zum Glück sind wenigstens die Eigenschaften des Eventobjekts gleich benannt):

```
meineP[i].onclick = function() {
    // direkter Zugriff auf window.event:
    alert('Klick an ' + event.clientX +
        ' und ' + event.clientY);
};
```

Nun müsste, um zu entscheiden, nach welchem Modell gearbeitet wird, festgestellt werden, ob der Funktion etwas übergeben wird oder nicht. Hierfür gibt es einen alten Programmierertrick, der den Übergabeparameter prüft, ohne umständlich auf eine *if-else*-Bedingung zurückzugreifen. Sie setzen einfach (sehr viel kürzer) den ternären Operator *? : ein* – hier ein erläuterndes Beispiel:

```
meineP[i].onclick = function(e) {
    // wurde ein e übergeben?
    e ? alert("e definiert!") : alert("e undefiniert!");
    // und nun weiter...
};
```

Geprüft wird das übergebene `e`. Ist es undefiniert (wurde das Skript also im IE ausgeführt) und damit `false`, wird der Ausdruck *rechts* vom Doppelpunkt verwendet, ansonsten (e wird zu `true` ausgewertet) *links* davon. Die Inkompatibilität überwinden Sie demnach, indem Sie, bei `false`, `e` gleich `window.event` setzen (ansonsten `e` einfach belassen) und anschließend nach »Schema F« weiterverfah-
ren:

```
meineP[i].onclick = function(e) {
    e? e : e = window.event;
    // Prima, auch in IE heißt window.event jetzt e:
    alert('Klick an ' + e.clientX + ' und ' + e.clientY);
};
```

Abgesehen von dieser kleinen »Verrenkung« ist die Lösung nun recht einfach browserübergreifend zu schreiben. Der gesamte Code sieht jetzt so aus:

```
window.onload = function() {
    var meineP = document.getElementsByTagName("p");
    for(var i =0, len=meineP.length; i<len; i++) {
        meineP[i].onclick = function(e) {
            e? e : e = window.event;
            alert('Klick an ' + e.clientX + " und " + e.clientY);
        };
    }
}
```

Schauen Sie aber nun einmal, wie jQuery dies löst:

```
$(document).ready( function() {
    $('p').click( function(e) {
        alert('Klick an ' + e.clientX + " und " + e.clientY);
    });
});
```

Offensichtlich spart jQuery nicht nur (wie Sie bereits wissen) die `for`-Schleife ein, sondern sorgt obendrein dafür, dass die durch ein Ereignis getriggerte Funktion stets ein Eventobjekt übergeben bekommt. Die `$()`-Funktion bietet aber noch mehr, lassen Sie sich überraschen!

3.6 Give me more! – Verkettung von jQuery-Methoden

Eine angenehme Eigenheit von jQuery ist, dass Sie mehrere jQuery-Methoden hintereinander an ein jQuery-Objekt hängen können. Worauf eine solche *Verkettung* prinzipiell aufbaut, haben Sie im vorangegangenen Kapitel erfahren. jQuery

unterstützt dies auf ganzer Linie. Das Geheimnis besteht darin, dass jede jQuery-Methode wieder ein jQuery-Objekt zurückgibt.

Dieses jQuery-Objekt (es handelt sich um das gleiche wie am Anfang, nur mit den »eingearbeiteten« Veränderungen der eben angewendeten Methode) steht somit quasi »am Ausgang« zur Verfügung, sodass eine weitere Methode darauf angewendet werden kann. Und dies funktioniert immer so weiter. Sie können sich das wie ein Werkstück vorstellen, das eine Montagestraße entlangläuft:

```
// die jQuery-Methoden werden einfach verkettet:
$(selektorausdruck).methode1().methode2().methode3();
```

... was Sie (nur zur Erinnerung) auch wie folgt schreiben können (Vorsicht, bloß kein Semikolon an die Zeilenenden setzen – außer ganz am Schluss):

```
// die jQuery-Methoden werden einfach verkettet:
$(selektorausdruck)
    .methode1()
    .methode2()
    .methode3();
```

Aber schauen Sie es sich einfach einmal genau an.

Sie haben bereits mit `$(this).addClass("green")` gesehen, dass Sie einem Element eine CSS-Klasse hinzufügen können. Lassen Sie uns dieses Beispiel erweitern.

3.6.1 Den Elternknoten eines Elements manipulieren

Sagen wir, wir wollen das Elternelement von `this` (den die `<p>`-Container umgebenden `<div>`-Container) mit den Klassen `boxColor-0` bis `boxColor-2` versehen.

Anklickbar sein sollen hierbei nach wie vor die Textabsätze selbst. Sie müssen es nun so einrichten, dass jQuery dem Div die CSS-Klasse abhängig davon zuweist, welcher `<p>`-Container geklickt wurde. Das heißt, für den ersten `<p>`-Container erhält der Div die Klasse `boxColor-0`, für den zweiten vergeben Sie `boxColor-1` etc.

Bei jedem Klick muss die vorher an den Div vergebene Klasse allerdings entfernt werden. Ebenso soll nur der eben angeklickte `<p>`-Container die Klasse `green` besitzen, von allen anderen muss sie wieder verschwinden. Eine ganze Menge an Randbedingungen. Allerdings halb so schlimm, wenn Sie sich Stück für Stück an die Lösung heranarbeiten.

Selektieren Sie zunächst alle `<p>`-Container in `Div#box`, und machen Sie sie klickbar:

```
$("#box p").click( function() {
    ...
});
```

So weit, so gut. Der geklickte `<p>`-Container soll die Klasse `green` erhalten. Diesen Container holen Sie mit `$(this)`. Auch das kennen Sie bereits:

```
$("#box p").click( function() {
    $(this).addClass("green");
});
```

Jetzt wird es interessanter – Sie haben den geklickten `<p>`-Container bereits in der Hand, wollen jetzt aber etwas mit dessen Elternknoten, dem Div, machen. Gut, wechseln Sie einfach vom `<p>`-Container dorthin.

jQuery bietet Ihnen hierfür die Methode `parent()`. Weil Sie im Baum von einem Element zum anderen klettern (Bergsteiger bezeichnen dies als *traversieren*), zählt jQuery die `parent()`-Methode zu den *Tree-traversing*-Methoden.

Gehen Sie also zum Div:

```
$("#box p").click( function() {
    $(this).addClass("green").parent();
});
```

Nochmals zum Verständnis – der Ausdruck `$(this).addClass("green").parent()` *ist* jetzt der `<div>`-Container. Kürzer hätten Sie `$(this).parent()` schreiben können, aber Sie hatten ja »unterwegs noch etwas zu erledigen« ...

Nun sorgen Sie dafür, dass am `<div>`-Container CSS-mäßig »Tabula rasa« gemacht wird: Mit `removeClass()` entfernen Sie eventuell dort befindliche CSS-Klassen:

```
$("#box p").click( function() {
    $(this).addClass("green").parent().removeClass();
});
```

Jetzt soll dem Div die gewünschte Klasse hinzugefügt werden. Dies machen Sie wiederum mit `addClass()`. Weichen Sie auf mehrzeilige Schreibweise aus, und kommentieren Sie das Geschehen ein wenig mit:

```
$("#box p").click( function() {
    $(this)                // der geklickte P-Container
    .addClass("green")     // erhält die Klasse green
    .parent()              // wir traversieren zum Elternknoten
    .removeClass()         // entfernen dort alle CSS-Klassen
    .addClass( ... );     // und fügen eine neue hinzu
});
```

Stopp – wo bekommen Sie nun den Klassennamen her, den Sie `.addClass()` übergeben müssen? Der Bezeichner beginnt immer mit `boxColor-` und geht dann mit einer der Ziffern 0, 1 oder 2 weiter, die den `<p>`-Container bezeichnen. Das erinnert an die Indexziffern eines Arrays.

Handelt es sich hier denn um ein Array? Sozusagen. Und zwar, wenn Sie den grundlegenden Ausdruck `$("#box p")` hernehmen. Von diesen drei `<p>`-Containern ist einer angeklickt worden, der im Rahmen unserer Funktion durch `this` bezeichnet wird. jQuery bietet Ihnen nun eine Methode `.index()`, die Ihnen die Position `pos` eines Objekts in einer Knotenliste nennen kann. Das Prinzip ist dieses:

```
var pos = $(knotenliste).index(vergleichsobjekt);
```

Auf unseren Fall angewendet, schreiben Sie beispielsweise:

```
var pos = $("#box p").index(this);
```

Speichern Sie aber den Rückgabewert nicht, sondern setzen Sie ihn direkt ein:

```
$("#box p").click( function() {
    $(this)           // der geklickte P-Container
    .addClass("green") // erhält die Klasse green
    .parent()         // wir traversieren zum Elternknoten
    .removeClass()     // entfernen dort alle CSS-Klassen
    .addClass("boxColor-" + $("#box p").index(this));
});
```

Sie haben nun ganz vergessen, dafür zu sorgen, dass noch die Klasse `green` von vorher geklickten `<p>`-Containern entfernt werden muss. Dies erledigen Sie einfach pauschal zu Beginn der Funktion. Macht nichts, dass einer der Absätze gleich wieder die gleiche Klasse erhält ...

```
$("#box p").click( function() {
    // pauschal überall die Klasse green entfernen:
    $("#box p").removeClass("green");
    $(this)           // der geklickte P-Container
    .addClass("green") // erhält die Klasse green
    .parent()         // wir traversieren zum Elternknoten
    .removeClass()     // entfernen dort alle CSS-Klassen
    .addClass("boxColor-" + $("#box p").index(this));
});
```

Das ist jetzt insoweit schön, als dass es funktioniert. Allerdings ist es ein wenig unordentlich. Außerdem haben Sie nun dreimal ein jQuery-Objekt mit demselben Ausdruck `$("#box p")` gebildet. Das wirkt unökonomisch. Sie könnten dieses Objekt auch speichern, also in eine Variable packen, und »recyceln«:


```
var obj = $("#box p");
```

Sie erhalten dann:

```
var obj = $("#box p");
obj.click( function() {
    // pauschal überall die Klasse green entfernen:
    obj.removeClass("green");
    $(this)                // der geklickte P-Container
    .addClass("green")     // erhält die Klasse green
    .parent()              // wir traversieren zum Elternknoten
    .removeClass()         // entfernen dort alle CSS-Klassen
    .addClass("boxColor-" + obj.index(this));
});
```

Damit lässt sich schon besser arbeiten. Nun könnten Sie auch das ständige Hin und Her mit Entfernen und Hinzufügen von Klassen sortieren:

```
var obj = $("#box p");
obj.click( function() {
    // erst alles, was Klassen entfernt
    obj.removeClass("green").parent().removeClass();
    // jetzt Klassen hinzufügen:
    $(this).addClass("green").parent()
    .addClass("boxColor-" + obj.index(this));
});
```

Lagern Sie nun noch die Indexerzeugung aus dem unteren Ausdruck aus:

```
var obj = $("#box p");
obj.click( function() {
    // welcher P-Container wurde geklickt?
    var pos = obj.index(this);
    // erst alles, was Klassen entfernt
    obj.removeClass("green").parent().removeClass();
    // jetzt Klassen hinzufügen:
    $(this).addClass("green").parent()
    .addClass("boxColor-" + obj.index(this));
});
```

So sieht die Funktion nun im Großen und Ganzen aus – das vorher Geschriebene muss lediglich noch in einen `ready`-Block eingefügt werden:

```
<html>
<head>
    <title>Listing 4</title>

    <!-- Zunächst die Styleangaben: -->
```

```

<style type="text/css">
  p {
    cursor:pointer;
  }
  .green {
    color:#009933;
    background-color:#E2FFEC;
    cursor:default;
  }
  .boxColor-0 {
    background-color:#CCCCCC;
  }
  .boxColor-1 {
    background-color:#EEEEEE;
  }
  .boxColor-2 {
    background-color:#999999;
  }
</style>

<!-- Nun das Framework einbinden: -->
<script type="text/javascript"
  src="../../jquery/jquery-1.4.min.js"></script>

<!-- ...und jetzt unseren eigenen Code: -->
<script type="text/javascript">
  $(document).ready(function() {
    var obj = $("#box p");
    obj.click(function() {
      var pos = obj.index(this);
      obj.removeClass()
      .parent().removeClass();
      $(this).addClass("green")
      .parent().addClass("boxColor-" + pos);
    });
  });
</script>
</head>
<body>
  <div id="box">
    <p>Erster Absatz</p>
    <p>Zweiter Absatz</p>
    <p>Dritter Absatz</p>
  </div>

```

```
</body>  
</html>
```

Listing 3.4 Verkettung von jQuery-Funktionen

3.7 Zusammenfassung

jQuery stellt Ihnen einen wichtigen *Event* zur Verfügung, mit dem Sie überprüfen können, ob ein HTML-Baum fertig geladen wurde, und zwar `$(document).ready(fn)`, um erst anschließend, nach der Prüfung, eine Funktion auszuführen. Es stellt Ihnen ein einfaches Modell zur Navigation durch den Dokumentenbaum zur Verfügung, *Selektoren*, die sich an die Spezifikationen der CSS-Selektoren orientieren.

Sie können aus einem Arsenal an Werkzeugen wählen, mit dem Sie eine HTML-Seite manipulieren können, wobei Sie bei Ihren ersten Gehversuchen `.addClass(name)` kennengelernt haben. Sie können nun zudem mehrere jQuery-Methoden miteinander verketteten. Alles in allem entsteht damit ein schlanker Code, und eine strikte Strukturierung Ihrer Projekte wird möglich, da Sie die HTML-Seite von allem möglichen Ballast befreien und diesen zu Ihrer besseren Übersicht in externe Dateien auslagern können.

Und wie jetzt weiterarbeiten?

Mit Kapitel 4, »jQuery – die Übersicht«, folgt ein längeres (zugegeben sehr langes) Kapitel, in dem wir Ihnen das gesamte Vokabular von jQuery vorstellen und es ausführlich besprechen. Irgendwo muss ja alles stehen. Sie können es natürlich hier und jetzt durchlesen, um einen Kompletteindruck zu erhalten. Vielleicht möchten Sie aber stattdessen die Ärmel hochkrempeln, die Siebenmeilenstiefel anlegen und direkt zum Praxisteil in Kapitel 5, »jQuery – der Praxiseinsatz«, springen. Dort werden die jQuery-Methoden angewendet, aber nicht erklärt. Sie können die vorgestellten Beispiele natürlich trotzdem nachvollziehen und nur in den Referenzteil zurückblättern, um Hintergrundinformationen zu erhalten.

»For me, jQuery is much more than a block of code. It's the sum total of experiences that have transpired over the years in order to make the library happen. The considerable ups and downs, the struggle of development together with the excitement of seeing it grow and succeed.«
John Resig

4 jQuery – die Übersicht

Es ist jetzt der Moment gekommen, sich einen Überblick über die gesammelten Fähigkeiten von jQuery zu verschaffen. Statt pauschal die Frage, was dieses Framework eigentlich so kann, mit »im Prinzip alles« zu beantworten, untergliedern wir dieses weite Gebiet in verschiedene funktionale Bereiche.

Das vorliegende, aus diesem Grund etwas längere Kapitel wird Ihnen eine Einführung in alle Features geben, die an jQuerys `$()`-Funktion und das `$`-Objekt geknüpft sind.

4.1 Im Zentrum – das jQuery-Objekt

Sie haben gesehen, dass der erste Schritt bei der Anwendung von jQuery in der Regel darin besteht, ein *jQuery-Objekt* zu bilden, das diejenigen Teile des Dokuments verkapselt (meist wird es sich um einen oder mehrere Elementknoten handeln), mit denen anschließend weitergearbeitet werden soll. Das jQuery-Objekt, das am Anfang steht (genauer gesagt, die HTML-Elemente, die in diesem Objekt verkapselt sind und denen es so seine Eigenschaften verleiht), bezeichnen wir als die *primäre Collection* (siehe Abbildung 4.1).

Die Anwendung einer jQuery-Methode oder eines Filters auf diese primäre Collection ergibt wieder ein jQuery-Objekt (nennen wir es *sekundäre Collection*), auf das ebenfalls wieder eine Methode oder ein Filter angewendet werden kann. Dieses Prinzip setzt sich beliebig fort – es wird als *Verkettbarkeit* (chainability) bezeichnet. Sie haben es bereits kennengelernt.

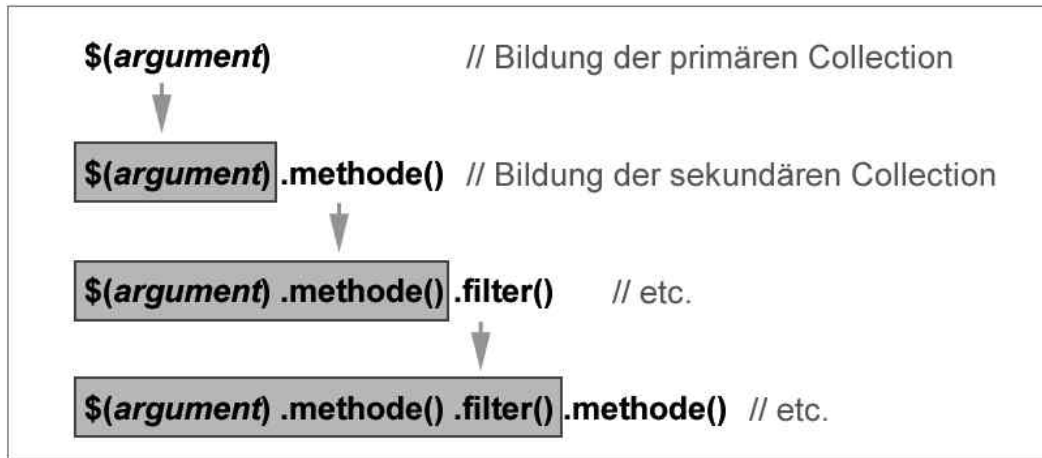


Abbildung 4.1 Verkettbarkeit von jQuery-Methoden

4.1.1 Drei Arten von jQuery-Methoden

Rufen Sie sich nochmals in Erinnerung, dass einem jQuery-Objekt *alle* jQuery-Methoden zur Verfügung stehen und dass Sie die Werte *aller* jQuery-Properties dieses Objekts auslesen können. Auf diesem Weg können die Eigenschaften einer Collection in Erfahrung gebracht werden, und Sie können sie infolgedessen sinnvoll einsetzen.

Die jQuery-Methoden sind dabei nicht völlig gleich. Es gibt drei Gruppen von Methoden, je nachdem, was ihr Rückgabewert ist:

1. Transparente jQuery-Methoden

Diese Methoden geben ein jQuery-Objekt zurück, das der Eingabe-Collection entspricht. Sie können Arbeiten an den Elementen der Collection verrichten, geben die Collection aber ansonsten unverändert an die nächste Methode weiter, die diese wieder als Input verwendet. Diese Methoden sind ohne Einschränkung verkettbar.

2. Destruktive jQuery-Methoden

Diese Methoden geben zwar ein jQuery-Objekt zurück, dieses entspricht allerdings nicht der Eingabe-Collection. Sie können eine reduzierte, erweiterte oder vollkommen andere Collection an die nächste Methode weitergeben. Auch diese Methoden sind uneingeschränkt verkettbar. jQuery kennt jedoch Wege, die Veränderungen an der Collection durch destruktive Methoden rückgängig zu machen,¹ um zur primären Collection zurückzukehren.

3. Terminierende jQuery-Methoden

Manche (die wenigsten) jQuery-Methoden geben gelegentlich gar kein jQuery-Objekt zurück, sondern etwas anderes, das nicht die jQuery-Eigen-

¹ Hier soll kein Geheimnis daraus gemacht werden: Es handelt sich um die Methode `end()`.

schaften besitzt. Es kann sich um einen DOM-Knoten, einen String oder einen anderen Datentyp handeln. Nach Anwendung einer solchen Methode steht keine Collection mehr zur Verfügung, die einer Folgemethode als Eingangswert dienen könnte, und so kann auch keine jQuery-Methode auf den Rückgabewert angewendet werden. Diese Methoden sind also *nicht verkettbar*, sondern beenden stets die jQuery-Kette. (Beispiele wären die Methode `text()`, sobald sie einen Textinhalt liest und daher einen String zurückgibt, oder `get()`, das ein Array aus DOM-Knoten erzeugt.)

4.2 Die Funktion `$()` und ihre Signatur

Betrachten wir aber jetzt zunächst die grundsätzlichen Möglichkeiten, die sich uns bieten, die primäre Collection zusammenzustellen. Dies geschieht, indem der jQuery-Funktion ein *Argument* übergeben wird – aber was für eine Art von Argument? Hier existiert eine Reihe von Möglichkeiten – man sagt, die Funktion `$()` verfügt über verschiedene »Signaturen«, die sich durch Art und Anzahl der Argumente unterscheiden.

4.2.1 DOM-Element oder jQuery-Objekt als Argument

Wie Sie im vorangegangenen Kapitel gesehen haben, ist die primäre Aufgabe der `$()`-Funktion das »jQueryifizieren« eines Elementknotens des Dokuments. Hierfür erhält die Funktion unmittelbar eine Referenz auf den betreffenden Knoten übergeben (wie beispielsweise durch `getElementById()` erzeugt), oder ein JavaScript-Objekt bzw. eine Variable, die eine solche Referenz darstellt (wie über das Schlüsselwort `this`). Ebenso gut gelingt das »jQueryifizieren« eines ganzen Arrays aus Knoten, wie Sie sie durch `getElementsByTagName()` erhalten.

In allen Fällen gibt die `$()`-Funktion ein jQuery-Objekt zurück, das als primäre Collection folgender Aktionen dient. Auch ein bereits existierendes jQuery-Objekt kann als Eingangswert verwendet werden – in diesem Fall ist der Rückgabewert ein mit diesem identisches, also »geklontes« jQuery-Objekt.

Signatur	Argument(e)	Erläuterung
<code>\$(elem)</code>	1) DOM-Element	Ein DOM-Knoten, der als Collection in ein jQuery-Objekt verkapselt werden soll
<code>\$(elems)</code>	1) Array aus DOM-Elementen	Ein Array aus DOM-Knoten, die als Collection in ein jQuery-Objekt verkapselt werden sollen

Tabelle 4.1 Die Signaturen der jQuery-Funktion: DOM-Elemente und jQuery-Objekt

Signatur	Argument(e)	Erläuterung
\$(jQuery)	1) jQuery-Objekt	Ein jQuery-Objekt, das hierbei in ein neues jQuery-Objekt verkapselt (geklont) wird

Tabelle 4.1 Die Signaturen der jQuery-Funktion: DOM-Elemente und jQuery-Objekt (Forts.)

Elementknoten als Argument

Eine Referenz auf DOM-Knoten kann direkt übergeben werden:

```
$(document.forms[0]).addClass("formclass");
```

Collection aus Elementknoten als Argument

Die `$()`-Funktion nimmt auch eine Tag-Collection entgegen und verkapselt die betreffenden Elemente in ihrer Gesamtheit in ein jQuery-Objekt:

```
var meineP = document.getElementsByTagName('p');
$(meineP).addClass("content");
```

jQuery-Objekt als Argument

Die `$()`-Funktion akzeptiert auch andere jQuery-Objekte als Argument:

```
var obj = $("#container");
$(obj).addClass("classname");
```

Seitenblick: Selektieren von DOM-Knoten in JavaScript

Das *Selektieren* eines Knotens im Dokumentbaum geschieht in JavaScript über eine Methode, die diesen Knoten als Objekt zurückliefert. Die direkteste Art geschieht über einen *Identifizierer*, der dem Knoten als `id`-Attribut beigegeben ist. Die `document`-Methode `getElementById()` nimmt einen ID-Namen als Zeichenkette entgegen (es wird also nicht wie beim CSS-Selektor die Raute vorangestellt). Sie liefert einen *Einzelknoten* als DOM-Element zurück:

```
// liefert einen Elementknoten als Objekt zurück:
var meinKnoten = document.getElementById("meinID");
```

Eine weitere Methode steht in Form `getElementsByTagName()` zur Verfügung, die einen *Elementnamen* als String akzeptiert. Elementnamen treten mehrfach im Dokument auf. Demzufolge gibt die Methode auch keinen Einzelknoten zurück, sondern mehrere, die in einer sogenannten *Collection* zusammengefasst sind.² Für unsere Begriffe kann eine Collection mit einem Array gleichgesetzt werden (technisch betrachtet ist sie jedoch keines):

```
// liefert eine Collection mehrerer Elementknoten zurück:
var meineCollection = document.getElementsByTagName("p");
```

² Achtung – auch ein Einzelknoten ist dann eine Collection der Länge 1.

Erwähnenswert ist, dass die Methode `getElementsByName()` auch jedem Elementknoten zur Verfügung steht. Dies bietet interessante Varianten. Holen Sie sich nun zunächst einen Knoten:

```
var meinKnoten = document.getElementById("inhalt");
```

Nehmen wir jetzt einmal an, `#inhalt` sei ein Container, der den Inhaltsbereich des Dokuments enthält. Wir wollen nun alle Textabsätze im Inhalt selektieren (aber nicht anderswo). In `meinKnoten` ist ein Elementknoten referenziert, daher geht Folgendes:

```
var dieTextabsaeetze = meinKnoten.getElementsByTagName("p");
```

Oder aber, ohne den Knoten in `meinKnoten` zwischenspeichern:

```
var dieTextabsaeetze = document.getElementById("inhalt")
    .getElementsByTagName("p");
```

Beachten Sie, dass wir hier aus Platzgründen wieder eine Schreibweise über mehrere Zeilen gewählt haben. Dies ist vollständig legal. Im Grunde reichen diese beiden Methoden aus, um auf alle Knoten des Dokuments zuzugreifen – sofern diese entweder einen ID besitzen oder der Tag-Name bekannt ist.

Eine dritte Methode `getElementsByName()` steht ebenfalls für den Dokumentknoten und als Elementknoten zur Verfügung – allerdings gehört sie zur DOM Level 3-Implementierung, die nicht allen Browsern zur Verfügung steht. Sie muss, so praktisch sie ist, daher mit Vorsicht eingesetzt werden.

4.2.2 HTML-String als Argument

Meist greift jQuery zur Erstellung der primären Collection auf ein bestehendes Dokument und darin enthaltene Elemente zu. Dies muss jedoch nicht so sein – die `$()`-Funktion kann auch als »Fabrik« für Elementknoten dienen, wenn diese ad hoc benötigt werden. Dies funktioniert analog zur DOM-Methode `document.createElement()`, nur einfacher. Hierbei nimmt `$()` ein oder zwei Argumente entgegen, von denen das erste immer eine Zeichenkette ist, aus der HTML konstruiert werden soll.

Signatur	Argument(e)	Erläuterung
<code>\$(htmlString)</code>	1) String	Ein String <code>htmlString</code> , aus dem ein DOM-Element mit Inhalt konstruiert wird, das als primäre Collection Verwendung findet
<code>\$(htmlString, doc)</code>	1) String 2) Dokumentknoten	Ein String <code>htmlString</code> , aus dem ein DOM-Element mit Inhalt konstruiert wird, das als primäre Collection Verwendung findet <i>und</i> ein Dokumentknoten, der das Dokument bestimmt, in dem das konstruierte DOM-Element verwendet werden soll

Tabelle 4.2 Die Signaturen der jQuery-Funktion: HTML-String

Signatur	Argument(e)	Erläuterung
<code>\$(htmlString, prop)</code>	1) String 2) Property-Map	Ein String <code>htmlString</code> , aus dem ein DOM-Element mit Inhalt konstruiert wird, das als primäre Collection Verwendung findet <i>und</i> eine Property-Map in Form eines Objekts, das Attribute, Events und Methoden bestimmt, die dem konstruierten DOM-Knoten zugeordnet werden sollen.

Tabelle 4.2 Die Signaturen der jQuery-Funktion: HTML-String (Forts.)

Einfacher HTML-String als Argument

Im einfachsten Fall nimmt `$()` ein einzelnes Argument in Form einer Zeichenkette, die ein HTML-Fragment enthält. Aus dieser wird derjenige Tag als Elementknoten erzeugt, der die äußere Instanz des Fragments bildet, der Rest wird als dessen Inhalt eingesetzt. Letzteres funktioniert analog zur Zuweisung zur Eigenschaft `innerHTML` eines Elementknotens, geschieht also nicht über Knotenerzeugung.

Fügt einen HTML-String einem angegebenen Element hinzu:

```
$("<p>Hallo Erde</p>").appendTo("#container");
```

Erzeugt im Dokument für

```
<div id="container">
  <p>Hallo Mond</p>
</div>
```

folgendes Ergebnis:

```
<div id="container">
  <p>Hallo Mond</p>
  <p>Hallo Erde</p>
</div>
```

Die HTML-Strings müssen wohlgeformt sein. Das einfache Übergeben eines Start-Tags wie in `$("<div>")` reicht also nicht – `$("<div></div>")` sollte es schon sein. Auch die (zugegeben, für dieses Element unorthodoxe) XML-Short-Tag-Schreibweise `$("<div />")` wird von jQuery akzeptiert (allerdings auf die »herkömmliche« Art umgesetzt).

HTML-String mit `ownerDocument`

Optional kann ein `ownerDocument` angegeben werden. Interessant wird dieses optionale Argument, wenn man Inhalte in einem `iFrame` namens `#myIframe` erzeugen will:

```
$("#myIframe").ready(function () {
    $("<p>Hallo iFrame</p>", frames['myIframe'].document)
    .appendTo("body");
});
```

4.2.3 Callback-Funktion als Argument

Ein weiterer Argumenttyp, den die jQuery-Funktion akzeptiert, ist ein Funktionsobjekt. Dieses wird automatisch als Callback-Funktion an den `ready()`-Event von jQuery gebunden.

Signatur	Argument(e)	Erläuterung
<code>\$(fn)</code>	1) Callback-Funktion	Diejenige Funktion, die als Callback ausgeführt werden soll, wenn das Dokument geladen ist Anmerkung: Abkürzung für <code>\$(document).ready(fn)</code>
<code>\$()</code>	– (leer)	Wird die jQuery-Funktion ohne Argumente aufgerufen, gibt sie ein jQuery-Objekt mit einer leeren Collection zurück. Achtung: In Vorgängerversionen von jQuery 1.4 bewirkte der leere Aufruf <code>\$()</code> die Selektion des Dokumentknotens als Default-Collection.

Tabelle 4.3 Die Signaturen der jQuery-Funktion: Callback-Funktion und leerer Aufruf

In diesem Sinne ist die direkte Übergabe der Callback-Funktion nur eine weitere (weniger umständlich zu schreibende) Methode, jQuery mitzuteilen, was nach Laden des Dokuments geschehen soll.

```
$( function() { ... } )
```

ist also eine Abkürzung von

```
$(document).ready( function() { ... } )
```

wobei letztere Schreibweise die übliche ist, weil expliziter.

4.2.4 CSS-Selektor als Argument

Der wichtigste Argumenttyp für die `$()`-Funktion sind die CSS-Selektoren. jQuery verwendet sie, um schnell und mit einer dem Webprogrammierer ohnehin geläufigen Syntax Elemente im Dokument zu finden und als Teil einer jQuery-Collection für weitere Bearbeitung zur Verfügung zu stellen. Hierbei unterstützt jQuery sämtliche CSS-Selektortypen, eingeschlossen deren neueste Spielarten aus CSS 3.

Signatur	Argument(e)	Erläuterung
<code>\$(sel)</code>	String	Ein String <code>sel</code> , der einen CSS-Selektor (mit optionalen jQuery-Filterkriterien) enthält, der Elemente des Dokuments selektiert
<code>\$(sel, context)</code>	String DOM-Element <i>oder</i> Dokumentknoten <i>oder</i> jQuery-Objekt	Ein String <code>sel</code> , der einen CSS-Selektor (mit optionalen jQuery-Filterkriterien) enthält, der Elemente des Dokuments selektiert <i>und</i> ein Objekt <code>context</code> , das den Kontext der Auswertung bestimmt (als DOM-Knoten, Dokumentknoten oder jQuery-Objekt)

Tabelle 4.4 Die Signaturen der jQuery-Funktion: CSS-Selektor und optionaler Kontext

Die Selektoren werden einfach in Form einer Zeichenkette als Argument der jQuery-Funktion übergeben. Hierbei gilt folgende allgemeine Schreibweise:

```
jQuery("CSS-Selektor"); // erzeugt eine Collection
```

Kürzer und deshalb verbreiteter ist die Schreibweise der Funktion als `$()`:

```
$("CSS-Selektor"); // erzeugt eine Collection
```

Achten Sie darauf, dass die Selektoren der `$()`-Funktion grundsätzlich als String, also *in Anführungszeichen*, übergeben werden müssen. Der Rückgabewert eines jQuery-Selektors ist jeweils eine Collection.

Die `$()`-Funktion ist hier auch mit einer *zweiten Signatur* in Gebrauch:

```
jQuery("CSS-Selektor", context); // erzeugt eine Collection
```

Das Argument `context` ist optional. Als Default (wenn nur ein Argument übergeben wird) dient das HTML-Dokument als Kontext. Als `context` kann aber jeder beliebige DOM-Knoten angegeben werden – beispielsweise ein Formular:

```
$("input:submit", document.forms[0]).addClass("button");
```

oder

```
$("#input:submit", "form#contact").addClass("button");
```

Dieser Ausdruck fügt einem SUBMIT-Button im Inneren des Formulars mit dem ID `contact` eine CSS-Klasse hinzu. Die Angabe eines Kontexts verringert unter Umständen den Aufwand bei der Erstellung der Collection, indem der zu durchsuchende Bereich des Dokuments eingeschränkt wird.

Sonderzeichen in Selektorstrings escapen

Da jQuerys *Sizzle Engine*, die die CSS-Selektorausdrücke interpretiert, bestimmte Zeichen als funktionale Zeichen (Steuerzeichen) betrachtet, dürfen Sie diese nicht als »literale Zeichen« in einem String verwenden. Stattdessen müssen Sie sie durch eine *Escape-Sequenz* ersetzen. Von Javascript her kennen ist Ihnen bekannt, dem zu maskierenden Zeichen als Symbol den Backslash `\` voranzustellen. Achtung – für jQuery-Selektoren genügt dies nicht: Die Maskierung der Escape-Sequenz (der Backslash) muss *selbst zusätzlich escaped* werden: Um den Backslash zu maskieren, müssen Sie diesen also *verdoppeln* `\\`.

Folgenden Sonderzeichen müssen in jQuery-Selektoren daher stets *zwei Backslashes* vorangestellt werden, wenn sie literal auftreten:

```
#; & , . + * ~ ' : " ! ^ $ [ ] ( ) = > | /
```

Als Teil eines ID-Selektors muss das Hash-Zeichen `#` nicht maskiert werden, ebenso wenig der Punkt als Teil des Class-Selektors. In anderen Fällen ist eine Maskierung von Zeichen erforderlich. Kritisch sind Punkt und Doppelpunkt, weil sie als Teil eines HTML-Namens (also beispielsweise im Rahmen eines IDs) auftreten dürfen. Für folgenden Container mit etwas konstruiert wirkendem ID-Bezeichner:

```
<div id="asdf:bce"></div>
```

liefert der Ausdruck `$("#asdf:bce").length` den Wert 0 zurück, das Element wurde also nicht erfasst. Mit `$("#asdf\\:bce").length` erhalten Sie hingegen als Länge der Collection den korrekten Wert 1 – der Container wurde gefunden.

4.3 CSS-Selektoren für die primäre Collection

Betrachten wir nun einmal das Gebiet der CSS-Selektoren etwas ausführlicher, beginnend mit den grundlegenden Selektoren und ihrer Wirkung in jQuery. Wir werden anschließend auch die komplexeren Selektoren betrachten und feststellen, dass jQuery über die herkömmlichen CSS-Selektoren hinaus (die allerdings so gut wie alle unterstützt werden) auch eigene Ausdrücke beisteuert, die eine Selektion noch weiter verfeinern können.

4.3.1 Die Basisselektoren

Die grundlegenden Selektoren in jQuery sind ID, Element (Typselektor), Klasse und der Universalselektor `*`.

Selektor	Typ	Erläuterung
<code>*</code>	Alle	Selektiert typunabhängig alle Elemente.
<code>E</code>	Typ	Selektiert alle Elemente mit dem Bezeichner E.
<code>.class</code>	Klasse	Selektiert alle Elemente mit der genannten CSS-Klasse.
<code>#id</code>	ID-Selektor	Selektiert ein Element mit dem übergebenen ID.

Tabelle 4.5 Basisselektoren in jQuery

Mit den Selektoren ID, Element und CSS-Klasse wählen Sie bestimmte Elemente im DOM-Baum aus. Der Universalselektor wiederum selektiert beliebige Elemente (wir sprechen hier auch von »Wildcard«) – er wird meist nur in Kombination mit anderen Selektoren eingesetzt. Die Anwendung dieser Selektoren ist selbsterklärend.

4.3.2 Mehrfachklassenselektor

Selektor	Typ	Erläuterung
<code>.class1.class2</code>	Klasse	Selektiert Elemente mit allen genannten CSS-Klassen.

Tabelle 4.6 Mehrfachklassenselektor

Eine Klassenselektion können Sie verschärfen, indem Sie zwei (oder mehr) gleichzeitig vorhandene Klassen in einem Element verlangen:

```
$(".detail.color")
```

Dieser Selektor findet die Elemente

```
<span class="detail color"></span>
```

aber nicht

```
<span class="detail"></span>
```

und auch nicht

```
<span class="color"></span>
```

4.3.3 Gruppen- und Kontextselektoren

Selektor	Typ	Erläuterung
<code>sel1, sel2, ... selN</code>	Selektorgruppe	Selektiert die Kombination aus allen durch die Selektoren <code>sel1</code> bis <code>selN</code> ausgewählten Knoten.
<code>E F</code>	Kontext	Selektiert alle Elemente <code>F</code> , die Nachfahren eines Elements <code>E</code> sind.
<code>E > F</code>	Kontext	Selektiert alle Elemente <code>F</code> , die Kindknoten eines Elements <code>E</code> sind.
<code>E + F</code>	Kontext	Selektiert alle jeweils unmittelbar auf ein Element von Typ <code>E</code> folgenden Elemente vom Typ <code>F</code> .
<code>E ~ F</code>	Kontext	Selektiert alle folgenden Geschwisterknoten eines Elements <code>E</code> , die den Typ <code>F</code> besitzen.

Tabelle 4.7 Gruppen- und Kontextselektoren

Gruppenselektor `sel1, sel2, sel3`

Auch Kombinationen sind mehrere Selektoren möglich. In diesem Fall addieren sich die Ergebnisse der Einzelselektoren in eine jQuery-Collection:

```
$("#id1, #id2, .detail")
```

Dann sieht eine Selektion folgendermaßen aus:

```
<html>
<head>
<title>jQuery - Beispiel 7</title>
</head>
<body>
  <div id="id1"></div>
  <div id="id2"></div>
  <span class="detail">
    <h1>Eine Überschrift</h1>
    <p>Ein Absatz</p>
    <a href="#">Ein Anker</a>
  </span>
</body>
</html>
```

Merken Sie sich, dass sich die Selektion nicht auf Unterelemente vererbt. Wenn Sie die Elemente für Überschrift, Absatz oder Anker selektieren wollen, nutzen Sie entweder den Universalselektor oder sprechen sie direkt über den Elementnamen an.

Kontextselektor E F

Sie können einfache Selektoren kombinieren, um so einen hierarchischen Kontext zu nennen, auf dem die Auswahl basieren soll:

`$("div #box .detail *")` selektiert in folgendem Quelltext die fett hervorgehobenen Elemente:

```
<html>
<head>
<title>jQuery - Beispiel 7</title>
</head>
<body>
  <div>
    <div id="box">
      <span class="detail">
        <h1>Eine Überschrift</h1>
        <p>Ein Absatz</p>
        <a href="#">Ein Anker</a>
        <span>Innerer Span</span>
      </span>
    </div>
  </div>
</body>
</html>
```

Kindknotenselektor E > F

Mit dem Kindknotenselektor werden die direkten Kindelemente eines Elements selektiert. Hierbei können Sie auch einen Typ angeben:

`$("#box > span")`

Dann werden folgende Elemente ausgewählt:

```
...
<body>
  <div id="box">
    <span class="detail">
      <h1>Eine Überschrift</h1>
      <p>Ein Absatz</p>
      <a href="#">Ein Anker</a>
      <span>Innerer Span</span>
    </span>
    <div id="id1"></div>
  </div>
</body>
...
```

Es werden nur die direkten Nachfahren selektiert, weitere Abkömmlinge bleiben außen vor.

Immediate Following Sibling-Selektor E + F

Der *Immediate Following Sibling*-Selektor dient dazu, ein Element in Abhängigkeit des unmittelbar vorhergehenden Geschwisterelements zu selektieren:

```
$("div + p")
```

selektiert einen `<p>`-Container, wenn er einem `<div>`-Container in der gleichen hierarchischen Ebene unmittelbar folgt:

```
<div>
  <div></div>
  <p id="eins"></p>
  <span></span>
  <div id="start"></div>
  <span></span>
  <p id="zwei"></p>
  <p id="drei"></p>
  <span></span>
  <p id="vier"></p>
</div>
```

Beachten Sie, dass der Absatz `#zwei` nicht selektiert wird, da er kein direkter Nachbar eines `<div>`-Containers ist (hier liegt ein `Span` dazwischen).

Following Sibling Selektor E ~ F

Ab einem bestimmten Element sollen alle folgenden Geschwisterelemente eines bestimmten Typs selektiert werden:

```
$("#start ~ p")
```

Damit werden alle `<p>`-Elemente ausgewählt, die im selben Parent auf `#start` folgen:

```
<div>
  <div></div>
  <p id="eins"></p>
  <span></span>
  <div id="start"></div>
  <span></span>
  <p id="zwei"></p>
  <p id="drei"></p>
  <span></span>
</div>
```



```
<p id="vier"></p>
</div>
```

Das dazwischenliegende `<span>`-Element bleibt unselektiert. Es folgt zwar dem Element `#start`, besitzt aber nicht den geforderten Typ.

4.3.4 Filterausdrücke für Selektoren

jQuery stellt eine Reihe von Filtern bereit, mit denen Sie eine Selektion einschränken können. Typischerweise wird dabei das Muster `E:filter` verwendet, wobei `E` für einen beliebigen Elementtyp und `:filter` für den gewählten Filter steht. Der Elementname vor dem Doppelpunkt kann weggelassen werden (in diesem Fall steht implizit der Universalselektor). Beachten Sie, dass der Filter innerhalb des Selektorstrings steht.

Selektor	Typ	Erläuterung
<code>:first</code>	Filter	Selektiert das erste Element des übergebenen Typs.
<code>:last</code>	Filter	Selektiert das letzte Element des übergebenen Typs.
<code>:even</code>	Filter	Selektiert alle Elemente mit geradem Index innerhalb der aktuellen Collection (0 ist <code>even</code>). Achtung: Das erste Item ist daher <code>even</code> .
<code>:odd</code>	Filter	Selektiert alle Elemente mit ungeradem Index innerhalb der aktuellen Collection (1 ist <code>odd</code>). Achtung: Das zweite Item ist daher <code>odd</code> .
<code>:not(sel)</code>	Filter	Selektiert alle Elemente, auf die der Selektor <code>sel</code> nicht zutrifft.
<code>:eq(n)</code>	Filter	Selektiert alle Elemente, die innerhalb der aktuellen Collection den übergebenen Index <code>n</code> besitzen.
<code>:lt(n)</code>	Filter	Selektiert alle Elemente, die innerhalb der aktuellen Collection einen Index kleiner als den übergebenen Index <code>n</code> besitzen.
<code>:gt(n)</code>	Filter	Selektiert alle Elemente, die innerhalb der aktuellen Collection einen Index größer als den übergebenen Index <code>n</code> besitzen.
<code>:animated</code>	Filter	Selektiert alle aktuell animierten Elemente.
<code>:header</code>	Filter	Selektiert alle Elemente, die Überschriften <code>h1</code> , <code>h2</code> , <code>h3</code> etc. entsprechen.

Tabelle 4.8 Filterausdrücke für Selektoren

Die Filter :first und :last

Betrachten wir nun einige der Filter im Einzelnen. Beginnen wir mit `:first` und werfen dabei einen Blick auf den Ausdruck `$("div p:first")`. Folgendes Element wird ausgewählt:

```
<div>
  <p></p>
  <p></p>
  <p></p>
</div>
```

Wie nicht anders zu erwarten, speichert `$("div p:last")` das letzte ausgewählte Element:

```
<div>
  <p></p>
  <p></p>
  <p></p>
</div>
```

Der Filterausdruck :not()

Hingegen wählt `$("div p:not(:first)")` folgende Elemente:

```
<div>
  <p></p>
  <p></p>
  <p></p>
</div>
```

Innerhalb der Klammer von `:not()` kann jeder Selektorausdruck stehen und, wie in vorhergehendem Beispiel zu sehen, selbstverständlich auch weitere Filter. Es können daher mehrere Filter nach dem Muster `$("div p:even:not(:first)")` kombiniert werden. Dies gilt auf für alle anderen Filter.

Die Filter :odd und :even

Blicken wir nun auf weitere Basisfilter. Elemente mit geradem und ungeradem Index in der ungefilterten Collection können Sie mit `$("div p:even")` und `$("div p:odd")` herauslesen. Hier besteht die Basis-Collection aus allen `<p>`-Containern, die sich in einem Div befinden. Die Selektion für `:even` sieht wie folgt aus (selektierte Elemente sind fett hervorgehoben):

```
<div>
  <p></p>
  <p></p>
  <p></p>
```

```

    <p></p>
    <p></p>
    <p></p>
    <p></p>
  </div>

```

Die Zählung des Indexes beginnt bei 0, wobei die 0 als »gerade« gilt. Dementsprechend wird das erste Element mit dem Index 0 selektiert und anschließend das dritte, fünfte und siebte (!) Element, da diese die geradzahligen Indizes 2, 4, 6 besitzen.

Praktische Verwendung findet dieser Filter, um beispielsweise Tabellenzeilen in sich abwechselnden Farben zu realisieren (durch Zuweisung einer CSS-Klasse).

Indexfilterausdrücke :eq(), :lt() und :gt()

Wollen Sie ein Element mit einem bestimmten Index herausfiltern, benutzen Sie `$("div p:eq(1)")`. Hier wird das *zweite* `<p>`-Element ausgelesen. Es wird zwar eine Collection zurückgegeben, deren Länge wird jedoch stets 1 sein, da lediglich ein einziges Element gefunden wird.

Den Filter `$("div p:gt(1)")` können Sie mit »greater than«, übersetzen. Er findet alle Elemente, deren Index größer als ein gesetzter Vergleichsindex ist. Das Pendant `$("div p:lt(1)")`, »less than«, findet dementsprechend alle Elemente, deren Index kleiner als der Vergleichsindex ist.

Für `$("div p:gt(1)")` werden alle Absätze nach jenem mit Index 1 ausgewählt:

```

<div>
  <p></p>
  <p></p>
  <p></p>
  <p></p>
  <p></p>
</div>

```

Filter zur Extraktion von Überschriften

`$(":header")` gibt Überschriften zurück, die mit `<h1></h1>` bis `<h6></h6>` ausgezeichnet sind. Diesen Filter an ein beliebiges Element zu hängen liefert kein Ergebnis, da dieser Filter ja ausschließlich Überschriften selektiert. Er eignet sich aber hervorragend, um aus längeren Texten Inhaltsverzeichnisse zu generieren.

Filter für animierte Elemente

Der Filter `$("div p:animated")` zeigt, ob sich ein Element gerade in einer Animations-Queue befindet:

```
function run() {
    $("div p:eq(0)").slideToggle("slow",run);
}
run();
```

Die Funktion `run()` wird kontinuierlich ausgeführt, also kann mit

```
$("div p:eq(0)").click(function() {
    $("div p:animated").addClass("green");
});
```

genau auf dieses animierte Element zugegriffen werden:

```
<div>
    <p>Textabsatz 1 (animiert)</p>
    <p>Textabsatz 2 (nicht animiert)</p>
</div>
```

4.3.5 Inhaltsfilter

Über normale Möglichkeiten von CSS hinaus geht jQuery mit der Option, ein Element anhand seines Inhalts zu selektieren (oder nicht zu selektieren). Die betreffenden Zusatzfilter bezeichnet man als *Inhaltsfilter*.

Selektor	Typ	Erläuterung
<code>:contains(<i>text</i>)</code>	Inhalt	Selektiert alle Elemente, die den übergebenen Text <i>text</i> enthalten.
<code>:empty</code>	Inhalt	Selektiert alle Elemente ohne Inhalte (Textknoten einbezogen).
<code>:has(<i>sel</i>)</code>	Inhalt	Selektiert alle Elemente, die mindestens ein Element enthalten, auf die der übergebene Selektor <i>sel</i> passt.
<code>:parent</code>	Inhalt	Selektiert alle Elemente, die Elternelement eines anderen Knotens sind (also sowohl von Element- als auch Textknoten).

Tabelle 4.9 Inhaltsfilter in jQuery

Der Inhaltsfilter :contains

Der Filter `:contains()` findet alle Elemente, die einen bestimmten Textinhalt haben.

```
$("#div:contains('Galileo')")
```

findet beide `<div>`-Container aus folgendem HTML-Fragment:

```
<div>Galileo Verlag</div>
<div><span>Galileo Verlag</span></div>
```

Nicht selektiert würde folgender Container:

```
<div>galileo verlag</div>
```

Die Filteranweisung findet »Galileo«, aber nicht »galileo«. Der Ausdruck ist also *case-sensitive*.

Der Inhaltsfilter :empty

Der Filter `:empty` findet alle Elemente, die keinerlei Inhalt, also weder Kindelemente noch Textinhalt (Textknoten) besitzen.

```
$("#div:empty")
```

findet

```
<div></div>
```

Ist jedoch in einem leeren Element ein Zeilenumbruch vorhanden, wird dies als (allerdings leerer) Textknoten verstanden. Das Element gilt folglich nicht mehr als »ohne Inhalt«. Diese Interpretation entspricht der XML-Spezifikation. Ihr folgen die standardkonformen Browser (Firefox, Webkit, Opera).

Achtung – der Internet Explorer verwirft solche *Whitespace-Nodes*, selbst wenn sie neben dem Umbruchzeichen auch weitere Leerzeichen oder Tabulatorzeichen enthalten:

```
<!-- würde in IE über :empty selektiert: -->
<div>
    </div>
```

Anmerkung

Selbstverständlich macht es keinen Sinn, Collections aus Elementen, die per definitionem keine Kinder haben, mit `:empty` zu durchsuchen, beispielsweise `<img>` oder `<input>`.

Der Inhaltsfilter :has()

Der Filter `:has()` findet Elemente, deren Inhalt dem übergebenen Selektor entspricht. Ist dieser ein Typselektor, muss das gesuchte Element mindestens ein Element dieses Typs beinhalten.

```
$("div:has('p')")
```

findet

```
<div>
  <p>Absatz</p>
</div>
```

Auch innerhalb des Negationsfilters `:not()` kann `:has()` angewendet werden.

```
$("div:not(:has('h2'))")
```

findet alle `<div>`-Elemente, die *keine* H2-Überschrift beinhalten.

Der Inhaltsfilter :parent

Der Filter `:parent` findet all jene Elemente, die selbst Elternelemente sind, also Kindknoten besitzen. Hierbei kann es sich um weitere Elemente oder um Text (also Textknoten) handeln.

`$("div:parent")` selektiert die `<div>`-Container `#eins` und `#drei`:

```
<div id="eins">
  <div id="zwei"></div>
</div>
<div id="drei">Beispieltext</div>
```

4.3.6 Sichtbarkeitsfilter

Über jQuery können explizit Elemente aufgrund ihrer Sichtbarkeit oder Nichtsichtbarkeit ausgewählt werden. jQuery bezeichnet nicht sichtbare Elemente als *hidden*, also als »versteckt«. Wie üblich, existiert auch ein dazu inverser Filter, der entsprechend Elemente selektiert, die jQuery als »sichtbar« erachtet.

Selektor	Typ	Erläuterung
<code>:hidden</code>	Sichtbarkeit	a) Selektiert alle Elemente mit <code>display-Property none</code> oder Höhe und Breite 0; b) Selektiert <code><input type="hidden"></code> .

Tabelle 4.10 Sichtbarkeitsfilter in jQuery

Selektor	Typ	Erläuterung
<code>:visible</code>	Sichtbarkeit	Selektiert alle Elemente, deren <code>display</code> -Property nicht <code>none</code> ist oder die sowohl Höhe als auch Breite ungleich 0 besitzen.

Tabelle 4.10 Sichtbarkeitsfilter in jQuery (Forts.)

Sie sollten wissen, dass beide Filter nicht etwa die CSS-Eigenschaft `visibility` als Kriterium verwenden (in früheren Versionen von jQuery war dies jedoch der Fall). Vielmehr ist nun ausschlaggebend, ob ein Element *Raum im Layout* einnimmt. Dies kann aber auch bei nach herkömmlichen Kriterien als »versteckt« geltenden Elementen der Fall sein.

Sichtbarkeitsfilter `:hidden`

Entsprechend findet der Selektor `:hidden` diejenigen Elemente, deren `display`-Property auf `none` steht oder deren Abmessungen (Breite und Höhe) den Wert 0 besitzen. Ein durch `visibility:hidden` oder mit `opacity:0` verstecktes Element besitzt seine Ausdehnung hingegen noch und würde konsequenterweise nicht selektiert.

Diese CSS-Eigenschaft `display:none` verbirgt das Element samt seiner Kindelemente so, dass es keinerlei Platz in der Dokumentendarstellung einnimmt. Dies gilt von Haus aus auch für `<input type="hidden" />`, weshalb solche Formularfelder ebenfalls selektiert werden.

Problem mit ungewollten Selektionen

Beachten Sie, dass in einigen Browsern auch die (unsichtbaren) Elemente im `<head>` durch `:hidden` selektiert werden. Es ist also sinnvoll, die Wirkung des Selektors auf den Body-Bereich oder noch expliziter zu beschränken.

Im folgenden Quelltext wird durch `$("body :hidden")` der gesamte Inhalt des Body-Tags untersucht. Die ersten beiden `<div>`-Container und das `<form>`-Element werden nicht selektiert, der dritte, mit `display:none`, hingegen schon. Ebenfalls gefunden wird der `Input type="hidden"`:

```
<div style="visibility:hidden">Unsichtbar</div>
<div style="opacity:0">Durchsichtig</div>
<div style="display:none">Verborgenen</div>
<form>
  <input type="hidden" />
</form>
```

`$("body :hidden").length` gibt die Zahl 2 zurück.

Sichtbarkeitsfilter :visible

Der Filter `:visible` findet alle Elemente, die im HTML-Dokument eine Abmessung mit Länge und Breite größer als 0 besitzen. Dieser Filter ist eigentlich nur die direkte Umkehrung des `:hidden`-Filters. Betrachten wir jetzt ein weiteres Mal das gleiche Quelltextfragment wie zuletzt:

```
<div style="visibility:hidden">Unsichtbar</div>
<div style="opacity:0">Durchsichtig</div>
<div style="display:none">Verborgen</div>
<form>
  <input type="hidden" />
</form>
```

In Anlehnung an dieses Beispiel wird `$("body :visible").length` die Zahl 3 zurückgeben, da `<div style="visibility:hidden">...</div>` und `<div style="opacity:0">...</div>` eine Ausdehnung im Dokument besitzen. Dasselbe gilt für das `<form>`-Element, das deshalb ebenfalls selektiert wird. Nicht selektiert sind die beiden Elemente, die die `display`-Eigenschaft `none` besitzen (das dritte Div und der Hidden-Input).

4.3.7 Attributfilter

Die CSS-Spezifikation erlaubt es auch, Elemente anhand ihrer Attribute zu identifizieren, wobei CSS 3 einige neue Selektoren in diesem Segment einführt. In vielen Browsern ist dieser Teil der Spezifikation (noch) nicht vollständig umgesetzt, was jedoch von jQuery ausgeglichen wird.

Diese Filterausdrücke werden in eckigen Klammern geschrieben, stets nach dem Muster `Attributname=Wert`. Vor der eckigen Klammer kann (muss aber nicht) zusätzlich ein Elementname stehen.

Selektor	Typ	Erläuterung
<code>[name]</code>	Attribut	Selektiert alle Elemente beliebigen Typs, die über ein Attribut <code>name</code> verfügen.
<code>E[name]</code>	Attribut	Selektiert alle Elemente vom Typ <code>E</code> , die über ein Attribut <code>name</code> verfügen.
<code>[name=wert]</code> <code>[name2=wert2]</code>	Attribut- gruppe	Selektiert alle Elemente, auf die <i>alle</i> genannten Attributfilter zutreffen.

Tabelle 4.11 Attributfilter-Ausdrücke für jQuery

Selektor	Typ	Erläuterung
[name =wert]	Attribute	Speziell für lang-Attribute; selektiert ein Element, wenn das benannte Attribut den übergebenen Wert besitzt, oder den übergebenen Wert, dem ein beliebiger, durch Bindestrich abgetrennter String folgt.
[name*=wert]	Attribute	Selektiert ein Element, wenn der Wert des benannten Attributs den übergebenen String als Substring enthält.
[name~=wert]	Attribute	Selektiert ein Element, wenn der Wert des benannten Attributs vom Listentyp den übergebenen String als Teilwert enthält.
[name\$=wert]	Attribute	Selektiert ein Element, wenn der Wert des Attributs mit dem übergebenen String endet.
[name=wert]	Attribute	Selektiert ein Element, wenn der Wert des Attributs dem übergebenen String entspricht.
[name!=wert]	Attribute	Selektiert ein Element, wenn der Wert des benannten Attributs nicht dem übergebenen String entspricht oder das Attribut nicht existiert.
[name^=wert]	Attribute	Selektiert ein Element, wenn der Wert des Attributs mit dem übergebenen String beginnt.

Tabelle 4.11 Attributfilter-Ausdrücke für jQuery (Forts.)

Attribut-Filter [name]

Die Anweisung `$("#img[alt]")` selektiert

```

```

aber nicht

```

```

Attribut-Filter [name=wert]

Statt lediglich nach Vorhandensein eines Attributs können Sie auch gegen einen konkreten Wert prüfen. Beachten Sie dabei die in JavaScript gebräuchliche Schreibweise für verschachtelte Zeichenketten:

```
$("#img[src='header-image.png']")
```

Möglich ist auch

```
$('#img[src="header-image.png"]')
```

oder, unter Weglassung der Anführungszeichen um die Attributwerte

```
$("#img[src=header-image.png]")
```

Selbstverständlich können die inneren Anführungszeichen auch über den Backslash maskiert werden:

```
$("#img[src=\"header-image.png\"]")
```

Lesen Sie dazu auch in Abschnitt 4.2.4, »CSS-Selektoren als Argument«, den Absatz über Escape-Zeichen. Welche Form Sie dabei verwenden, ist allein eine Frage Ihres Geschmacks.

Multi-Attribut-Filter

Es ist auch möglich, mehrere Attributausdrücke durch Reihung zu verknüpfen. Hierbei darf nur vor dem ersten Ausdruck ein Typbezeichner stehen:

```
$("#img[src='header-image.png'][alt='Header']")
```

Attribut-Filter [name|=wert]

Dieser Filter selektiert Elemente, bei denen der übergebene String entweder dem Attributwert entspricht oder den Anfang des Attributwerts bildet, dem ein Bindestrich und ein weiterer beliebiger String folgen. Zu kryptisch ausgedrückt? Schauen wir uns also lieber ein Beispiel an – wir suchen hier nach dem Präfix `header`:

```
$("#img[src|=header]");
```

jQuery findet beispielsweise dieses `<img>`-Element:

```

```

Meist wird der Selektor jedoch für das `lang`-Attribut eingesetzt. Sollen beispielsweise Container beliebigen Typs selektiert werden, deren Inhalt als »englisch« gekennzeichnet ist, schreibt man `$("#[lang|=en]")`.

Dies selektiert folgende drei Elemente:

```
<div lang="en">English without further specification.</div>
<p lang="en-UK">This is British English, isn't it?</p>
<span lang="en-US">Supposed to be American English.</span>
```

Attribut-Filter [name*=wert]

Dieser Filter selektiert einen beliebigen Teilstring eines Attributwerts. So findet `$("#img[src*=head]")` Elemente, die im Attribut `src` den Wert `header`, `headache` oder `Egghead` haben:

```

```

Attribut-Filter [name~=wert]

Dieser Filter selektiert ein Element, wenn der übergebene String einen durch *Leerzeichen* abgetrennten Teilwert des betreffenden Attributs darstellt oder dem vollständigen Attributwert entspricht. So etwas kommt im Rahmen von `title`- oder `alt`-Attributen vor.

```
$("img[alt~=jQuery]")
```

findet

```

```

Anmerkung

Der Selektor richtet sich prinzipiell an Attribute vom Inhaltstyp »Liste«, also solche, die durch Leerzeichen getrennte Einzelwerte enthalten dürfen. Hierzu gehört das `class`-Attribut (für das aber eher der `class`-Selektor eingesetzt wird), aber auch die Attribute `rel`, `rev` (von `<a>` oder `<link>`), `accept-charset` (von `<form>`) oder `archive` (von `<object>`).

Attribut-Filter [name\$=wert]

Dieser Selektor erkennt Teilstrings am Ende eines Attributwerts und selektiert das dazugehörige Element.

```
$("input[value$=buch]")
```

findet

```
<input type="text" value="Das jQuery Praxisbuch" />
```

Attribut-Filter [name!=wert]

Ebenfalls testen lässt sich die Ungleichheit gegenüber einem Vergleichswert. Es werden dann alle Elemente ausgewählt, bei denen das betreffende Attribut einen anderen als den übergebenen Wert besitzt. Achtung – ebenfalls ausgewählt würden auch Elemente, bei denen das bezeichnete Attribut gar nicht existiert!

```
$("img[src!=header-image.png]");
```

selektiert

```


```

aber nicht

```

```

Attribut-Filter [name^=value]

Dieser Selektor erkennt Teilstrings am Anfang eines Attributwerts und selektiert das dazugehörige Element.

```
$("img[src^=head]")
```

findet

```

```

aber nicht

```

```

4.3.8 Child-Filter

Diese Filter wählen für eine Collection diejenigen Elemente aus, die sich als Kindknoten ihres Elternknotens durch eine bestimmte Position in der Reihenfolge auszeichnen.

Selektor	Typ	Erläuterung
:first-child	Kindknoten	Selektiert alle Knoten, die erster Kindknoten ihres Elternelements sind.
:last-child	Kindknoten	Selektiert alle Elemente, die letzter Kindknoten ihres Elternelements sind.
:only-child	Kindknoten	Selektiert alle Elemente, die einziger Kindknoten ihres Elternelements sind.
:nth-child(<i>n</i>)	Kindknoten	Selektiert alle Elemente, die als Kindknoten ihres Elternelements an der Stelle <i>n</i> stehen. Achtung: <i>n</i> darf ein Ausdruck sein!

Tabelle 4.12 Filterausdrücke für Kindknoten

Unterschiede zwischen :first und :first-child

Der Filter `:first-child` selektiert, im Gegensatz zu `:first`, *alle* Elemente des genannten Typs, die erstes Kindelement ihres Elternknotens sind. Hingegen selektiert `:first` nur und ausschließlich das erste gefundene Element des übergebenen Typs in der übergebenen Collection.

Die Selektoren :first-child, :last-child und :only-child

```
$("li:first-child")
```

selektiert bei dem zugrunde gelegten HTML-Code folgende Elemente (fett hervorgehoben):

```

<ul>
  <li>erster Listeneintrag</li>
  <li>zweiter Listeneintrag</li>
  <li>dritter Listeneintrag</li>
</ul>
<ul>
  <li>noch eine Liste</li>
  <li>mit weiteren</li>
  <li>Listeneinträgen</li>
</ul>

```

Analog gibt `:last-child` die letzten Elemente seiner Elternelemente zurück.

Der dritte im Bunde der einfachen Child-Selektoren ist `:only-child`:

```

<ul>
  <li>einziger Listeneintrag</li>
</ul>
<ul>
  <li>noch eine Liste</li>
  <li>mit weiteren</li>
  <li>Listeneinträgen</li>
</ul>

```

Ist ein Child-Element das einzige Kind, wird es selektiert.

Der Selektor `:nth-child()`

Der Selektor `:nth-child()` bietet Ihnen die vielfältigsten Möglichkeiten, nach Kindelementen zu suchen. Er kann die Selektoren `:first-child` und `:last-child` prinzipiell ersetzen, die damit im Grunde Spezialfälle von `:nth-child()` darstellen. Statt `:first-child` können Sie ebenso gut `:nth-child(1)` schreiben, mit gleichem Ergebnis.

Die runde Klammer von `:nth-child()` nimmt einen Vergleichsindex entgegen, der entweder aus einer Ganzzahl oder einem Ausdruck besteht. Im Rahmen dieses Ausdrucks steht die Variable *n* für einen Zählindex, der ab 1 läuft. (Der Ausdruck `:nth-child(n)` liefert entsprechend *alle* Kindknoten zurück, macht also für sich gesehen wenig Sinn.)

Zu *n* können jedoch *Faktoren* hinzukommen und *Offsets* addiert werden. So gibt Ihnen `:nth-child(2n)` jedes *zweite* Element (Faktor: 2) zurück, `:nth-child(3n+2)` wiederum selektiert jedes dritte Element (Faktor: 3) ab dem zweiten (Offset: +2).

Statt Ausdrücke und Zahlen können auch die Schlüsselworte `odd` und `even` übergeben werden. So selektieren `:nth-child(even)` und `:nth-child(odd)` alle geraden, respektive alle ungeraden Kindelemente. Achtung – das erste Kindelement hat den Index `n=1`. Mit `even` wählen Sie daher (also im Prinzip naheliegend) das zweite, vierte, sechste etc. Kindelement:

Der Ausdruck `$("li:nth-child(even)")` ergibt:

```
<ul>
  <li>erster Listeneintrag</li>
  <li>zweiter Listeneintrag</li>
  <li>dritter Listeneintrag</li>
  <li>vierter Listeneintrag</li>
  <li>fünfter Listeneintrag</li>
</ul>
<ul>
  <li>noch eine Liste</li>
  <li>mit weiteren</li>
  <li>Listeneinträgen</li>
</ul>
```

Unterschied zwischen `:nth-child(even)` und `:even`

Das Verhalten von `:nth-child(even)` steht im Gegensatz zu `:even`. Die Selektion `$("li:even")` sähe folgendermaßen aus, weil über die Elternelemente hinaus selektiert wird und in Dokumentreihenfolge ab Index 0 gezählt wird:

```
<ul>
  <li>erster Listeneintrag</li>
  <li>zweiter Listeneintrag</li>
  <li>dritter Listeneintrag</li>
  <li>vierter Listeneintrag</li>
  <li>fünfter Listeneintrag</li>
</ul>
<ul>
  <li>noch eine Liste</li>
  <li>mit weiteren</li>
  <li>Listeneinträgen</li>
</ul>
```

Als Erstes stellen wir fest, dass `:even` mit dem Zähler 0 beginnt, im Gegensatz zu `:nth-child(even)`. Letzterer beginnt mit 1. Während `:even` einfach durch alle gefundenen Elemente iteriert, bezieht sich `:nth-child(even)` nur auf eine jeweilige Kindelementgruppe. Der Unterschied zwischen `:nth-child(odd)` und `:odd` ist analog.

Unterschiede zur W3C-Spezifikation

Das W3C hat für die CSS-Spezifikation weitere Child-Selektoren wie beispielsweise `:nth-last-child()` definiert. Nach Meinung der Autoren von jQuery sind dieser und weitere Child-Selektoren wenig praxisrelevant. Daher werden sie von jQuery nicht unterstützt.

4.4 Accessoren – Eigenschaften der Collection

Von einer erstellten Collection aus kann eine weitere Verarbeitung stattfinden. Die erste Gruppe von Methoden und Eigenschaften, die hier betrachtet werden soll, sind die »Object Accessors«. Diese Methoden und Eigenschaften verändern nicht die Collection (d.h. erweitern oder verschieben sie nicht), sondern machen entweder etwas mit den Elementen der Collection oder geben Informationen über die Collection zurück.

Methode	Erläuterung
<code>.context</code>	Gibt das Kontextobjekt (als Elementknoten) zurück, wenn ein solches an <code>\$()</code> übergeben wurde; gibt ansonsten das <code>document</code> -Objekt zurück.
<code>.each(fn(i, elem))</code>	Führt die Funktion <code>fn</code> für jedes Element der Collection aus, wobei <code>fn</code> den Index <code>i</code> der Loop-Iteration übergeben bekommt und das aktuelle Element <code>elem</code> als Kontext verwendet.
<code>.get()</code> <code>.get(n)</code>	Erstellt ein Array aus den DOM-Elementen der aktuellen Collection. Alternativ: Gibt den Elementknoten des Items der Collection zurück, das sich an der Indexposition <code>n</code> befindet. Anmerkung: Für <code>n < 0</code> wird ab dem letzten Item rückwärts gezählt.
<code>.index(sel)</code> <code>.index(elem)</code> <code>.index()</code>	Gibt den Index (Basis 0) des ersten Elements in der aktuellen Collection zurück, das durch den Selektor <code>sel</code> oder ein DOM-Element <code>elem</code> bezeichnet wird. Gibt <code>-1</code> zurück, wenn kein entsprechendes Element gefunden wird. Alternativ: Ohne Parameter gibt <code>.index()</code> die Position des ersten Items der Collection unter den Kindknoten seines Parents zurück.
<code>.is(sel)</code>	Gibt <code>true</code> oder <code>false</code> zurück, je nachdem, ob mindestens ein Element der aktuellen Collection dem Selektor <code>sel</code> entspricht.

Tabelle 4.13 jQuery-Methoden: Object Accessors

Methode	Erläuterung
<code>.length</code>	Enthält eine Ganzzahl (die Anzahl der Items in der aktuellen Collection).
<code>.selector</code>	Gibt den der aktuellen Collection zugrunde liegenden Selektor zurück.
<code>.size()</code>	Gibt die Anzahl der DOM-Elemente in der aktuellen Collection zurück.
<code>.toArray()</code>	Gibt die Elemente der Collection als Array aus DOM-Knoten zurück.

Tabelle 4.13 jQuery-Methoden: Object Accessors (Forts.)

Betrachten wir nun die wichtigsten Methoden und Eigenschaften der Object Accessors im Detail.

Methode `.each(callback)`

Diese Methode führt eine Funktion `callback` im Kontext jedes einzelnen gefundenen Elements eines jQuery-Objekts aus. Dabei iteriert `.each(callback)` über alle Elemente der Collection. Das bedeutet, dass wenn mit einem Selector `div p` eine Anzahl von Absätzen gefunden wurde, für jeden Absatz eine Funktion `callback` ausgeführt wird:

```
$(document).ready(function () {
    $("#box p").each(

        // Funktion wird auf alle Collection-Items angewandt:
        function(i) {
            var inc = i + 1;
            // Funktion bildet eigene Collection:
            $(this).text(" Dies ist der " + inc + ". Absatz");
        }

    );
});
```

Wenn die Funktion ausgeführt wird, wird ihr automatisch ein Argument mit auf den Weg gegeben, das die aktuelle Position innerhalb der jQuery-Collection zurückgibt. Das Argument repräsentiert den Index des aktuellen Elements. Da hier mit diesem Index gearbeitet werden soll, wird der Funktion ein Eingabeparameter `i` zur Verfügung gestellt. (Ansonsten kann der Übergabeparameter weggelassen werden, falls er nicht gebraucht wird. Dass an Callback-Funktionen Argu-

mente weitergereicht werden, ist ein häufiger Fall bei jQuery. Es steht Ihnen stets frei, diese Argumente aufzufangen und zu nutzen oder nicht.)

Normalerweise verarbeitet `.each()` automatisch in einer Schleife die gesamte Collection. Der Vorgang kann abgebrochen werden, indem Sie der Callback-Funktion `false` zurückgeben lassen, was Sie sinnvollerweise an eine passende Bedingung binden sollten. Gibt die Callback-Funktion `true` (oder keinen Wert) zurück, läuft die Schleife bis zum Ende.

Kein `.each()` für die Anwendung von jQuery-Methoden nötig

Wenden Sie eine jQuery-Methode auf eine Collection an, geschieht dies stets in einer impliziten Schleife. Wollen Sie in alle Absätze des vorigen Beispiels denselben Text schreiben, ginge dies auch einfacher:

```
$("#box p").text(" Dies ist ein Absatz");
```

Der Clou an `.each()` ist jedoch, dass es keine jQuery-Methode ist, die auf die Items der Collection wirkt, sondern ein anonymes Funktionsobjekt, das beliebigen Inhalt besitzen kann (und im vorigen Beispiel mit `$(this)` jeweils seine eigene Collection aus dem aktuell verarbeiteten Element bildet).

Methode `.get()` und `.get(index)`

Die `.get()`-Methode ohne Parametereingabe gibt die gesamten Items der Collection als Array aus DOM-Elementen zurück:

```
var images = $("#box img").get();
alert(images.src);
```

Beachten Sie, dass die Methode selbst kein jQuery-Objekt erzeugt! An einen `.get()`-Aufruf können daher keine weiteren jQuery-Methoden angereicht werden. Die `.get()`-Methode gehört damit zu den terminierenden Methoden. Allerdings stehen für das DOM-Array die herkömmlichen DOM-Methoden und Eigenschaften zur Verfügung (was ja auch genau erwünscht ist).

Rufen Sie die Methode mit einem Parameter, genauer einer Ganzzahl `index` größer gleich 0, auf, gibt sie ein einziges, nämlich das mit `index` spezifizierte DOM-Element zurück. Diesem stehen keine jQuery-Methoden zur Verfügung. Versuchen Sie daher, mit einer jQuery-Methode auf ein durch `.get(0)` zurückgegebenes Element zuzugreifen,

```
// schlägt fehl:
alert($("#box p").get(0).attr("id"));
```

wird (in der Fehlerkonsole des Firefox) folgende Fehlermeldung erzeugen:

```
Fehler: $("#box p").get(0).attr is not a function
```

Die jQuery-Methode wird als ungültig in diesem Kontext erkannt. Dagegen gibt

```
alert($("#box p").get(0).id);
```

das Ergebnis ohne Fehlermeldung aus: Die hier aufgerufene Eigenschaft `id` ist eine DOM-Eigenschaft, auf die nun zugegriffen werden kann.

Alternative zu `.get()` – eckige Array-Klammern

Ebenfalls ein DOM-Element erhalten Sie aus einer Collection, indem Sie sie wie ein Array behandeln. Anstatt `.get(0)` können Sie einfach schreiben `[0]`:

```
alert($("#box p")[0].id);
```

Dieser Aufruf ist wirkungsgleich mit dem in der davor stehenden Zeile dargestellten Aufruf.

Methode `.index(subject)`

`.index(subject)` gibt die Position eines einzelnen Elements innerhalb eines jQuery-Objekts an. Als Argument `subject` wird das Objekt angegeben, nach dem gesucht werden soll. Hier wird im Rahmen einer `.each()`-Schleife das jeweils aktuelle Objekt über `this` an `.index()` übergeben:

```
var obj = $("#box p");
obj.each( function() {
    $(this).text("Index: " + (obj.index(this)));
});
```

Methode `.size()`

Die Methode `size()` gibt die Anzahl an Elementen zurück, die sich in einer jQuery-Collection befinden. Diese Methode entspricht im Prinzip der Eigenschaft `.length` des jQuery-Objekts, die den gleichen Wert enthält. Da sie eine Zahl und kein jQuery-Objekt zurückgibt, zählt sie zu den terminierenden (nicht verkettbaren) Methoden:

```
$(document).ready(function () {
    // obj ist ein jQuery-Objekt:
    var obj = $("#box p")
    obj.each(function() {
        $(this).text("obj hat " + obj.size() + " Elemente");
    });
});
```

Tipp: Besser .length als .size() einsetzen

Zwar ist die Methode `.size()` derzeit noch gültig, die Autoren von jQuery raten aber dazu, stattdessen die Eigenschaft `.length` zu verwenden, da der Zugriff auf die Länge der Collection über sie performanter ist. Es gibt bereits Überlegungen, die Methode `.size()` irgendwann ganz abzuschaffen.

Eigenschaft .length

Die Eigenschaft ermöglicht Ihnen den direkten Zugriff auf die Anzahl der Elemente, die sich in einem jQuery-Objekt befinden:

```
$(document).ready(function () {
    var obj = $("#box p")
    obj.each(function() {
        $(this).text("obj hat " + obj.length + " Elemente");
    });
});
```

Eigenschaften .selector und .context

Die Eigenschaft `.selector` gibt einen String zurück, der den Selektor enthält, mit dem ein Element gefunden wurde, `.context` gibt den Kontext zurück, in dem sich das aktuelle von jQuery selektierte Element befindet:

```
$(document).ready(function () {
    var obj = $("#box p")
    obj.each(function() {
        $(this).text("Selektor: " + obj.selector);
        $(this).append("Context: " + $(this).context.nodeName);
    });
});
```

Diese Eigenschaften sind hauptsächlich für Plugin-Entwickler interessant, um die Position von Elementen zu bestimmen. Mit anderen Worten: Sie können hiermit abfragen, in welchem Kontext eine Funktion gerade ausgeführt wird.

Methode .toArray()

Die Methode `.toArray()` gibt ein Array aus DOM-Knoten aus den in einem jQuery-Objekt enthaltenen Elementen zurück. Die Methode gehört zu den terminierenden Methoden, da sie kein jQuery-Objekt zurückgibt. Ein Beispiel:

```
// Zeigt, das aus der Collection ein DOM-Array wird:
alert($('p').toArray());
```

Die einzelnen Objekte werden als Array zurückgegeben.

Auf das DOM-Array können zwar keine jQuery-Methoden angewendet werden, jedoch stehen alle Methoden für DOM-Arrays zur Verfügung (bzw. die meisten der Array-Methoden von JavaScript). So könnte die Reihenfolge der Knoten im Array umgekehrt werden:

```
// Achtung: reverse() ist eine DOM.Methode:
alert($('p').toArray().reverse());
```

Eine jQuery-Methode würde an dieser Stelle einen Fehler verursachen, eine DOM-Methode ist problemlos anwendbar.

Anmerkung

`.toArray()` ist im Grunde genommen identisch mit `.get()`, mit dem Unterschied, dass kein Argument übergeben werden kann.

Methode `.is()`

Die Methode `.is()` gibt einen Booleschen Wert zurück, und zwar `true`, wenn der übergebene Selektor auf den aktuellen Knoten der Collection zutrifft, ansonsten `false`. Da sie kein jQuery-Objekt zurückgibt, zählt sie zu den terminierenden Methoden (nicht verkettbar). Man verwendet sie häufig zum Formulieren von Bedingungen:

```
$('#div').each( function() {
    if($(this).is('#box')) {
        $(this).text('Dieses Div hat den ID "box".');
    } else {
        $(this).text('Dies ist ein Div wie jedes andere.');
```

4.5 Traversieren – ausgehend von Collections

Eine nächste Gruppe von Methoden verändert die aktuelle Collection, indem sie entweder weitere Elemente hinzufügt oder, ausgehend von den aktuellen Elementen, einen Schritt auf einer der DOM-Achsen vornimmt. Dies wird als *Traversieren* bezeichnet. Der Schritt kann hierbei in der gleichen hierarchischen Ebene (Sibling-Achse) hin zu den Geschwisterelementen, aber auch zu den Kindknoten (Child- und Descendant-Achse) oder in Richtung des Elternknotens (Parent- und Ancestor-Achse) erfolgen.

4.5.1 Was versteht man unter »Traversieren«?

Eine Bewegung im DOM-Baum, das *Traversieren*, ist dann erforderlich, wenn der zu selektierende Knoten nicht auf direktem Weg erreicht werden kann. Manchmal ist es lediglich möglich, den Eltern- oder einen der Kindknoten des eigentlichen Zielknotens zu fassen zu bekommen. In diesem Fall wird von diesem aus per entsprechende DOM-Eigenschaft eine Traverse vorgenommen.

Dies kann in verschiedenen Richtungen geschehen, die im DOM als *Achsen* bezeichnet werden. Die eindeutigste hiervon ist die *Parent-Achse*:

```
// adressiert den Elternknoten des Knotens #meinID:
var elem = document.getElementById("meinID").parentNode;
```

Die Eigenschaft `parentNode` besitzt jeder Elementknoten. Sie enthält eine Referenz auf den Elternknoten (als DOM-Knoten). Da jeder Elementknoten über genau einen Elternknoten verfügt, ist dies eindeutig.

Gehört der gewünschte Knoten zum Inhalt des erreichbaren Knotens, finden Sie ihn auf der *Child-Achse*. Allerdings kann die Ergebnismenge aus beliebig vielen Knoten bestehen (u.a. sowohl Text- als auch Elementknoten), je nach Struktur des Inhalts. Ein Schritt auf der Child-Achse gibt daher eine arrayähnliche »Collection« (gemeint ist *nicht* eine jQuery-Collection) zurück:

```
// adressiert die Kindknoten des Knotens #meinID:
var elems = document.getElementById("meinID").childNodes;
```

Um entweder den ersten oder den letzten Kindknoten eines Bezugselements zu erreichen, dienen die Eigenschaften `firstChild` und `lastChild`, die bequeme Aliase von `childNodes[0]` und `childNodes[childNodes.length]` darstellen.

Beim horizontalen Traversieren bewegen Sie sich innerhalb eines Elementinhalts auf der Ebene der Geschwisterknoten (*Sibling-Achse*). Dies ist in zwei Richtungen möglich – vorhergehende Geschwisterknoten (`preceding siblings`) und folgende Geschwisterknoten (`following siblings`). Hierzu stehen Ihnen die Eigenschaften `previousSibling` und `nextSibling` zur Verfügung.

4.5.2 jQuery-Methoden zur Achsen-Traversal

jQuery bietet, aufbauend auf den zuvor nur kurz angerissenen DOM-Properties, eigene Methoden, die wieder jQuery-Objekte zurückgeben. Eine erste Gruppe von Methoden kann als Entsprechung der DOM-Eigenschaften gesehen werden. Sie dient dazu, Kindknoten, Elternknoten oder Geschwisterknoten ausgehend von der aktuellen Collection zu selektieren und diese dann für die weitere Verarbeitung zur neuen Collection zu machen.

Methode	Erläuterung
<code>.children()</code> <code>.children(sel)</code>	Holt die Kindelemente jedes Elements der Collection. (Nimmt optional einen Selektor <code>sel</code> als Filterkriterium entgegen.)
<code>.contents()</code>	Erstellt eine Collection aus allen Kindknoten (inklusive Textknoten) der Elemente der aktuellen Collection.
<code>.parent()</code> <code>.parent(sel)</code>	Erzeugt eine neue Collection aus den Elternknoten jedes Elements der aktuellen Collection. Optional: Der Selektor <code>sel</code> reduziert die neue Collection auf diejenigen Elternknoten, auf die der Selektor passt.
<code>.next()</code> <code>.next(sel)</code>	Erzeugt ein neues jQuery-Objekt aus den unmittelbar folgenden Geschwisterknoten der Elemente der aktuellen Collection. Optional: Ein Folgeknoten wird nur dann der Collection hinzugefügt, wenn er dem Selektorausdruck <code>sel</code> entspricht.
<code>.prev()</code> <code>.prev(sel)</code>	Erzeugt ein neues jQuery-Objekt aus den unmittelbar vorangehenden Geschwisterknoten der Elemente der aktuellen Collection. Optional: Ein Vorgängerknoten wird nur dann der Collection hinzugefügt, wenn er dem Selektorausdruck <code>sel</code> entspricht.

Tabelle 4.14 jQuery-Traversierungsmethoden (DOM-Achsen)

Ausgehend von folgendem HTML-Fragment

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div id="d1">
    <p id="p3">Dies ist ein Div-Container
      in div#box1, der einen Textabsatz enthält.
    </p>
  </div>
</div>
```

sollen jetzt einige dieser Methoden demonstriert werden.

Methode `.children()`

Zunächst wird aus dem `<div>`-Container `#box1` die primäre Collection gebildet und ein grüner Rahmen und ein Padding hinzugefügt. Danach wird eine Collection der Kindelemente von `#box1` zusammengestellt, die ihrerseits einen roten Rahmen erhalten:

```
$(document).ready(function() {
  $('#box1')
    .css({border:'1px solid green',padding:'15px'})
```

```
.children().css({border:'1px solid red'})
});
```

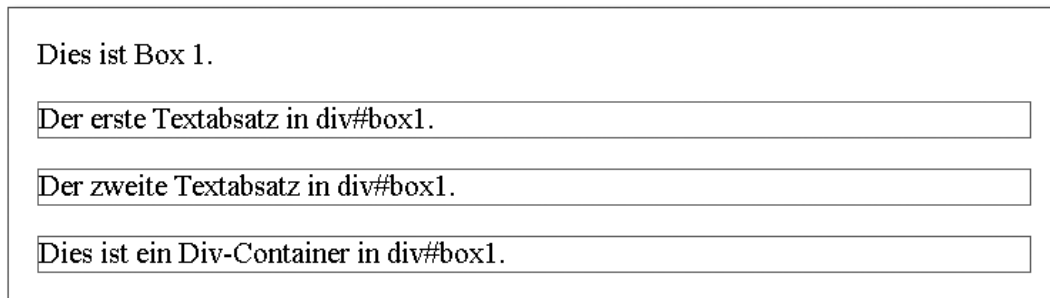


Abbildung 4.2 06_children.html

Beachten Sie, dass der unmittelbare Textinhalt von `#box1` keinen Rahmen erhält. *Textknoten* werden von der `.children()`-Methode nicht erfasst. Der `<p>`-Container und der innere Div werden umrahmt. Sollen nur die `<p>`-Tags einen Rahmen bekommen, können Sie noch zusätzlich einen Selektor als Filterkriterium an `.children()` übergeben:

```
$(document).ready(function() {
    $('#box1')
        .css({border:'1px solid green',padding:'15px'})
        .children('p').css({border:'1px solid red'})
});
```

Hierbei wird der im inneren Div liegende `<p>`-Container nicht erfasst, da er nicht zu den Kindknoten von `div#box1` gehört.

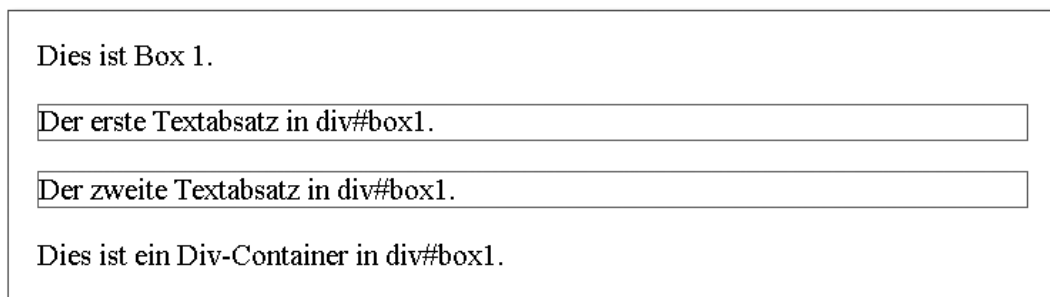


Abbildung 4.3 06_children2.html

Methode `.contents()`

Die Methode `.contents()` selektiert, im Gegensatz zu `.children()`, auch Textknoten. Allerdings beinhaltet dies auch leere Textknoten, wie sie typischerweise zwischen Elementknoten auftreten. Auf den Inhalt von `div#box1` angewendet, bildet `.contents()` eine Collection aus sieben Items, von denen drei Element- und vier Textknoten sind. Der erste Textknoten ist der String »Dies ist Box 1.«:

```
$(document).ready(function() {
    alert($('#box1').contents().length) //-> 7
});
```

Methode .parent()

Mit der `.parent()`-Methode wird eine Collection der Parentknoten der aktuellen Collection zusammengestellt. Hier ist dies der `div#box1`, ausgehend vom Container `p#p1`, der in dessen Inneren liegt:

```
$(document).ready(function() {
    $('#p1')
        .css({border:'1px solid red'})
        .parent()
        .css({border:'1px solid green',padding:'15px'})
});
```

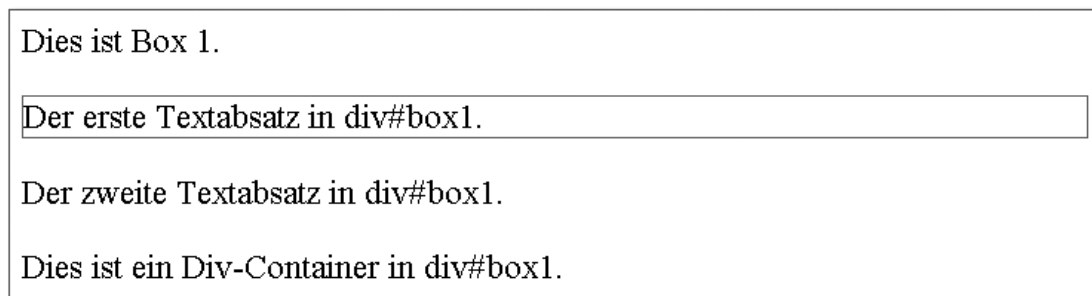


Abbildung 4.4 06_parent.html

Möchten Sie einen Parentknoten nur dann selektieren, wenn er bestimmte Eigenschaften besitzt, können Sie dies durch einen Selektor ausdrücken, den Sie der Methode übergeben.

Methode .next()

Die Methode `.next()` wählt für alle Elemente der primären Collection den jeweilig folgenden Elementknoten in Quelltextreihenfolge aus und bildet aus ihm die neue Collection:

```
$(document).ready(function() {
    $('#p1')
        .css({border:'1px solid red'})
        .next()
        .css({border:'1px solid green'})
});
```

Hier erhält der auf `p#p1` folgende Tag `p#p2` einen grünen Rahmen.

Methode prev()

Analog wählt `.prev()` den vorangehenden Knoten aus:

```
$(document).ready(function() {
    $('#p2')
        .css({border:'1px solid red'})
        .prev()
        .css({border:'1px solid green'})
});
```

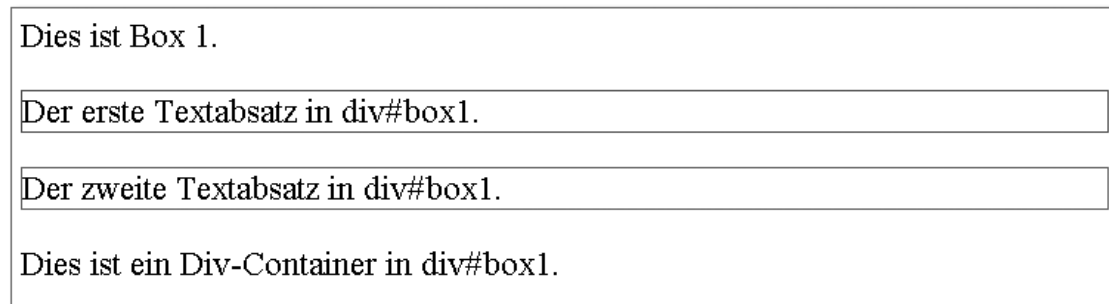


Abbildung 4.5 06_prev.html

Die Auswahl reicht immer genau einen Knoten weit. Um alle Geschwisterelemente auf der Preceding- oder Following-Achse auszuwählen, verwenden Sie `.nextAll()` oder `.prevAll()`. Diese Methoden werden gleich noch besprochen.

4.5.3 jQuery-Methoden zur erweiterten Achsen-Traverse

Neben den Pendanten zu den gewöhnlichen DOM-Traversierungen stellt jQuery weitere zur Verfügung, die ausgehend von der aktuellen Collection weitere Selektionen in Richtung verschiedener Achsen vornehmen. Im Gegensatz zu den eben vorgestellten Methoden ist die Reichweite des Auswahlvorgangs auf der gewählten Achse jedoch unbegrenzt.

Methode	Erläuterung
<code>.closest(sel)</code> <code>.closest(sel, cont)</code>	Selektiert das erste auf den Selektor <code>sel</code> passende Element auf der Ancestor-Achse. (Nimmt als optionalen zweiten Parameter den Kontext <code>cont</code> entgegen.)
<code>.find(sel)</code>	Bildet auf der Decendant-Achse der Elemente der aktuellen Collection eine neue Collection aus Elementen, die dem übergebenen Selektor <code>sel</code> entsprechen.

Tabelle 4.15 Erweiterte jQuery-Traversierungsmethoden (Ancestors und Descendants)

Methode	Erläuterung
<code>.parents()</code> <code>.parents(sel)</code>	Erzeugt eine neue Collection aus allen Elementen auf der Ancestor-Achse jedes Elements der aktuellen Collection. Optional: Der Selektor <code>sel</code> reduziert die neue Collection auf diejenigen Ancestor-Knoten, auf die der Selektor passt.
<code>.parentsUntil(sel)</code> <code>.parentsUntil()</code>	Erzeugt ein neues jQuery-Objekt aus den Elementen der Ancestor-Achse der Elemente der aktuellen Collection bis unmittelbar vor (d.h. nicht inklusive) der Ebene des ersten Elements, auf das der Selektor <code>sel</code> passt. Optional: Wird kein Selektor übergeben (oder tritt kein Match auf), verhält sich die Methode analog zu <code>.parents()</code> .

Tabelle 4.15 Erweiterte jQuery-Traversierungsmethoden (Ancestors und Descendants) (Forts.)

Wir gehen erneut vom bereits zuvor verwendeten HTML-Fragment aus:

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div id="d1">
    <p id="p3">Dies ist ein Div-Container
      in div#box1, der einen Textabsatz enthält.
    </p>
  </div>
</div>
```

Methode `.find()`

Die Methode `.find()` selektiert auf der Descendant-Achse ausgehend von der aktuellen Collection, trifft also eine Auswahl unter allen Abkömmlingen in beliebiger Hierarchietiefe. Sinnvollerweise erhält sie ein Selektorargument, das die Auswahl auf bestimmte Knotengruppen beschränkt. Hier werden ausgehend von `div#box1` alle `<p>`-Container ausgewählt:

```
$(document).ready(function() {
  $('#box1')
    .css({border:'1px solid red'})
    .find('p')
    .css({border:'1px solid green'})
});
```

Gefunden werden die beiden unmittelbar in `div#box1` liegenden `<p>`-Container, aber auch derjenige, der sich *im Inneren* des auf diesen folgenden Divs befindet.

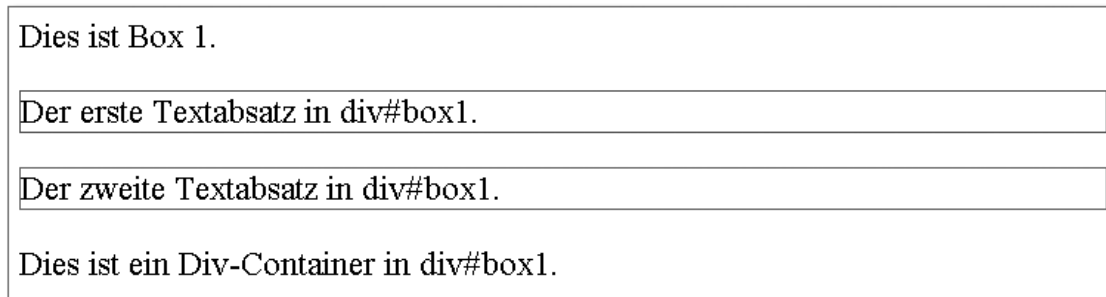


Abbildung 4.6 06_find.html

Methode .parents()

Ähnlich wie `.find()` auf der Descendant-Achse, arbeitet auch die Methode `.parents()` auf der Ancestor-Achse. Hier sollen, ausgehend vom innersten Textabsatz `p#p3`, diese ihn umgebenden `<div>`-Container gefunden werden. Es sind genau zwei, nämlich `div#d1` und `div#box1`:

```
$(document).ready(function() {
    $('#p3')
        .css({border:'1px solid red'})
        .parents('div')
        .css({border:'1px solid green'})
});
```

Ließen Sie den Selektor weg, würden auch der Body- und der HTML-Container ausgewählt (im Falle des Letzteren bräuchte das allerdings nicht viel).

Methode .parentsUntil()

Wollen Sie auf der Ancestor-Achse zwar zunächst alle Elemente aufsammeln, dann aber an einem bestimmten Punkt haltmachen, bietet sich dafür die Methode `.parentsUntil()` an:

```
$(document).ready(function() {
    $('#p3')
        .css({border:'1px solid red'})
        .parentsUntil('#box1')
        .css({border:'1px solid green'})
});
```

In diesem Fall wird nur der direkt um `p#p3` liegende `<div>`-Container erfasst (in diesem Fall also der direkte Parent). Der Container `div#box1` ist *nicht* mehr Teil der Collection, da er dem Filterkriterium entspricht.

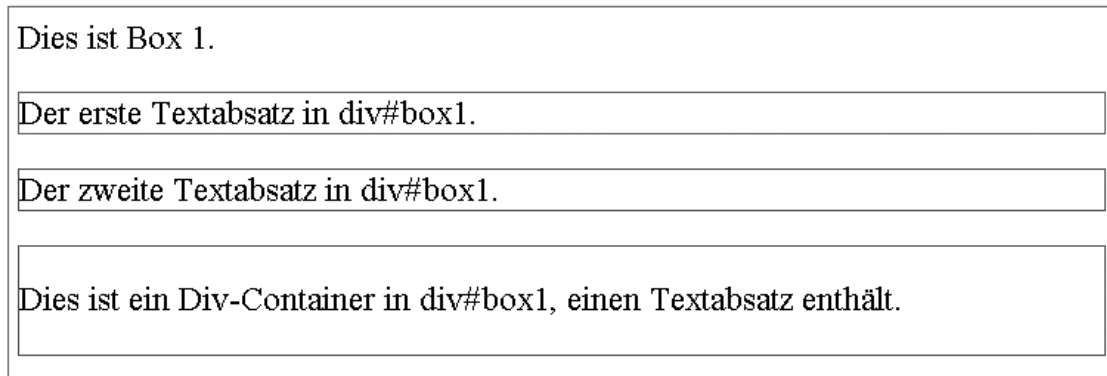


Abbildung 4.7 06_parentsUntil.html

Methode .closest()

Wollen Sie auf der Ancestor-Achse suchen, aber nur den Knoten erfassen, der hierarchisch am nächsten liegt (dies kann, muss aber nicht, auch der eigentliche Parent sein), setzen Sie die Methode `.closest()` ein. Hier findet der Befehl, ausgehend von `p#p3`, den diesen umgebenden `<div>`-Container (in diesem Fall wieder identisch mit dem Parent):

```
$(document).ready(function() {
    $('#p3')
        .css({border:'1px solid red'})
        .closest('div')
        .css({border:'1px solid green'})
});
```

Methode	Erläuterung
<code>.nextAll()</code> <code>.nextAll(sel)</code>	Erzeugt ein neues jQuery-Objekt aus allen folgenden Geschwisterknoten der Elemente der aktuellen Collection. Optional: Von den Folgeknoten werden nur diejenigen der Collection hinzugefügt, die dem Selektorausdruck <code>sel</code> entsprechen.
<code>.nextUntil(sel)</code> <code>.nextUntil()</code>	Erzeugt ein neues jQuery-Objekt aus allen folgenden Geschwisterknoten der Elemente der aktuellen Collection jeweils bis unmittelbar <i>vor</i> das erste Element (d.h. <i>nicht inklusive</i>), auf das der Selektor <code>sel</code> passt. Optional: Wird kein Selektor übergeben (oder tritt kein Match auf), verhält sich die Methode analog zu <code>.nextAll()</code> .

Tabelle 4.16 Erweiterte jQuery-Traversierungsmethoden (Preceding und Following)

Methode	Erläuterung
<code>.prevAll()</code> <code>.prevAll(sel)</code>	Erzeugt ein neues jQuery-Objekt aus allen vorangehenden Geschwisterknoten der Elemente der aktuellen Collection. Optional: Von den Vorgängerknoten werden nur diejenigen der Collection hinzugefügt, die dem Selektorausdruck <code>sel</code> entsprechen.
<code>.prevUntil(sel)</code> <code>.prevUntil()</code>	Erzeugt ein neues jQuery-Objekt aus allen vorangehenden Geschwisterknoten der Elemente der aktuellen Collection jeweils bis unmittelbar <i>vor</i> das erste Element (d.h. <i>nicht inklusive</i>), auf das der Selektor <code>sel</code> passt. Optional: Wird kein Selektor übergeben (oder tritt kein Match auf), verhält sich die Methode analog zu <code>.prevAll()</code> .
<code>.siblings()</code> <code>.siblings(sel)</code>	Bildet eine Collection aus <i>allen</i> Geschwisterknoten der Elemente der aktuellen Collection. Optional: Der Selektor <code>sel</code> reduziert die Collection auf die Geschwisterelemente, die diesem Selektor entsprechen.

Tabelle 4.16 Erweiterte jQuery-Traversierungsmethoden (Preceding und Following) (Forts.)

Methode .nextAll()

Die `.nextAll()`-Methode selektiert alle Folgeelemente ausgehend von den Elementen der aktuellen Collection. Hier werden der auf `p#p1` folgende Textabsatz und auch der diesem folgende `<div>`-Container gewählt und eingerahmt:

```
$(document).ready(function() {
    $('#p1')
        .css({border:'1px solid red'})
        .nextAll()
        .css({border:'1px solid green'})
});
```

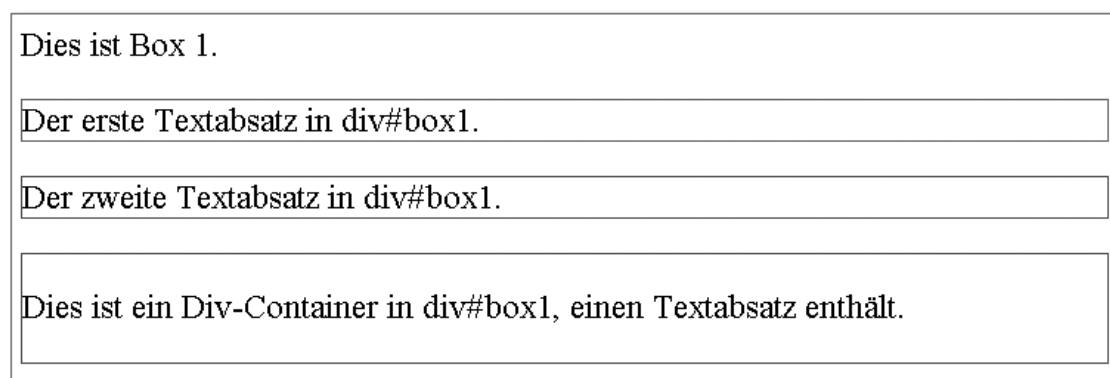


Abbildung 4.8 06_nextAll.html

Bei Bedarf können Sie die Auswahl wieder durch Übergabe eines Selektors beschränken. Auf gleiche Weise arbeitet `.prevAll()`.

Methode `.nextUntil()`

Die Methode `.nextUntil()` versteht sich wie `.nextAll()` mit »eingebautem Haltesignal«: Es kann ein Selektor übergeben werden, der den Endpunkt der Selektion beschreibt. So können Sie bewirken, dass nur diejenigen Geschwisterknoten bis zum Auftreten eines bestimmten Knotens ausgewählt werden sollen:

```
$(document).ready(function() {
    $('#p1')
        .css({border:'1px solid red'})
        .nextUntil('div')
        .css({border:'1px solid green'})
});
```

Hier wird noch der auf `p#p1` folgende Textabsatz ausgewählt, der diesem folgende `<div>`-Container (und auch eventuell auf diesen folgende Geschwisterknoten) jedoch nicht mehr, da er dem Filterkriterium entspricht. Die Methode `.prevUntil()` funktioniert analog, nur in die andere Richtung.

Methode `.siblings()`

Wollen Sie einfach nur die Geschwisterknoten eines Elements auswählen, um sich um dessen Position in der Hierarchieebene keine Gedanken machen zu müssen, ist `.siblings()` die Methode der Wahl. Sie selektiert ausgehend vom Ursprungselement in beide Richtungen, stellt also eine Synthese aus `.nextAll()` und `.prevAll()` dar:

```
$(document).ready(function() {
    $('#p2')
        .css({border:'1px solid red'})
        .siblings()
        .css({border:'1px solid green'})
});
```

Hier wurden, ausgehend vom zweiten Textabsatz, dessen Vorgängerabsatz `p#p1` und der folgende `<div>`-Container `div#d1` ausgewählt und gerahmt.

Collection durch Traversierung erweitern

Die bisher vorgestellten Traversierungen ersetzen die bis dahin aktuelle Collection durch eine neue. Mithilfe zweier weiterer Methoden, `.add()` und `.andSelf()`, kann jQuery die Elemente der Ausgangs-Collection beibehalten und mittels Traversierungsmethoden durch weitere Items erweitern.

Methode	Erläuterung
<code>.add(sel)</code>	Fügt der aktuellen Collection diejenigen weiteren Elemente hinzu, auf die der Selektor <code>sel</code> passt.
<code>.add(sel, context)</code>	Fügt der aktuellen Collection diejenigen weiteren Elemente hinzu, auf die der Selektor <code>sel</code> passt. Nimmt als optionalen zweiten Parameter den Kontext <code>context</code> entgegen.
<code>.add(elems)</code>	Fügt übergebene DOM-Elemente <code>elems</code> der aktuellen Collection hinzu.
<code>.add(HTML-String)</code>	Bildet ein Element aus einem übergebenen HTML-String und fügt dieses der aktuellen Collection hinzu.
<code>.andSelf()</code>	Fügt die ursprüngliche Collection der aktuellen Collection hinzu, wenn eine Traversierungsmethode aufgerufen wurde.

Tabelle 4.17 Methoden zur Erweiterung der Collection

Methode `.add()`

Die Methode `.add()` existiert in verschiedenen Signaturen, die jeweils andere Argumenttypen erwarten. Alle folgenden Beispiele basieren auf diesem einfachen HTML-Code:

```
<div id="box1">
  <p id="p1">Klick für Add-Test</p>
  <p id="p2">Beide Absätze färben sich rot.</p>
</div>
```

Mit `.add()` kann eine bestehende Collection erweitert werden. Hierbei werden ihr die mit dem Selektor in `.add()` festgelegten Elemente hinzugefügt. Etwaige angehängte jQuery-Methoden werden dann auf die vergrößerte Collection angewendet:

```
$(document).ready(function() {
  $('#p1').click( function() {
    alert(this.id);
    $('#p1').add("#p2")
               .css({color:'red'})
  });
});
```

Der Selektor von `.add()` arbeitet kontextabhängig. Liegt der Kontext auf dem Dokument selbst, arbeiten die Selektoren wie erwartet. Liegt der Kontext wie im folgenden Beispiel auf einem anderen Knoten (hier ist es der angeklickte `<p>`-Container), können Sie als zweites Argument den gewünschten *Kontext* nennen:

```
$(document).ready(function() {
    $('#p1').click( function() {
        alert(this.id);
        $(this).add("#p2",document)
            .css({color:'red'})
    });
});
```

Anstelle eines jQuery-Selektors kann auch ein *DOM-Element* übergeben werden. Hier ist es eine Referenz, die in einer Variablen `absatz2` abgelegt wurde:

```
$(document).ready(function() {
    var absatz2 = document.getElementById('p2');

    $('#p1').click( function() {
        alert(this.id);
        $(this).add(absatz2)
            .css({color:'red'})
    });
});
```

Ein an `.add()` übergebener *HTML-String* wird in einen Elementknoten umgewandelt, der dann ins Dokument eingehängt werden kann. Hier wird zu dem ausgewählten `<p>`-Container ein weiterer erzeugt, und beide werden an das Ende des Dokumentinhalts angehängt (der Container `p#p1` wird dabei von seiner Originalposition entfernt):

```
$(document).ready(function() {
    $('#p1').click( function() {
        alert(this.id);
        $(this).add('<p>Ich bin neu!</p>')
            .appendTo(document.body).css({color:'red'})
    });
});
```

Methode `.andSelf()`

Beim Anwenden von Traversierungsmethoden verschiebt sich die Collection auf die Elemente, auf die sich die Traversierung richtet. Möchten Sie traversieren, aber gleichzeitig auch die ursprüngliche Collection erhalten, können dies über `.andSelf()` vornehmen:

```
$(document).ready(function() {
    $('#box1')
        .children('p')
        .andSelf()
```



```
.css({border:'1px solid green'})
});
```

Ausgehend von `div#box1`, wird zuerst eine Collection aus deren `<p>`-Children zusammengestellt (also traversiert). Anschließend wird die primäre Collection über `.andSelf()` wieder hinzugefügt. Im Ergebnis erhalten sowohl `div#box1` als auch ihre Textabsätze einen grünen Rahmen.

4.5.4 Filtern von Collections

Anstatt eine Collection wie im Falle der Traversierung zu verschieben oder die Zahl der Elemente zu erweitern, kann jQuery auch das Gegenteil tun, nämlich einen Filterausdruck auf die Collection anwenden, um nur solche Elemente zu belassen, die einer Bedingung entsprechen.

Methode	Erläuterung
<code>.clone()</code> <code>.clone(mitEvents)</code>	Erstellt eine Kopie (Clone) der aktuellen Collection. (Der optionale Boolesche Parameter <code>mitEvents</code> bestimmt, ob dabei Eventbindings mitkopiert werden.)
<code>.eq(n)</code>	Reduziert die aktuelle Collection auf die Elemente, die innerhalb der Collection den übergebenen Index <code>n</code> besitzen. Anmerkung: Für <code>n < 0</code> wird ab dem letzten Item rückwärts gezählt.
<code>.filter(sel)</code> <code>.filter(fn(i))</code>	Reduziert die aktuelle Collection auf diejenigen Elemente, die dem übergebenen Selektor <code>sel</code> entsprechen. Alternativ: Reduziert die Collection auf die Elemente, die den Test der Prüffunktion <code>fn</code> bestehen (die Funktion erhält den Index <code>i</code> des geprüften Elements als Parameter).
<code>.first()</code>	Reduziert die aktuelle Collection auf ihr erstes Element.
<code>.has(sel)</code>	Reduziert die aktuelle Collection auf Elemente, die auf der Descendant-Achse Nachfahren besitzen, auf die der Selektor <code>sel</code> passt.
<code>.last()</code>	Reduziert die aktuelle Collection auf ihr letztes Element.
<code>.map(fn(i, elem))</code>	Wendet auf alle Elemente der Collection die Funktion <code>fn</code> an, die die Indexposition <code>i</code> und den DOM-Knoten <code>elem</code> des aktuellen Elements übergeben bekommt. Erzeugt ein <i>jQuery-Objekt</i> aus allen Rückgabewerten. Anmerkung: <code>this</code> bezieht sich in <code>fn</code> auf den DOM-Knoten des aktuellen Items.

Tabelle 4.18 Methoden zum Filtern der Collection

Methode	Erläuterung
<code>.not(sel)</code> <code>.not(elems)</code> <code>.not(fn(i))</code>	Entfernt alle Elemente aus der aktuellen Collection, die dem Selektor <code>sel</code> entsprechen. Alternativ: Entfernt die Elemente aus der aktuellen Collection, die dem DOM-Knoten <code>elems</code> entsprechen. Oder: Filtert anhand einer Funktion <code>fn</code>, die den Collection-Index <code>i</code> erhält und pro Item <code>true</code> (entfernen) oder <code>false</code> (behalten) zurückgibt.
<code>.slice(start, ende)</code> <code>.slice(start)</code>	Reduziert die aktuelle Collection auf diejenigen, deren Index durch <code>start</code> und <code>ende</code> begrenzt wird. Wird nur ein Parameter <code>start</code> übergeben, wird die Collection auf die Elemente ab dem entsprechenden Index reduziert.

Tabelle 4.18 Methoden zum Filtern der Collection (Forts.)

Methode `.eq(position)`

Diese Methode reduziert eine Menge von `n` jQuery-Elementen auf ein einziges Element mit dem Argument `position`:

```
$("#box p").eq(1).text("Nur dieses Element wird geändert");
```

Der Zähler beginnt bei 0 und endet bei `length-1`. Eine `position` größer als die Anzahl der tatsächlich vorhandenen Elemente (»out of range«) gibt ein leeres Objekt zurück.

Methode `.filter()`

Die `.filter()`-Methode reduziert die primäre Collection auf diejenigen Elemente, die dem Filterkriterium entsprechen. Hierfür werden alle Elemente der Collection in einer impliziten Schleife geprüft. Pro geprüftes Item gibt die Methode entweder `true` oder `false` zurück und behält das Element entsprechend in der Collection oder entfernt es.

Ausgehend vom Quelltext

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div id="d1">
    <p id="p3">Dies ist ein Div-Container in div#box1,
      einen Textabsatz enthält.</p>
  </div>
</div>
```

sollen nur die `<p>`-Container ausgewählt werden, die einen bestimmten Textstring enthalten. Hierfür wird von einer Collection aus allen `<p>`-Containern ausgegangen, die anschließend gefiltert wird. Das Filterkriterium können Sie als beliebigen Selektor formulieren:

```
$(document).ready(function() {
    $('p')
    .filter(':contains(zweite)')
    .css({border:'1px solid green'})
});
```

Hier wird `p#p2` ausgefiltert und gerahmt, da nur dieser Absatz das Wort »zweite« enthält.

Als Filterargument kann auch eine Funktion eingesetzt werden, die einen Booleschen Wert zurückgibt. Diejenigen Items, für die `true` zurückgegeben wird, verbleiben in der Collection:

```
$(document).ready(function() {
    $('p')
    .filter( function(){
        return $(this).parents('div').length < 2;
    } )
    .css({border:'1px solid green'})
});
```

Die `<p>`-Container `p#p1` und `p#p2` werden gerahmt, der dritte, im verschachtelten Div liegende Absatz erhält keinen Rahmen, da er dem Filterkriterium nicht entspricht.

Methode `.not()`

Im Grunde kann man `.not()` als die Umkehrung von `.filter()` auffassen, wobei `.not()` neben Selektoren und Funktionen mit Booleschem Rückgabewert auch *DOM-Elemente* als Argument entgegennimmt. Zunächst folgt eine Umsetzung der beiden vorangegangenen Beispiele mit `.not()` anstelle von `.filter()`, wobei vom gleichen Quelltext ausgegangen wird:

```
$(document).ready(function() {
    $('p')
    .not(':contains(zweite)')
    .css({border:'1px solid green'})
});
```

Hier verbleiben die `<p>`-Container in der Collection, die den gesuchten Text *nicht* enthalten. Gerahmt werden also `p#p1` und `p#p3`. Analog ist das Ergebnis des Aufrufs mit dem Funktionsargument exakt umgekehrt zu `.filter()`:

```
$(document).ready(function() {
    $('p')
        .not( function(){
            return $(this).parents('div').length < 2;
        } )
        .css({border:'1px solid green'})
});
```

Diesmal wird `p#p3` selektiert, die anderen Textabsätze bleiben ohne Rahmen. Übergeben Sie `.not()` eine Referenz auf einen DOM-Knoten, entfernt die Methode das entsprechende Item aus der Collection:

```
$(document).ready(function() {
    var absatz2 = document.getElementById('p2');

    $('p')
        // DOM-Element als Argument:
        .not(absatz2)
        .css({border:'1px solid green'})
});
```

Hier werden, wie zu erwarten war, der erste und der dritte Textabsatz gerahmt.

Methoden `.first()` und `.last()`

Soll aus einer Collection nur das erste bzw. das letzte Item verarbeitet werden, können Sie diese mittels `.first()` oder `.last()` herausfiltern. Hier wird nur der erste `<p>`-Container der primären Collection mit einem Rahmen versehen:

```
$(document).ready(function() {
    $('p')
        .first()
        .css({border:'1px solid green'})
});
```

Die Methode `.last()` funktioniert analog.

Methode `.slice()`

Die `.slice()`-Methode greift aus einer Collection einen Indexbereich heraus, der durch das oder die übergebenen Argumente festgelegt wird. Der Index wird, wie üblich, ab 0 gezählt. Übergeben Sie der Methode nur ein Argument (eine Ganzzahl), belässt sie alle Items ab diesem Index (»inclusive«) in der Collection. Eine

zweite Ganzzahl bezeichnet entsprechend eine Indexposition, bis zu der die Collection übernommen wird, wobei das als Endposition genannte Item selbst nicht mehr enthalten ist (»exclusive«):

```
$(document).ready(function() {
    $('p')
        .slice(1,2)
        .css({border:'1px solid green'})
});
```

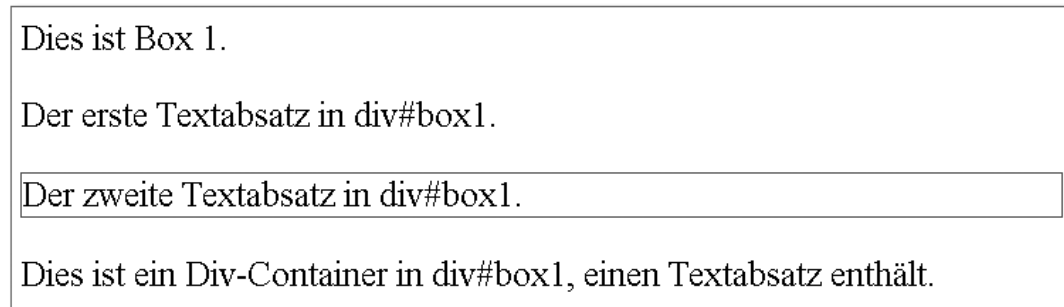


Abbildung 4.9 06_slice.html

Erläuterung

Der Wert `.slice(1)` behält demnach alle Items ab dem Item mit Indexposition 1 (also ab einschließlich des zweiten Items). Im Beispiel bewirkt das zweite Argument in `.slice(1,2)`, dass Items mit einer Indexposition größer gleich 2 ausgefiltert werden.

Übergeben Sie ein einziges, *negatives Argument*, wird der Indexpunkt (Start des Slices) ab der Länge der Collection rückwärts berechnet: Der Aufruf `.slice(-1)` bei einer Länge der Collection von 3 wird interpretiert als `.slice((3-1))`, also `.slice(2)`, und belässt daher das letzte Item in der Collection.

Methode `.has()`

Mit der Methode `.has()` können die Items einer Collection anhand ihres Elementinhalts gefiltert werden. Hierfür kann `.has()` ein gewöhnlicher Selektor übergeben werden. In diesem Beispiel wird zunächst eine Collection aus allen `<div>`-Containern des Dokuments zusammengestellt (das wären in diesem Fall zwei), die dann auf diejenigen Dives reduziert werden, die selbst ein Div enthalten:

```
$(document).ready(function() {
    $('div')
        .has('div')
        .css({border:'1px solid green'})
});
```

Gewählt und umrahmt wird der äußere Div. Alternativ kann `.has()` auch eine DOM-Referenz übergeben werden, sofern Sie eine zur Hand haben. Hier ist dies eine Referenz auf den innersten `<p>`-Container `p#p3`. Diesmal wird die Ausgangs-Collection jedoch *nicht* reduziert, da beide Container das Kriterium erfüllen. Die Methode schaut nicht auf die Hierarchie, sondern nur darauf, ob das Vergleichselement in den untersuchten Containern »irgendwo« enthalten ist:

```
$(document).ready(function() {
    var absatz3 = document.getElementById('p3');
    $('div')
        .has(absatz3)
        .css({border:'1px solid green'})
});
```

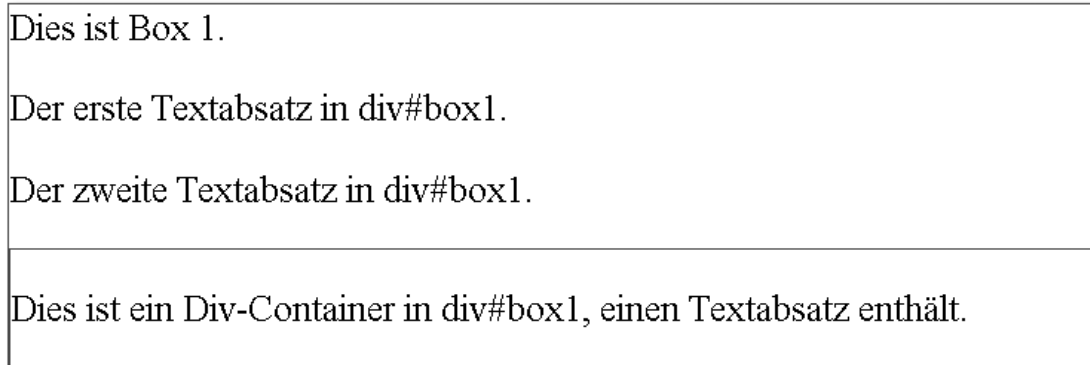


Abbildung 4.10 06_has2.html

4.5.5 Aufheben einer Filterung

Wollen Sie von einer gefilterten Collection zu deren Zusammensetzung vor der Filterung zurückgehen, setzen Sie die Methode `.end()` ein. Dies nimmt alle Filterungen zurück (auch mehrere hintereinander ausgeführte), die nach Erstellung der Basis-Collection vorgenommen wurden, und stellt die Basis-Collection wieder her.

Methode	Erläuterung
<code>.end()</code>	Macht den letzten auf die Collection angewendeten Filtervorgang rückgängig und kehrt zur Collection davor zurück.

Tabelle 4.19 Methode zur Rücknahme einer Filterung

Methode `.end()`

Die `.end()`-Methode hebt stets alle Filterungen auf und geht auf die Collection am Anfang der Kette zurück. Dies kann ohne Weiteres auch mehrmals geschehen.

In diesem einfachen Beispiel werden `.end()`, `.first()` und `.last()` auf diesem Weg miteinander kombiniert:

```
$(document).ready(function() {
    $('p')
        .first()
        .css({border:'1px solid green'})
        .end()
        .last()
        .css({border:'1px dotted red'})
        .end()
        .css({'font-style':'italic'})
});
```

Hier wird die erstellte Collection zweimal jeweils gefiltert und zurückgesetzt, um am Ende komplett nochmals mit einem Style belegt zu werden. Alle Absätze werden am Ende kursiv ausgegeben, der erste und der letzte erhalten (über `.first()` und `.last()`) zusätzlich verschiedene Rahmen.

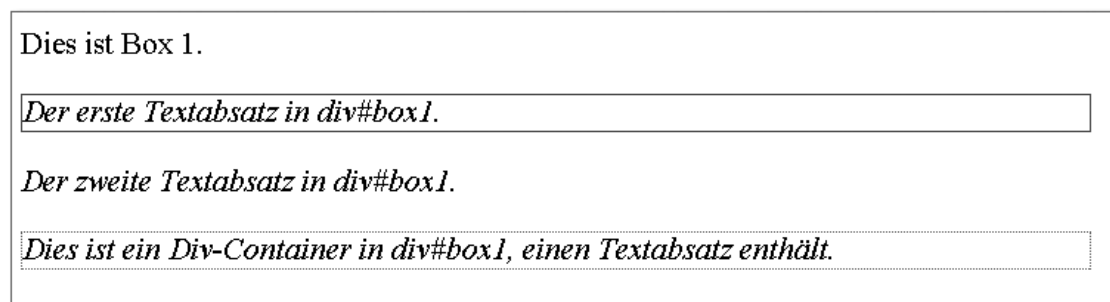


Abbildung 4.11 06_end.html

4.5.6 Kopieren einer Collection

Praktisch ist die Möglichkeit, Elemente einer Collection oder eine gesamte Collection kopieren zu können. Hierzu leistet die jQuery-Methode `.clone()` gute Dienste.

Methode	Erläuterung
<code>.clone()</code>	Erstellt eine Kopie (Clone) der aktuellen Collection.
<code>.clone(mitEvents)</code>	Der optionale Boolesche Parameter <code>mitEvents</code> bestimmt, ob dabei Eventbindings mitkopiert werden. Sinnvollerweise wird meist <code>true</code> übergeben.

Tabelle 4.20 Methode zum Kopieren einer Collection

Beachten Sie jedoch, dass die so erzeugten Collection-Items nicht automatisch Teil des Dokuments werden. Woher sollte jQuery auch wissen, an welcher Stelle

sie benötigt werden? Für die folgenden Beispiele muss daher eine Methode vorweggenommen werden, die Elemente in das Dokument einfügt.

Methode `.clone()`

Verwendet wird wieder der bereits bekannte Quelltext:

```
<div id="box1">Dies ist Box 1.
  <p id="p1">Der erste Textabsatz in div#box1.</p>
  <p id="p2">Der zweite Textabsatz in div#box1.</p>
  <div id="d1">
    <p id="p3">Dies ist ein Div-Container in div#box1,
      der einen Textabsatz enthält.
    </p>
  </div>
</div>
```

Nun sollen alle existierenden `<p>`-Container geklont und in `div#d1` eingefügt werden. Dies soll nach dem dort bereits vorhandenen `p#p3` geschehen, und die neu eingefügten `<p>`-Container sollen (wie üblich) gerahmt werden. Im Grunde ist dies ganz einfach:

```
$(document).ready(function() {
  // Alle P-Container auswählen ...
  $('p')
  // ... und kopieren
  .clone()
  // Einen Rahmen für die Kopien:
  .css({border:'1px solid green'})
  // ... und in div#d1 einfügen:
  .appendTo('#d1')
});
```

Die geklonten `<p>`-Container erhalten einen grünen Rahmen und werden danach an das Ende der Inhalte von `div#d1` angehängt (eben dies bewirkt `.appendTo()`). Allerdings produzieren Sie hier invaliden Quellcode, da die `id`-Attribute mitkopiert werden und daher unerlaubterweise doppelt vergeben sind. Sie sollten also bei den geklonten Elementen entfernt werden. Dies können Sie mit `.removeAttr()` bewirken:

```
$(document).ready(function() {
  $('p')
  .clone()
  .removeAttr('id')
  .css({border:'1px solid green'})
});
```



```

        .appendTo('#d1')
    });

```

Interessant wird es, wenn an ein zu klonendes Element eine Eventbindung erfolgt ist. Um dies zu demonstrieren, erhalten alle Textabsätze einen Click-Event-Listener, der die Hintergrundfarbe eines angeklickten Absatzes auf Hellgrau ändert. Danach wird wieder, wie eben, geklont:

```

$(document).ready(function() {
    $('p')
    // Click-Event für alle existierenden P:
    .click( function(){
        $(this).css({'background':'#ddd'});
    })
    // Eventbindung wird nicht mitkopiert!
    .clone()
    .removeAttr('id')
    .css({'border':'1px solid green'})
    .appendTo('#d1')
});

```

Die geklonten `<p>`-Container zeigen beim Anklicken keine Wirkung. Dies ist normal, denn Eventbindungen werden beim Klonen nicht automatisch mitkopiert. Über ein simples Flag-Argument (der `.clone()`-Methode wird der Wert `true` übergeben) können Sie dies jedoch veranlassen:

```

$(document).ready(function() {
    $('p')
    // Click-Event für alle existierenden P:
    .click( function(){
        $(this).css({'background':'#ddd'});
    })
    // Click-Eventbindung mitkopieren:
    .clone(true)
    .removeAttr('id')
    .css({'border':'1px solid green'})
    .appendTo('#d1')
});

```

4.6 Events und Event Handling

Der Bereich Event Handling nimmt im jQuery-Kern eine gewichtige Rolle ein. Nicht nur wird ein vereinheitlichtes Eventobjekt zur Verfügung gestellt, es finden sich auch Methoden zur browserübergreifenden Eventbindung sowie eine Reihe

von spezialisierten Methoden, die für oft benötigte Routinen praktische Abkürzungen bieten.

4.6.1 Das Eventobjekt in JavaScript

Ereignisse in einer HTML-Seite treten auf, wenn etwas mit beliebigen Elementen der Seite geschieht. Dies kann mit Nutzeraktionen zusammenhängen, wie dem Anklicken, Selektieren oder Ziehen von Objekten, aber auch mit Vorgängen, wie dem Laden oder Entladen des Dokuments. Damit ein Event sinnvoll behandelt werden kann, sind, salopp gesagt, drei Dinge von Bedeutung:

- ▶ Das Ereignis muss überhaupt bemerkt werden.
- ▶ Es muss bekannt sein, um was für ein Ereignis es sich handelt, an welchem Element und bei welchen Koordinaten es auftritt.
- ▶ Eine Funktion muss mit dem Auftreten des Events verknüpft sein.

Für die Erfüllung von Punkt eins sorgt der *Event Listener*, der mit dem Elementknoten verknüpft sein muss, an dem das Ereignis auftritt – vereinfacht gesagt (Sie werden gleich sehen, dass dies nicht ganz stimmt). Der Event Listener kann über ein Attribut in den HTML-Code geschrieben oder per JavaScript nachträglich (also »unobtrusive«) hinzugefügt worden sein. Aufgabe des Listeners ist es, den sogenannten *Event Handler* zu starten, ein Funktionsobjekt, das die JavaScript-Anweisungen enthält, die beim Eintritt des Ereignisses, auf das der Handler wartet, ausgeführt werden. Dies erfüllt Punkt drei:

```
<p id="p1" onclick="alert('Ein Ereignis trat auf.')">
  Bitte hier klicken!</p>
```

Würde das Ereignis per Script gebunden, sähe dies etwa so aus (dies ist wirkungsgleich zur eben gezeigten Inline-Variante):

```
var meinP = document.getElementById('p1');

meinP.addEventListener('click', function() {
    alert('Ein Ereignis trat auf.');
```

Das Eventobjekt

Welche Informationen sind über das Ereignis bekannt, sprich, was ist mit Punkt zwei der eben angeführten Liste? Klar ist, dass es sich um ein Ereignis vom Typ »Klick« handelte, das an einem <p>-Container auftrat, den man deshalb als *Target* bezeichnet. Der Klick fand auch an bestimmten Koordinaten (genannt »pageX« und »pageY«) innerhalb des Textabsatzes statt, die vielleicht interessant sein könnten.

Praktischerweise fasst JavaScript eben diese Informationen (und noch mehr) zusammen und stellt sie als Objekt zur Verfügung, das deshalb als *Eventobjekt* bezeichnet wird. Ein solches Eventobjekt wird bei jedem auftretenden Ereignis gebildet. Bevor Sie zu überlegen beginnen, wie Sie an dieses Objekt herankommen, um es zu verwenden – hierfür ist bereits gesorgt: Das Eventobjekt wird stets dem Event Handler übergeben.

An dieser Stelle ist es nötig, zu präzisieren, *was genau* der Event Handler eigentlich ist – und zwar ist dies die an das Ereignis gebundene *Funktion*. Einfacher zu zeigen ist dies bei der Scriptbindung – die Funktion ist fett markiert:

```
meinP.addEventListener('click', function() {
    alert('Ein Ereignis trat auf.');
```

```
}, false);
```

Ein Wert, der einer Funktion übergeben wird, muss auch in einem *Übergabeparameter* aufgefangen werden, sonst landet er im Datennirvana (bislang geschieht das noch mit dem Eventobjekt). Nennen Sie die Variable *e*, und nehmen Sie weiter an, dass das Eventobjekt tatsächlich dort landet. Sie können die Art des Ereignisses aus dem übergebenen Objekt dann wie folgt auslesen:

```
meinP.addEventListener('click', function(e) {
    alert('Ein ' + e.type + '-Ereignis trat auf.');
```

```
}, false);
```

Eine analoge Methode ist für das Event-Handler-Attribut nicht möglich. Zwar wird *implizit* ebenfalls ein anonymes Funktionsobjekt um die Anweisungen im Attributwert gehüllt (Achtung – das Beispiel ist eine rein illustrative Verdeutlichung des Geschehens):

```
<!-- So kann man sich das vorstellen: -->
<p id="p1"
  onclick="function(){ alert('Ein Ereignis trat auf.')} ">
  Bitte hier klicken!</p>
```

Da aber keine Möglichkeit besteht, den Wrap in ein Funktionsobjekt explizit vorzunehmen, kann kein Übergabeparameter bestimmt werden, der das Eventobjekt in den Anweisungsblock hineinreicht. Das ist aber nicht weiter schlimm, da Inline-Event-Handler sowieso möglichst wenig eingesetzt werden sollten.

Das Eventobjekt ist im Prinzip eine gute Sache. Leider ist auch dieser Aspekt nicht plattformübergreifend implementiert und wie immer folgt Internet Explorer nicht dem Standard und definiert das Eventobjekt als globales Objekt – immerhin mit »weitestgehend« ähnlichen Eigenschaften. (Da die erste Implementierung durch Microsoft allerdings vor der Standardisierung stattfand, sei dies verziehen.

Tatsache ist, dass so vom Programmierer einiges an Verrenkungen verlangt würde, um plattformübergreifenden Code zu schreiben.)

4.6.2 Das jQuery-Eventobjekt

jQuery bietet uns seine eigene Interpretation des Eventobjekts, die im Rahmen des Frameworks benutzt wird. Sowohl Eigenschaften als auch das Erreichen des Objekts für die jQuery-Eventmethoden ist *im Sinne des W3C vereinheitlicht* und steht so allen Browsern in identischer Form zur Verfügung. Es lohnt sich also eine kurze Beschäftigung damit durchaus.

Eigenschaften des Ereignisses

Die in folgender Tabelle dargestellten Eigenschaften stellt das jQuery-Eventobjekt zur Verfügung.

Property	Erläuterung
<code>event.currentTarget</code>	Das DOM-Element, das während der Bubbling-Phase Ziel des Ereignisses ist
<code>event.data</code>	Die Daten, die für das aktuelle Ereignis an <code>jQuery.fn.bind</code> übergeben werden, sofern vorhanden
<code>event.pageX</code>	Mausposition relativ zum linken Rand des Dokuments
<code>event.pageY</code>	Mausposition relativ zum oberen Rand des Dokuments
<code>event.relatedTarget</code>	Ein anderes DOM-Element, das am Ereignis beteiligt ist, falls vorhanden; sonst <code>null</code>
<code>event.result</code>	Der letzte Rückgabewert eines Event Handlers, sofern dieser nicht <code>undefined</code> war
<code>event.target</code>	Das DOM-Element, an dem das Ereignis stattfand
<code>event.timeStamp</code>	Zeitstempel des Ereignisses (Zahl der Millisekunden seit 1. Januar 1970)
<code>event.type</code>	Die Art des Ereignisses
<code>event.which</code>	Die Keyboardtaste oder Maustaste, die betätigt wurde

Tabelle 4.21 Eigenschaften des jQuery-Eventobjekts

Das Target-Element und das »currentTarget«

Im Rahmen des Eventobjekts sind zwei Informationen über den Listener erreichbar, der aktuell »feuert«, also den Event gerade verarbeitet. Hierbei enthält der Wert `target` stets eine DOM-Referenz auf das Element, an dem die Useraktion stattfindet, während `currentTarget` das Element wiedergibt, das in der Event-

kette (Capture- oder Bubbling-Phase, dazu im Anschluss mehr) aktuell den Event wahrnimmt. (Das `currentTarget` ist dabei auch das Objekt, auf das sich das `this` in der dann ausgelösten Handler-Funktion beziehen wird.)

```
$("#*").click(function(e){
    alert('Klick an: ' + e.target +
        ', erkannt an: ' + e.currentTarget);
});
```

Das »relatedTarget«

Bestimmte Events treten (beinahe) stets mit einem gegensätzlichen Event gemeinsam auf. So bedeutet ein *Focus*-Ereignis an Element A ein *Blur*-Ereignis an Element B, wenn Letzteres vorher den Focus besessen hat. Analog ist ein *Mouseover*-Event mit einem *Mouseout*-Event mit demjenigen Element verbunden, über dessen Bereich sich der Mauszeiger zuvor befand.

Jenes Element wird, ausgehend von demjenigen, an dem der erfasste Event auftritt, als *relatedTarget* bezeichnet. Folglich nennt sich auch die Eigenschaft des Eventobjekts `relatedTarget`, über die Sie eine DOM-Referenz des betreffenden Knotens erhalten:

```
$('#input#test').bind('mouseover', function(e){
    alert('Mouseover an: ' + e.target +
        ', Mouseover erkannt an: ' + e.relatedTarget);
});
```

Existiert kein `relatedTarget`, ist der Wert der Eigenschaft `undefined`.

Koordinaten eines Ereignisses

Mittels der Eigenschaften `pageX` und `pageY` des Eventobjekts lassen sich die Koordinaten eines Ereignisses auslesen – beispielsweise die eines Klicks auf den Dokumentkörper:

```
$('#body').bind('click', function(e){
    var position = e.pageX + 'px, ' + e.pageY + 'px';
    alert('Klick auf ' + e.target + ' an ' + position);
});
```

Zeitpunkt eines Ereignisses

Über die `timestamp`-Eigenschaft können Sie den Zeitpunkt eines Ereignisses erkennen. Die Eigenschaft gibt einen Zeitstempelwert in Millisekunden zurück, den Sie als Argument eines `Date`-Objekts einsetzen können:

```

$("a.zeitmelder").click(function(e){
    var zeitstempel =Date(e.timeStamp);
    alert('Ressource angefordert am: ' + zeitstempel);
});

```

Art und Trigger eines Ereignisses

Über die `type`-Eigenschaft lesen Sie die Art des Ereignisses aus, falls mehrere Events an einem Element auftreten können. Auch die Art des Triggerings (per Maus oder Tastatur) können Sie hiermit erkennen. Dies erfolgt über die Eigenschaft `which`, die einen numerischen Wert zurückgibt:

```

$('body').bind('mousedown mouseup', function(e){
    alert(e.type + ' mit ' + e.which );
});

```

Wird ein Mausereignis aufgefangen, gibt `which` für die linke Maustaste die 1, für die mittlere die 2 und für die rechte Maustaste die 3 zurück.

```

$('body').bind('keydown ', function(e){
    alert(e.type + ' mit ' + e.which );
});

```

Analog arbeitet `which` auch mit Tastaturereignissen zusammen. Es folgt eine Tabelle für die wichtigsten Tasten.

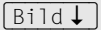
Taste	Code	Taste	Code
0 – 9	48 – 57	a – z	65 – 90
	8		9
	13		16
	17		18
	20		27
	32		33
	34		37
	38		39
	40		46

Tabelle 4.22 Tastaturcodes für Keyboard-Events (`event.which`-Property)

Anmerkung

Die `keyCode`-Eigenschaft, die ansonsten zum Erkennen der Taste eines Keyboard-Events verwendet wird, ist in jQuery nicht ausdrücklich implementiert, scheint aber meist zu funktionieren und gibt denselben Wert zurück.

Behandlung des Weitertragens des Events

Damit Sie die jQuery-Methoden zur Ereignisbehandlung wirklich verstehen, ist es erforderlich, ein wenig mehr über das *Weitertragen von Events* im DOM-Baum zu wissen. Dies ist vom Prinzip her einfach, von den Konsequenzen her jedoch unübersichtlich.

Tritt irgendwo im DOM-Baum ein Ereignis auf, beginnt seine (mögliche) Erfassung stets an der DOM-Wurzel. Von dort aus wandert das Ereignis ins Innere des Baums bis zum sogenannten *Target*, dem eigentlichen Ort, an dem das Ereignis stattgefunden hat. An jedem Knoten, der dabei passiert wird (der als *Observer* bezeichnet wird), kann das Ereignis über einen Listener erfasst werden. Diese Phase der Ereignisbehandlung ist die *Capture-Phase*.

In Abbildung 4.12 tritt am Link (dem Target) ein Klick auf, der während der Capture-Phase jedoch zuerst am Body, dann an einem Div und schließlich am `<p>`-Container, der den Link umgibt, erkannt werden kann.

Anschließend feuert der Event am Target selbst (Phase 2, die *Target-Phase*) und wandert dann von dort den Baum wieder nach oben (daher *Bubbling*) zum Elternknoten, dessen Elternknoten und schließlich, durch alle Instanzen, bis zur Dokumentwurzel (dem HTML-Tag). Dies ist die dritte Phase, die *Bubbling-Phase*. Auch hier kann ein Ereignis an jedem Punkt des Weges über einen Listener erkannt und mit einem Handler verarbeitet werden.

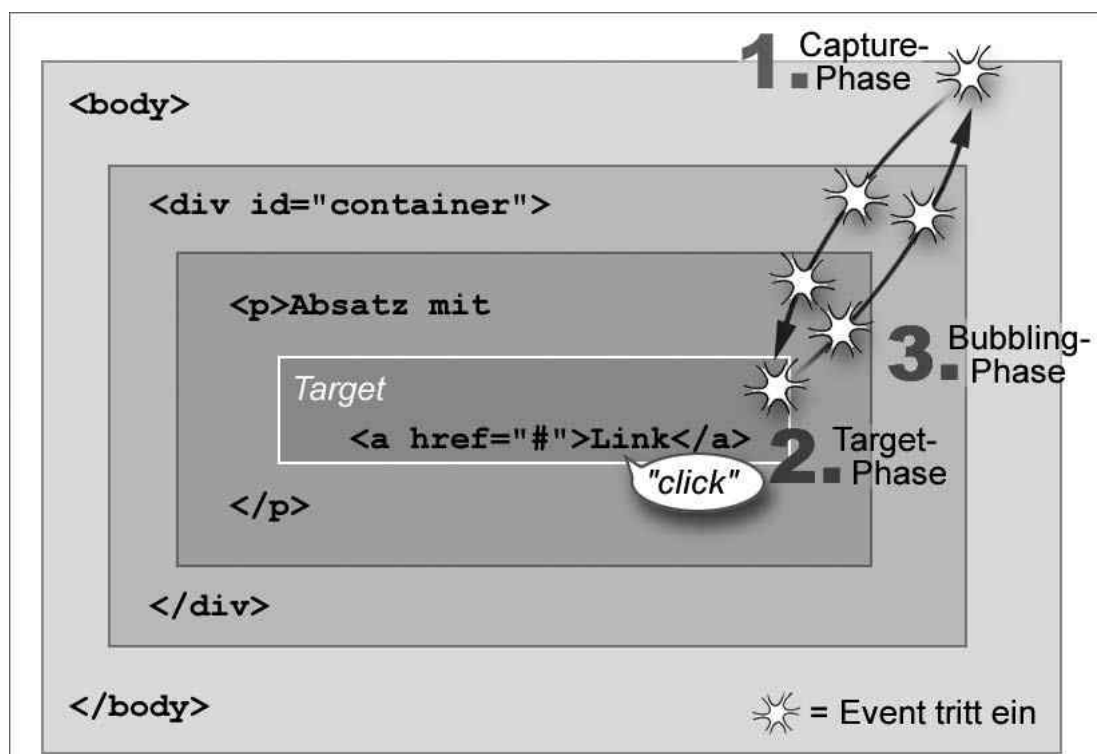


Abbildung 4.12 Capturing und Bubbling von Events

In welcher der Phasen ein Ereignis erkannt wird, hängt davon ab, wie der Listener definiert worden ist. Am Target selbst tritt das Ereignis nur einmal auf, vor Beginn der Bubbling-Phase.

Soll ein Event Listener für die Capture-Phase definiert werden, bekommt der dritte Parameter der `.addEventListener()`-Methode den Wert `true`:

```
document.body.addEventListener('click', function(){
    alert('Klick am Body erfasst (Capture)!');
}, true);
```

Der Witz dabei ist, dass in der Capture-Phase das Ereignis am Body erfasst wird, *bevor* es das eigentliche Target erreicht hat. Dort kann es natürlich ebenfalls aufgefangen werden. Anschließend beginnt die Bubbling-Phase. Ist ein zweiter Listener für den Body definiert, kann das Ereignis auf dem Rückweg ein zweites Mal aufgefangen werden. Der dritte Parameter ist `false`:

```
document.body.addEventListener('click', function(){
    alert('Klick am Body! (Bubbling)');
}, false);
```

Es ist wichtig, zu verstehen, dass die Weitergabe des Ereignisses (auch *Propagation* genannt) das Standardverhalten ist. Es ist jedoch möglich, die Weitergabe zu unterbinden. So können Sie dem Event Listener im Body für die Capturing-Phase befehlen, den Event nicht weiterzuleiten:

```
// Fängt das Event für confirm=true ab:
document.body.addEventListener( 'click', function(e){
    if(confirm('Capture anhaltent?')) e.stopPropagation();
}, true );
```

Halten Sie den Event durch `stopPropagation()` in der Capture-Phase auf, kann er an weiter innen im Baum liegenden Elementen nicht mehr erfasst werden.³ Eine Weitergabe kann aber auf gleiche Art auch erst auf dem Rückweg, vor oder während der Bubbling-Phase unterbrochen werden.

Methode	Erläuterung
<code>event.isImmediatePropagationStopped()</code>	Speichert den Aufruf von <code>event.stopImmediatePropagation()</code> .
<code>event.isPropagationStopped()</code>	Speichert den Aufruf von <code>event.stopPropagation()</code> .

Tabelle 4.23 jQuery-Methoden für Event-Propagation

³ Ein eventueller Bubbling-Listener im Body erfasst das Ereignis jedoch ein zweites Mal: Das Ereignis wird nicht »weggefangen«, sondern kehrt nur » auf der Stelle« um.

Methode	Erläuterung
<code>event.stopImmediatePropagation()</code>	Verhindert das Feuern weiterer Event Handler der gleichen Registrierungsgruppe.
<code>event.stopPropagation()</code>	Verhindert Weitergabe des Ereignisses an folgende Observer.

Tabelle 4.23 jQuery-Methoden für Event-Propagation (Forts.)

Existiert ein Link im Inneren eines Textabsatzes, der ebenfalls auf ein Klickereignis reagiert, wird durch Bubbling der `<p>`-Handler im Anschluss an den `<a>`-Handler ausgelöst:

```
<p id="p3" onclick="alert('Der Absatz wurde geklickt.')">
  Externer Link: <a class="extern" href="#">Ein Link</a>
</p>
```

Dies lässt sich über `.stopPropagation()` verhindern:

```
$("#a.extern").click(function(e){
  e.stopPropagation();
});
```

Behandlung des Defaultverhaltens

Bestimmte HTML-Elemente besitzen ein Defaultverhalten gegenüber bestimmten Ereignistypen. Ein `<a>`-Element beispielsweise erkennt einen Click-Event am Link und reagiert, indem es das Neuladen der Seite anhand des ihm übergebenen URL vornimmt. Dies können Sie verhindern, indem Sie die Methode `event.preventDefault()` für das Eventobjekt dieses Ereignisses aufrufen.

Property/Methode	Erläuterung
<code>event.isDefaultPrevented()</code>	Gibt den Zustand von <code>event.preventDefault()</code> als <code>true</code> oder <code>false</code> wieder.
<code>event.preventDefault()</code>	Verhindert die Defaultaktion, die mit dem Ereignis verbunden ist.

Tabelle 4.24 jQuery-Methoden zur Steuerung des Event-Defaultverhaltens

Der an das Element gebundene Handler wird stets ausgelöst, bevor das Defaultverhalten in Aktion tritt. So können Sie das Ausführen eines Links in einer Linkliste unterbinden und stattdessen eine Meldung ausgeben, wenn seine Gültigkeit zweifelhaft ist (hier wird dies über die an den Link gebundene CSS-Klasse »inaktiv« entschieden):

```
$("#a.inaktiv").click(function(e){
    alert('Link wird überprüft und ist derzeit deaktiviert');
    e.preventDefault();
});
```

Ob ein Ereignis das Standardverhalten auslöst oder nicht, können Sie über die Methode `event.isDefaultPrevented()` erkennen, die in diesem Fall `false` zurückgibt:

```
$("#a.inaktiv").click(function(e){
    e.preventDefault();
    if(e.isDefaultPrevented())
        alert('Link wird überprüft und ist derzeit deaktiviert');
});
```

Eine andere Möglichkeit, das Defaultverhalten zu unterbinden, ist, der Handler-Funktion den Rückgabewert `false` zurückzugeben:

```
$("#a.inaktiv").click(function(e){
    alert('Link wird überprüft und ist derzeit deaktiviert');
    return false;
});
```

Dies ist jedoch nicht dasselbe, wie Sie in dieser Konfiguration sehen können:

```
<p id="p3" onclick="alert('Der Absatz wurde geklickt.')">
  Inaktiver Link: <a class="inaktiv" href="#">Ein Link</a>
</p>
```

Hier existiert am Textabsatz ein weiterer Listener für den Klick, der also in der Bubbling-Phase dort aufgefangen und erkannt würde. Die Rückgabe von `false` im Click-Handler unterbindet neben dem Defaultverhalten auch das Bubbling, so dass das Ereignis nie am `<p>`-Container ankommt. Die Methode `event.preventDefault()` arbeitet subtiler und lässt anschließendes Bubbling des Events zu.

jQuery normalisiert die Methoden des Eventobjekts W3C-konform

Alle soeben beschriebenen Methoden des jQuery-Eventobjekts zur Event-Propagation und zum Defaultverhalten entsprechen den namensgleichen Methoden des W3C-Eventobjekts. Auch hier brauchen Sie also keinen Gedanken mehr an browserspezifische Implementierungen verschwenden (... und ja, hier ist der Internet Explorer gemeint!).

4.6.3 Allgemeine Methoden zu Eventbindung

Das jQuery Framework kennt drei Methodenpärchen, die allgemein Event Handler an Elemente binden können: `.bind()`, `.live()`, `.delegate()` und ihre Gegenparts `.unbind()`, `.die()` und `.undelegate()`. Hierbei müssen Sie zwischen einer

Bindung an existierende Elementknoten und an, nennen wir dies »Bindung im Voraus«, durch jQuery zukünftig neu erzeugte Elementknoten unterscheiden.

Einfaches Binden von Events an die Collection

Die einfachste allgemeine Bindung geschieht durch die Methode `.bind()`.

Methode	Erläuterung
<code>.bind(typ, fn)</code> <code>.bind(typ,[data], fn)</code>	Bindet einen Handler <code>fn</code> an den Event <code>typ</code> (und nimmt optional Daten <code>data</code> als zweiten Parameter entgegen). <code>fn</code> bekommt das Eventobjekt übergeben.
<code>.bind(events)</code>	Alternativ: Nimmt ein Mapping aus Events <code>events</code> in Form eines Objekts entgegen.

Tabelle 4.25 Allgemeines Binden von Events

Im einfachsten Fall wird ein Event an alle Elemente einer Collection gebunden, indem `.bind()` der Eventtyp und die Handler-Funktion als Argumente übergeben wird. Hier erhalten alle `<p>`-Container einen Click-Event:

```
$( "p" ).bind( "click", function() {
    alert( "Du hast mich angeklickt!" );
});
```

Zwei (oder mehr) Events können gleichzeitig gebunden werden, indem die Eventbezeichner durch Leerzeichen getrennt als ein Stringargument übergeben werden. Allen Events wird die gleiche Handler-Funktion zugewiesen:

```
$( 'p' ).bind( "mouseenter mouseleave", function() {
    // bindet oder entfernt eine CSS-Klasse:
    $( this ).toggleClass( "hover" );
});
```

Eine weitere, wesentlich flexiblere Möglichkeit zum gleichzeitigen Binden mehrerer Events über `.bind()` besteht darin, der Methode ein *JavaScript-Objekt* als Argument zu übergeben, das Eventname und zuständige Handler-Funktion als Properties enthält. Dies wird als *Eventmap* bezeichnet. Hier werden drei Events gebunden:

```
$( "p" ).bind( {
    mouseenter: function() {
        $( this ).addClass( "hover" );
    },
    mouseleave: function() {
        $( this ).removeClass( "hover" );
    },
});
```

```

    click: function(){
        $(this).addClass("clicked");
    }
});

```

Die Bindung wirkt leider nur auf diejenigen Elemente im DOM, die zum Zeitpunkt des Aufrufs der Bindung existieren. Hier werden alle Elemente der Klasse »klickbar« mit einem Click-Event-Handler versehen:

```

$('.klickbar').bind('click', function() {
    alert("Element wurde angeklickt.");
});

```

Fügen Sie nun Elemente mit dieser Klasse nachträglich hinzu, erstreckt sich die Eventbindung auf diese nicht automatisch:

```

$('body').append('<p class="klickbar">Nicht klickbar!</p>');

```

Ein erneuter Aufruf der `.bind()`-Methode wäre vonnöten. Um dies zu vermeiden, existieren Alternativmethoden zur Eventbindung, die Bindungen für später hinzugefügte Elemente vormerken. Doch dazu gleich noch mehr.

Lösen von Events

Das Lösen von Events, die mittels `.bind()` gebunden wurden, geschieht durch die Methode `.unbind()`. Ein Ablösen eines Events ist dann notwendig, wenn dieses nicht, wie sonst der Fall, bis zum Entladen der Seite an einem Element gültig bleiben soll.

Methode	Erläuterung
<code>.unbind()</code>	Entfernt alle Eventbindungen von den Elementen der Collection.
<code>.unbind(typ)</code>	Wird an <code>.unbind()</code> ein Eventtyp <code>typ</code> übergeben, wird nur dieser Eventtyp entfernt.
<code>.unbind(typ, fn)</code>	Wird zusätzlich zum Typ <code>typ</code> eine Handler-Funktion <code>fn</code> übergeben, wird nur dieser Handler für den betreffenden Event entfernt.
<code>.unbind(e)</code>	Alternativ: Nimmt ein Eventobjekt <code>e</code> entgegen, das den zu entfernenden Event bezeichnet.

Tabelle 4.26 Allgemeines Lösen von Events

Meist wird `.unbind()` mit dem zu lösenden Eventtyp als Argument verwendet. Folgender Aufruf löst den Click-Event eines angeklickten `<p>`-Containers:

```

$("p").bind("click", function() {
    alert("Anklicken geht nur einmal!");
});

```

```
$(this).unbind("click"); // kein weiterer Klick möglich
});
```

Dasselbe Ergebnis wäre einfacher mit dem Convenience-Eventbinder `.one()` zu erreichen gewesen. (Führen Sie einen zusätzlichen Counter ein, könnten Sie den Event Handler auch erst nach dem zweiten, dritten etc. Klick lösen.)

Binden von Events an zukünftige Elemente – `.live()`

Mit `.bind()` kann, wie vorhin erwähnt, eine Eventbindung nur an bestehende Elemente des Dokuments erfolgen. Dies bedeutet, dass nachträglich hinzugefügten Bestandteilen der Seite (die beispielsweise per Ajax nachgeladen wurden), die entsprechenden Event Handler fehlen.

Da es umständlich ist, nach jeder DOM-Manipulation die Eventbindung nachzuholen, gibt es Methoden, die eine Eventbindung an Elemente bestimmten Typs quasi »vormerken« und beim Erzeugen eines entsprechenden Containers automatisch vornehmen. Dies sind die Methoden `.live()` und `.delegate()`, für die jeweils die Pendanten `.die()` bzw. `.undelegate()` zum Lösen der Bindungen existieren.

Methode	Erläuterung
<code>.live(typ, fn)</code> <code>.live(typ, data, fn)</code>	Bindet über einen Event <code>typ</code> eine Handler-Funktion <code>fn</code> an alle Elemente der Collection und zukünftig hinzugefügte Elemente, die demselben Selektor entsprechen. (Nimmt gemappte Daten <code>data</code> als optionalen zweiten Parameter entgegen, die an die Handler-Funktion weitergereicht werden.)
<code>.die()</code> <code>.die(typ)</code> <code>.die(typ, fn)</code>	Entfernt über die Methode <code>.live()</code> gebundene Events. (Optional kann der Eventtyp <code>typ</code> sowie der abzukoppelnde Handler <code>fn</code> übergeben werden.)

Tabelle 4.27 Binden/Lösen von Events für bestehende und zukünftig existierende Elemente

Die Methode `.live()` funktioniert nach außen analog zu `.bind()`. Die Bindung erfolgt unverzüglich an alle Elemente der Collection, die die Methode aufruft:

```
$('.klickbar').live('click', function() {
    alert("Element wurde angeklickt.");
});
```

Ein nachträglich hinzugefügtes Element verfügt ebenfalls über die Bindung:

```
$('#body').append('<p class="klickbar">Auch klickbar!</p>');
```

Diese Bindung ist allerdings nur eine scheinbare, da nicht wirklich die eigentlichen Target-Kandidaten mit dem Event belegt werden. Stattdessen wird das Ereignis am Ende seiner Bubbling-Phase am Root aufgefangen. Ist ein solcher Event über `.live()` registriert, wird nun geprüft, ob das Target-Element den Bindungskriterien entspricht. Ist das der Fall, wird der Handler ausgelöst.

Der Kontext für das Auswerten von `.live()` ist also in der Regel das gesamte Dokument. Dies kann eingeschränkt werden, indem die Bindung im Rahmen eines explizit definierten Kontexts erfolgt.

Hier ist dieser Kontext ein Element mit dem ID `inhalt`, innerhalb dessen Textabsätze auf `Mouseover` reagieren sollen:

```
$("#inhalt").each(function(){
    // #inhalt dient als Kontext für .live():
    $("p", this).live("hover", function(){
        $(this).toggleClass("hervorheben");
    });
});
```

Die Auswertung von `.live()` erfolgt auch hier während der Bubbling-Phase, allerdings wird diese bereits am Element `#inhalt` unterbrochen. Danach wird das Event-Target mit dem Selektorkriterium von `.live()` verglichen und gegebenenfalls der Handler mit am Target ausgeführt.

Anmerkung

Der hier gebundene "hover"-Event ist ein Synonym für die beiden Events "mouseenter" und "mouseleave", die sonst einzeln gebunden werden müssten. Anders gesagt, das Argument "hover" entspricht dem Argument "mouseenter mouseleave".

Ein Lösen von Events, die durch `.live()` gebunden wurden, kann nicht durch `.unbind()` erfolgen – wie bereits erwähnt, existiert gar keine »echte« Bindung am Target selbst. Stattdessen muss die Methode `.die()` aufgerufen werden:

```
$("p").live("click", function() {
    alert("Anklicken geht nur einmal!");
    $(this).die("click"); // kein weiterer Klick möglich
});
```

Binden von Events an zukünftige Elemente – `.delegate()`

Vergleichbar mit `.live()` arbeitet `.delegate()`, wobei damit allerdings nicht an die aktuelle Collection und analoge Elemente gebunden wird. Stattdessen bringt `.delegate()` ein eigenes Selektorargument mit und verwendet die aktuelle Coll-

action als Kontext der späteren Auswertung. Delegiert wird also ausgehend von der Collection an selektierte Nachfahren.

Methode	Erläuterung
<code>.delegate(sel, typ, [data,] fn)</code>	Bindet den Handler <code>fn</code> über den Event <code>typ</code> an alle Elemente, auf die <code>sel</code> passt, wobei dies sowohl die aktuell existierenden als auch später hinzugefügte Elemente dieses Typs betrifft. (Nimmt optional als dritten Parameter Daten <code>data</code> entgegen, die dem Handler übergeben werden.)
<code>.undelegate()</code> <code>.undelegate(sel, typ)</code> <code>.undelegate(sel, typ, fn)</code>	Entfernt alle durch <code>.delegate()</code> vorgenommenen Eventbindungen von den Elementen der aktuellen Collection. Werden ein Selektor <code>sel</code> und ein Eventtyp <code>typ</code> übergeben, werden auch nur entsprechende Events an den durch <code>sel</code> bezeichneten Elementen der aktuellen Collection entfernt. Wird als dritter Parameter eine Handler-Funktion <code>fn</code> übergeben, wird nur dieser Handler für den betreffenden Event entfernt.

Tabelle 4.28 Binden/Lösen von Events für bestehende und zukünftig existierende Elemente

Ein analoges Beispiel finden Sie bei `.live()`. Auch hier wird an `<p>`-Container innerhalb von `#inhalt` ein Hover-Event-Handler »delegiert«, was auch später hinzugefügte Elemente erfasst. Da Sie hier den Kontext nicht bei der Bindung setzen müssen, sondern sich dieser aus der Collection ergibt, sparen Sie die `.each()`-Schleife in diesem Fall ein:

```
$("#inhalt").delegate("p", "hover", function(){
    $(this).toggleClass("hervorheben");
});
```

Ähnlich wie im Fall von `.bind()` und `.live()` können durch `.delegate()` gebundene Events nur durch eine eigene Methode gelöst werden, die sich `.undelegate()` nennt.

4.6.4 Spezielle Eventbindungen (Convenience-Methoden)

Selbstverständlich lassen sich per `.bind()` auch komplexere Eventsituationen behandeln. Für bestimmte, häufig auftretende Fälle stellt jQuery drei spezialisierte Eventbindungsmethoden zur Verfügung, die es Ihnen ersparen, lästiges Beiwerk zu schreiben. Alle drei Methoden können Sie jedoch nur auf existierende DOM-Elemente anwenden – Convenience-Pendants zu `.live()` und `.delegate()` gibt es nämlich nicht.

Methode	Erläuterung
<code>.hover(fnI, fnO)</code> <code>.hover(fnIO)</code>	Bindet die Callback-Funktion <code>fnI</code> an den Mouseenter-Event und die Callback-Funktion <code>fnO</code> an den Mouse-Leave-Event. Wird nur eine Callback-Funktion <code>fnIO</code> übergeben, wird diese an beide Events gebunden. An die Callback-Funktionen wird das Eventobjekt übergeben.
<code>.one(typ, fn(e))</code> <code>.one(typ, d, fn(e))</code>	Bindet zur <i>maximal einmaligen</i> Ausführung einen Handler <code>fn</code> an den Event <code>typ</code> (und nimmt optional Daten <code>data</code> als zweiten Parameter entgegen). <code>fn</code> bekommt das Eventobjekt übergeben.
<code>.toggle(fn1(e), fn2(e))</code> <code>.toggle(fn1(e), fn2(e)</code> <code>[, ..., fnN(e)])</code>	Bindet mindestens zwei Handler-Funktionen <code>fn1</code> und <code>fn2</code> an die Elemente der Collection und führt diese auf Klick <i>abwechselnd</i> aus. Werden weitere Handler <code>fn3</code>, ... <code>fnN</code> übergeben, werden alle gebundenen Handler auf Klick <i>nacheinander</i> ausgeführt, und es wird am Ende von vorn begonnen. Allen Funktionen wird das Eventobjekt <code>e</code> als Parameter übergeben.

Tabelle 4.29 Convenience-Methoden zur Eventbindung

Methode `.hover()`

Mit der Methode `.hover()` lassen sich gleichzeitig der Mouseenter- und der Mouseleave-Event binden. Entsprechend nimmt die Methode zwei Callback-Funktionen als Argumente entgegen, die jeweils ausgelöst werden:

```

$("p").hover(
  // 1) Callback für Mouseenter:
  function () {
    $(this).addClass("hervorheben ");
  },
  // 2) Callback für Mouseleave:
  function () {
    $(this).removeClass("hervorheben ");
  }
);

```

Da es kein `.unhover()` gibt, muss `.unbind()` bemüht werden – wobei die beiden tatsächlich gebundenen Events auch genannt werden müssen:

```

// löst eine Eventbindung durch .hover():
$("p").unbind("mouseenter mouseleave");

```


Die Methode `.one()` sorgt für eine einmalige Bindung eines Events. Sobald der Event an einem Target ausgelöst wurde, wird er automatisch (analog zu `.unbind()`) wieder entfernt, steht also nur ein einziges Mal zur Verfügung:

```
$("p").one("click", function() {
    alert("Anklicken geht nur einmal!");
});
```

Ein Lösen eines durch `.one()` gebundenen Events kann ebenfalls durch `.unbind()` vorgenommen werden, falls das erforderlich sein sollte – betroffen wären natürlich nur diejenigen Elemente mit Bindung, an denen der Event noch nicht aufgetreten ist.

Methode `.toggle()`

So wie `.hover()` eine Convenience-Bindung für Mouseenter und Mouseleave darstellt, arbeitet `.toggle()` mit dem Click-Event. Die Methode arbeitet verschieden, abhängig davon, welche und wie viele Argumente übergeben werden.

Meist erhält die Methode einfach zwei Callbacks, die dann auf Klick abwechselnd aufgerufen werden (und so praktisch eine »Ein-/Aus-Funktion« darstellen):

```
$('p').toggle(
    function() {
        $(this).addClass("hervorheben ");
    },
    function() {
        $(this).removeClass("hervorheben ");
    }
);
```

Übergeben Sie weitere Callbacks, werden diese der Reihe nach ausgeführt. Auf diese Weise können beliebig viele Funktionen nacheinander ausgelöst werden (das Target merkt sich, welche als Nächste an der Reihe ist):

```
$('p').toggle(
    function() {
        $(this).addClass("rot");
    },
    function() {
        $(this).addClass("gelb");
    },
    function() {
        $(this).addClass("gruen");
    }
);
```

4.6.5 Erzeugen von Events

Ein Ereignis kann durch jQuery auch simuliert bzw. künstlich zu einem geeigneten Zeitpunkt erzeugt werden. Hierfür existieren zwei Methoden, die sich allerdings geringfügig unterscheiden.

Methode	Erläuterung
<code>.trigger(typ, arrPm)</code>	Triggert an den Elementen der Collection einen Event <code>typ</code> und übergibt gleichzeitig ein Array aus weiteren Parametern. (Dieses Array darf leer sein.)
<code>.triggerHandler(typ, arrPm)</code>	Triggert an den Elementen der Collection einen Event <code>typ</code> , ohne dabei das Defaultverhalten auszulösen, und übergibt gleichzeitig ein Array aus weiteren Parametern. (Dieses Array darf leer sein.)

Tabelle 4.30 Methoden zum Generieren von Events

Liegt an einer Collection von Elementen eine Eventbindung wie hier vor

```
$('#inhalt p').bind('click', function() {
    alert($(this).text());
});
```

kann über die Methode `.trigger()` das Ereignis für alle Elemente ausgelöst werden – beispielsweise für Testzwecke.

```
$('#inhalt p').trigger('click');
```

Beachten Sie, dass hierbei das Defaultverhalten des Elements gegenüber dem getriggerten Ereignis ebenfalls ausgelöst sowie eine Bubbling-Phase ab dem Target eingeleitet wird. Möchten Sie dies nicht, können Sie alternativ die Methode `.triggerHandler()` einsetzen, die nur und ausschließlich die Handler-Funktion des Ereignisses startet:

```
$('#a[title]').bind('click', function() {
    alert($(this).attr('title'));
});
```

Um nicht den Link selbst auszulösen, sondern nur die Handler-Funktion zu starten, schreiben Sie:

```
$('#a[title]').triggerHandler('click');
```

Allerdings hat diese Methode zwei Nachteile: Das Ereignis wird zum Ersten nur am ersten Element der Collection getriggert, und der Rückgabewert ist zum Zweiten kein jQuery-Objekt. Stattdessen wird der Wert zurückgegeben, den die Handler-Funktion erzeugt. Des Weiteren bleibt das Ereignis »vor Ort« – es gibt also

keine anschließende Bubbling-Phase, in der ein Event Handling durch Ancestor-Elemente erfolgen könnte.

Events triggern durch andere Methoden

Alle auf einen bestimmten Eventtyp gerichteten Eventbinder wie `.click()`, `.keydown()`, `.focus()` etc. dienen, ohne Argument eingesetzt, auch als Trigger für »ihren« Event. Sie müssen dafür lediglich von einer Collection ausgehend aufgerufen werden.

```
$('#beispiel').click();
```

ist demnach nur eine Abkürzung für

```
$('#beispiel').trigger('click');
```

Eine Ausnahme bilden die Eventbinder-Methoden `.load()` und `.unload()`, die lediglich als Binder, aber nicht als Trigger arbeiten können.

4.6.6 Shortcut-Methoden für bestimmte Eventtypen

Natürlich könnten Bindungen für jede Art von Event durch allgemeine Methoden wie `.bind()` erfolgen. Da die Nennung des Eventtyps hierbei jedoch ein wenig Aufwand bedeutet, bietet jQuery für alle Eventtypen eigene, spezialisierte Bindungsmethoden an, die ihren Namen aus dem jeweiligen Event ableiten. So bindet die Methode `.click()` einen Click-Event, die Methode `.submit()` einen Submit-Event.

Vorteil der spezialisierten Methoden ist ihre leichtere Handhabung. Allerdings müssen die Elemente, an die die Bindung erfolgt, bereits im Dokument existieren. Für »vorweggenommene« Bindungen müssen Sie auf die hierfür vorgesehenen allgemeinen Methoden `.live()` und `.delegate()` zurückgreifen. Auch das Lösen von spezialisierten Methoden muss durch `.unbind()` erfolgen, da es (beispielsweise) keine Methode wie `.unclick()` als Pendant zu `.click()` gibt.

Binden von Mouse-Events

Wollen Sie einen beliebigen Mouse-Event binden, können Sie dies natürlich immer per `.bind()` bewirken. Ähnlich wie im DOM Level 1 gibt es in jQuery allerdings auch hierfür Shortcuts, die den Eventtyp bereits festlegen. In diesen Fällen ist nur noch die Handler-Funktion zu übergeben.

Methode	Erläuterung
<code>.click(fn(e))</code> <code>.click()</code>	Bindet einen Handler <code>fn</code> an den Click-Event. <code>fn</code> bekommt das Eventobjekt übergeben. Alternativ: Triggert den Click-Event.

Tabelle 4.31 Methoden zum Binden von Mouse-Events

Methode	Erläuterung
<code>.dblclick(fn(e))</code> <code>.dblclick()</code>	Bindet einen Handler <code>fn</code> an den Dblclick-Event. <code>fn</code> bekommt das Eventobjekt übergeben. Alternativ: Triggert den Dblclick-Event.
<code>.mousedown(fn(e))</code> <code>.mousedown()</code>	Bindet einen Handler <code>fn</code> an den Mousedown-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Mousedown-Event.
<code>.mouseenter(fn(e))</code> <code>.mouseenter()</code>	Bindet einen Handler <code>fn</code> an den Mouseenter-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Mouseenter-Event.
<code>.mouseleave(fn(e))</code> <code>.mouseleave()</code>	Bindet einen Handler <code>fn</code> an den Mouseleave-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Mouseleave-Event.
<code>.mousemove(fn(e))</code> <code>.mousemove()</code>	Bindet einen Handler <code>fn</code> an den Mousemove-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Mousemove-Event.
<code>.mouseout(fn(e))</code> <code>.mouseout()</code>	Bindet einen Handler <code>fn</code> an den Mouseout-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Mouseout-Event.
<code>.mouseover(fn(e))</code> <code>.mouseover()</code>	Bindet einen Handler <code>fn</code> an den Mouseover-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Mouseover-Event.
<code>.mouseup(fn(e))</code> <code>.mouseup()</code>	Bindet einen Handler <code>fn</code> an den Mouseup-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Mouseup-Event.

Tabelle 4.31 Methoden zum Binden von Mouse-Events (Forts.)

Ein Click-Event wird über die `.click()`-Methode gebunden:

```
$('#inhalt p').click(function() {
    alert('Der Absatz wurde angeklickt.');
```

Dies ist eigentlich nichts anderes als eine Abkürzung für:

```
$('#inhalt p').bind('click', function() {
    alert('Der Absatz wurde angeklickt.');
```

Ohne Argument aufgerufen, triggert die Methode den jeweilige Event.

Binden von Keyboard-Events

Ebenso wie für Mausereignisse gibt es Convenience-Methoden zur Bindung von Keyboard-Events. Die Verwendung ist analog zu der für Mouse-Events.

Methode	Erläuterung
<code>.keydown(fn(e))</code> <code>.keydown()</code>	Bindet einen Handler <code>fn</code> an den Keydown-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Keydown-Event.
<code>.keypress(fn(e))</code> <code>.keypress()</code>	Bindet einen Handler <code>fn</code> an den Keypress-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Keypress-Event.
<code>.keyup(fn(e))</code> <code>.keyup()</code>	Bindet einen Handler <code>fn</code> an den Keyup-Event. <code>fn</code> bekommt das Eventobjekt übergeben. Alternativ: Triggert den Keyup-Event.

Tabelle 4.32 Methoden zum Binden von Keyboard-Events

Ein Keydown-Event-Handler wird an ein Formular gebunden. Die Funktion nimmt das Eventobjekt entgegen und prüft, ob die `Enter`-Taste betätigt wurde. In diesem Fall wird das Defaultverhalten, nämlich das Abschicken der Formular-daten, unterbunden. Der User muss hierfür den SUBMIT-Button einsetzen:

```
$('form').keydown(function(e) {
    if (e.which == '13') {
        event.preventDefault();
    }
})
```

Binden von Fenster-Events

Ereignisse in Zusammenhang mit dem Browserfenster können ebenfalls erfasst werden. Dies sind das Scrollen oder auch eine Größenveränderung des Fensters sowie das Entladen des aktuellen Dokuments. Auch für diese Ereignistypen existieren Bindungsmethoden.

Methode	Erläuterung
<code>.load(fn(e))</code>	Bindet eine Handler-Funktion <code>fn</code> an den Load-Event. Die Funktion bekommt das Eventobjekt <code>e</code> übergeben. Anmerkung: <code>.load()</code> (Ajax) und <code>.load()</code> (Event) werden nur anhand ihrer Parameter unterschieden.

Tabelle 4.33 Methoden zum Binden fensterbezogener Events

Methode	Erläuterung
<code>.resize(fn(e)),</code> <code>.resize()</code>	Bindet einen Handler <code>fn</code> an den Resize-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Resize-Event.
<code>.scroll(fn(e))</code> <code>.scroll()</code>	Bindet einen Handler <code>fn</code> an den Scroll-Event <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Scroll-Event.
<code>.unload(fn(e))</code>	Bindet eine Handler-Funktion <code>fn</code> an den Unload-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben.

Tabelle 4.33 Methoden zum Binden fensterbezogener Events (Forts.)

Eventbinder `.load()`

jQuery stellt für den Load-Event einen eigenen Eventbinder namens `.load()` zur Verfügung, der die ihm übergebene Callback-Funktion aufruft, sobald das entsprechend überwachte Objekt (dies kann das Dokument selbst sein, aber auch andere Objekte, wie beispielsweise Bilder) vollständig geladen wurde:

```
$('img.illustration').load( function() {
    alert("Illustration fertig geladen.");
});
```

Achtung – Namensgleichheit zur Ajax-Methode `.load()`

Eine Namensgleichheit, aber eigentlich keine Verwechslungsgefahr besteht zur Methode `.load()`, die per Ajax eine Ressource ins Dokument lädt. Jene Methode besitzt eine vollständig andere Signatur (Zahl und Art der Argumente), sodass jQuery stets »erkennt«, welcher Aufruf gemeint ist.

Eventbinder `.resize()`

Der Resize-Event kann direkt an das `window`-Objekt gebunden werden. Hier wird er dazu verwendet, die jeweils neue Fenstergröße in `div#box1` auszugeben. In diesen werden beim Laden des Dokuments zunächst die in diesem Augenblick gültigen Fensterabmessungen geschrieben:

```
$(document).ready(function() {

    $('#box1').text('Fenstergröße: '
        + $(window).width() + 'px, '
        + $(window).height() + 'px.');
```

```
$(window).resize(function() {
    $('#box1').text('Fenstergröße: '
```

```

        + $(window).width() + 'px, '
        + $(window).height() + 'px. ');
    });

});

```

Eventbinder .scroll()

Der Scroll-Event kann in beliebigen Containern mit Scrollbalken aufgefangen werden. Der häufigste Fall wird jedoch das Scrollen des Fensters selbst sein. Hier wird der Scroll-Event erkannt und die Scrollposition mithilfe der Methode `.scrollTop()` in ein Div mit fixierter Position geschrieben (die erste Anweisung füllt das Dokument mit Textabsätzen, die zweite schreibt die Startposition in das Div:

```

$(document).ready(function() {
    for (var i=0;i<100;i++) {
        $('<p>Textabsatz</p>').appendTo(document.body);
    }

    $('#box1').text('Scrollposition: '
        + $(window).scrollTop() + 'px von oben. ');

    $(window).scroll( function() {
        $('#box1').text('Scrollposition: '
            + $(window).scrollTop() + 'px von oben. ');
    });
});

```

Eventbinder .unload()

Dieser Eventbinder stellt das Gegenteil des `.load()`-Eventbinders dar. Seine Call-back-Funktion feuert, wenn die beobachtete Resource entladen (also aus dem Fenster entfernt) wird. Sinnvollerweise setzen Sie `.unload()` entweder auf das Dokument selbst oder auf das `window`-Objekt an:

```

$(window).unload(function() {
    // Versendet ein Formularobjekt beim Verlassen der Seite:
    $('form#tracking').submit();
});

```

4.6.7 Binden sonstiger Events

Eine eigene Eventgruppe bildet das Error-Ereignis, das zum Erfassen von Fehlern im Zusammenhang mit dem Laden von Ressourcen (beispielsweise Bildern) oder von Scriptfehlern verwendet wird. Es wird über die `.error()`-Methode gebunden.

Methode	Erläuterung
<code>.error(fn(e))</code>	Bindet eine Handler-Funktion <code>fn</code> an das Error-Objekt. Der Funktion wird das Eventobjekt übergeben. Anmerkung: Die Methode entspricht <code>.bind('error', fn)</code> .

Tabelle 4.34 Methode zum Binden des Error-Events

4.7 Inhalte, Attribute, Datenspeicher

Das DOM bietet zum Auslesen oder Überschreiben des Inhalts eines Knotens mehrere Ansätze. Handelt es sich um einen Textknoten, kann entweder das `nodeValue` oder das `data-Property` des Knotens verwendet werden. Ein Attributwert hingegen lässt sich über `getAttribute()` (oder ebenfalls `nodeValue`) auslesen und mit `setAttribute()` schreiben.

Komplizierter ist es mit Elementknoten – hier nämlich gibt die `nodeValue`-Eigenschaft den Wert `null` zurück, egal, ob Inhalte vorhanden sind oder nicht: Das DOM definiert, dass ein Element entweder weitere Elemente oder aber Textknoten enthält, deren Werte dann einzeln ausgelesen werden müssen.

Eine Eigenschaft, die den gesammelten Textinhalt eines Elements ausliest, fehlt (wie gesagt, ist laut DOM der Text im Textknoten, nicht im Element!) – immerhin existieren hierfür nicht standardisierte Eigenschaften wie `innerHTML`, `innerText` oder `textContent`, die allerdings browserspezifisch sind.

Auf dieser etwas disparaten Sachlage baut jQuery ein Gebäude aus Methoden auf, die das Auslesen und Manipulieren von Inhalten erstens vereinfachen und zweitens auch die browsertypischen Eigenheiten berücksichtigen. Es ist daher nicht erforderlich, dass Sie sich mit den DOM-Eigenschaften selbst befassen, solange Sie die infrage kommenden Knoten in einer jQuery-Collection zur Verfügung haben.

4.7.1 Lesen, Ändern und Entfernen von Attributen

Methode	Erläuterung
<code>.attr(name)</code>	Holt den Wert des bezeichneten Attributs <code>name</code> des ersten Elements der Collection.
<code>.attr(name, val)</code>	Schreibt den Wert <code>val</code> in das bezeichnete Attribut <code>name</code> des ersten Elements der Collection.

Tabelle 4.35 jQuery-Methoden zur Manipulation von Attributen

Methode	Erläuterung
<code>.removeAttr(att)</code>	Entfernt das bezeichnete Attribut <code>att</code> aus allen Elementen der aktuellen Collection.

Tabelle 4.35 jQuery-Methoden zur Manipulation von Attributen (Forts.)

Methode `.attr()`

Sie können nicht nur `class`-Attribute hinzufügen, sondern beliebige Attribute. Mit der allgemeinen Methode `.attr()` können Sie auch Attribute abfragen, und um noch mehr Verwirrung zu stiften, Sie könnten mittels `.attr()` selbstverständlich auch `class`-Attribute hinzufügen. Sie können mit der Methode `.attr()` auch lesend auf ein Attribut zugreifen, diese Möglichkeiten wollen wir uns nun kurz einmal ansehen:

Der entsprechende HTML-Abschnitt:

```
<a href="#" title="Titel des Links">Link</a>
```

Und mit folgendem jQuery-Code greifen Sie lesend darauf zu:

```
$(document).ready(function() {
    $("a").click(function() {
        alert($(this).attr("title"));
    });
});
```

Wenn Sie den Wert eines Attributs setzen wollen, fügen Sie den Wert als zweiten Parameter hinzu, betrachten das vorige Beispiel und fügen folgende Zeile hinzu:

```
$(document).ready(function() {
    $("a").attr("title","Der neue Titel des Links");
    $("a").click(function() {
        alert($(this).attr("title"));
    });
});
```

Wenn Sie mehrere Attribute übergeben wollen, können Sie auch mehrere Attributwertpaare setzen:

```
$("a").attr({
    title : "Der Titel",
    alt : "Der Alternativtext",
});
```

Und wie bei anderen jQuery-Methoden können Sie auch hier eine anonyme Funktion übergeben:

```

$("a").attr("title",function() {
    return "Der Titel mit dem Index:" + $(this).index();
});

```

Methode `.removeAttr()`

Sie können Attribute mittels `.removeAttr()` aus einem Element entfernen. Ein allgemeines Beispiel:

```

$("a").removeAttr("title");

```

Das Attribut als Parameter müssen Sie in jedem Fall vergeben.

4.7.2 Manipulation von Text- und Elementinhalt

Methode	Erläuterung
<code>.html()</code>	Gibt den HTML-Inhalt des ersten Elements der aktuellen Collection zurück (entspricht <code>innerHTML</code>).
<code>.html(htmlString)</code>	Ersetzt den HTML-Inhalt aller Elemente der aktuellen Collection durch den übergebenen HTML-String <code>htmlString</code> .
<code>.html(fn(i, oHtml))</code>	Ersetzt den HTML-Inhalt aller Elemente durch den Rückgabewert der Funktion <code>fn</code> , die den Index <code>i</code> in der Collection und den alten Inhalt <code>oHtml</code> jedes Elements als Parameter erhält.
<code>.text()</code>	Gibt die in Dokumentreihenfolge verketteten Textknoten der Elemente der Collection und ihrer Kindelemente als Textstring zurück.
<code>.text(txt)</code>	Setzt den Textinhalt der Elemente der Collection auf den übergebenen Textstring <code>txt</code> .
<code>.text(fn(i, txtAlt))</code>	Nimmt eine Funktion <code>fn</code> entgegen, die den Textinhalt als Rückgabewert liefert. Die Funktion bekommt den Collection-Index <code>i</code> und den alten Textinhalt <code>txtAlt</code> übergeben.
<code>.empty()</code>	Entfernt den Inhalt (d.h. alle Kindknoten) aus den Elementen der Collection.

Tabelle 4.36 jQuery-Methoden zur Manipulation von Inhalten

Methode `.html()`

Mittels der Methode `.html()` können Elemente samt Inhalt gelesen und geschrieben werden. (Im Extremfall könnten Sie mit dieser Methode aus einer komplett leeren Seite, die nichts ein leeres Grundgerüst enthält – und natürlich die `<script>`-Tags –, komplett dynamisch eine inhaltsreiche Seite zaubern.)

Fangen wir aber mit dem Lesen von Elementen an. Ohne Parameter gibt `.html()` den Inhalt des Selektors zurück:

```
$(document).ready(function() {
    alert($(".container").html());
});
```

Dies würde den kompletten Inhalt des Containers `<div id="container">...</div>` als String zurückgeben und in einer Alert-Box anzeigen.

Wichtig

Enthält der Selektor mehr als ein Element, wird nur das erste gefundene Element zurückgegeben.

Setzen Sie einen String als Parameter, den Sie der Methode übergeben, in das Element, wird dieser String in das Element geschrieben:

```
$(document).ready(function() {
    $(".container").html("<div><p>Ein Absatz</p></div>");
});
```

Wenn bereits Elemente und Inhalt in dem Container enthalten waren, werden diese komplett überschrieben. Im Gegensatz zu `.append()` oder `.prepend()` ersetzt diese Methode sämtliche eingeschlossenen Elemente.

Sie können `.html()` (wie auch der im Anschluss vorgestellten Methode `.text()`) ebenfalls eine anonyme Funktion als Parameter übergeben. Hier ein kurzes Beispiel:

```
$(document).ready(function() {
    $('.target').html(function() {
        var str = '<strong>Diese Seite enthält '
            + $('p').length + ' Absätze</strong>';
        return str;
    });
});
```

Dieses Beispiel durchsucht das Dokument nach allen `<p>`-Elementen. Mit `.length` erhalten Sie die Anzahl der Elemente. Das Ergebnis wird an das Element mit dem ID `target` angefügt.

Methode `.text()`

Im Gegensatz zu `.html()` gibt die Methode `.text()` nur den Textinhalt eines Elements, d.h. ohne Kindelemente, zurück. Da der Rückgabewert ein String ist, ist

die Methode terminierend, also nicht verkettbar. Nehmen Sie also den HTML-Code

```
<div class="container">
  <p>Absatz</p>
  <p>Ein weiterer Absatz</p>
</div>
<div class="target">
</div>
```

und wenden folgende jQuery-Anweisung an:

```
$(document).ready(function() {
  $(".target").append($(".container").text());
});
```

Sie sehen, wenn Sie den generierten Quelltext betrachten, dass die Anweisung nur den Textinhalt ohne umgebenden HTML-Code zurückgibt:

```
<div class="container">
  <p>Absatz</p>
  <p>Ein weiterer Absatz</p>
</div>
<div class="target">
  Absatz
  Ein weiterer Absatz
</div>
```

Hier wird mittels des Ausdrucks `$(".container").text()` ein String erzeugt und in ein Zielelement geschrieben. Interessant mag sein, dass auf den String alle Stringmethoden von JavaScript direkt anwendbar sind, beispielsweise die Methode `replace()`:

```
// Achtung: replace() ist eine Stringmethode:
$(".container").text().replace(/weiterer/ig, 'anderer')
```

Wenden Sie dies in Zusammenhang mit dem vorigen Beispiel an, erhalten Sie:

```
$(document).ready(function() {
  $(".target")
    .append($(".container")
      .text()
      // Stringmethode:
      .replace(/weiterer/ig, 'anderer'));
});
```

Und als Ergebnis:

```
<div class="container">
  <p>Absatz</p>
  <p>Ein weiterer Absatz</p>
</div>
<div class="target">
  Absatz
  Ein anderer Absatz
</div>
```

Der Methode `.text()` kann, genauso wie `.html()`, auch wieder ein Stringparameter übergeben werden. Was aber passiert, wenn hierbei ein HTML-String übergeben wird, wie im folgenden Beispiel?

```
$(document).ready(function() {
  $(".target").text("<p>Ein Absatz</p>");
});
```

Der generierte Quelltext zeigt, dass die Tag-Klammern der HTML-Elemente durch Entitäts ersetzt werden. Es werden keine Kindelemente erzeugt:

```
<div class="target">&lt;p&gt;Ein Absatz&lt;/p&gt;</div>
```

Methode `.empty()`

Ähnlich verhält es sich mit der Methode `.empty()`, die alle Inhalte eines selektierten Elements löscht. Ausgehend vom üblichen HTML-Gerüst

```
<div class="container">
  <p class="target">Absatz</p>
  <p>Ein weiterer Absatz</p>
</div>
```

und mit den Zeilen

```
$(document).ready(function() {
  $(".target").empty();
});
```

erhalten Sie folgendes Ergebnis:

```
<div class="container">
  <p class="target"></p>
  <p>Ein weiterer Absatz</p>
</div>
```

Es werden alle Inhalte und Elemente innerhalb des selektierten Elements gelöscht. Würden Sie innerhalb von `<p class="target">` weitere verschachtelte

DOM-Elemente vorfinden, würden auch diese gelöscht. (Im Gegensatz dazu entfernt `.remove()` alles, *inklusive* des selektierten Elements.)

Der Methode `.empty()` kann kein Parameter übergeben werden, der die Selektion weiter einschränkt. Aber genauso wie bei `.remove()` werden alle gebunden Events und `.data()` gelöscht.

4.7.3 Daten in Objekten speichern

Mit der Methode `.data()` können Daten an ein jQuery-Objekt gebunden werden. Analog dient `.removeData()` dazu, derart gebundene Daten wieder zu entfernen.

Methode	Erläuterung
<code>.data(key, val)</code> <code>.data(obj)</code> <code>.data(key)</code> <code>.data()</code>	<i>Speichert</i> den Wert <code>val</code> unter dem Namen <code>key</code> an die Elemente der aktuellen Collection, nimmt alternativ ein Objekt <code>obj</code> aus Key-Value-Pärchen entgegen. <i>Holt</i> aus der aktuellen Collection den Wert des übergebenen <code>key</code> (wenn kein Wert <code>val</code> übergeben wurde oder alle Key-Value-Pärchen als Objekt <code>obj</code>, wenn kein Parameter übergeben wurde).
<code>.removeData()</code> <code>.removeData(name)</code>	Entfernt <i>alle</i> durch jQuery an die Elemente der Collection gebundenen Datenwerte. Optional: Wird ein Bezeichner <code>name</code> übergeben, werden nur die entsprechend benannten Datenwerte entfernt.

Tabelle 4.37 jQuery-Methoden zur Datenbindung und -lösung

.data(name), .data(name,value)

Mit `$(this).data("info","Der Index ist ...")` wird ein Wert unter dem Schlüsselbezeichner `info` an das aktuell verarbeitete Element der Collection (ein `<p>`-Container) gebunden. In der an den Click-Event desselben Elements gebundenen Funktion wird durch Aufruf des Schlüssels `$(this).data("info")` der gebundene Wert mit `$(this).append()` an den Elementinhalt angehängt:

```
$(document).ready(function() {
    var obj = $("#box p");
    obj.each(function() {
        // Daten binden (key, val):
        $(this).data("info"," Der Index ist: " +
            obj.index(this));
    });
    $("#box p").click(function() {
        // Daten auslesen (key)
```

```

        $(this).append($(this).data("info"));
    });
});

```

Aus dem Beispielcode

```

<div id="box">
    <p>Eins.</p>
    <p>Zwei.</p>
    <p>Drei.</p>
    <p>Vier.</p>
</div>

```

wird nach (einmaligem) Anklicken jedes Absatzes:

```

<div id="box">
    <p>Eins. Der Index ist: 0</p>
    <p>Zwei. Der Index ist: 1</p>
    <p>Drei. Der Index ist: 2</p>
    <p>Vier. Der Index ist: 3</p>
</div>

```

Enthält die jQuery-Collection, auf die `.data()` angewendet wird, mehrere Elemente, so werden die Daten nur an das erste Element gehängt.

Anmerkung

Nicht nur Metainformationen lassen sich an ein DOM-Element binden, auch über Ajax lassen sich Daten laden und an ein Element hängen, um später abgerufen zu werden. Die Erweiterung jQuery UI macht ausgiebig Gebrauch von diesem Ansatz.

.removeData(name)

Mit dieser Methode lassen sich gebundene Daten wieder entfernen:

```

// an p#p1 werden Daten unter zwei Schlüsseln gebunden:
$("p#p1").data("info","Wert");
$("p#p1").data("info2","Wert2");

```

Ein pauschaler Aufruf von `removeData()` entfernt Daten von allen `<p>`-Containern. Hierdurch verliert `p#p1` beide gebundenen Informationen:

```

// alle gebundenen Daten werden entfernt:
$("p").removeData();

```

Möchten Sie subtiler arbeiten, können Sie den Schlüsselbezeichner der zu entfernenden Information übergeben. Nach folgendem Aufruf besitzt `p#p1` noch die mit `info2` benannten Daten:

```
// nur unter Key 'info' gebundene Daten werden entfernt:
$("p").removeData("info");
```

Data-Utilities – jQuery.data() und jQuery.removeData()

Passend zur jQuery-Methode `.data()`, die Daten an DOM-Objekte bindet oder sie ausliest, existieren Utility-Methoden, die gleichfalls Daten lesen oder lösen können. Da dies ausgehend vom jQuery-Objekt geschieht (das also Kontextobjekt ist), muss das nach Daten zu untersuchende Objekt als Argument übergeben werden. Eine Datenbindung ist auf diesem Wege nicht möglich – dies geschieht ausschließlich für jQuery-Collections durch die `.data()`-Methode.

Utility	Erläuterung
<code>jQuery.data(obj)</code>	Liest alle gebundene Daten aus einem übergebenen Objekt <code>obj</code> .
<code>jQuery.data(obj, key)</code>	Liest gebundene Daten mit dem Schlüssel <code>key</code> aus einem übergebenen Objekt <code>obj</code> .
<code>jQuery.removeData()</code>	Entfernt ein jQuery-Datenobjekt.

Tabelle 4.38 jQuery-Utilities für Datenbindungen an Objekte

4.8 Formulare verarbeiten mit jQuery

Zur Verarbeitung von Formulardaten bietet jQuery einige Erleichterungen, die einerseits Selektoren, andererseits Event Handler und drittens Methoden zur Vorverarbeitung der eingegebenen Daten umfassen.

4.8.1 Filterausdrücke für Formularelemente

Formularselektoren und `-filter` lassen sich sinnvoll einsetzen, um Formulareingaben zu validieren und die Eingabemöglichkeiten benutzerfreundlich zu gestalten. Sie erhalten mit diesen Selektoren Zugriff auf alle Formularelemente, um ihre Werte zu validieren, bestimmte Elemente in Abhängigkeit von anderen ein- oder auszublenden oder bei bestimmten Benutzeraktionen hervorzuheben.

Filter für Input-Elemente

Selektor	Typ	Erläuterung
<code>:button</code>	Formulare	Selektiert <code><button></code> und <code><input type="button"></code> .
<code>:checkbox</code>	Formulare	Selektiert <code><input type="checkbox"></code> .

Tabelle 4.39 jQuery-Formular-Filterselektoren (Inputelemente)

Selektor	Typ	Erläuterung
:file	Formulare	Selektiert <code><input type="file"></code> .
:image	Formulare	Selektiert <code><input type="image"></code> .
:input	Formulare	Selektiert alle Formularinputelemente (inklusive Textareas, Selektfelder und Buttons).
:password	Formulare	Selektiert <code><input type="password"></code> .
:radio	Formulare	Selektiert <code><input type="radio"></code> .
:reset	Formulare	Selektiert <code><input type="reset"></code> .
:submit	Formulare	Selektiert <code><input type="submit"></code> .
:text	Formulare	Selektiert <code><input type="text"></code> .

Tabelle 4.39 jQuery-Formular-Filterselektoren (Inputelemente) (Forts.)

Für jeden Typ von Eingabeelement existiert in jQuery ein spezieller Filterselektor, der nach der allgemeinen Regel `:typename` benannt ist. Ausgenommen sind das Element `<select>` und die `<textarea>`, für die Sie herkömmliche Selektoren einsetzen müssen, wenn Sie sie gezielt erreichen wollen. Ansonsten lassen sie sich (dann aber ununterscheidbar von den »echten« Elementen vom Typ `<input>`) über den allgemeinen Filterselektor `:input` erreichen.

So selektieren Sie mit `:button` sowohl ein `<button>`-Element als auch `<input type="button"/>`. (Selbstverständlich können Sie auch einen Attributselektor und einen Typselektor kombinieren: `$('input[type=button], button')`.)

Analog selektiert `$("input:checkbox")` Elemente vom Typ `<input type="checkbox"/>` oder `$("input:file")` den File-Upload-Input `<input type="file"/>`.

Besser explizit mit Elementtyp

Selbstverständlich könnten Sie zur Selektion `$(":checkbox")` schreiben, und erhielten Ihr erwartetes Ergebnis. Dahinter steckt aber eigentlich der Ausdruck `$("*:checkbox")`. Das bedeutet, mittels des Universalselektors wird der gesamte DOM-Baum durchsucht. Da aber klar ist, dass ein *Input* gesucht wird, schreiben Sie besser (und performanter) direkt: `$("input:checkbox")`. Dies trifft auf alle Pseudoklassen zu, die hier vorgestellt werden.

:input etc. – Filter für Inputelemente

Sehr praktisch ist der *allgemeine Filterausdruck* `:input`, der aber nicht lediglich Elemente `<input>` selektiert (dies wäre der Selektorausdruck `"input"`, ohne vorangehenden Doppelpunkt), sondern alle in Formularen als Eingabeelemente funktierenden Elementtypen.

In folgendem Beispielformular sind insgesamt 14 Eingabeelemente versammelt:

```
<form id="formular" action="#">
  Textfeld:
  <input type="text"
    name="t1" id="t1"/><br/>
  Passwortfeld:
  <input type="password"
    name="text" name="p1" id="p1"/><br/>

  3 Checkboxes:
  <input type="checkbox"
    name="cb" id="cb1" value="cb1"/>
  <input type="checkbox"
    name="cb" id="cb2" value="cb1"/>
  <input type="checkbox"
    name="cb" id="cb3" value="cb1"/><br/>

  3 Radiobuttons:
  <input type="radio"
    name="rb" id="rd1" value="rd1"/>
  <input type="radio"
    name="rb" id="rd2" value="rd2"/>
  <input type="radio"
    name="rb" id="rd3" value="rd3"/><br/>

  (1 Verstecktes Feld)
  <input type="hidden"
    name="hd1" id="hd1" value="hd1"/><br/>

  Textarea:
  <textarea name="ta1" id="ta1"></textarea><br/>

  Selectbox:
  <select name="s11" id="s11">
    <option value="op1">Option 1</option>
    <option value="op2">Option 2</option>
  </select><br/>

  3 Buttons:
  <input type="reset" value="Zurücksetzen"/>
  <input type="submit" value="Abschicken" />
  <input type="button" value="Noch ein Button"/>
</form>
```

Wenden Sie folgenden Ausdruck an

```
$(document).ready(function() {

    // selektiert Eingabefelder jeden Elementtyps:
    alert($('#formular :input').length); //-> 14

});
```

gibt die Message-Box den Wert 14 aus, was der Anzahl der Eingabeelemente entspricht (das versteckte Feld wird mitgezählt).

Vorsicht jedoch – begrenzen Sie die Wirkung von `:input` auf den Elementtyp `<input>`, werden Selectbox und Textarea (wie eigentlich auch zu erwarten) ausgefiltert:

```
// Typselektor beschränkt auf Elementtyp <input>:
alert($('#formular input:input').length); //-> 12
```

Merke

Dem Filter `:input` sollten Sie besser keinen Typselektor hinzufügen.

Weitere Filter dienen dazu, gezielt Eingabefelder eines bestimmten Untertyps herauszugreifen: `:text`, `:password`, `:radio`, `:checkbox`, `:submit`, `:image`, `:reset`, `:file` und `:button`.

So beispielsweise erhalten die Anzahl der Checkboxes

```
// suboptimal: Filter ohne Typselektor:
alert($('#formular :checkbox').length); //-> 3
```

wobei es diesmal ökonomischer ist (aus Sicht von jQuery jedenfalls), den Typselektor dem Filter beizugeben. Das Framework ist so nicht gezwungen, alle Elemente mit dem Filter zu überprüfen, wo es dies doch nur mit den Inputs zu machen bräuchte:

```
// besser: Filter mit Typselektor:
alert($('#formular input:checkbox').length); //-> 3
```

4.8.2 Filter für Zustände von Formularinputs

Interessant ist die Möglichkeit, einen Input anhand seines aktuellen *Zustands* auswählen zu können. Hierzu dienen die Zustandsfilter, die in der folgenden Tabelle gelistet sind.

Selektor	Erläuterung
:checked	Selektiert alle Inputs, deren checked-Property true ist (Radiobuttons und Checkboxes).
:disabled	Selektiert alle Formularelemente, deren disabled-Property den Wert true hat.
:enabled	Selektiert alle Formularelemente, deren enabled-Property den Wert true hat.
:selected	Selektiert alle Option-Elemente, deren selected-Property den Wert true hat (für Radiobuttons, Checkboxes siehe :checked).

Tabelle 4.40 jQuery-Formular-Filterselektoren (Zustände von Inputelementen)

Filter :enabled und :disabled

Allgemein reagieren die Filterausdrücke :disabled und :enabled auf den Wert des disabled-Property, der durch JavaScript gesetzt worden sein kann oder über das disabled-Attribut festgelegt wurde.

Für das folgende Beispiel wird ein einfaches Formular eingesetzt. Das Textinputfeld soll über zwei Buttons alternativ aktiviert und deaktiviert werden. Der aktuelle Zustand wird in einen (anfangs leeren) Textabsatz geschrieben:

```
<form>
  <input type="text" name="" id="inpBeispiel" value="Test"/>
  <input type="button" id="butDisable" value="Disable"/>>
  <input type="button" id="butEnable" value="Enable"/>
</form>
<p id="message"></p>
```

Um die Aktion durchzuführen, wird an jeden Button ein Click-Event-Handler gebunden, innerhalb dessen Callback-Funktion der Textinput jeweils aktiviert bzw. deaktiviert wird (alles innerhalb des .ready()-Blocks):

```
// Deaktivieren
$('#butDisable').click( function(){
    if($('#inpBeispiel').is(':enabled')) {
        $('#inpBeispiel').attr('disabled', 'disabled');
    }
    $('#message').html('Textinput inaktiv geschaltet.');
```

```
});
$('#butEnable').click( function(){
    if($('#inpBeispiel').is(':disabled')) {
        $('#inpBeispiel').removeAttr('disabled');
```

```
    }
    $('#message').html('Textinput aktiv geschaltet.');
```

```
});
```

Das Deaktivieren geschieht über das Hinzufügen des `disabled`-Attributs, was Sie mit der Methode `.attr()` bewirken. Für das Reaktivieren wird das Attribut über `.removeAttr()` wieder entfernt. Um diese Aktionen nicht überflüssigerweise vorzunehmen, prüfen Sie vorher mit einer Bedingung, ob der Input möglicherweise bereits im angestrebten Zustand ist. Hierfür werden die Zustandsfilter `:disabled` und `:enabled` von jQuery in Zusammenhang mit der Methode `.is()` eingesetzt, die einen Booleschen Wert zurückgibt:

```
// true, wenn der Input disabled ist:
$('#inpBeispiel').is(':disabled')
// true, wenn der Input enabled ist:
$('#inpBeispiel').is(':enabled')
```

Achtung

Beachten Sie, dass die Werte deaktivierter Felder beim Datenversand nicht berücksichtigt werden!

Filter `:checked`

Für *Checkboxes* und *Radiobuttons* ist es außerdem von Bedeutung, zu erfahren, ob diese durch den User ausgewählt wurden, um entsprechend zu reagieren. Hierzu findet der Filter `:checked` Anwendung, der das `checked`-Property als Kriterium nutzt.

Im folgenden Beispiel soll sich der User entscheiden, ob er für seine Bestellung eine Pizza mit allen verfügbaren Belägen ordert oder eine Einzelauswahl bevorzugt. Die Wahl zwischen Einzelauswahl und Gesamtauswahl wird über zwei Radiobuttons vorgenommen. In Abhängigkeit von der Wahl werden Checkboxes für die Beläge selektiert.

Hier der relevante Ausschnitt aus dem HTML:

```
<form>
<fieldset>
<legend>Auswahl</legend>
  <input type="radio" name="auswahl"
    id="a1" /> Alles<br/>
  <input type="radio" name="auswahl"
    id="a2" checked="checked"/> Einzelauswahl<br/>
</fieldset>

<fieldset>
<legend>Belag</legend>
  <input type="checkbox" name="belag"
```

```

        id="b1" />Tomaten<br/>
<input type="checkbox" name="belag"
        id="b2" />Zwiebeln<br/>
<input type="checkbox" name="belag"
        id="b3" />Paprika<br/>
<input type="checkbox" name="belag"
        id="b4" />Sardellen<br/>
<input type="checkbox" name="belag"
        id="b5" />Pilze
</fieldset>
</form>
<p id="message"></p>

```

Hier soll nun zweierlei erreicht werden. Beim Wählen der Option ALLES sollen automatisch alle Checkboxes für Pizzabelag auf CHECKED gesetzt werden. Es soll nur möglich sein, eine Checkbox individuell zu betätigen, wenn die Option EINZELAUSWAHL gewählt ist. Ein leerer Textabsatz dient zur Ausgabe einer Nachricht. Dies geschieht über folgende Funktionen (wiederum in einem `ready()`-Block enthalten):

```

// wenn Option "alles" gewählt wird:
$('#a1').click( function(){
    // alle Belag-Checkboxen auf "checked" setzen:
    $('input[type=checkbox]').attr('checked', 'checked');
    // Nachricht ausgeben:
    $('#message').html('Sie haben alle Beläge gewählt.');
```

Wenn explizit EINZELAUSWAHL gewählt wird, wird lediglich eine andere Nachricht ausgegeben. Der Zustand der Checkboxes bleibt unverändert:

```

$('#a2').click( function(){
    $('#message').html('Sie haben Einzelauswahl gewählt.');
```

Wenn eine Checkbox betätigt wird, wird zunächst geprüft, ob die Option ALLES gewählt ist. In diesem Fall soll zunächst auf EINZELAUSWAHL gewechselt werden (hierzu erfolgt eine Alert-Meldung). Das Abwählen der Checkbox wird durch das Statement `return false` in der Funktion verhindert:

```

// eine Checkbox wird betätigt:
$('input[type=checkbox]').click( function(){
    // ist die Option "alles" gewählt?
    if($('#a1').is(':checked')) {
        // ... dann Meldung ausgeben:
        alert("Bitte wählen Sie zunächst Einzelauswahl!");
        return false;
    }
});

```

```

        // ...und Aktion abbrechen:
        return false;
    }
});

```

Ist die Option ALLES nicht gewählt, lässt sich jede Checkbox problemlos an- und abwählen (wie es sich bei einer Einzelauswahl ja auch gehört).

Filter :selected

Für die Optionen innerhalb von *Auswahlboxen* `<select>` muss der `:selected`-Filter verwendet werden, da eine Option kein `checked`-Property besitzt. Um zu überprüfen, ob eine `<option>` angeklickt wurde, schreiben Sie also `$("option:selected")`. In folgendem Beispiel werden die Werte aller selektierten Optionen eines Auswahlfelds zu einem String zusammengestellt und in einen Textabsatz ausgegeben.

Zunächst wieder das zugrunde liegende HTML-Fragment:

```

<form id="meinFormular">
  <select name="blumen" id="blumen"
    multiple="multiple" size="5">
    <option value="Rosen">Rosen</option>
    <option value="Tulpen">Tulpen</option>
    <option value="Nelken">Nelken</option>
    <option value="Veilchen">Veilchen</option>
    <option value="Margeriten">Margeriten</option>
  </select>
  <input type="submit" value="Selektion ausgeben"/>
</form>
<p id="message"></p>

```

Für das Script (wieder im `.ready()`-Block) bilden Sie zunächst mit dem Ausdruck `$('select option:selected')` eine Collection aus allen selektierten Optionselementen. Über diese läuft eine `.each()`-Schleife, die über die Methode `.val()` den Wert extrahiert und in einem String zusammenfasst (ist der String nicht leer, d.h. enthält er nach dem ersten Schleifendurchlauf bereits einen Wert, werden noch ein Komma und ein Leerzeichen angefügt):

```

// Erinnerung: e ist das Eventobjekt:
$('#meinFormular').submit( function(e){
  // wir wollen keine Daten versenden:
  e.preventDefault();
  var auswahl = '';
  $('select option:selected').each(
    function(){

```

```

        if(auswahl) {
            auswahl += ", ";
        }
        auswahl += $(this).val();
    });

    $('#message').html('Sie haben ' + auswahl + ' gewählt.');
```

Die Methode `.val()`, die hier vorweggenommen wird, dient zum Extrahieren des Inhalts des `value`-Attributs eines Eingabeelements. Sie wird im weiteren Verlauf noch häufiger vorkommen.

4.8.3 Binden von Events an Formularelemente

Für Formulare und ihre Verarbeitung müssen bestimmte Ereignisse überwacht werden, die an Eingabefeldern auftreten oder beim Versenden der Formulardaten ausgelöst werden. Für diese Events bietet jQuery entsprechende Methoden zur Bindung an.

Methode	Erläuterung
<code>.blur(fn(e))</code> <code>.blur()</code>	Bindet einen Handler <code>fn</code> an den Blur-Event. <code>fn</code> bekommt das Eventobjekt übergeben. Alternativ: Triggert den Blur-Event (z. B. an einem Formular-input).
<code>.change(fn(e))</code> <code>.change()</code>	Bindet einen Handler <code>fn</code> an den Change-Event. <code>fn</code> bekommt das Eventobjekt übergeben. Alternativ: Triggert den Change-Event (z. B. an einem Formular-input).
<code>.focus(fn(e))</code> <code>.focus()</code>	Bindet einen Handler <code>fn</code> an den Focus-Event. <code>fn</code> bekommt das Eventobjekt übergeben. Alternativ: Triggert den Focus-Event (z. B. an einem Formular-input).
<code>.focusin(fn(e))</code>	Bindet einen Handler <code>fn</code> an den FocusIn-Event. <code>fn</code> bekommt das Eventobjekt übergeben. Anmerkung: Der FocusIn-Event geht dem Focus-Event unmittelbar voraus, wenn ein Element den Focus erhält.
<code>.focusout(fn(e))</code>	Bindet einen Handler <code>fn</code> an den FocusOut-Event. <code>fn</code> bekommt das Eventobjekt übergeben. Anmerkung: Der FocusOut-Event geht dem Blur-Event unmittelbar voraus, wenn ein Element den Focus verliert.

Tabelle 4.41 Methoden zum Binden formularbezogener Events

Methode	Erläuterung
<code>.select(fn(e))</code> <code>.select()</code>	Bindet einen Handler <code>fn</code> an den Select-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Select-Event.
<code>.submit(fn(e))</code> <code>.submit()</code>	Bindet einen Handler <code>fn</code> an den Submit-Event. <code>fn</code> bekommt das Eventobjekt <code>e</code> übergeben. Alternativ: Triggert den Submit-Event (z. B. an einem Formularobjekt).

Tabelle 4.41 Methoden zum Binden formularbezogener Events (Forts.)

Die Anwendung der Methoden ist im Prinzip selbsterklärend und kann der Tabelle 4.41 entnommen werden. Exemplarisch wird hier als Beispiel eine Funktion an den *Change*-Event eines Inputs gebunden und ausgeführt, sobald der User nach einer erfolgten Änderung den Cursor aus dem betreffenden Eingabefeld nimmt. Diese kleine Routine entfernt führenden und folgenden Whitespace aus Texteingaben (das hier verwendete Utility `$.trim()` wird später noch vorgestellt):

```
$(document).ready(function() {
    $('#input#i1').change( function() {
        // Manipuliert wird die DOM-Eigenschaft:
        this.value = $.trim(this.value);
    });
});
```

4.8.4 Serialisierung von Formulardaten

Die Daten, die in ein Formular eingegeben werden, werden beim Versand automatisch vom Browser in eine gebrauchsfertige Form umgewandelt. Dies kann jedoch auch abseits des regulären Versands erforderlich sein, beispielsweise, wenn Sie Formulardaten im Rahmen eines Ajax-Requests als Nutzdaten benötigen oder wenn Sie sie zur Konstruktion eines JSON-Objekts verwenden möchten. Für beide Fälle stellt jQuery eigene Methoden zur Verfügung.

Methode	Erläuterung
<code>.serialize()</code>	Serialisiert die Werte aus Formularelementen der aktuellen Collection als URL-encodierten Textstring.
<code>.serializeArray()</code>	Serialisiert die Werte aus Formularelementen der aktuellen Collection als JavaScript-Array. Anmerkung: Dient gewöhnlich als Vorstufe eines JSON-Objekts.

Tabelle 4.42 Methoden zur Serialisierung von Formulardaten

Die Methode `.serialize()`

Die Methode `.serialize()` dient dazu, die Nutzdaten des Formulars in Zusammenhang mit den `name`-Bezeichnern der Inputs, die die Daten enthalten, zu serialisieren. Hierbei werden nur die Daten derjenigen Inputs verwendet, die als *successful* gewertet werden,⁴ was bedeutet, dass sie

1. ein `name`-Attribut mit gültigem Bezeichner besitzen
2. nicht deaktiviert (*disabled*) sind
3. die `checked`-Eigenschaft auf `true` steht (für Radiobuttons und Checkboxes)
4. die `selected`-Eigenschaft auf `true` steht (für Optionen in Select-Feldern)

Diese Methode kann, als jQuery-Methode natürlich auf jede beliebige Collection angewendet werden, die aus Formularinputs besteht. Üblich ist es allerdings, sie direkt auf das betreffende Formularobjekt anzuwenden, das hierfür in eine Collection geladen wird:

```
$('#meinFormular').submit(function() {
    // Formulardaten serialisiert über Alert ausgeben:
    alert($(this).serialize());
    // Datenversand unterbinden:
    return false;
});
```

Die Ausgabe der `.serialize()`-Methode besteht in einem üblichen Querystring. Für das in einem der vorigen Beispiele verwendete Formular mit der Multi-Select-Box

```
<form id="meinFormular">
  <select name="blumen" id="blumen"
    multiple="multiple" size="5">
    <option value="Rosen">Rosen</option>
    <option value="Tulpen">Tulpen</option>
    <option value="Nelken">Nelken</option>
    <option value="Veilchen">Veilchen</option>
    <option value="Margeriten">Margeriten</option>
  </select>
  <input type="submit" value="Selektion ausgeben"/>
</form>
<p id="message"></p>
```

ergibt sich mit folgendem Script (auch hier wird wieder das Eventobjekt zur Unterbindung des eigentlichen Datenversands eingesetzt)

⁴ Mehr Informationen hierzu unter <http://www.w3.org/TR/html401/interact/forms.html#successful-controls>

```
$('#meinFormular').submit( function(e){
    e.preventDefault();
    var auswahl =$(this).serialize();
    $('#message').html('Sie haben '+ auswahl + ' gewählt.');
```

im Ausgabeabsatz ein Querystring wie

```
blumen=Rosen&blumen=Nelken&blumen=Margeriten
```

Achtung – Option stets mit value-Attribut verwenden

Die Optionen besitzen im Beispiel ein `value`-Attribut, sodass dessen Wert für die Konstruktion des Querystrings eingesetzt wird. Standardmäßig wird bei Abwesenheit des `value`-Attributs hierfür der Textinhalt des `<option>`-Containers verwendet. Eine Select-Box der Form

```
<select name="blumen" id="blumen"
    multiple="multiple" size="5">
    <option >Rosen</option>
    <option>Tulpen</option>
    <option>Nelken</option>
    <option>Veilchen</option>
    <option>Margeriten</option>
</select>
```

würde eigentlich das gleiche Ergebnis erzeugen. Vorsicht ist allerdings bei einigen Versionen des Internet Explorers geboten, der hier einen Fehler wirft. Als Faustregel sollte man grundsätzlich einem Option-Element ein `value`-Attribut zuweisen, auch wenn dessen Inhalt dem Textinhalt des Elements entspräche.

Die Methode `.serializeArray()`

Mit der `.serializeArray()`-Methode wird aus den Daten eines verarbeiteten Formularobjekts ein Array erzeugt, das seinerseits Objekte enthält, die jeweils ein Eingabeelement repräsentieren. Hierbei werden nur diejenigen Inputs berücksichtigt, die als *successful* angesehen werden.

Für das bereits bekannte Formular

```
<form id="meinFormular">
    <select name="blumen" id="blumen"
        multiple="multiple" size="5">
        <option value="Rosen">Rosen</option>
        <option value="Tulpen">Tulpen</option>
        <option value="Nelken">Nelken</option>
        <option value="Veilchen">Veilchen</option>
        <option value="Margeriten">Margeriten</option>
    </select>
    <input type="submit" value="Selektion ausgeben"/>
```

```
</form>
<p id="message"></p>
```

ergibt sich mit diesem Script

```
$('#meinFormular').submit( function(e){
    e.preventDefault();
    var auswahl =$(this).serializeArray();
    $.each(auswahl, function(i, item){
        $("#message").append(i +
            ". " + item.name +
            ": " + item.value + "; ");
    });
});
```

beispielsweise folgendes Ausgabeergebnis (drei Optionen gewählt):

0. blumen: Tulpen; 1. blumen: Nelken; 2. blumen: Veilchen;

Intern ist das durch `.serializeArray()` erzeugte Datenarray folgendermaßen organisiert – gehen wir dafür von den gewählten drei Optionen aus:

```
[{name:"blumen",value:"Tulpen"},
{name:"blumen",value:"Nelken"},
{name:"blumen",value:"Veilchen"}]
```

Es existiert also für jede gewählte Option (hätte das Formular mehr Elemente, würden Sie sehen, dass dies für jedes *erfolgreiche* Eingabefeld gilt) ein Objekt mit einem `name`- und einem `value`-Property. Dies ist die Grundlage für die Datenextraktion, die im Beispiel mittels einer `$.each()`-Schleife vorgenommen wird:

```
$.each(auswahl, function(i, item){
    $("#message").append(i +
        ". " + item.name +
        ": " + item.value + "; ");
});
```

4.8.5 Extraktion von Formularfeldwerten

Regelmäßig ergibt sich die Notwendigkeit, Einzelwerte von Eingabefeldern zu lesen oder zu schreiben. Dies könnte zwar über das `value`-Property des entsprechenden DOM-Knotens erfolgen, allerdings existiert dieses nicht für alle Eingabeelemente bzw. arbeitet nicht konsistent. Mit `.val()` stellt jQuery eine Methode zur Verfügung, die allen Erwartungen entspricht. Die Methode ist *terminierend*, da sie einen String (in manchen Fällen auch ein Array) zurückgibt. Sie steht daher stets am Ende einer jQuery-Kette.

Methode	Erläuterung
<code>.val()</code>	Die Methode wird vor allem für Formularinputs eingesetzt. Extrahiert den <code>value</code> des ersten Elements der aktuellen Collection.
<code>.val(wert)</code>	Setzt den <code>value</code> aller Elemente der aktuellen Collection auf den übergebenen Wert <code>wert</code> .
<code>.val(fn(i, oVal))</code>	Setzt den <code>value</code> aller Elemente der aktuellen Collection über den Rückgabewert der Funktion <code>fn</code> . Diese erhält den Collection-Index <code>i</code> und den alten Value <code>oVal</code> als Parameter.

Tabelle 4.43 Methoden zur Extraktion von Formularfeldwerten

Die Methode `.val()`

Die Methode `.val()` wird vorwiegend zum Auslesen des Wertes eines Eingabefelds eingesetzt, kann diesen bei Bedarf aber auch schreiben. Eine dritte Einsatzmöglichkeit von `.val()` besteht im Auswählen von Checkboxes, Radiobuttons oder Optionen eines Select-Felds. Aber nun besser der Reihe nach ...

Um den Wert eines Formularelements *auszulesen*, wird `.val()` einfach auf eine Collection angewendet, die aus dem gewünschten Eingabeelement besteht:

```
// holt den 'value' eines Inputs beliebigen Typs:
var meinWert = $("#meinInput").val();
```

Dies ist prinzipiell für alle Arten von Eingabeelementen möglich, wobei auf das `value`-Attribut zugegriffen wird (bzw. das `value`-Property, in dem sich der Userinput befindet). Der aus dem Input geholte Wert ist in allen Fällen ein *String*.

Es können sowohl statische Values (beispielsweise einer Checkbox oder eines Hidden Fields) ausgelesen werden wie auch dynamisch gesetzte Werte (wie in einem Textinput). Hierbei sind Sie jedoch nicht auf Elemente beschränkt, die explizit ein `value`-Attribut besitzen. Auch der Text in `<textarea>`-Elementen kann per `.val()` gelesen werden:

```
var textfeldWert = $('textarea').val();
```

Ist in der Collection mehr als ein Formularelement enthalten, wird nur der Wert des *ersten* von ihnen ausgelesen.

Bilden Sie aus folgenden drei Checkboxes eine Collection:

```
<input type="checkbox" name="cb" id="cb1" value="check1"/>
<input type="checkbox" name="cb" id="cb2" value="check2"/>
<input type="checkbox" name="cb" id="cb3" value="check3"/>
```

und wenden `.val()` unmittelbar an, erhalten Sie nur den Wert von `#cb1`:

```
alert($("#input[name=cb]").val()); // gibt check1 aus
```

Es bleibt also nur die Möglichkeit, die Collection Item für Item abzuarbeiten, was mit der `.each()-Methode` geht – hier werden in der Callback-Funktion die Werte per `alert()` ausgegeben:

```
$("#input[name=cb]").each( function() {
    // diese Collection hat nur ein Item:
    alert($(this).val());
})
```

Sie können also nicht per `.val()` auf alle Werte eines Checkbox-Verbunds gleichzeitig zugreifen. Anders ist die Lage bei Select-Feldern. Wird `.val()` auf ein Select-Element mit gesetzter `multiple`-Eigenschaft angewendet, gibt die Methode ein *Array* zurück, das die Werte aller selektierten Optionen enthält (es wird wieder das Formular aus dem Beispiel zu `.serialize()` verwendet):

```
$('#meinFormular').submit( function(e){
    e.preventDefault();
    // erzeugt ein Array:
    var auswahl = $('#blumen').val();
    // iteriert durch das Array:
    $.each(auswahl, function(i){
        var ausgabe = '';
        if(ausgabe) {
            ausgabe += ', ';
        }
        ausgabe += auswahl[i];
    });
    $("#message").append(ausgabe);
});
```

Eine Option ist immer unmittelbar eine der *Array-Methoden*, die JavaScript anbietet, um sie auf das ausgegebene Array anzuwenden. Hier wird exemplarisch mit `.join()` aus den Items des Arrays ein String gebildet (ein Schönheitsfehler gegenüber dem vorangegangenen Beispiel ist das Semikolon am Ende, dafür ist es allerdings sehr einfach):

```
// Anmerkung: .join() ist eine Array-Methode:
var auswahl = $('#select').val().join(', ');
$('#message').html('Sie haben ' + auswahl + ' gewählt.');
```

Die Methode `.val()` ist umgekehrt auch in der Lage, einen Wert zu schreiben. Hierzu braucht ihr lediglich ein entsprechender Wert übergeben zu werden (oder eine Callback-Funktion, die den gewünschten Wert erzeugt):

Texteingabe:

```
<input type="text" name="meinInput" id="meinInput" />
```

Textausgabe:

```
<input type="text" name="meinOutput" id="meinOutput" />
```

Um die Eingabe aus dem Feld `#meinInput` in das Ausgabetextfeld `#meinOutput` zu schreiben, genügen die Zeilen

```
var meinWert = $("#meinInput").val();
$("#meinOutput").val(meinWert);
```

Sinnvoller ist dies natürlich, wenn die zu übertragenden Werte von Checkboxes, Radiobuttons oder versteckten Feldern dynamisch zur Laufzeit bestimmt werden sollen.

Betrachten wir nochmals die Testkonstellation aus den drei Checkboxes:

```
<input type="checkbox" name="cb" id="cb1" value="check1"/>
<input type="checkbox" name="cb" id="cb2" value="check2"/>
<input type="checkbox" name="cb" id="cb3" value="check3"/>
```

Der Wert einer Checkbox lässt sich überschreiben:

```
$("#cb3").val('checkX');
alert($("#cb3").val()); // gibt checkX aus
```

Übergeben Sie den bereits als `value` vorliegenden Wert, ändert sich nichts:

```
$("#cb2").val('check2');
alert($("#cb2").val()); // gibt check2 aus
```

Interessant ist aber, was geschieht, wenn Sie `.val()` ein Array übergeben, das den momentanen Value einer Checkbox enthält:

```
// setzt #cb2 auf "checked":
$("#cb2").val(['check2']);
```

Ein Array triggert das `checked`-Property desjenigen Inputs, dessen Value es enthält – dies gilt für Checkboxes, Radiobuttons oder Select-Felder, die im Multiple-Modus definiert sind. Dies lässt sich zeigen, wenn die Collection mehrere Checkboxes enthält:

```
// setzt #cb2 und #cb3 auf "checked":
$("input[name=cb]").val(['check2', 'check3']);
```

Auf das Blumen-Auswahlfeld angewendet:

```
<select name="blumen" id="blumen"
    multiple="multiple" size="5">
    <option value="Rosen">Rosen</option>
```

```

<option value="Tulpen">Tulpen</option>
<option value="Nelken">Nelken</option>
<option value="Veilchen">Veilchen</option>
<option value="Margheriten">Margheriten</option>
</select>

```

Hier erreichen Sie die Auswahl von Rosen und Nelken, indem Sie folgendes anordnen (dies richtet sich auf das `value`-Attribut der Optionen):

```
$("#blumen").val(['Rosen', 'Nelken']);
```

4.9 DOM-Manipulation

Meist soll das Dokument – oder anders ausgedrückt, das DOM – während der Lebensdauer der Seite nicht unverändert bestehen bleiben; man möchte Inhalte (beispielsweise Elementknoten) hinzufügen oder löschen können. Das Erzeugen von Knoten obliegt dem `document`-Objekt höchstpersönlich. Es besitzt eigene Methoden, um Element-, Attribut- oder Textknoten »herzustellen«. Auf diese Funktionen greift jQuery letztendlich zwar ebenfalls zurück, »verpackt« die eher spartanischen DOM-Funktionen aber zum einfachen Gebrauch durch den Programmierer als handliche, praktische Helferfunktionen.

Erläuternd wollen wir zunächst ein Blick auf die DOM-Methode zur Erzeugung von Elementen werfen. So wird beispielsweise ein `<div>`-Element erzeugt:

```

// erzeugt: <div></div>
var neuesDiv = document.createElement("div");

```

Der neue Elementknoten wird vorübergehend in eine Variable abgelegt und kann nun ein Attribut erhalten, das mittels der Elementmethode `setAttribute()` gleich auch mit einem Wert gefüllt werden kann:

```

// erzeugt: <div align="center"></div>
neuesDiv.setAttribute("align", "center");

```

Ein Textknoten muss separat über `createTextNode()` erzeugt werden. Er kann anschließend über `appendChild()` als Kindknoten in den frisch erzeugten Div-Knoten eingefügt werden:

```

var textinhalt = document.createTextNode("Textinhalt");
// erzeugt: <div align="center">Textinhalt</div>
neuesDiv.appendChild(textinhalt);

```

Das neu erzeugte Div ist im Moment immer noch nicht mehr als eine Referenz auf einen Elementknoten, die sich in einer Variable befindet – das Element ist

noch nicht Teil des DOM. Um es in den Baum, sprich ins Dokument, zu integrieren, muss es dort *eingehängt* werden.

Hierfür kann neben der bereits bekannten Methode `appendChild()`, die einen Inhalt stets als letzten Kindknoten eines Zielknotens einfügt, auch eine weitere Methode eingesetzt werden: `insertBefore()`.

Angenommen, irgendwo im Dokument befindet sich ein Div mit `id="ziel"`:

```
<div>
  <div id="ziel">Der Zielknoten</div>
</div>
```

Nun kann das vorhin erzeugte Div *vor* diesem ins Dokument eingefügt werden. Die DOM-Methode `insertBefore()` arbeitet nur ausgehend vom Elternknoten desjenigen Knotens, vor dem der Inhalt erscheinen soll. Dies ist dadurch etwas unhandlich in der Handhabung (jQuery verspricht hier Abhilfe):

```
var ziel = document.getElementById("ziel");
ziel.parentNode.insertBefore(ziel, neuesDiv);
```

Das Ergebnis sieht wie folgt aus:

```
<div>
  <div align="center">Textinhalt</div>
  <div id="ziel">Der Zielknoten</div>
</div>
```

Das Löschen eines Knotens funktioniert analog von dessen Elternknoten aus – die dazugehörige Anweisung lautet `removeChild()`. Hier soll der Knoten `#ziel` entfernt werden:

```
var ziel = document.getElementById("ziel");
ziel.parentNode.removeChild(ziel);
```

Nach der Operation ist nur noch der vorher eingefügte Div im Container:

```
<div>
  <div align="center">Textinhalt</div>
</div>
```

jQuery und DOM-Manipulation

Das nachträgliche Erstellen, Einfügen oder Löschen von Dokumentteilen ist in der aktuellen Webprogrammierung unabdingbar (beispielsweise in Zusammenhang mit Ajax-Aktionen). Leider sind die Standardmethoden des DOM unhandlich und lückenhaft – z. B. fehlt ein Pendant `.insertAfter()` zu `.insertBefore()`. jQuery füllt diese Lücken und bietet zu den DOM-Methoden einfacher zu handhabende und mächtigere Pendants.

Betrachten wir nun die Alternativen, die jQuery zu den DOM-Methoden zur Verfügung stellt. Einige von ihnen sind beinahe namensgleich zu ihren »offiziellen« Varianten, was im Sinne der Selbsterklärbarkeit durchaus gewünscht ist. Auffällig ist, dass Methoden zur expliziten Erzeugung von Knoten hier zu fehlen scheinen. Der Grund hierfür ist, dass dies bereits Aufgabe der `$()`-Funktion selbst ist – erinnern Sie sich dabei an das Übergeben eines HTML-Strings an `$()`. Auch im Rahmen der DOM-Manipulation wird die Knotenerzeugung quasi »nebenbei« erledigt.

4.9.1 Methoden zum Einfügen von Knoten

Generierte Knoten in den DOM-Baum einzuhängen ist mit den Standardmethoden eine komplizierte, manchmal durchaus auch heikle Aktion. Die jQuery-Methoden, die dem gleichen Zweck dienen, erweisen sich im täglichen Gebrauch als wesentlich einfacher zu handhaben.

Methode	Erläuterung
<code>.after(inhalt)</code> <code>.after(fn(i))</code>	Fügt den durch den Parameter <code>inhalt</code> bestimmten Inhalt nach jedem Element der Collection in den DOM-Baum ein. Alternativ: Nimmt eine Funktion <code>fn</code> entgegen, die den Inhalt erzeugt und die Indexposition <code>i</code> des aktuellen Elements der Collection erhält.
<code>.append(inhalt)</code> <code>.append(fn(i,old))</code>	Fügt den durch den Parameter <code>inhalt</code> bestimmten Inhalt ans Ende des Elementinhalts jedes Elements in der Collection ein. Alternativ: Nimmt eine Funktion entgegen, die den Inhalt erzeugt und den Collection-Index <code>i</code> des aktuellen Elements sowie dessen alten Inhalt <code>old</code> als Parameter erhält.
<code>.appendTo()</code>	Fügt alle Elemente der Collection ans Ende des Elementinhalts des Zielements ein.
<code>.insertAfter(elem)</code>	Hängt alle Elemente der aktuellen Collection <i>hinter</i> das Target-Element <code>elem</code> in den DOM-Baum ein.
<code>.insertBefore(elem)</code>	Hängt alle Elemente der aktuellen Collection <i>vor</i> das Target-Element <code>elem</code> in den DOM-Baum ein.
<code>.prepend(inhalt)</code> <code>.prepend(fn(i,old))</code>	Fügt den durch den Parameter <code>inhalt</code> bestimmten Inhalt <i>am Anfang</i> des Elementinhalts jedes Elements in der Collection ein. Alternativ: Nimmt eine Funktion entgegen, die den Inhalt erzeugt und die Indexposition <code>i</code> des aktuellen Elements der Collection sowie den alten Inhalt <code>old</code> als Parameter erhält.
<code>.prependTo()</code>	Fügt alle Elemente der Collection an den Anfang des Elementinhalts des Zielements ein.

Tabelle 4.44 jQuery-Methoden zur DOM-Manipulation (Einfügen von Knoten)

Methode .after()

Die Methoden `.after()` und `.insertAfter()` bewältigen die gleichen Aufgaben, nur die Syntax ist unterschiedlich. Die Aufgabe ist, nach einem bestimmten Element Inhalt oder Inhalt mit enthaltenen HTML-Elementen einzusetzen.

Hier ein paar Beispiele, zunächst eines, das mit `.after()` ausgeführt wurde. Zunächst wieder erst einmal der relevante HTML-Ausschnitt:

```
<h1 class="target">Überschrift</h1>
```

Und nun der jQuery-Code:

```
$(document).ready(function() {
    $('.target').after('<p>Absatz</p>');
});
```

Sie fügen an jedes selektierte Element mit dem Selektor `.target` einen Absatz an, also hinter das schließende Tag `</h1>`. Betrachten wir jetzt den generierten Code:

```
<h1 class="target">Überschrift</h1><p>Absatz</p>
```

Sie können auch Elemente, die Sie in einem Dokument vorfinden, an ein anderes umhängen. Sie verändern damit den HTML-Aufbau:

```
<h1>Überschrift</h1>
<p class="target">Absatz</p>
```

Der jQuery-Aufbau sieht dann folgendermaßen aus:

```
$(document).ready(function() {
    $('.target').after($('h1'));
});
```

Und der generierte Quellcode präsentiert sich so:

```
<p class="target">Absatz</p>
<h1>Überschrift</h1>
```

Die Überschrift wurde nun an das Zielobjekt angehängt. Im Unterschied zum ersten Beispiel verpacken Sie in der Methode `.after()` nur einen weiteren Selektor `$('h1')` und sortieren ihn nur an einer anderen Stelle wieder ein. Sie bemerken vielleicht, dass dieses `<h1>`-Element nicht kopiert, sondern ausgeschnitten und an anderer Stelle wieder eingefügt wurde.

Methode .insertAfter()

Nun soll das gleiche Ergebnis mit `.insertAfter()` erzielt werden:

```
$(document).ready(function() {
    $('<p>Absatz</p>').insertAfter('.target');
});
```

Das Ergebnis wird dasselbe sein, es wird an ein Element mit der CSS-Klasse `target` ein Absatz an das selektierte Element eingefügt.

Einen wichtigen Unterschied zwischen `.after()` und `.insertAfter()` müssen Sie allerdings stets bedenken: Sie können der Methode `.after()` eine anonyme Funktion übergeben, `.insertAfter()` bietet das nicht:

```
$(document).ready(function() {
    var count = 0;
    $('.target').after(function() {
        count++;
        return '<p>Absatz mit dem Index: '+ count + '</p>';
    });
});
```

Den HTML-Abschnitt erweitern Sie um eine Zeile (oder, wenn Sie wollen, um mehrere):

```
<h1 class="target">Überschrift</h1>
<h2 class="target">Subline</h2>
```

In diesem Beispiel lassen Sie sich nun die Anzahl der eingefügten Absätze anzeigen. Der Zähler `count` wird bei jeder Iteration über das Element mit der CSS-Klasse `target` um eins erhöht. Diese Variable lassen Sie sich anschließend als Inhalt des `<p>`-Elements ausgeben.

Methode `.clone()`

Gesetzt den Fall, Sie wollen nun erreichen, dass das Element, das Sie herauslösen und an anderer Stelle wieder einfügen wollen, nicht ausgeschnitten und eingefügt, sondern dupliziert und eingefügt werden soll, dann verwenden Sie die Methode `.clone()`, die bereits als Accessor-Methode vorgestellt wurde. Hier ein Beispiel:

```
$(document).ready(function() {
    $('.target').clone().after($('h1'));
});
```

Der generierte Output sieht so aus:

```
<h1>Überschrift</h1>
<p class="target">Absatz</p>
<h1>Überschrift</h1>
```

Es werden alle Daten, die z. B. mit `.data()` an dieses Element angehängt wurden, und alle Event Handler mitkopiert. Falls Sie dies nicht wünschen, können Sie den optionalen Parameter `withDataAndEvents` auf `false` setzen. Dieser optionale Boolesche Wert sorgt dafür, dass Sie entscheiden können, ob die »unsichtbaren« Daten und Events mitkopiert werden oder nicht.

Methoden `.before()` und `.insertBefore()`

Für `.after()` und `.insertAfter()` gibt es die Pendanten `.before()` und `.insertBefore()`. Wie Sie richtig erraten werden, fügen diese Methoden vor einem selektierten Element einen Inhalt mit oder ohne neue Elemente hinzu. Beide Methoden verhalten sich analog zu eingangs beschriebenen Methoden, daher können wir uns hier mit einem kurzen Beispielfragment begnügen:

```
$('.target').before('<p>Absatz</p>');
```

Und das Ganze mit `.insertBefore()`:

```
$('<p>Absatz</p>').insertBefore('.target');
```

Sie können auch analog zu `.after()` für `.before()` eine anonyme Funktion übergeben, für `.insertBefore()` geht das nicht.

Methode `.append()`

Mit der Methode `.append()` können Sie an ein selektiertes Element weitere Elemente oder auch Inhalt anhängen. Haben Sie diese Beschreibung nicht eben schon gehört? Worin liegt nun der Unterschied zu `.after()` oder `.insertAfter()`? Sehen wir uns dazu wieder ein Beispiel an:

```
<div class="target">
  <p>Absatz</p>
</div>
```

Und nun der dazugehörige Scriptteil:

```
$(document).ready(function() {
  $('.target').append('<p>Ein weiterer Absatz</p>');
});
```

Der generierte Code sieht dann folgendermaßen aus:

```
<div class="target">
  <p>Absatz</p>
  <p>Ein weiterer Absatz</p>
</div>
```

Der Absatz wird an das letzte Kind innerhalb des oder der selektierten Elemente angehängt. Mit `.after()` hingegen würde der Absatz nach dem `<div>`-Container angehängt werden.

An `.append()` können Sie genauso wie an `.after()` eine anonyme Funktion übergeben. Wir modifizieren das Beispiel, das wir bereits von `.after()` kennen:

```
$(document).ready(function() {
    var count = 0;
    $('.target').append(function() {
        count++;
        return '<p>Absatz mit dem Index: '+ count +'</p>';
    });
});
```

Die HTML-Ausgabe sparen wir uns an dieser Stelle, sehen Sie es als kleine Herausforderung, das Beispiel selbst zu komplettieren, im Kopf oder mit Ihrem Editor.

Weitere Methoden

Wie systematisch John Resig und sein jQuery-Team hier vorgegangen sind, sehen Sie an den Methoden: `.append()`, `.appendTo()`, `.prepend()` und `.prependTo()`. Wir können voraussagen, dass sich `.append()` zu `.appendTo()` genauso verhält wie `.before()` zu `.insertBefore()` und `.prepend()` zu `.prependTo()`. Vergleichen wir:

```
$('.target').after('<p>Absatz</p>');
$('.<p>Absatz</p>').insertAfter('.target');

$('.target').append('<p>Absatz</p>');
$('.<p>Absatz</p>').appendTo('.target');
```

Die Beispiele führen jeweils zu denselben Ergebnissen, also `.after()` und `.insertAfter()` einerseits, `.append()` und `.appendTo()` andererseits.

Das Beispiel mit `.prepend()` und `.prependTo()` probieren Sie wieder selbst aus, es folgt vorigem Muster. Eine Bemerkung vielleicht noch, `.prepend()` und `.prependTo()` hängen den übergebenen Parameter an das erste Kind innerhalb des selektierten Elements an und nicht an das letzte Kind, aber das dachten Sie sich wahrscheinlich schon.

Sie können auch an `.prepend()` eine anonyme Funktion übergeben, hingegen an `.prependTo()` nicht.

4.9.2 Entfernen von Knoten

Das Entfernen von Knoten ist mittels der regulären DOM-Funktionen möglich, aber gelegentlich knifflig. jQuery weiß auch hier wieder Rat und hält zudem unerschwellige Verbesserungen für diese Standardaktionen bereit.

Methode	Erläuterung
<code>.remove()</code> <code>.remove(sel)</code>	Entfernt die Elemente der aktuellen Collection (mit Inhalten, gebundenen Daten und Events) aus dem DOM. Optional: Entfernt die Elemente der aktuellen Collection aus dem DOM, auf die der übergebene Selektor <code>sel</code> passt.
<code>.detach()</code> <code>.detach(sel)</code>	Entfernt alle Elemente der aktuellen Collection aus dem DOM, behält aber deren Datenbindungen, falls sie in den DOM zurückgeschrieben werden (anders als <code>.remove()</code>). (Nimmt optional einen Selektor <code>sel</code> entgegen, der die aktuelle Collection filtert.)

Tabelle 4.45 jQuery-Methoden zur DOM-Manipulation (Entfernen von Knoten)

Methode `.remove()`

Die Methode `.remove()` entfernt erwartungsgemäß DOM-Elemente, zuerst wieder das DOM-Gerüst:

```
<div class="container">
  <p class="target">Absatz</p>
  <p>Ein weiterer Absatz</p>
</div>
```

Und hier die entsprechende jQuery-Anweisung:

```
$(document).ready(function() {
  $(".target").remove();
});
```

Sie erwarten nun sicherlich, dass der Absatz, der mit der CSS-Klasse `target` versehen war, nun entfernt wird. Wir können Ihnen versichern, dass das auch eintrifft. Wären in diesem `<p>`-Element weitere Elemente verschachtelt, würden sie ebenfalls entfernt werden. Alternativ können Sie auch folgende Syntax verwenden:

```
$(document).ready(function() {
  $("p").remove(".target");
});
```

Dies führt zum selben Ergebnis. Beachten Sie dabei, dass auch gebundene Events und über `.data()` gebundene Daten mitgelöscht werden.

Methode `.detach()`

Es existiert noch eine weitere Methode, um Elemente aus dem DOM-Baum zu entfernen: `.detach()` entfernt zwar das oder die Elemente, hält aber dieses Objekt für eine weitere Verwendung im Speicher, sodass Sie es zu einem späteren Zeitpunkt wieder einfügen und weiterverwenden können. Die Elemente werden zwar aus dem Baum entfernt, selbst Events und `.data()` bleiben aber gebunden. Fügen wir die Elemente später wieder ein, bleiben auch Zustände erhalten. War ein Element über `.toggle()` angeklickt, ist dieser Status auch nach der `.detach()`-Operation vorhanden. Schauen wir uns dazu wieder ein Beispiel an, in dem wir zunächst dem HTML-Gerüst noch einen Button hinzufügen:

```
<button>Entfernen/Hinzufügen</button>
...
<div class="container">
  <p class="target">Ein Absatz</p>
  <p>Ein weiterer Absatz</p>
</div>
```

Im jQuery-Teil werden wir nun zwei `.toggle()` verwenden, um hin- und herschalten zu können:

```
$(document).ready(function() {
  var hold = $(".container");
  $("button").toggle(function(){
    hold.detach();
  }, function() {
    hold.appendTo("body");
  });
  $("p").toggle(function() {
    $(this).css("background-color", "#FF0000");
  }, function() {
    $(this).css("background-color", "#CCCCCC");
  });
});
```

Das erste `.toggle()`-Event löscht mit `.detach()` das Element und fügt das selektierte Element mit `.append()` wieder ein, mit dem zweiten Element schalten Sie eine Hintergrundfarbe für jedes `<p>`-Elemente ein und wieder aus. Probieren Sie dieses Beispiel aus. Sie werden sehen, dass der gelöschte Container die Hintergrundfarbe zeigt, die zuletzt eingeschaltet wurde.

4.9.3 Ersetzen von Knoten

Ergänzend zum Löschen oder Hinzufügen von Knoten fungiert deren Ersetzung. Dies ist zwar auch mit Standard DOM-Methoden möglich, jQuery setzt jedoch, wie üblich, in Bezug auf Handhabung und Kompatibilität noch einen obendrauf.

Methode	Erläuterung
<code>.replaceAll(target)</code>	Ersetzt die Elemente, die durch den Selektor <code>target</code> bestimmt werden, durch die aktuelle Collection.
<code>.replaceWith(cont)</code> <code>.replaceWith(fn)</code>	Ersetzt die Elemente der aktuellen Collection durch <code>cont</code> (DOM-Knoten, HTML-String oder jQuery-Objekt). Alternativ: Ersetzt die Elemente der aktuellen Collection durch den Rückgabewert einer Funktion <code>fn</code> (einen HTML-String).

Tabelle 4.46 jQuery-Methoden zur DOM-Manipulation (Ersetzen von Knoten)

Methode `.replaceAll()`

Mittels der Methode `.replaceAll()` wird ein in einer Collection selektiertes Element ersetzt. Diese Methode führt prinzipiell zum selben Ergebnis wie `.replaceWith()`, lediglich die Syntax ist umgekehrt. Betrachten wir dazu ein Beispiel mit dem bereits bekannten HTML-Fragment. Zunächst der jQuery-Befehl:

```
$(document).ready(function() {
    $("<h1>Neue Überschrift</h1>").replaceAll(".target");
});
```

Es sollen also alle Elemente der Klasse `target` durch die aktuelle Collection ersetzt werden, die aus dem übergebenen HTML-String gebildet wurde. Aus

```
<div class="container">
  <p class="target">Ein Absatz</p>
  <p>Ein weiterer Absatz</p>
</div>
```

wird daher

```
<div class="container">
  <h1>Neue Überschrift</h1>
  <p>Ein weiterer Absatz</p>
</div>
```

Wie Sie es bereits von `.insertBefore()` oder `.insertAfter()` kennen, steht das neue Element in einem frisch erzeugten jQuery-Objekt am Beginn der Kette. Das Ziel hingegen, also das zu ersetzende Element, befindet sich in der Replace-Anweisung als zu übergebender Parameter.

Methode `.replaceWith()`

Die Methode `.replaceWith()` ist vom Handling her ein wenig anders als `.replaceAll()`, erzielt aber auf Wunsch dasselbe Ergebnis. Um dies zu demonstrieren, wird die jQuery-Anweisung ein wenig modifiziert, indem die Methode ausgetauscht wird:

```
$(document).ready(function() {
    $(".target").replaceWith("<h1>Neue Überschrift</h1>");
});
```

Und wir erhalten dasselbe Ergebnis. Der Methode kann auch ein jQuery-Objekt übergeben werden:

```
$(document).ready(function() {
    $(".target").replaceWith( $("div p:last") );
});
```

Hierbei werden die beiden `<p>`-Elemente vertauscht. Der Quelltext sieht anschließend so aus:

```
<div class="container">
    <p>Ein weiterer Absatz</p>
    <p class="target">Ein Absatz</p>
</div>
```

Anders als bei `.replaceAll()` kann `.replaceWith()` wieder eine Funktion übergeben werden. Ein abstraktes Beispiel schriebe sich wie folgt:

```
$(document).ready(function() {
    $(".target").replaceWith( function() {
        // Den Ersatzstring erzeugen ...
        // und zurückgeben:
        return ersatzstring;
    } );
});
```

4.9.4 Wrapping-Methoden

Mit den hier folgenden Methoden können Sie vorhandene, im Rahmen einer Collection selektierte Elemente innen, außen oder in Gesamtheit mit generierten Inhalten umschließen. Im Gegenzug besteht auch die Möglichkeit, ausgehend von einem selektierten Element dessen umschließende Elemente zu entfernen.

Methode	Erläuterung
<code>.wrap(wElem)</code>	Umgibt die Elemente der aktuellen Collection durch die HTML-Struktur, die sich aus dem Ausdruck <code>wElem</code> ergibt. (Hierbei kann <code>wElem</code> ein HTML-Fragment, ein Selektorausdruck, ein jQuery-Objekt oder ein DOM-Element sein.)
<code>.wrap(wFn)</code>	Ist das Argument eine Funktion <code>wFn</code> , generiert diese die Wrap-Struktur aus ihrem Rückgabewert.
<code>.wrapAll(wElem)</code>	Umgibt die Elemente der aktuellen Collection <i>in ihrer Gesamtheit</i> durch die HTML-Struktur, die sich aus dem Ausdruck <code>wElem</code> ergibt. (Hierbei kann <code>wElem</code> ein HTML-Fragment, ein Selektorausdruck, ein jQuery-Objekt oder ein DOM-Element sein.)
<code>.wrapAll(wFn)</code>	Ist das Argument eine Funktion <code>wFn</code> , generiert diese die Wrap-Struktur aus ihrem Rückgabewert.
<code>.wrapInner(wElem)</code>	Umgibt <i>den Inhalt</i> der Elemente der aktuellen Collection durch die HTML-Struktur, die sich aus dem Ausdruck <code>wElem</code> ergibt. (Hierbei kann <code>wElem</code> ein HTML-Fragment, ein Selektorausdruck, ein jQuery-Objekt oder ein DOM-Element sein.)
<code>.wrapInner(wFn)</code>	Ist das Argument eine Funktion <code>wFn</code> , generiert diese die Wrap-Struktur aus ihrem Rückgabewert.
<code>.unwrap()</code>	Entfernt die Parents der Elemente der aktuellen Collection aus dem DOM, belässt Letztere aber an Ort und Stelle.

Tabelle 4.47 jQuery-Methoden zum Wrappen selektierter Elemente

Methode .wrap()

Beginnen wir mit der grundlegenden Methode `.wrap()` und betrachten zunächst das bekannte HTML-Gerüst:

```
<div class="container">
  <p>Ein Absatz</p>
  <p class="target">Ein weiterer Absatz</p>
</div>
```

Dies sind die wirksamen jQuery-Anweisungen:

```
$(document).ready(function() {
  $("p").wrap("<div class='wrap' />");
});
```

Diese Zeilen beschieren folgendes Ergebnis:

```
<div class="container">
  <div class="wrap">
    <p>Ein Absatz</p>
```

```

</div>
<div class="wrap">
  <p class="target">Ein weiterer Absatz</p>
</div>
</div>

```

Im Ergebnis wird jedes einzelne der selektierten Elemente mit dem neuen Element `<div class="wrap">` umhüllt.

Methode `.wrapAll()`

Im Gegensatz zu `.wrap()`, die sich einzeln auf jedes Item der Collection bezieht, wird die Methode `.wrapAll()` alle selektierten Elemente gemeinsam umhüllen. Zunächst der jQuery-Code

```

$(document).ready(function() {
  $("p").wrap('<div class="wrap" />');
});

```

und dann das generierte Ergebnis

```

<div class="container">
  <div class="wrap">
    <p class="target">Ein Absatz</p>
    <p>Ein weiterer Absatz</p>
  </div>
</div>

```

Es gibt noch einen Unterschied zwischen den beiden Methoden: `.wrap()` können Sie eine Funktion als Parameter übergeben. Dies sieht in Kurzform folgendermaßen aus:

```

$(document).ready(function() {
  $("p").wrap(function() {
    return "irgendwas"; //Anweisung hier
  });
});

```

Methode `.wrapInner()`

Eine weitere Variante der Umhüllung können Sie ebenfalls nutzen – die Methode `.wrapInner()`. Wie der Name schon andeutet, umhüllt sie innerhalb der Selektion mit einem Element. Gleiches HTML-Gerüst, aber modifizierte jQuery-Anweisung:

```

$(document).ready(function() {
  $(".container").wrapInner('<div class="wrap" />');
});

```

Und hier das generierte Ergebnis:

```
...
<div class="container">
  <div class="wrap">
    <p>Ein Absatz</p>
    <p class="target">Ein weiterer Absatz</p>
  </div>
</div>
...
```

Der neue Container `<div class="wrap">` wird nun innerhalb des selektierten Elements eingefügt, umschließt aber alle `<p>`-Elemente der Collection in der Gesamtheit. Auch hier könnte eine anonyme Funktion übergeben werden; ein Syntaxbeispiel wurde im Vorfeld bereits gezeigt.

Methode `.unwrap()`

Als letzte zu betrachtende Methode bleibt noch `.unwrap()`, das umschließende Elemente entfernt. Als Beispiel dient folgendes HTML-Fragment:

```
<div class="container">
  <div class="wrap">
    <p>Ein Absatz</p>
  </div>
  <div class="wrap">
    <p class="target">Ein weiterer Absatz</p>
  </div>
</div>
```

Hier der jQuery-Code (die Methode nimmt keinen Parameter entgegen):

```
$(document).ready(function() {
  $("p").unwrap();
});
...
```

Das Ergebnis sieht wie folgt aus (es entspricht, wie Sie sehen, dem Code, von dem wir ursprünglich ausgegangen sind):

```
<div class="container">
  <p>Ein Absatz</p>
  <p class="target">Ein weiterer Absatz</p>
</div>
```

Alle die selektierte `<p>`-Elemente jeweils unmittelbar umgebenden Container `<div class="wrap">` sind nun verschwunden.

4.10 CSS und Styleeigenschaften

Ein wichtiges Gebiet, auf dem jQuery tätig wird, ist die Manipulation der CSS-Eigenschaften von Seitenelementen. Dies geschieht auf Grundlage einer Collection, die aus den zu bearbeitenden Elementen zusammengestellt wurde. Zur Stylebearbeitung bietet jQuery eine Reihe von Methoden, die auf das `class`-Attribut zugreifen oder direkt das `style`-Property eines Knotens adressieren.

Auch verschiedene Wege existieren, bestimmte, ansonsten schwer fassbare CSS-Werte auszulesen – wie die aktuelle Breite oder Position eines Elements im Layout. Mit herkömmlichen Ansätzen ist dies oft nur schwer möglich, da solche Eigenschaften vom Browser während des Renderingvorgangs erst berechnet werden (z. B. als *computed height*).

4.10.1 Methoden für das class-Attribut

Verschaffen wir uns zunächst eine Übersicht über die Methoden des jQuery-Objekts, die auf das `class`-Attribut einwirken. Ein Grundbedürfnis besteht darin, festzustellen, ob eine bestimmte Klasse einem Element zugewiesen ist.

Eine Klasse mit `.hasClass()` überprüfen

Methode	Erläuterung
<code>.hasClass(class)</code>	Gibt für jedes Element der aktuellen Collection <code>true</code> oder <code>false</code> zurück, je nachdem, ob es die CSS-Klasse <code>class</code> besitzt oder nicht.

Tabelle 4.48 Methode zur Prüfung der Existenz einer Klasse

Die Methode `.hasClass()` gibt `true` zurück, falls eine entsprechende Klasse in einem Element gefunden wurde. Prüfen Sie beispielsweise

```
<div id="d1" class="container"> ... </div>
```

mit

```
$('#d1').hasClass('container');
```

erhalten Sie, wie erwartet, den Rückgabewert `true`. Dies gilt auch, wenn mehrere Klassen zugewiesen sind, darunter die geprüfte:

```
<div id="d1" class="container test"> ... </div>
```

Auch für diesen Fall ergibt der dieser Test den Wert `true`: jQuery prüft nämlich den Wert des `class`-Attributs dahingehend, ob es den übergebenen Prüfstring als Teilstring enthält.

Als problematisch erweist sich dieser Umstand, sobald mehrere Klassen gleichzeitig getestet werden sollen. So gibt der Ausdruck

```
$("#ul").hasClass("class-1 class-2");
```

nur dann `true` zurück, wenn beide Klassen in dieser Reihenfolge als Teilwert des Klassenattributs vorkommen:

```
<ul class="class-1 class-2">
```

Für diesen Fall gäbe der Ausdruck `$("#ul").hasClass("class-2 class-1")` allerdings `false` zurück! Eine Reihung von `.hasClass()`, also das Prüfen zweier Klassen gleichzeitig, ist nicht möglich. Statt dies über eine Boolesche Verknüpfung zweier `.hasClass()`-Aufrufe zu versuchen (wäre möglich), ist es sinnvoller, auf eine andere Methode auszuweichen, die einen CSS-Selektor anstatt eines Klassennamens akzeptiert:

```
$("#ul").is(".class-2.class-1") // gibt true zurück
```

Dieser Ausdruck erzeugt ebenfalls einen Booleschen Wert, kann also analog zu `.hasClass()` eingesetzt werden, sofern Sie einen Class-Selektor übergeben.

Nicht jede jQuery-Methode gibt ein jQuery-Objekt zurück

Es ist zu beachten, dass der gesamte Ausdruck durch `.hasClass()` einen Booleschen Wert ergibt. In diesem Fall entsteht also *kein jQuery-Objekt*. Das hat zur Folge, dass an die Methode `.hasClass()` keine weitere jQuery-Methode angereicht werden kann!

Methode	Erläuterung
<code>.addClass(class)</code>	Fügt allen Elementen der aktuellen Collection die betreffende(n) Klasse(n) hinzu.
<code>.addClass(fn(i, oCl))</code>	Fügt eine Klasse per Funktion hinzu, die die Indexposition <code>i</code> des aktuellen Elements der Collection und die alte Klasse <code>oCl</code> als Parameter erhält.
<code>.removeClass(cls)</code> <code>.removeClass()</code> <code>.removeClass(fn(i, oCls))</code>	Entfernt die übergebene(n) CSS-Klasse(n) <code>cls</code> von allen Elementen der aktuellen Collection. Entfernt <i>alle</i> CSS-Klassen von den Elementen der Collection, solange kein Parameter übergeben wird.

Tabelle 4.49 jQuery-Methoden zur Behandlung von CSS-Klassen

Methode	Erläuterung
	Alternativ: Nimmt eine Funktion <i>fn</i> entgegen, die zu entfernende Klassen als Rückgabewert liefert. Die Funktion bekommt den Collection-Index <i>i</i> und den alten <i>class</i> -Wert <i>oCls</i> übergeben.
<pre>.toggleClass(cCls) .toggleClass(cCls, s) .toggleClass(fn(i, cClsAlt)) .toggleClass(fn(i, oCls), s)</pre>	Entfernt eine vorhandene Klasse <i>cCls</i> von den Elementen der Collection oder fügt sie hinzu, wenn sie fehlt. Wird als zweiter Parameter ein Boolescher Ausdruck <i>s</i> übergeben, bedeutet <i>true</i> <i>hinzufügen</i>, <i>false</i> <i>entfernen</i>. Alternativ: Nimmt eine Funktion <i>fn</i> entgegen, die zu toggelnde Klassen als Rückgabewert liefert. Die Funktion bekommt den Collection-Index <i>i</i> und den alten <i>class</i>-Wert <i>oCls</i> übergeben. Wird als zweiter Parameter ein Boolescher Ausdruck <i>s</i> übergeben, bedeutet <i>true</i> <i>hinzufügen</i>, <i>false</i> <i>entfernen</i>.

Tabelle 4.49 jQuery-Methoden zur Behandlung von CSS-Klassen (Forts.)

Eine Klasse mit `.addClass()` hinzufügen

Um eine Klasse zu einem Element hinzuzufügen, verwenden Sie die Methode `.addClass()`. Im Folgenden ein kleines Beispiel. Beginnen Sie wieder mit dem HTML-Gerüst. HTML-, Head- und Body-Elemente werden hier weggelassen, wobei Sie im `<head>` noch folgenden Stil notieren:

```
.container {
  width:300px;
  height:150px;
  background-color:#CCCCCC;
}
```

Im Body befindet sich ein `<div>`-Container mit beliebigem Inhalt:

```
<div>
  <p>Ullique volori in eiciliqui aut rehenduciet
    asitio occae ped cipsunti.</p>
</div>
```


Im Scriptteil notieren wir:

```
$(document).ready(function() {
    $("div").addClass("container");
})
```

Wir erwarten nun keine Überraschungen, wenn wir das Ergebnis im Browser betrachten, wenn Sie im Firefox den generierten Quelltext der Seite über das Web Developer Plugin ansehen, stellen Sie fest, dass das Attribut tatsächlich hinzugefügt wurde:

```
<div class="container"> ... </div>
```

Seit der Version 1.4 besitzt diese Methode noch die Möglichkeit, eine anonyme Funktion zu übergeben:

```
$(document).ready(function() {
    $("li").addClass(function(index, class) {
        if (class != selected) {
            return "list-element-" + index;
        }
    });
})
```

Diese Funktion übergibt zwei Argumente, den Namen einer Klasse, der vielleicht schon eingesetzt wurde, und den Index des Elements. Nehmen wir einmal an, wir haben eine unsortierte Liste:

```
<ul>
    <li>Punkt 1</li>
    <li class="selected">Punkt 2</li>
    <li>Punkt 3</li>
</ul>
```

So erhalten wir in der Ausgabe des generierten Quelltexts:

```
<ul>
    <li class="list-element-0">Punkt 1</li>
    <li class="selected">Punkt 2</li>
    <li class="list-element-2">Punkt 3</li>
</ul>
```

Sie können damit jedem der einzelnen Listenelemente eine eigene Klasse zuweisen, die natürlich in Ihrem Stylesheet definiert werden muss.

Eine Klasse mit `.removeClass()` entfernen

Dementsprechend können Sie mit der Methode `.removeClass()` eine oder mehrere Klasse auch wieder entfernen.

```
$("#div").removeClass("container");
```

entfernt die Klasse `container`. Dagegen entfernt

```
$("#div").removeClass("container","selected");
```

die Klassen `container` und `selected`; und `$("#div").removeClass();`, also ohne Parameter, entfernt alle Klassen eines Elements. Auch `.removeClass()` bietet seit der Version 1.4 die Möglichkeit, eine anonyme Funktion zu übergeben.

Sinnvoll ist es auch, beide Methoden miteinander zu verketten:

```
$("#div").removeClass().addClass("new");
```

So entfernen Sie erst alle Klassen eines Elements, um anschließend eine neue Klasse hinzuzufügen.

Klasse an- und abschalten mit `.toggleClass()`

Mit `.toggleClass()` können Sie per Mouse-Event eine Klasse hinzufügen und wieder entfernen. Schauen wir uns dazu ein Beispiel an:

```
$(document).ready(function() {
    $("#p").click(function () {
        $(this).toggleClass("clicked");
    });
});
```

Im `<head>` des Dokuments notieren Sie die nötigen Stilangaben. Beachten Sie dabei, dass für `<p>` das Aussehen des Cursors geändert wurde, um zu signalisieren, dass der Container angeklickt werden kann:

```
<style type="text/css">
.clicked {
    background-color:#CCCCCC;
}
p {
    margin:10px 50px 0 10px;
    padding:5px;
    cursor:pointer;
}
</style>
```

Und der relevante HTML-Teil:

```
<p>Schalt mich um.</p>
<p>Schalt mich auch um.</p>
<p>Und mich auch!</p>
```

Haben Sie einen Absatz geklickt, erscheint die jeweilige Zeile hervorgehoben. Klicken Sie denselben Absatz erneut, verschwindet dessen Hervorhebung; die Klasse ist gelöscht.

Zusätzlich zum Parameter *Klassenname* kann der Methode `toggle()` ein zweiter Parameter vom Datentyp *Boolean* übergeben werden. Meist ist dies, anstelle eines konkreten Booleschen Wertes, ein Ausdruck, der von einer Eingangsgröße abhängig ist und im Ergebnis entweder `true` oder `false` ergibt. Im folgenden Beispiel ist es ein Vergleichsausdruck, der von einem Zähler `i` abhängt (im Code fett hervorgehoben):

```
$(document).ready(function() {
    var i = 0;
    $("p").click(function () {
        i++;
        $(this).toggleClass("clicked", (i % 3 == 0));
        $(this).html(" Jeder dritte Klick: " + (i % 3 == 0))
    });
});
```

Werden die Absätze der Reihe nach angeklickt, bekommt nur der dritte Absatz die Hintergrundfarbe zugewiesen. Zuerst wurde eine Variable `i = 0` deklariert, zu deren Wert bei jedem Click-Event 1 hinzugezählt wird: `i++`.

Der Ausdruck `(i % 3 == 0)` gibt `true` zurück, sobald die Variable `i` geteilt durch 3 keinen Rest ergibt, das Ergebnis der Operation also 0 ist. Der Operator `%`, der hier verwendet wird, nennt sich *Modulo*. Er gibt den ganzzahligen Rest einer Division zurück und wird daher auch als *Restwertoperator* bezeichnet.

Die Klasse `clicked` wird hinzugefügt, wenn der Ausdruck `true` zurückgibt – im betrachteten Fall bei jedem dritten Klick.

Des Weiteren bietet Ihnen `toggleClass()` die Möglichkeit, eine anonyme Funktion zu übergeben. (Auch hier könnte optional ein zweiter Parameter `switch` übergeben werden.) Ein allgemeines Beispiel:

```
$("p").click(function() {
    $(this).toggleClass(function() {
        return "classname-" + $(this).index();
    });
});
```

Sanftes Überblenden zwischen zwei Klassen

Eigentlich ist `toggleClass()` nur zu einem harten Umschalten zwischen der einen und der anderen Klasse fähig. Binden Sie jedoch die Erweiterungen von *jQuery UI* ein, kann die Methode als weiteren Parameter eine Zeitdauer entgegennehmen. Diese bestimmt, wie lange ein sanftes Überblenden zwischen den CSS-Eigenschaften beider Klassen dauert. (Ein Beispiel hierzu finden Sie im Praxisteil.)

4.10.2 CSS-Eigenschaften manipulieren

Die Methode `.css()` bietet gleich zweierlei. Zum einen kann sie beliebige, für ein Element gültige CSS-Eigenschaften auslesen, diese bei Bedarf aber auch schreiben. Als wichtigste Rahmenbedingung ist zu beachten, dass die Methode immer nur das erste Element einer jQuery-Collection erfasst, sowohl beim Lesen als auch beim Schreiben. Die Methode besitzt mehrere Signaturen, richtet ihre Arbeitsweise also nach Zahl und Art der übergebenen Argumente.

Methode	Erläuterung
<code>.css(prop)</code>	Holt den Wert des CSS-Properties <code>prop</code> aus dem <i>ersten</i> Element der Collection.
<code>.css(prop, val)</code>	Setzt den Wert des CSS-Properties <code>prop</code> aus dem <i>ersten</i> Element der Collection auf den Wert <code>val</code> .
<code>.css(prop, fn(i, oVal))</code>	Setzt den Wert <code>val</code> für <code>prop</code> durch eine Funktion <code>fn</code> , die die Indexposition <code>i</code> des aktuellen Elements der Collection und den alten Wert <code>oVal</code> von <code>prop</code> übergeben bekommt.
<code>.css({ prop1:val1, ..., propN:valN })</code>	Nimmt ein <i>Objekt</i> (Map) entgegen, das mehrere CSS-Eigenschaften und deren Werte enthält und setzt diese alle gleichzeitig.

Tabelle 4.50 jQuery-Methode zum Lesen und Schreiben von CSS-Eigenschaften

Zum Lesen einer Eigenschaft müssen Sie lediglich den Namen des infrage kommenden Properties (als String) übergeben:

```
$("#div").click(function () {
    // Hintergrundfarbe auslesen:
    var farbe = $(this).css("background-color");
    alert("Dieser Div hat die Hintergrundfarbe "+ color);
});
```

Um eine Eigenschaft zu setzen, wird als *zweites Argument* am einfachsten direkt der gewünschte Wert übergeben:

```

$("div").click(function () {
    // Schriftfarbe auf Orange setzen:
    $(this).css("color", "#ff9900");
});

```

Der zu setzende Wert kann auch aus dem Rückgabewert einer Funktion entnommen werden. Beispielsweise können Sie die Schriftfarbe eines Containers mit der Schriftfarbe eines anderen steuern, indem Sie deren Wert auslesen und übergeben:

```

$("div").click(function () {
    // Schriftfarbe auf Schriftfarbe von #p1 setzen:
    $(this).css("color", function(){
        // Schriftfarbe von #p1 auslesen und zurückgeben:
        return $("#p1").css("color");
    });
});

```

Sollen *mehrere CSS-Eigenschaften* gleichzeitig gesetzt werden, geschieht dies durch ein an die Methode übergebenes JavaScript-Objekt. In jQuery wird dies als *Map* bezeichnet, also eine Reihe von Werten, die einzelnen Eigenschaften zugewiesen sind. Hier bekommt ein Container gleichzeitig Hintergrundfarbe, Textfarbe und den Fettegrad »bold« zugewiesen:

```

$("div").click(function () {
    // Drei Eigenschaften über ein Objekt setzen:
    $(this).css( {"color": "#ff9900",
                  "background-color": "black",
                  "font-weight": "bold"} );
});

```

Die CSS-Eigenschaftsnamen müssen hierbei nur dann unbedingt in *Anführungszeichen* übergeben werden, wenn sie gegen die Konvention für JavaScript-Bezeichner verstoßen. Dies gilt für alle Eigenschaftsnamen, die Bindestriche enthalten. Im vorigen Beispiel sind dies "background-color" und "font-weight". Der Bezeichner "color" wäre erlaubt und dürfte daher im Prinzip auch ohne Anführungszeichen stehen. Allerdings ist eine konsequent durchgehaltene Schreibweise anzuraten.

Eine Alternative besteht im Ausweichen auf konforme Aliase der CSS-Eigenschaften in JavaScript-Binnengroßschreibung (camel caps). In diesem Fall können die Anführungszeichen (müssen jedoch nicht) entfallen:

```

$(document).ready(function() {
    $("div").click(function () {
        // Drei Eigenschaften über ein Objekt setzen:

```

```

$(this).css( { color:"#ff9900",
               backgroundColor:"black",
               fontWeight:"bold" } );
});
});

```

Setzen einer einzigen Eigenschaft – Caveats

Beachten Sie, dass es zwei Wege gibt, mit der Methode `.css()` eine einzelne Eigenschaft zu setzen – Sie verwenden entweder zwei Argumente (Bezeichner und Wert) oder übergeben ein Objekt (nur ein Argument). Hier ein Vergleich:

```

// zwei Argumente:
$(this).css( "color", "#ff9900" );
// ein Argument (Objekt):
$(this).css( {color:"#ff9900"} );

```

Beliebte Fehler sind das Vergessen der Objektklammern und der Versuch, den Wert mit Doppelpunkt innerhalb eines Strings zuzuweisen.

4.10.3 Abmessungen von Containern

Die Breite und vor allem die Höhe von Containern im Layout sind zum Teil schwer vorherzusagen und hängen von zahlreichen Randbedingungen ab. In der Regel werden sie vom Browser erst beim Rendering der Seite bestimmt (*computed height*, *computed width*). jQuery bietet verlässliche Methoden, diese in manchen Browsern nicht greifbaren Größen auszulesen.

Methode	Erläuterung
<code>.height()</code>	Gibt eine Ganzzahl zurück, die dem Pixelwert der Innenhöhe (ohne Padding und Border) in der Viewport-Darstellung des ersten Elements der Collection entspricht (<i>computed height</i>).
<code>.innerHeight()</code>	Gibt eine Ganzzahl zurück, die dem Pixelwert der Innenhöhe inklusive Padding der Viewport-Darstellung des ersten Elements der Collection entspricht (<i>computed height</i>).
<code>.innerWidth()</code>	Gibt eine Ganzzahl zurück, die dem Pixelwert der Innenbreite inklusive Padding der Viewport-Darstellung des ersten Elements der Collection entspricht (<i>computed width</i>).
<code>.outerHeight()</code>	Gibt eine Ganzzahl zurück, die dem Pixelwert der Außenhöhe inklusive Padding und Border der Viewport-darstellung des ersten Elements der Collection entspricht (<i>computed height</i>).

Tabelle 4.51 jQuery-Methoden zum Auslesen von Containerabmessungen

Methode	Erläuterung
<code>.outerWidth()</code>	Gibt eine Ganzzahl zurück, die dem Pixelwert der Außenbreite inklusive Padding und Border der Viewport-Darstellung des ersten Elements der Collection entspricht (<i>computed width</i>).
<code>.width()</code>	Gibt eine Ganzzahl zurück, die dem Pixelwert der Innenbreite (ohne Padding und Border) der Viewport-Darstellung des ersten Elements der Collection entspricht (<i>computed width</i>).

Tabelle 4.51 jQuery-Methoden zum Auslesen von Containerabmessungen (Forts.)

Die ausgegebenen Breiten und Höhen betreffen die Innenbreite des Containers (*width*) oder dessen Innenbreite inklusive Padding (*innerWidth*). Was jQuery als Außenbreite bezeichnet, entspricht der Innenbreite zuzüglich Padding und Rahmenbreite (*outerWidth*). Das Margin, das laut Boxmodell auch zur Gesamtbreite des Objekts zählt, wird in keinem Fall berücksichtigt. Analog ist die Interpretation der Containerhöhen zu verstehen.

Anhand folgenden HTML-Codes werden die Methoden zur Bestimmung der Abmessungen demonstriert:

```
<div id="box1">
  <p id="p1"></p>
  <p id="p2">Lorem ipsum ...</p>
  <p id="p3">Lorem ipsum ...</p>
  <p id="p4">Lorem ipsum ...</p>
</div>
```

Die Ergebnisse der Methodenanwendung auf `div#box1` werden in den ersten (anfangs noch leeren) `<p>`-Container geschrieben:

```
$(document).ready(function() {
  $('p').text(
    'Box1 hat eine width() von ' + $('#box1').width() +
    'px und eine .innerWidth() von ' + $('#box1').innerWidth() +
    'px. Ihre Höhe height() beträgt ' + $('#box1').height() +
    'px, ihre innerHeight()' + $('#box1').innerHeight() +
    'px. Die outerWidth() ist ' + $('#box1').outerWidth() +
    'px, die outerHeight()' + $('#box1').outerHeight() + 'px.';
  });
```

4.10.4 Position von Containern

Ähnlich wie Breiten und Höhen bestimmt sich die Position von Containern (seien diese im Flow oder positioniert bzw. gefloatet) erst beim Rendering der Seite durch den Browser. Oft werden Sie dabei jedoch auf die Koordinaten eines Ele-

ments zugreifen müssen, um entsprechend reagieren zu können. Auch für diese Anforderung bietet jQuery einige Hilfsmittel in Form eigener Methoden.

Methode	Erläuterung
<code>.offset()</code>	Gibt relativ zum Dokument die Koordinaten des ersten Elements der aktuellen Collection zurück.
<code>.offset(coords)</code>	Verschiebt alle Elemente der aktuellen Collection an die übergebenen Koordinaten <code>coords</code> relativ zum Dokument.
<code>.offset(fn(i,coords))</code>	Verschiebt alle Elemente der aktuellen Collection an die Koordinaten relativ zum Dokument, die durch die Funktion <code>fn</code> erzeugt werden. Diese Funktion erhält jeweils den Collection-Index <code>i</code> des Elements und dessen ursprünglichen Koordinaten <code>coords</code> als Parameter. Die Methode <code>.position()</code> gibt hingegen die Koordinaten relativ zum <code>offset parent</code> zurück.
<code>.offsetParent()</code>	Durchsucht die Ancestor-Achse und gibt das erste gefundene positionierte Element zurück.
<code>.position()</code>	Gibt die Koordinaten des ersten Elements der Collection zurück, relativ zum ersten positionierten Element auf dessen Ancestor-Achse (<code>offset parent</code>). Vergleich: Die Methode <code>.offset()</code> gibt hingegen die Koordinaten relativ zum Dokument zurück.

Tabelle 4.52 jQuery-Methoden zur Bestimmung von Containerpositionen

Der HTML-Quellcode dieses Beispiels besteht aus drei ineinander verschachtelten `<div>`-Containern, die zum Teil relativ, zum Teil absolut positioniert sind:

```
<div id="wrap">
  <div id="box1">Dies ist Box 1.
    <div id="box2">
      Dies ist Box 2 in div#box1.
    </div>
  </div>
</div>
```

Das hierfür verwendete CSS lautet wie folgt:

```
div#wrap {
  margin:30px; padding:10px;
  height: 200px; width:500px;
  border:1px solid #aaa;
  background-color:#eee;
```



```

}

div#box1 {
    position:relative;
    height: 120px; width:250px; margin:20px;
    border:1px solid #aaa;
    background-color:#ddd;
}

div#box2 {
    position:absolute;
    top:65px; left:180px;
    height: 70px; width:200px;
    border:1px solid #aaa;
    background-color:#bbb;
}

```

Das Wrapper-Div `#wrap` dient nur als allgemeiner Rahmen und stellt Margin und Padding zur Verfügung. Box 1 ist *relativ* positioniert, aber nicht verschoben und dient so als Koordinatenursprung für Box 2, die *absolut* positioniert ist. Die Koordinaten sind `top:65px` und `left:180px`.

Dem Webdesigner stellt sich jetzt die Frage, an welcher *Koordinatenposition* sich denn nun ein Element in der Seite befindet und welchem Element es gegenüber positioniert ist. Die erste Frage beantwortet jQuery wahlweise mit `.offset()` oder `.position()`, die zweite mit der Methode `.offsetParent()`, die den Bezugspunkt nennt.

Methode `.offset()`

Die `.offset()`-Methode gibt ein Objekt zurück, das in den Eigenschaften `top` und `left` die Koordinaten des übergebenen Objekts enthält. Hier wird der Rückgabewert zwischengespeichert und anschließend über ein Alert ausgegeben:

```

$(document).ready(function() {
    $("div#box2").click(function () {
        var myOffset = $(this).offset();
        alert('Box 2: offset() ist ' +
            myOffset.left + 'px left, ' +
            myOffset.top + 'px top.');
```

Die im Beispiel ausgegebenen Koordinaten sind `top:127px` und `left:250px`. Wie Sie beim Vergleich mit der Positionierung in CSS sehen, sind dies offenbar *nicht*

die Koordinaten gegenüber Box 1, sondern *gegenüber dem Dokumentursprung*. Dies ist auch tatsächlich der Fall und kann somit auch genau die gewünschte Information sein.

Unübersichtlicher wird die Lage, wenn ein Element mittels `offset()` verschoben werden soll. Hierfür wird ein JavaScript-Objekt übergeben, das dem Rückgabewert des vorigen Beispiels gleicht, also zwei benannte Koordinatenwerte enthält. Hier wird Box 2 an die Koordinaten `top:150px` und `left:300px` verschoben, was wiederum gegenüber dem Dokumentursprung zu verstehen ist:

```
$(document).ready(function() {
    $("#div#box2").click(function () {
        $(this).offset( {top:150, left:300} );
        var myOffset = $(this).offset();
        alert('Box 2: offset() ist ' +
            myOffset.left + 'px left, ' +
            myOffset.top + 'px top.');
```

Um den Offset anschließend auszugeben, wurde er nach Zuweisung nochmals in `myOffset` gespeichert. Feste Koordinaten zu übergeben ist nicht unbedingt notwendig. Sie können auch die `.offset()`-Funktion als Quelle für die aktuelle Position heranziehen, um das Objekt anschließend um einen bestimmten Betrag zu verschieben:

```
$(document).ready(function() {
    $("#div#box2").click(function () {
        var myOffset = $(this).offset();
        $(this).offset({ top: myOffset.top+50,
            left: myOffset.left+50 });
        var myOffset = $(this).offset();
        alert('Box 2: offset() ist ' +
            myOffset.left + 'px left, ' +
            myOffset.top + 'px top.');
```

Der Container rückt bei jedem Klick um 50 Pixel nach rechts und unten. Die Methode `.offset()` wird hier dreimal aufgerufen, um die Position erst zu bestimmen, dann zu ändern und schließlich erneut auszulesen.

Methode .offsetParent()

Die Methode `.offsetParent()` gibt den Bezugscontainer, bei dem es sich um den nächsten positionierten Container nach oben innerhalb der Dokumenthierarchie handelt, als jQuery-Objekt zurück:

```
$(document).ready(function() {
    $("#div#box2").click(function () {
        var myOffsetParent = $(this).offsetParent();
        myOffsetParent.css({border:'2px dashed #555'});
        alert('Box 2: offsetParent() ist ' +
            myOffsetParent.attr('id') + '
        ');
    });
});
```

Hier erfahren Sie, dass Box 2 gegenüber Box 1 positioniert ist.

Methode .position()

Die Methode `.position()` ähnelt der Methode `.offset()`, kann jedoch die Koordinaten nur auslesen und nicht verändern. Anders als `.offset()` wird die Entfernung der linken oberen Eckkoordinate von der des `offsetParent` ausgegeben (`.offset()` bezieht sich, wie gesagt, auf den Dokumentursprung). Analog zum vorigen Beispiel angewendet, gibt `.position()` exakt die Werte aus, mit denen `div#box2` laut CSS absolut gegenüber `div#box1` positioniert wurde:

```
$(document).ready(function() {
    $("#div#box2").click(function () {
        var myPosition = $(this).position();
        alert('Box 2: position() ist ' +
            myPosition.left + 'px left, ' +
            myPosition.top + 'px top.');
```

Die ausgegebenen Koordinaten sind `top:65px` und `left:180px`.

4.11 Scrollen und Scrollposition

Ist der Viewport kleiner als das Layout, können sich Seitenbestandteile außerhalb des Sichtbereichs befinden, entweder in horizontaler oder vertikaler Richtung. Sie möchten das Dokument dergestalt scrollen können, dass das betreffende Element sichtbar wird. In Form von `.scrollLeft()` und `.scrollTop()` bietet jQuery zwei Methoden, die diesbezüglich alle Wünsche erfüllen.

Methode	Erläuterung
<code>.scrollLeft()</code> <code>.scrollLeft(integer)</code>	Gibt den Pixelwert der horizontalen Scrollposition des ersten Elements der Collection als Ganzzahl zurück. Alternativ: Scrollt das erste Element der Collection horizontal an die Position, die dem Wert <code>integer</code> entspricht.
<code>.scrollTop()</code> <code>.scrollTop(integer)</code>	Gibt den Pixelwert der vertikalen Scrollposition des ersten Elements der Collection als Ganzzahl zurück. Alternativ: Scrollt das erste Element der Collection vertikal an die Position, die dem Wert <code>integer</code> entspricht.

Tabelle 4.53 jQuery-Methoden zum Bestimmen von Scrollpositionen

Beachten Sie, dass `.scrollTop()` und `.scrollLeft()` nicht die Position eines Elements gegenüber dem Viewport auslesen, sondern die Scrollposition *innerhalb* eines Elements (z. B. des Body-Elements). Eine Positionsbestimmung eines Elements im Layout erfolgt mit `.offset()` (oder gegebenenfalls `.position()`).

Betrachten wir nun folgenden Testaufbau, um das Scrollen des Dokuments auszuprobieren:

```
<div id="box1">
  <b>Scrollposition:</b>
  <div><!-- Ausgabe --></div>
  <a href="javascript:jump()">
    Spring zu Box 2</a>
</div>
<div id="box2"></div>
```

Im Dokument befinden sich zwei `<div>`-Container, von denen der eine (`div#box1`) fixiert positioniert ist und dazu dient, die Scrollposition auszugeben. Der andere (`div#box2`) ist absolut außerhalb des Viewports positioniert. Der Body ist per CSS-Angabe auf eine ausreichende Größe gebracht worden. Hier der relevante Ausschnitt aus dem CSS (die weggelassenen Passagen sind rein präsentativer Natur):

```
body {
  width:5000px;
  height:5000px;
}

#box1{
  position:fixed;
  top:10px;
  right:10px;
```

```

    ...
}

#box2{
    position: absolute;
    top: 350px;
    left: 1200px;
    ...
}

```

Mithilfe von jQuery wurde eine Funktion geschrieben, die den Body (hier interagieren Sie eigentlich mit dem `window`-Objekt) zur Position von Box 2 scrollt. Hierfür werden zunächst per `.offset()` deren Koordinaten ausgelesen. Die ermittelten Werte werden als `.scrollTop()` und `.scrollLeft()` übergeben und über `$(window)` auf das `window`-Objekt angewendet:

```

function jump() {
    // Position von Box 2 bestimmen:
    var sTop = $('#box2').offset().top;
    var sLeft = $('#box2').offset().left;
    // zur ausgelesenen Position scrollen:
    $(window).scrollTop(sTop);
    $(window).scrollLeft(sLeft);
}

```

Anmerkung

Andere potenzielle Kandidaten für das Scrolling wären das Dokument und der Body bzw. der HTML-Container, was als `$(document).scrollTop(sTop)`, `$('#html').scrollTop(sTop)` oder `$('#body').scrollTop(sTop)` geschrieben würde. Dies würde jedoch in den jeweilig verschiedenen Browsern nicht wie erwartet funktionieren.

Die Bestimmung der aktuellen Scrollposition geschieht ebenfalls über `.scrollLeft()` und `.scrollTop()`, diesmal jedoch ohne Argument. Die Ausgabe erfolgt in einen Div innerhalb des fixierten `<div>`-Containers:

```

$(document).ready(function() {
    $(window).scroll( function() {
        $('#box1 div').html($(window).scrollTop() +
            'px vertikal,<br/>' + $(window).scrollLeft() +
            'px horizontal.');
```

Um die Position des Scrollbalkens innerhalb eines Elements zu bestimmen (oder auch, um dort eine Scrollposition anzusteuern) können diese Methoden ebenfalls

eingesetzt werden. In Abwandlung des vorigen Quelltexts bekommt `div#box2` in diesem Fall die Eigenschaft `overflow:scroll` und wird mit einer Reihe von Textabsätzen gefüllt.

Im CSS steht (stark gekürzt) nun Folgendes:

```
#box2{
    position:absolute;
    top:10px;
    left:50px;
    ...
    overflow:scroll;
}
```

Im Scriptblock fügt jQuery zunächst die Textabsätze in Box 2 ein und verwendet hierfür die Methode `.html()`. Die Scrollposition wird wieder in der fest positionierten Box 1 ausgegeben, deren Inhalt anfangs initialisiert wird. An `div#box2` wird (mit `.scroll()`) ein Scroll-Event-Handler gebunden, der über `.scrollTop()` die vertikale Scrollposition ermittelt und in Box 1 schreibt:

```
$(document).ready(function() {
// Textabsätze in Box 2 schreiben:
    for (var i=0;i<100;i++) {
        $('<p>Textabsatz</p>').appendTo('#box2');
    }

// Ausgabebox initialisieren:
    $('#box1 div').html('<b>Scrollposition:</b><br/>'
        + $('#box2').scrollTop() + 'px vertikal');

// beim Scrollen von Box 2 neue Position in Box 1 schreiben:
    $('#box2').scroll( function() {
        $('#box1 div').html('<b>Scrollposition:</b><br/>'
            + $('#box2').scrollTop() + 'px vertikal');
    });
});
```

4.11.1 Animationen mit jQuery

Bereits im Rahmen des Sprachkerns stellt jQuery eine Reihe von Methoden (*Effects*) zur Verfügung, die für grundlegende Animationen wie Ein- und Ausblenden, Zeigen und Verstecken oder Fades eingesetzt werden können.

4.11.2 Zeigen und Verstecken

Einfache Animationsmethoden müssen nicht über Plugins oder das jQuery UI-Tool aufgerufen werden, sondern gehören als »Grundbedürfnis« zum jQuery-Core.

Die erste hier vorgestellte Gruppe ist für das Sichtbarmachen bzw. Verstecken von Seitenelementen zuständig. Ihre genaue Arbeitsweise ist, wie üblich, abhängig von Art und Anzahl der ihr übergebenen Argumente.

Methode	Erläuterung
<code>.hide()</code>	Setzt die Elemente der Collection auf <code>display:none</code> und speichert dabei den ursprünglichen Display-Wert
<code>.hide(d)</code>	Sobald eine Dauer <i>d</i> übergeben wird fadet <code>.hide()</code> die Elemente der Collection über Reduzierung der Abmessungen und Opacity aus und setzt sie dann auf <code>display:none</code> .
<code>.hide(d, fn)</code>	Sobald eine Dauer <i>d</i> übergeben wird fadet <code>.hide()</code> die Elemente der Collection über Reduzierung der Abmessungen und Opacity aus und setzt sie dann auf <code>display:none</code> . Ruft anschließend die Callbackfunktion <i>fn</i> auf.
<code>.show()</code>	Macht versteckte Elemente der Collection sichtbar und verwendet dabei den ursprünglichen Display-Wert.
<code>.show(d)</code>	Sobald eine Dauer <i>d</i> übergeben wird fadet <code>.show()</code> die Elemente der Collection über Vergrößerung der Abmessungen und Opacity ein und setzt sie dann auf die ursprüngliche Display-Eigenschaft.
<code>.show(d, fn)</code>	Sobald eine Dauer <i>d</i> übergeben wird fadet <code>.show()</code> die Elemente der Collection über Vergrößerung der Abmessungen und Opacity ein und setzt sie dann auf die ursprüngliche Display-Eigenschaft. Ruft anschließend die Callbackfunktion <i>fn</i> auf.
<code>.toggle()</code>	Werden ihr keine Argumente übergeben, so wendet <code>.toggle()</code> die Methoden <code>.show()</code> und <code>.hide()</code> auf die Elemente der Collection an.
<code>.toggle(d)</code>	Werden ihr eine Animationsdauer <i>d</i> übergeben, so wendet <code>.toggle()</code> die Methoden <code>.show()</code> und <code>.hide()</code> mit dieser Dauer auf die Elemente der Collection an.
<code>.toggle([d,] fn)</code>	Werden ihr eine Callbackfunktion <i>fn</i> übergeben, so wendet <code>.toggle()</code> die Methoden <code>.show()</code> und <code>.hide()</code> auf die Elemente der Collection an, und ruft anschließend die Callbackfunktion auf. Optional kann zusätzlich eine Animationsdauer <i>d</i> übergeben werden.
<code>.toggle>ShowHide)</code>	Wird ein Boolescher Ausdruck <i>ShowHide</i> übergeben, so wird für <code>true</code> die Methode <code>.show()</code> angewandt, für <code>false</code> die Methode <code>.hide()</code> .

Tabelle 4.54 jQuery-Methoden zum Zeigen und Verstecken

Methoden `.show()` und `.hide()`

Wie die Namen bereits implizieren, dient `.show()` zum Einblenden, `.hide()` hingegen zum Verstecken eines Elementcontainers. Eine dritte Methode, `.toggle()`, wirkt als Kombination beider Grundmethoden, versteckt sichtbare Elemente, blendet dafür versteckte im Gegenzug ein. Die Methoden werden einfach auf eine Collection angewendet und bearbeiten alle deren Items gleichzeitig.

Dies ermöglicht es, einfache Funktionen zu schreiben, die einen Container aus- bzw. einblenden:

```
function hide1() {
    $('#box1').hide();
}

function show1() {
    $('#box1').show();
}
```

Methode `.toggle()`

Die Methode `.toggle()` wertet die Sichtbarkeit des bearbeiteten Elements aus und zeigt oder versteckt es entsprechend:

```
function toggle1() {
    $('#box1').toggle();
}
```

Hierbei wird der betroffene Container praktisch schlagartig (also ohne Verzögerung) versteckt oder gezeigt. Dies entspricht der Anwendung von `display:none`, der Container wird also aus dem Flow entfernt. Diese Funktionen wären (äußerlich) wirkungsgleich:

```
function wegDisplay() {
    $('#box1').css({'display':'none'});
}

function herDisplay() {
    $('#box1').css({'display':'block'});
}
```

Dauer einer Animation

Interessanter wird es schon, wenn man weiß, dass man den jQuery-Methoden einen Parameter übergeben kann, der die Zeitdauer des Ein- und Ausblendens steuert. Hierbei kann es sich um einen Zahlenwert handeln, der als Dauer in Millisekunden verstanden wird, oder um ein Schlüsselwort wie »slow« oder »fast«:


```
function hide2() {
    $('#box1').hide('fast');
}
function show2() {
    $('#box1').show('slow');
}
```

Oder, mit Zeitangabe von drei Sekunden (als Zahl, nicht als String!):

```
function show2() {
    // keine Anführungszeichen um den ms-Wert:
    $('#box1').show(3000);
}
```

Zeitdauerangabe	Dauer in Millisekunden
'fast'	200
(ohne)	400 (Default)
'slow'	600
positive Ganzzahl, größer oder gleich 0	Übergebener Wert als ms

Tabelle 4.55 Mögliche Werte zur Angabe der Duration

Als möglicher zweiter Parameter dient ein Funktionsobjekt, das die Rolle eines Callbacks besitzt, also nach Abschluss des Vorgangs ausgelöst wird:

```
function hide3() {
    $('#box1').hide('fast', function() {
        $('#box2').html("Box 1 ist dann mal weg...");
    });
}
function show3() {
    $('#box1').show('slow', function() {
        $(this).html("Bin wieder da.");
        $('#box2').html("Box 2");
    });
}
```

Diese Funktionen schreiben beim Ausblenden eine Nachricht in eine benachbarte Box und beim Einblenden in die animierte Box (wobei in der zweiten Box der ursprüngliche Inhalt wiederhergestellt wird).

4.11.3 Slides – Zeigen und Verstecken mit Animation

Wie Sie bei den zeitgesteuerten Varianten von `.show()` und `.hide()` sehen konnten, erfolgen das Verbergen und Zeigen über die Manipulation zweier Größen, nämlich der *Abmessungen* (Breite und Höhe) und der *Transparenz* (Opacity).

Manchmal wirkt dies zu aufdringlich, sodass man auf Varianten ausweichen möchte, die nur entweder die Abmessungen oder die Transparenz manipulieren. Die erste Gruppe alternativer Methoden, die hier vorgestellt wird, beschränkt sich auf eine *Animation der Abmessungen*: `.slideDown()`, `.slideUp()` und `.slideToggle()`.

Methode	Erläuterung
<code>.slideDown()</code> <code>.slideDown(d)</code> <code>.slideDown(d, fn)</code>	Blendet versteckte Elemente der Collection durch Vergrößern der vertikalen Abmessung ab der Oberkante des Elements nach unten ein. Die Defaultdauer der Animation beträgt 400 ms, wenn keine Dauer <code>d</code> als Parameter übergeben wurde. (Ruft anschließend eine optionale Callback-Funktion <code>fn</code> auf.)
<code>.slideUp()</code> <code>.slideUp(d)</code> <code>.slideUp(d, fn)</code>	Versteckt die Elemente der Collection durch Verringern der vertikalen Abmessung; setzt sie dann auf <code>display:none</code> . Die Defaultdauer der Animation beträgt 400 ms, wenn keine Dauer <code>d</code> als Parameter übergeben wurde. (Ruft anschließend eine optionale Callback-Funktion <code>fn</code> auf.)
<code>.slideToggle()</code> <code>.slideToggle(d)</code> <code>.slideToggle(d, fn)</code>	Blendet Elemente der Collection durch Animation der vertikalen Abmessung des Elements abwechselnd ein oder aus. Die Defaultdauer der Animation beträgt 400 ms, wenn keine Dauer <code>d</code> als Parameter übergeben wurde. (Ruft anschließend eine optionale Callback-Funktion <code>fn</code> auf.)

Tabelle 4.56 jQuery-Methoden zum Zeigen und Verstecken (Animation)

Methoden `.slideDown()` und `.slideUp()`

Mittels `.slideDown()` wird ein auf `display:none` gesetzter Container eingeblendet, mittels `.slideUp()` wieder versteckt. Im folgenden Beispiel werden drei Boxen unmittelbar nach dem Laden des Dokuments per `slideDown` sichtbar gemacht. Die oberste Box `div#box1` erhält per `.toggle()` einen Click-Event-Handler zugewiesen, der zwischen zwei Funktionen alterniert. Die erste Funktion slidet die mittlere Box aus der Sicht und schreibt per Callback eine Nachricht in die dritte Box. Die zweite Funktion blendet die Box per Slide wieder ein und ändert wiederum den Text in Box 3. Dieser Vorgang lässt sich beliebig wiederholen:

```
$(document).ready(function() {

    // Boxen per Slide einblenden:
    $('#box1, #box2, #box3').slideDown(1000);

    // Box 2 aus- oder einsliden (Klick auf Box 1):
    $('#box1').toggle(
```

```

function() {
    // Verstecken mit Callback:
    $('#box2').slideUp('slow', function() {
        $('#box3').html("Die grüne Box ist weg.");
    })
},
function() {
    // Einblenden mit Callback:
    $('#box2').slideDown('slow', function() {
        $('#box3').html("...und wieder da!");
    })
}
);

});

```

Die HTML-Struktur ist einfach – der umliegende Div `div#wrap` ist lediglich für die Gesamtbreite zuständig. Alle drei inneren Container sind auf `display:none` gesetzt, um sie beim Laden einfahren zu können:

```

<div id="wrap">
    <div id="box1">Bitte hier klicken!</div>
    <div id="box2"></div>
    <div id="box3"></div>
</div>

```



Abbildung 4.13 Animation mit SlideDown

Methode `.slideToggle()`

Wollen Sie einen Container per Slide abwechselnd ein- und ausblenden, können Sie anstelle der Methoden `.slideDown()` und `.slideUp()` auch die Methode `.slideToggle()` einsetzen, die sich in ihrer Arbeitsweise nach den augenblicklichen Display-Eigenschaften des Elements richtet: Ist es ausgeblendet (`display:none`), wird ein `slideDown` ausgeführt, ansonsten ein `slideUp`.

Das Ein- und Ausblenden der mittleren Box aus dem vorigen Beispiel ließe sich mit `.slideToggle()` grundsätzlich genauso gestalten, wobei dann aber nur eine

Callback-Funktion zur Verfügung stünde (auf die Textänderung wurde daher verzichtet):

```
// Box 2 ein- oder aussliden:
$('#box1').click( function() {
    // Zeigen oder Verstecken:
    $('#box2').slideToggle('slow');
})
```

Mithilfe der Methode `.slideToggle()` können Sie aber auch leicht ein Mini-Akkordeonpanel bauen, bei dem bei Klick auf eine Überschriftenleiste abwechselnd zwei Inhaltsbereiche auf- und zugeschoben werden.

Der Quelltext ist von der Struktur her einfach – es werden nur ein paar verschachtelte `<div>`-Container benötigt:

```
<div id="wrap">

    <!-- Panel 1 -->
    <div id="panelhead1">Alice I</div>
    <div id="panel1">Dies war grade nicht sehr...</div>

    <!-- Panel 2 -->
    <div id="panelhead2">Alice II</div>
    <div id="panel2">Der Eingang zum Kaninchenbau lief...</div>

</div>
```

Der Div `div#wrap` dient nur zur Festlegung der Gesamtbreite. Das Wichtigste ist, dass `div#panel2` (der Textbereich des zweiten Panels) zu Beginn mit `display:none` versteckt ist, während der Container `div#panel1` sichtbar ist. Im CSS wurde ihm zur Verdeutlichung explizit `display:block` zugewiesen:

```
#panel1 {
    display:block;
}

#panel2 {
    display:none;
}
```

Die Scriptbefehle, die die Animation beschreiben, sind dank jQuery wieder sehr einfach. Die Grundidee ist, stets zwei Container parallel bearbeiten zu können. Zunächst bekommt die Collection aus beiden Panel-Headern `div#panelhead1` und `div#panelhead2` einen Click-Event-Handler zugewiesen, der das Aus- und Einfahren der Panels triggert.

Dies geschieht über die Click-Callback-Funktion, die `.slideToggle()` auf eine Collection aus `div#panel1` und `div#panel2` anwendet. Da von diesen beiden Containern immer der eine unsichtbar ist, während der andere gezeigt wird, bewirkt dies ein abwechselndes Bewegen beider Panels:

```
$(document).ready(function() {
    $('#panelhead1,#panelhead2').click( function () {
        $('#panel1,#panel2').slideToggle(400);
    });
});
```

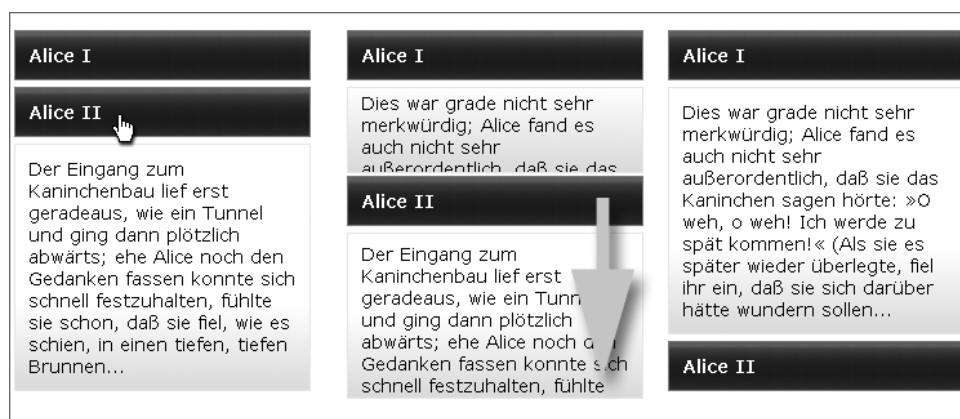


Abbildung 4.14 Mini-Akkordeon mit `slideToggle()`

Ein ausführlicheres Beispiel zum Bau von Akkordeonpanels finden Sie weiter hinten im Buch im Praxisteil.

4.11.4 Fades – Zeigen und Verstecken über Opacity

So wie die Slide-Funktionen eine Variante von Show und Hide darstellen, die sich auf die Animation der Abmessungen beschränkt, so arbeiten die Fade-Methoden mit der Manipulation der Transparenz. Im Hintergrund untersucht jQuery den Browser und wendet beispielsweise im Internet Explorer den Alpha-Filter an; in Browsern, die dies unterstützen, findet die CSS3-Eigenschaft `opacity` Verwendung. Summa summarum arbeiten `.fadeIn()`, `.fadeOut()` und `.fadeTo()` damit browserübergreifend.

Methode	Erläuterung
<code>.fadeIn()</code>	Blendet (versteckte) Elemente der aktuellen Collection durch Erhöhung der Opacity auf den Wert 1 ein.
<code>.fadeIn(d)</code>	(Akzeptiert optional Parameter zu Dauer <code>d</code> des Einblendevorgangs sowie eine Callback-Funktion <code>fn</code> , die nach Ende des Vorgangs aufgerufen wird.)
<code>.fadeIn(d, fn)</code>	

Tabelle 4.57 jQuery-Methoden zum Zeigen und Verstecken (Fades)

Methode	Erläuterung
<code>.fadeOut()</code> <code>.fadeOut(d)</code> <code>.fadeOut(d, fn)</code>	Versteckt (sichtbare) Elemente der aktuellen Collection durch Verringerung der Opacity bis zum Wert 0. (Akzeptiert optional Parameter zu Dauer <code>d</code> des Ausblendevorgangs sowie eine Callback-Funktion <code>fn</code> , die nach Ende des Vorgangs aufgerufen wird.)
<code>.fadeTo(d, op)</code> <code>.fadeTo(d, op, fn)</code>	Verändert die Opacity der Elemente der aktuellen Collection über die Dauer <code>d</code> hin zum Opacity-Wert <code>op</code> . (Akzeptiert optional eine Callback-Funktion <code>fn</code> , die nach Ende des Vorgangs aufgerufen wird.)

Tabelle 4.57 jQuery-Methoden zum Zeigen und Verstecken (Fades) (Forts.)

Die Fade-Methoden arbeiten analog zu den bisher vorgestellten Grundanimationen. In Anlehnung an das Akkordeonbeispiel, das sich auf `.slideToggle()` stützt, soll ein vergleichbares Szenario durch Überblenden zweier Inhalte erreicht werden. Dies soll mit `.fadeOut()` und `.fadeIn()` geschehen. Hierbei soll zwischen zwei Inhalten durch Klick auf denselben Panel Header gewechselt werden. Dieser muss also jeden zweiten Klick sein Vorgehen ändern – mit einer normalen Klickbindung geht dies nicht. Wir nehmen also die bereits vorgestellte `.toggle()`-Eventbindung zu Hilfe, die pro Klick zwischen zwei oder mehr Funktionen alternieren kann.

Zunächst ein Blick auf das HTML (selbstverständlich gekürzt):

```
<div id="wrap">
  <div id="panelhead">Fading Alice</div>
  <!-- Panel 1 -->
  <div id="panel1">Dies war grade nicht sehr...</div>
  <!-- Panel 2 -->
  <div id="panel2">Der Eingang zum Kaninchenbau lief...</div>
</div>
```

Das CSS ist nicht weiter bemerkenswert. Wichtig ist nur, `div#wrap` zu positionieren (hier mit `position:relative`), damit die beiden Inhaltspanels an deckungsgleiche Positionen gebracht werden können (dies geschieht hier durch absolute Positionierung). Panel 2 ist dabei »vorbeugend«, gewissermaßen routinemäßig, über `display:none` ausgeblendet:

```
div#wrap {
  ...
  position:relative;
}
```

```

#panelhead {
    height:20px;
    ...
    cursor:pointer;
}

#panel1, #panel2 {
    ...
    position:absolute;
    top:40px;
}

#panel1 {
    display:block;
}

#panel2 {
    display:none;
}

```

Das Script ist, typisch für jQuery, wieder sehr einfach. An den Panel Header werden mittels `.toggle()` »onclick« zwei Funktionen gebunden, die alternierend getriggert werden. Beide blenden jeweils das eine oder das andere Panel aus und dessen Gegenpart ein:

```

$(document).ready(function() {
    $('#panelhead').toggle(
        function () {
            $('#panel2').fadeIn(1100);
            $('#panel1').fadeOut(600);
        },
        function () {
            $('#panel1').fadeIn(1100);
            $('#panel2').fadeOut(600);
        }
    );
});

```

Beachten Sie, dass die Ein- und Ausblendzeiten nicht gleich sind! Es ist sinnvoll, das Einblenden im Verhältnis etwas zu verzögern, um nicht eine verwirrende optische Überlagerung beider Inhalte zu erzielen (wie in der Darstellung der Übergangsphase in Abbildung 4.15).



Abbildung 4.15 Übergang mit `fadeOut()` und `fadeIn()`

4.11.5 Utility-Methoden für Animationen

Um die Ihnen nun bekannten Animationen weiter zu steuern, bietet jQuery zwei Hilfsmethoden, die sich in verkettete Aufrufe von Animationsmethoden einreihen können. Die erste bewirkt eine Verzögerung der anschließenden Animation, die zweite löscht die aktuell laufende Animation, um sofort zur folgenden überzugehen.

Methode	Erläuterung
<code>.delay(d [, name])</code>	Nimmt einen Wert <code>d</code> als Verzögerungszeit für folgende Items der aktuellen Eventkette entgegen. (Der optionale Parameter <code>name</code> nennt den Namen einer alternativen Eventkette.)
<code>.stop()</code> <code>.stop(bCQ)</code> <code>.stop(bCQ, bCSS)</code>	Beendet augenblicklich die Animationen von Elementen der aktuellen Collection. Wird für den Booleschen Parameter <code>bCQ</code> der Wert <code>true</code> übergeben, wird die Effektkette gelöscht (<code>clear queue</code>). Default ist <code>false</code>. Wird für den Booleschen Parameter <code>bCSS</code> der Wert <code>true</code> übergeben, springen die CSS-Eigenschaften des Elements auf die für das Animationsende vorgesehenen Werte (<code>jumpToEnd</code>). Default ist <code>false</code>.

Tabelle 4.58 jQuery-Utility-Methoden für Animationen

Methode `.delay()`

Zwei oder mehr Animationsfunktionen können verkettet, also auf die gleiche Collection angewendet werden. Üblicherweise wartet jQuery, bis die erste Animation vollständig abgeschlossen ist, und beginnt dann unverzüglich mit der Abarbeitung des nächsten Effekts. Sie können, in Anlehnung an ein vorausgegan-

nes Beispiel, eine Reihe von Containern per Slide ein- und dann gleich wieder ausblenden:

```
$('#box1, #box2, #box3').slideDown(1000).slideUp(1000);
```

Dies geschieht jedoch in vielen Fällen ungewollt zügig. Soll jQuery eine »Denkpause« einlegen, bauen Sie ein »Verzögerungsglied« in die Kette mit ein. Hierzu dient die Methode `delay()`, die eine *Verzögerungszeit* als Parameter erwartet. So wird das Ausblenden per Slide um zwei Sekunden verschoben:

```
$(document).ready(function() {
    // alle Boxen per Slide einblenden:
    $('#box1, #box2, #box3')
        .slideDown(1000)
    // zwei Sekunden abwarten:
    .delay(2000)
    // und per Slide wieder ausblenden:
    .slideUp(1000);
});
```

Anmerkung

Außer Zahlenwerte für Millisekunden können an `.delay()` auch die Keywords `fast` und `slow` übergeben werden. Diese repräsentieren dann jeweils Verzögerungszeiten von (respektive) 200 und 600 Millisekunden. Einen Defaultwert gibt es jedoch nicht (oder treffender gesagt: Er ist gleich 0) – ein Aufruf von `.delay()` ohne Argument bewirkt rein gar nichts.

Methode `.stop()`

Mit der Methode `.stop()` können Sie eine Animation für das aktuelle Element abbrechen. Dies ist jedoch nicht ganz so simpel, wie es zunächst erscheint, wie sich aus dem vorigen Beispiel herleiten lässt. Beginnen wir damit, im `.ready()`-Block noch eine weitere Anweisung zu hinterlegen, die alle animierten Boxen klickbar macht – in der Absicht, den `SlideUp` nach einem Klick auf eine (beliebige) von ihnen abzubrechen. Gewünschtes Ergebnis: Die Boxen sollen dann *nicht* wieder per Slide versteckt werden:

```
$('#box1, #box2, #box3').click(function(){
    $(this).stop();
});
```

Dies ist im Prinzip zwar korrekt geschrieben, jedoch entspricht das Ergebnis nicht den Erwartungen – alle Container werden (meistens jedenfalls) dennoch weggeslidet, der angeklickte verschwindet dabei (unter Umständen jedenfalls) sogar früher! Erratisch? Scheinbar. Was ist also die Ursache?

Nochmals: Die Methode `.stop()` löscht die *aktuelle Animation* des *Kontextelements*. Lassen Sie sich dies auf der Zunge zergehen, so wird klar, dass deshalb, erstens, nur das angeklickte Element betroffen ist (alle anderen werden stur nach Vorschrift weiter animiert!) und, zweitens, nur die zum Zeitpunkt des Aufrufs der `.stop()`-Methode aktive Animationsmethode relevant ist.

Okay, wenn dies nun aber `.delay()` ist? In diesem Fall wird die *Verzögerung* gelöscht und *sofort* zur folgenden Animation der Kette (den Begriff *Kette* sollten Sie sich merken) gesprungen. Da dies das SlideUp ist, slidet das Element *augenblicklich* außer Sicht. Sollten Sie zufällig erst *nach* den zwei Sekunden Verzögerung geklickt haben,⁵ wird stattdessen das SlideUp gelöscht und der Container bleibt Ihnen (wenigstens teilweise) erhalten.

Was ist zu tun? Nehmen wir die zwei potenziellen Argumente zu Hilfe, die `.stop()` akzeptiert. Beide sind Boolesche Werte (Defaultwert ist in beiden Fällen `false`, sodass es nichts macht, wenn kein Argument übergeben wird), wobei der erste (`clear queue`) aussagt, dass die aktuelle Animationskette komplett gelöscht werden soll.

Das bedeutet, dass die gerade laufende Animation, aber auch alle, die auf sie folgen, gestoppt werden. Unterm Strich läuft dies darauf hinaus, dass es egal ist, ob Sie während der Delay-Animation oder während der SlideUp-Animation eines Elements klicken (... nun ja, mit eventuell halb weggeslideten Containern müssten Sie dann leben). Der Code wird jedenfalls wie folgt modifiziert:

```
$('#box1, #box2, #box3').click(function(){
    $(this).stop(true);
});
```

Nach wie vor müssen *alle drei* Container angeklickt werden, um sie zu erhalten: Das Löschen der Kette betrifft *nur* das Kontextelement. Hier hilft Ihnen der zweite Parameter (`jumpToEnd`) auch nicht weiter, denn er bestimmt lediglich, dass durch eine Methode animierte CSS-Eigenschaften umgehend auf den Zielwert gesetzt werden. (Wir kommen später noch darauf zurück.)

Die Lösung im betrachteten Fall liegt in der Modifikation des Selektors: Es kann nicht mit `$(this)` gearbeitet werden, weil dann nicht die gesamte animierte Collection betroffen ist, sondern eben nur das angeklickte Element. Schreiben Sie also:

```
$('#box1, #box2, #box3').click(function(){
    $('#box1, #box2, #box3').stop(true);
});
```

⁵ Allerdings – ein schneller zweiter Klick löscht auch die Folgeanimation. Sagten wir nicht etwas wie »erratisch«?

Warum geht dies nun, obwohl nach wie vor nur ein Element angeklickt wird? Weil innerhalb der Funktion eine neue Collection gebildet wird, die aus allen animierten Elementen besteht (welch Zufall), sodass zuverlässig die Animationsketten jedes beteiligten Containers gestoppt werden. Eine etwas ökonomischere Methode, dies zu schreiben, könnte etwa so lauten:

```
var anim = $('#box1, #box2, #box3');

anim.click(function(){
    anim.stop(true);
});
```

Warum sollte man Collections speichern?

Es kann sich durchaus lohnen, eine mehrfach verwendete Collection in eine Variable zu referenzieren. Sie sollten dabei aber nicht vergessen, dass der Aufruf der jQuery-Funktion mit einem Selektor bedeutet, dass das gesamte Dokument nochmals durchsucht wird. Dies sollten Sie nicht leichtfertig veranlassen, wenn Sie sich das Ergebnis einer solchen Operation auch einfach »merken« können.

4.11.6 Animation mehrerer CSS-Parameter

Die Methode `.animate()` ist jQuerys »eierlegende Wollmilchsau« bzw. seine »Geheimwaffe« bezüglich Animationen. Die unscheinbar benannte Methode bietet bei genauem Hinsehen enorme Möglichkeiten, die (zunächst leider prinzipbedingt nur in Ansätzen) hier unbedingt vorgestellt werden sollten.

Methode	Erläuterung
<code>.animate(p[,d,e,fn])</code>	Ermöglicht gleichzeitige Animation verschiedener CSS-Properties <code>p</code> . (Nimmt optionale Parameter zu Dauer <code>d</code> , Easing <code>e</code> und eine Callback-Funktion <code>fn</code> entgegen.)

Tabelle 4.59 Allgemeine jQuery-Methode für Animationen

Sie könnten sich jetzt auf den Standpunkt stellen, dass die bisher vorgestellten Animationsmethoden für Ein- und Ausblenden, Slides und Fades nur Spezialfälle darstellen, die ebenso durch `.animate()` abgedeckt werden könnten. Das ist im Prinzip zwar richtig, macht jedoch die anderen Methoden aufgrund ihrer einfacheren Anwendung nicht überflüssig. Die Methode `.animate()` als allgemeiner Ansatz jedoch verdient hier eine ausführlichere Abhandlung.

Zunächst stellen wir Ihnen die vier Argumente vor, die die Methode akzeptiert. Das erste *muss* vorhanden sein, die folgenden drei sind optional, dürfen also ent-

fallen (es gibt demnach Defaultwerte, bzw. die Parameter können schlicht übergeben werden). In dieser Reihenfolge übergeben Sie:

1. **CSS-Eigenschaften** (obligatorisch)

Die zu animierenden Eigenschaften nebst deren Zielwerten. Diese werden stets als JavaScript-Objekt übergeben, das `property-value`-Pärchen enthält.

2. **Zeitdauer**

Die Dauer der angeordneten Animation in Millisekunden (als Zahl). Die Keywords `slow` und `fast` werden im Rahmen eines alternativ möglichen *Duration-Objekts* (das auch noch mehr kann) ebenfalls akzeptiert.

3. **Easing**

Soll die Animation nicht gleichmäßig erfolgen, beispielsweise schnell beginnen, aber langsam auslaufen, wird dieser Parameter übergeben.

4. **Callback-Funktion**

Ein Funktionsobjekt (oder eine Referenz darauf), das wie üblich getriggert wird, sobald die Animation beendet ist.

Nun aber genauer und der Reihe nach:

Die Property-Map – zu animierende CSS-Eigenschaften

Der erste und wichtigste Parameter ist eine *Eigenschaften-Map*, wie jQuery ein zu diesem Zweck verwendetes JavaScript-Objekt nennt. Einiges hierzu (wie beispielsweise die gelegentliche Übergabe der Eigenschaftsnamen als String) wurde bereits bei der Vorstellung der Methode `.css()` gesagt.

Stellen wir zunächst fest, dass nicht einfach eine »irgendwie« zu animierende *Eigenschaft* genannt wird, sondern deren Name und deren *Zielwert*. Dies könnten die Höhe und die Breite eines zu animierenden Containers sein:

```
// eine "Map" ist immer ein Objektliteral:
{ height:'300px', width:'600px' }
```

Maßeinheit übergeben oder nicht?

Für Pixelwerte kann auf die Angabe einer Maßeinheit verzichtet werden. jQuery nimmt von sich aus an, dass es sich um Pixel handelt, wenn das Property eine Abmessung beschreibt. Andere Einheiten (`%`, `em`, `pt` etc.) müssen jedoch angegeben werden.

In diesem Fall wird also von den aktuell für das Objekt gültigen Abmessungen ausgegangen, die über den Zeitraum der Animation (dazu gleich noch mehr) auf den Zielwert gebracht werden. Besitzt eine der angesprochenen Eigenschaften des animierten Objekts bereits den Zielwert, passiert in diesem Fall nichts. Es

können beliebig viele Eigenschaften im Objekt beschrieben werden, die während der Animation alle gleichzeitig bearbeitet werden:

```
$('p').animate({height:'300px', width:'600px'});
```

Vorsicht – obwohl das Argument auf den ersten Blick aussieht, wie eine CSS-Regel (geschweifte Klammern außen, Doppelpunkt zwischen Name und Wert), bei der man den Selektor weggelassen hat, ist die Schreibweise im Detail anders – die Eigenschaften werden nicht mit Semikolon beendet, sondern (es ist ja ein JavaScript-Objekt) durch Kommata getrennt.

Animation von Farbwerten

Sollen Farbwerte mit jQuery animiert werden, übersteigt dies die Möglichkeiten des jQuery-Core. Entsprechende Anfragen (sei es für `color` oder `backgroundColor`) werden also ignoriert. Jedoch kann eine Farbanimation erfolgen, wenn *jQuery UI* oder hierfür konzipierte Plugins, wie *Color Animations*⁶, eingebunden sind. Ein vertiefendes Beispiel dazu finden Sie im Praxisteil.

Duration – Dauer der Animation

Eine Animation läuft auch ohne Angabe einer Zeitdauer. In diesem Fall wird einfach von einer Defaultzeitdauer von 400 ms ausgegangen. Expliziter wird es, sobald ein Wert übergeben wird, der den Zeitrahmen setzt. Gehen wir für den Anfang davon aus, dass es sich bei der Duration um einen einfachen Zahlenwert handeln wird, beispielsweise 1.300 für »1.300 Millisekunden«. Das ist selbsterklärend. Aber was geschieht nun »intern«?

Nehmen wir an, es soll die Left-Position eines Containers animiert werden, der sich bei `left:0` befindet und nach `left:1000px` bewegt werden soll. Das Framework kennt nun Anfangs- und Endposition und weiß, wie lange es sich zum Erreichen der Endposition Zeit zu nehmen hat:

```
$('p').animate( {left:'100px'}, 1300 );
```

Damit eine Animation erzeugt wird, müssen Zwischenpositionen errechnet werden (Tweening) – bezeichnen wir diese als *Keyframes*. jQuery arbeitet hierfür mit einer fest eingestellten Schrittdauer von 13 ms. Eine Dauer von 1.300 ms bedeutet also 100 Schritte bzw. Keyframes. Diese Anzahl wird verwendet, um die Zwischenwerte der CSS-Eigenschaften zu errechnen. Bei einer Strecke von 1.000 px, die der Beispielcontainer zurücklegen soll, kommen wir damit auf 10 px pro Schritt.

⁶ Nähere Informationen siehe unter <http://plugins.jquery.com/project/color>

Anmerkung

Anstelle eines Zahlenwerts für Millisekunden können auch die Keywords `slow` und `fast` übergeben werden. Dies muss jedoch (was es komplizierter macht), im Rahmen eines Optionsobjekts geschehen.

Hier ein Anschauungsbeispiel, das wir im Anschluss allerdings nicht weiterverfolgen werden:

```
$('p').animate( {left:'100px'}, {duration:'slow'} );
```

Easing – Verlaufskurve der Animation

Was aber, wenn Sie nun keinen gleichmäßigen Ablauf der Animation haben möchten, wie er sich durch das Erzeugen der Keyframes im regelmäßigen Abstand von 13 ms ergibt? In diesem Fall wird nicht der Abstand der Keyframes verändert (diese bleiben gleichmäßig verteilt), sondern die in jedem Keyframe gespeicherten Eigenschaftswerte werden durch eine Funktion neu gewichtet (verkleinert oder vergrößert, je nachdem). Die Funktion, die dies vornimmt, bezeichnet man als *Easing*-Funktion.

Um Ihnen dies zu verdeutlichen, wird hier die Wertänderung (vom Ausgangswert zum Zielwert) und der Zeitablauf (vom Start bis zum Ende der Animation) in einem Graph gegeneinander aufgetragen. In der angenommenen Ausgangskonfiguration ergibt dies eine gerade Linie (siehe Abbildung 4.16, links).

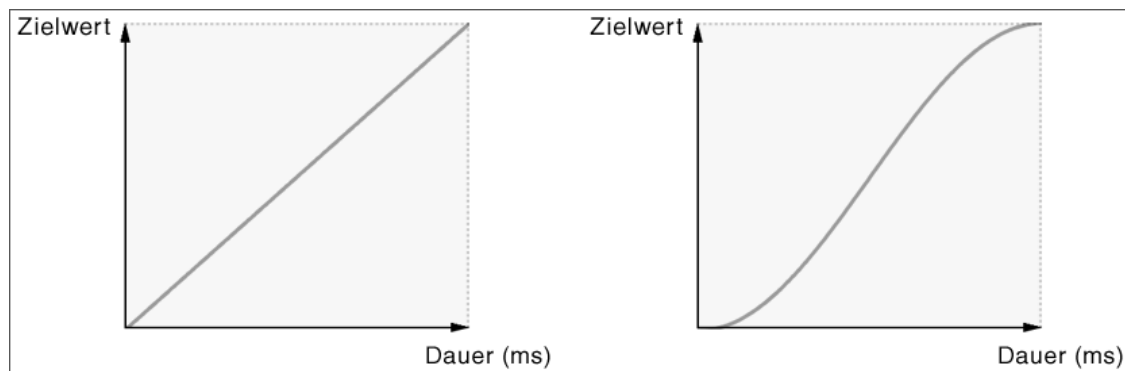


Abbildung 4.16 Lineares und Sinus-Easing (»Swing-Kurve«)

Verändern Sie den Wert nun nicht gleichmäßig, sondern anfangs pro Schritt etwas weniger, dann stärker und gegen Ende wieder etwas weniger, erhalten Sie eine Kurve wie in Abbildung 4.16, rechts. Eine solche Kurve lässt sich von einer Sinus-Funktion ableiten, und in der Tat geschieht genau dies. jQuery bezeichnet diese Kurve, die eine Animation sanfte beginnt, dann steigert und schließlich sanft auslaufen lässt, als *Swing-Kurve*. Es handelt sich hier um eine der eingebauten Easing-Funktionen, die zur Steuerung der Animationen zur Verfügung stehen.

Genauer gesagt handelt es sich bei *Swing* sogar um die *Default-Easing-Funktion*: Wird kein Easing-Parameter übergeben, wird automatisch diese Kurve der Animation zugrunde gelegt. Die (vorläufig) einzige Alternative, die vorhanden ist, besteht im anfangs hergeleiteten Verlauf, den jQuery als *linear* bezeichnet. So ergeben sich die beiden zulässigen Werte für Easing: 'swing' (Default) und 'linear'. Eigentlich, wie eben konstatiert, müsste der Wert 'swing' gar nicht extra übergeben werden – hier aber zur Demonstration:

```
$('p').animate( {left:'100px'}, 1300, 'swing' );
```

Wenn Sie es explizit »linear« möchten, müssen Sie dies auch sagen:

```
$('p').animate( {left:'100px'}, 1300, 'linear' );
```

Weitere Easing-Funktionen benötigt?

Werden spezielle Animationskurven benötigt, beispielsweise eine, die schnell beginnt und langsam ausläuft, oder eine, bei der sich das Objekt elastisch um die Zielposition einpendelt (Bouncing), kann einfach ein zusätzliches Plugin bemüht werden, das solche (und viele weitere) Easing-Methoden zur Verfügung stellt: das *jQuery Easing-Plugin*.

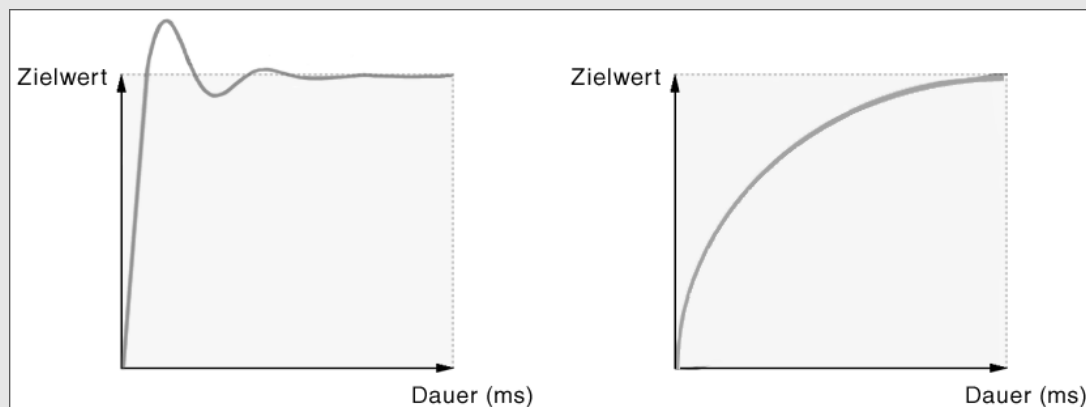


Abbildung 4.17 Zwei Easing-Kurven aus dem Easing-Plugin

Mithilfe dieses Plugins verfügen Sie über fast drei Dutzend weitere Animationsverläufe, die unter »bouncing«, »elastic« oder »exponential« firmieren und jeweils in den Varianten »Ease-In«, »Ease-Out« bzw. »Ease-InOut« vorliegen. (Reicht Ihnen das nicht, können Sie immer noch eigene Easing-Methoden schreiben.) Das Plugin ist Teil von *jQuery UI*. Auf das Einbinden und den Einsatz wird später noch im Praxisteil näher eingegangen, in dem Sie auch weitere Hintergründe zu Easing-Funktionen erfahren.

Callback – was passiert danach?

Das letzte optionale Argument, das `.animate()` entgegennimmt, ist eine Callback-Funktion. Diese wird nach Abschluss der Animation ausgelöst:

```

$('p').animate(
  {left:'100px'},
  1300,
  'linear' ,
  function() {
    alert("Mission abgeschlossen! Bin am Ziel.")
  }
);

```

Nachdem der theoretische Teil hiermit abgedeckt wurde, geht es jetzt »in medias res« – versuchen wir uns an ein paar einfachen Beispielen zu `.animate()`.

Methode `.animate()`

Animiert werden sollen mehrere `<div>`-Container, die der Einfachheit halber alle mit denselben CSS-Eigenschaften belegt sind (hier leicht gekürzt):

```

#box1, #box2, #box3, #box4, #box5 {
  padding:10px;
  height:50px;
  width:250px;
  margin-bottom:10px;
  border:1px solid #ccc;
  ...
}

```

Animiert werden sollen Höhe, Breite und Transparenz (wobei Letztere nicht angegeben ist – wir gehen hier vom Defaultwert `opacity:1.0` aus). Jede Animation (bis auf die letzte) soll per Klick auf den jeweiligen Container gestartet werden.

Zunächst sollen für die ersten zwei Boxen `div#box1` und `div#box2` jeweils Breite und Höhe animiert werden. Beide Abschnitte stehen im `.ready()`-Block, der hier der Übersichtlichkeit wegen weggelassen wurde:

```

$('#box1').click(function() {
  $(this)
    .animate( { width:'500px' }, 1000)
    .animate( { height:'150px'}, 1000) ;
});
$('#box2').click(function() {
  $(this)
    .animate( { width:'500px',
                height:'150px' }, 1000);
});

```

Obwohl hier zweimal die gleichen Größen mit gleichen Zielwerten und Dauern animiert werden, geschieht nicht dasselbe. Bei Box 1 werden einzeln nacheinan-

der zuerst die Breite und anschließend die Höhe animiert. Bei Box 2 geschieht die Manipulation beider Größen gleichzeitig. Es findet also eine *Kettenbildung* statt, die der Reihenfolge der aneinandergehängten Animationsaufrufe entspricht. Im Grunde ist dies naheliegend (aber nicht in Stein gemeißelt, wie im weiteren Verlauf noch gezeigt wird).

An Box 3 werden die Höhe und die Transparenz geändert, um die Box auszublenden. Dies soll aber nicht mit dem üblichen Swing-Easing geschehen, sondern »linear«, weshalb ein entsprechendes Easing-Argument übergeben wurde. Eine Callback-Funktion meldet den Abschluss der Animation:

```
$('#box3').click(function() {
    $(this)
        .animate( {opacity:'0',height:'0'},
            1500,
            'linear',
            function(){
                alert('Bin dann mal weg!');
            }
        );
});
```

Betrachten wir eine Animation von Box 4, um tiefer in das Eigenschaftsobjekt und seine Möglichkeiten einzusteigen. Sie brauchen nämlich keinen konkreten Zielwert anzugeben. Da jQuery den Startwert ohnehin ausliest, können Sie diesen auch zur Grundlage einer *Zielwertberechnung* machen:

```
$('#box4').click(function() {
    $(this)
        .animate(
            {width:'+=50px'},
            500
        );
});
```

Hier wird die Breite pro Animation um 50 px vergrößert – der Zielwert berechnet sich aus `zielwert = (startwert + aenderung)`. Analog ist auch eine Verringerung über `-=` möglich. (Eine Multiplikation oder Division mit einem Faktor gibt es jedoch nicht.)

Eine andere Art der Kettenbildung lernen Sie über folgendes Beispiel kennen. Über Box 5 soll gehovert werden können, wobei sich die Box bei Mouseover ausdehnt und bei Mouseout wieder in den Ausgangszustand zurückfährt. Dies wird mit dem Convenience-Eventbinder `.hover()` erreicht, der zwei Callbacks als Argumente erhält. Beide enthalten eine Animation:

```

$('#box5').hover(
  function() {
    $(this).animate(
      {width:'500px'},
      500
    ),
  function() {
    $(this).animate(
      {width:'250px'},
      500
    )
  }
);

```

Dies funktioniert so weit sehr gut. Hovern Sie jedoch mehrmals kurz hintereinander über dem Element, indem Sie den Mauszeiger schnell darüber hinein- und wieder hinausbewegen, sehen Sie, dass sich die Box dies »merkt«. Alle angestoßenen Hovers werden der Reihenfolge nach ausgeführt – die Animation »läuft nach«. Auch dies ist wieder eine Animationskette – die auszuführenden Funktionen werden sozusagen auf einen »Stack« gelegt und abgearbeitet.

Betrachten wir nun abschließend eine Möglichkeit, die Verkettung auszuhebeln, bevor wir uns Mittel verschaffen, sie korrekt zu steuern. Gehen wir zum ersten Beispiel zurück:

```

$('#box1').click(function() {
  $(this)
    .animate( { width:'500px' }, 1000)
    .animate( { height:'150px' }, 1000);
});

```

Diese Methoden werden nacheinander ausgeführt, so wie sie nacheinander zugewiesen wurden. Möchten Sie dies anders haben, können Sie veranlassen, dass eine Animation nicht zu dem Zeitpunkt ausgeführt wird, der ihrer Einbindung entspricht, sondern gleich zu Beginn – und zwar zusätzlich und gleichzeitig mit der ersten Animation.

De facto wird diese »aus der Reihe« geholte Animation zur ersten Animation »addiert«. Hierzu benötigen Sie das vorhin kurz gestreifte *Optionsobjekt*. Dieses kann neben `duration` noch eine Eigenschaft `queue` enthalten, die bestimmt, ob sich eine Animation in die Kette einreicht (`queue:true`) oder nicht (`queue:false`). Folgende kleine Manipulation lässt beide Animationen plötzlich synchron starten:

```

$('#box1').click(function() {
  $(this)
    .animate( {width:'500px'}, 1000)

```

```

        .animate( {height:'150px'},
                    {duration:1000, queue:false}
        );
    });

```

Nehmen wir noch eine weitere Animation hinzu, die die Transparenz auf 30 % reduziert, und linken diese an zweiter Stelle ein:

```

$('#box1').click(function() {
    $(this)
    .animate( { width:'500px' }, 1000)
    .animate( {opacity:'0.3'}, 1500)
    .animate( {height:'150px'},
              {duration:1000, queue:false}
    );
});

```

Die Reduktion der Transparenz wird erst zum Schluss vorgenommen, da die beiden anderen Animationen gleichzeitig starten. Auch die Transparenzänderung könnte nun aus der Kette genommen werden, um sie gleichzeitig mit den anderen zu starten. Da sie aber eine andere Dauer hat, läuft sie nach Abschluss der anderen beiden Animationen noch weiter. Sie können die Konfiguration von Einzelanimationen mit dem Optionsobjekt dabei noch weiter treiben:

```

$('#box1').click(function() {
    $(this)
    .animate( { width:'500px' }, 1000
    )
    .animate( {opacity:'0.3'},
              {duration: 2000,
               queue: false,
               complete: function(){
                   alert('Habe fertig!');
               }
    })
    .animate( {height:'150px'},
              {duration: 1500,
               queue: false,
               easing:'linear'}
    );
});

```

Das Optionsobjekt kann sogar noch mehr, wobei an dieser Stelle eine noch ausführlichere Erläuterung den Rahmen sprengen würde. Wir möchten daher in diesem Fall ausnahmsweise auf die offizielle Dokumentation verweisen. Da jetzt aber

bereits seit einiger Zeit von der Animationskette *Queue* die Rede war, sollen jetzt endlich ein paar bislang vernachlässigte Hintergründe dazu aufgezeigt werden.

4.11.7 Die Queue – Warteschlange für Effekte

In den vorangegangenen Abschnitten wurde deutlich gemacht, dass eine Reihe von Animationen, die einer Collection zugewiesen werden, eine Kette bildet. Dies gilt auch für an einem Objekt getriggerte Animationen, die über Event Handler gestartet werden – auch diese werden in der Reihenfolge der Triggerung auf einen Stapel gelegt und entsprechend abgearbeitet. In der Summe bezeichnet jQuery die an einem Objekt abzuarbeitenden Animationen als *Queue* (Kette) und bietet uns auch drei Methoden, in diese Verarbeitungskette einzugreifen. Es sind dies die Methode `.queue()`, die mit drei Signaturen auftritt, sowie die Methoden `.clearQueue()` und `.dequeue()`.

Methode	Erläuterung
<code>.queue()</code> <code>.queue(name)</code>	Gibt die Default-Effektkette (Queue) <code>fx</code> der Items der Collection zurück. Wird ein Bezeichner <code>name</code> übergeben, wird stattdessen die entsprechend benannte Effektkette zurückgegeben.
<code>.queue(newQ)</code> <code>.queue(name, newQ)</code>	Ersetzt die Default-Effektkette <code>fx</code> durch ein Array aus neuen Funktionen. Wird als erster Parameter ein Bezeichner <code>name</code> übergeben, wird die entsprechend benannte Effektkette ersetzt.
<code>.queue(fn(next))</code> <code>.queue(name, fn(next))</code>	Setzt die Funktion <code>fn</code> an das Ende der Effektkette und führt sie dann als Callback aus. Wird als erster Parameter ein Bezeichner <code>name</code> übergeben, wird die Funktion <code>fn</code> an die entsprechend benannte Effektkette gesetzt.
<code>.clearQueue()</code> <code>.clearQueue(name)</code>	Entfernt alle noch nicht abgearbeiteten Items aus der Default-Eventkette von <code>fx</code> (oder der durch <code>name</code> benannten Eventkette).
<code>.dequeue()</code> <code>.dequeue(name)</code>	Führt die nächste Funktion der Verarbeitungskette für die Elemente der Collection aus. (Nimmt optional einen Namen <code>name</code> einer benannten Verarbeitungskette entgegen.)

Tabelle 4.60 jQuery-Methoden zur Beeinflussung der Effektkette »Queue«

jQuery arbeitet Effekte wie `.show()`, `.fadeOut()` oder `.animate()` nacheinander ab, wenn sie entsprechend verkettet sind. Dies bezeichnet man als *Animations-*

kette oder *Queue* – der Begriff wurde bei der Beschreibung zu `.stop()` eingeführt. Das Prinzip der Verkettung und potenzielle Gegenmaßnahmen wurden vorhin bereits angerissen. Betrachten wir es aber erneut, um es anschließend vertiefen zu können.

Folgende Animationen laufen hintereinander, wobei die letzte ein Callback namens `animateBox` startet:

```
$("#moveable")
    .show("slow")
    .animate({left:'+=200'},2000)
    ...
    .animate({left:'+=-200'},200)
    .fadeOut("slow", animateBox);
```

Dieselbe Wirkung erzielen Sie übrigens auch, wenn pro Animation eine neue Collection gebildet wird. Die Animationsketten werden nämlich nicht den jeweiligen Collections zugewiesen (dann könnten sie unabhängig voneinander sein), sondern den Elementen, die die Collections enthalten. Da dies hier stets das gleiche Element `#moveable` ist, addieren sich die Animationen implizit ebenfalls zu einer einzigen Kette:

```
$("#moveable").show("slow")
$("#moveable").animate({left:'+=200'},2000)
...
$("#moveable").animate({left:'+=-200'},200)
$("#moveable").fadeOut("slow", animateBox);
```

Nach der Regel, dass diejenige Funktion, die zuerst in die Animationskette (Queue) gelangt, auch zuerst wieder aus ihr entfernt wird (»first in, first out«), werden die einzelnen Funktionen in einem »Stack« (genauer, einem Array) abgelegt. Auf dieses Array müssen Sie zunächst einmal überhaupt zugreifen können. Selbstverständlich gibt Ihnen jQuery hierzu eine Möglichkeit.

.queue([name])

Die Methode `.queue()` gibt eine Referenz auf die in einer Warteschlange angemeldeten Funktionen zurück. Hierfür wird zunächst eine Collection gebildet, die ein animiertes Objekt enthält. In diesem Fall ist es `#moveable`. Der Rückgabewert besitzt Array-Charakter, daher kann mittels der Eigenschaft `length` festgestellt werden, wie viele Animationsschritte in der aktuellen Queue des betrachteten Objekts enthalten sind:

```
$(document).ready(function() {
    $("p").click(function(e) {
        // Die Queue wird in Variable abgelegt:
```

```

    var value = $("#moveable").queue("fx");
    // val.length ist die Länge der Queue:
    $(this).text("Anzahl: " + value.length);
  });
  function animateBox() {
    $("#moveable").show("slow")
    .animate({left:'+=200'},2000)
    .animate({left:'+=-200'},200)
    .fadeOut("slow", animateBox);
  }
  animateBox();
});

```

Das Argument `name` ist optional und hätte in diesem Fall auch unterbleiben können, da der hier für die Queue übergebene Bezeichner `fx` auch der Defaultbezeichner für die Kette ist. Wollen Sie allerdings nicht die Standard-Queue, sondern eine andere betrachten, müssen Sie natürlich deren Namen übergeben.

.queue([name],Callback) und .dequeue()

Als zweites Argument kann eine Callback-Funktion übergeben werden, die zu den bereits gültigen noch weitere Animationen hinzufügt. Das erste Argument, der Bezeichner der Queue, kann aber auch (wie hier) problemlos entfallen, da jQuery ein übergebenes Funktionsobjekt stets als Callback behandeln wird:

```

$(document).ready(function() {
  $("#moveable").show("slow")
  $("#moveable").animate({left:'+=200'},2000)
  $("#moveable").queue(function() {
    // zwei zusätzliche Animationsschritte:
    $(this).animate({opacity:0.5},1000);
    $(this).animate({opacity:1},100);
    // forciert den Schritt zur Folgeanimation:
    $(this).dequeue();
  });
  $("#moveable").animate({left:'+=-200'},200)
  $("#moveable").fadeOut("slow", animateBox);
});

```

Fügen Sie als letztes Glied der Callback-Funktion noch die Anweisung `.dequeue()` hinzu. Diese Methode bewirkt die Fortführung der Warteschleife mit der nächstfolgenden Animation. Dies ist hier erforderlich, da die beiden durch das Callback hinzugefügten Animationsschritte außerhalb der regulären Queue-

Sequenz stehen. (Für ein »normales« Glied der Queue wird `.dequeue()` quasi implizit aufgerufen, sodass Sie sich nicht extra darum zu kümmern brauchen.)

.queue([name],queue)

Mit dieser Methode können Sie eine Warteschlange entfernen oder eine neue setzen:

```
var newQueue = function() {
    $("#moveable").show("fast")
    $("#moveable").hide("slow")
}
$("#moveable").queue("fx", [newQueue] );
```

Mit Aufruf der Funktion `newQueue` wird eine alte Warteschlange `fx` durch eine neue ersetzt. Mit einem leeren Array wird die alte Warteschlange gelöscht:

```
$("#moveable").queue("fx",[]);
```

4.11.8 Vollständiges Beispiel zu .queue()

Um das Thema Warteschlange im Zusammenhang zu erläutern, wird nun ein Beispiel als Ganzes betrachtet. Es wird ein `div#moveable` animiert, indem es horizontal hin- und herbewegt wird, seine Transparenz (Opacity) wird auf 50% herabgesetzt. Nach einem Klick auf den Button RICHTUNGSWECHSEL soll sich das Objekt in der Vertikalen bewegen. In dieser zweiten Kette unterbleibt die Änderung der Opacity. Der HTML-Teil sieht folgendermaßen aus:

```
<html>
<head>
    <title>jQuery - Beispiel 5</title>
    <link rel="stylesheet"
        href="styles.css" type="text/css"/>
    <script type="text/javascript"src="jquery-1.4.js"></script>
    <script type="text/javascript"src="queue.js"></script>
</head>
<body>
    <div id="moveable"></div>
    <button id="change">Richtungswechsel</button>
    <button id="reset">Zum Ausgangspunkt</button>
    <div>
        <span></span>
    </div>
</body>
</html>
```

Listing 4.1 Queue – komplettes Beispiel, HTML Gerüst

Es wird noch ein Stylesheet mit dem Namen *styles.css* benötigt:

```
div#moveable {
    position:absolute;
    width:50px;
    height:50px;
    left:10px;
    top:100px;
    background:green;
    display:none;
}
```

Listing 4.2 Queue – komplettes Beispiel, CSS

Die Funktionen werden in die Datei *queue.js* ausgelagert:

```
$(document).ready(function() {
    // Event
    $("button").toggle(function() {
        $("#moveable")
        .queue("fx",[newQueue]);
    }, function() {
        $("#moveable")
        .queue("fx",[moveRectangle]);
    });

    $("button#reset").click(function() {
        $("#moveable")
        .stop(true)
        .css({left:'10px',top:'100px'});
        moveRectangle();
    });

    // Erste Funktion, die eine Warteschlange erzeugt
    function moveRectangle() {
        $("#moveable").fadeIn("slow")
        .animate({left:'+=200'},2000)
        .animate({opacity:0.5},1000)
        .animate({opacity:1},1000)
        .animate({left:'+=-200'},2000)
        .fadeOut("slow", moveRectangle);
        message();
    }
    moveRectangle();
    // Zweite Funktion, die eine Warteschlange erzeugt
    function newQueue() {
```



```

        $("#moveable").fadeIn("slow")
        .animate({top: '+=200'},2000)
        .animate({top: '+=-200'},2000)
        .fadeOut("slow",newQueue);
        message();
    }

    // Hilfsfunktion zur Anzeige der Meldung
    function message() {
        var queueLength = $("#moveable").queue("fx").length;
        $("span").text("Es befinden sich " + queueLength);
        $("span").append(" Funktionen in der Queue");
    }
});

```

Listing 4.3 Queue – komplettes Beispiel, jQuery Code

Innerhalb der Eventfunktion `.toggle()` tauschen Sie die Warteschlangen aus:

```
.queue("fx",[newQueue]);
```

wobei `newQueue` eine Funktion aufruft, in der Sie den grünen Kasten vertikal bewegen. Innerhalb dieser Funktion wird eine Warteschlange angelegt und abgearbeitet. Klicken Sie erneut auf den Button, wird mit

```
.queue("fx",[moveRectangle]);
```

wieder die ursprüngliche Funktion aufgerufen und `newQueue` überschrieben. Mit der Funktion `message` wird mit `$("#moveable").queue("fx").length` die Anzahl der gerade in der Pipe befindlichen Funktionen in das Element `<span>` geschrieben.

Der Button `RESET` setzt die Warteschlange zurück. (Die Methode `stop()` wird in Abschnitt 4.11.5, »Utility-Methoden für Animationen«, besprochen.)

4.11.9 Utilities für Queue und Animationen

Neben den auf Collections anwendbaren jQuery-Methoden existieren zusätzlich noch globale Utilities (Low-level-Methoden), die direkt vom jQuery-Objekt aus eingesetzt werden können.

Der Einsatz dieser Low-level-Methoden ist analog zu den vorhin vorgestellten regulären Methoden, abgesehen davon, dass hier ein DOM-Element übergeben werden muss, an das die betrachtete oder zu bearbeitende Queue gebunden ist.

Utility	Erläuterung
<code>jQuery.queue(elem)</code> <code>jQuery.queue(elem, name)</code>	Zeigt die Funktionen der aktuellen Verarbeitungskette <code>fx</code> für das übergebene DOM-Element <code>elem</code> . Nimmt optional einen Queue-Namen <code>name</code> entgegen.
<code>jQuery.queue(elem, name, newQ)</code>	Überschreibt die Verarbeitungskette <code>name</code> für das übergebene DOM-Element <code>elem</code> durch die neue Kette <code>newQ</code> .
<code>jQuery.queue(elem, name, fn)</code>	Fügt der aktuellen Verarbeitungskette <code>fx</code> für das übergebene DOM-Element <code>elem</code> über die Callback-Funktion <code>fn</code> weitere Animationsschritte hinzu.
<code>jQuery.dequeue(elem)</code> <code>jQuery.dequeue(elem, name)</code>	Springt zur folgenden Funktion in der Verarbeitungskette <code>fx</code> für das übergebene DOM-Element <code>elem</code> . Nimmt optional einen Queue-Namen <code>name</code> entgegen.

Tabelle 4.61 jQuery-Utilities für Animationen und Queue

jQuery.fx.off

Interessant sind jedoch die Möglichkeiten, global alle Animationen zu stoppen. Hierzu dient die Eigenschaft `jQuery.fx.off`, die auch kürzer `$.fx.off` geschrieben werden kann.

Utility	Erläuterung
<code>jQuery.fx.off</code>	Boolescher Wert. Deaktiviert bzw. stoppt für <code>false</code> allgemein alle Animationen.

Tabelle 4.62 jQuery-Utility zum Stoppen aller Animationen

Diese Eigenschaft enthält einen Booleschen Wert – per Default ist sie `true`, sodass alle anstehenden Animationen auch ausgeführt werden. Setzen Sie die Eigenschaft künstlich auf `false`, werden sämtliche Animationen augenblicklich angehalten:

```

$("button").click(function() {
    // schaltet den Wert von $.fx.off hin und her:
    $.fx.off = !$.fx.off;
})
    
```

4.12 Ajax & JSON

Aus dem aktuellen Webdesign ist Ajax nicht mehr wegzudenken. Zu sehr hat man sich daran gewöhnt, dass Webseiten nach einer Interaktion Daten nachladen und dass diese dann flüssig in die Seite eingebaut werden. Im früheren, »statischen« Seitenmodell war dies anders – der Server lieferte HTML und CSS, und der Client stellte dies dar. Eine gewisse Dynamik war möglich, aber nur, wenn die dafür erforderlichen Daten (Scripte, Informationen, Bilder) bereits clientseitig vorhanden waren. Eine erneute Anfrage des Browsers bedeutete stets, eine komplette Seite anzufordern und diese dann neu im Viewport aufzubauen (siehe Abbildung 4.18, oben).

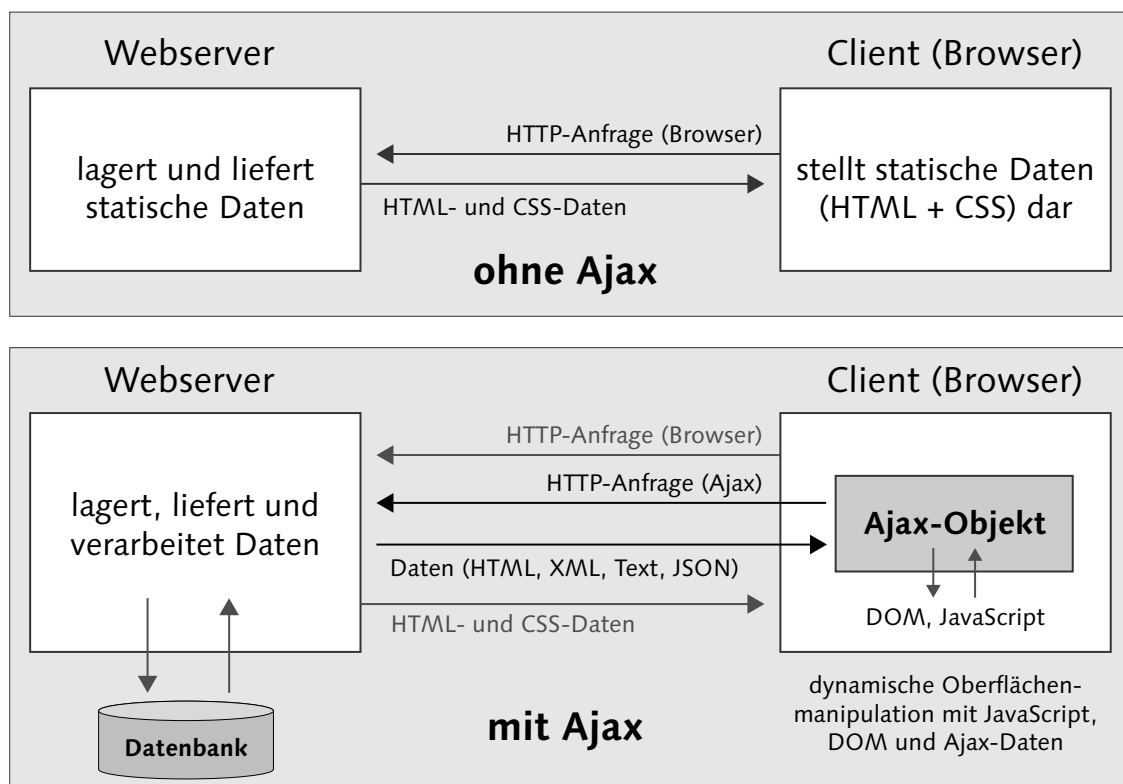


Abbildung 4.18 Statisches Web (oben) und Ajax-Web (unten) im Vergleich

Mit dem in JavaScript implementierten Ajax-Objekt `XMLHttpRequest` wurde es möglich, aus einer dargestellten Seite heraus, gewissermaßen an den üblichen Kanälen des Browsers vorbei, separate HTTP-Anfragen zu starten, mittels derer auch nachträglich Informationen geholt werden konnten (die auch durch eine Datenbankabfrage etc. generierbar waren). Dies geschieht obendrein »asynchron«, d.h., noch während eine Anfrage läuft, können Scripte weiterverarbeitet werden. Das Ajax-Objekt sendet und empfängt Daten, bemerkt und meldet also, wenn die

gewünschten Informationen eingetroffen sind. Diese werden dann durch JavaScript in das DOM des Dokuments eingebaut und erscheinen somit in der Seite (siehe Abbildung 4.18, unten).

4.12.1 Grundlagen zu Ajax

Prinzipiell ist eine Ajax-Anfrage keine besonders komplexe Sache – der Teufel steckt (wie üblich) im Detail, sprich, es gibt eine aktuelle standardisierte Implementierung und mehrere ältere von Microsoft.⁷

Eine Ajax-Anfrage kann in vier Schritte unterteilt werden:

1. Ajax-Objekt erzeugen
2. Verbindung zum Server definieren
3. Daten abfragen (d.h. die Anfrage schicken, gegebenenfalls mit Parametern)
4. Antwort (oder Fehlermeldung) des Servers entgegennehmen

Schritt 1:

Per JavaScript muss ein Objekt generiert werden, das die HTTP-Anfrage vornimmt und die Daten entgegennimmt. Das Objekt wird traditionell `xhr` genannt. Die standardisierte Methode (und für Internet Explorer ab 7.0), ein *XHR-Objekt* zu erzeugen, funktioniert so:

```
var xhr = new XMLHttpRequest();
```

Schritt 2:

Das erzeugte Objekt muss eine Verbindung zum Server aufnehmen. Hierfür dient die Methode `open()` des `XMLHttpRequest`-Objekts. Die Methode erwartet drei Parameter:

```
xhr.open(HTTP-Methode, Zieladresse, Requestmethode);
```

Parameter 1 – die HTTP-Methode:

Die Methode wird als String übergeben. Wie bei Formularen, muss hier entweder »GET« oder »POST« angegeben werden. Groß- und Kleinschreibung sind hier irrelevant – eingebürgert hat sich die Großschreibung:

```
xhr.open("POST", ..., ...); // nehmen wir mal POST
```

⁷ Hier trägt Microsoft als Vorreiter und Miterfinder der Technologie keine Schuld. Neuere Internet Explorer ab Version 7 implementieren ebenfalls den Standard.

Parameter 2 – die Zieladresse:

Die Adresse wird als String übergeben. Dieser bezeichnet (eventuell mit Pfad) den URL der Datei, die angefordert wird, bzw. die Ressource, die die Daten erzeugt, die verwendet werden sollen:

```
xhr.open("POST", "beispiel.html", ...);
```

Parameter 3 – die Request-Methode:

Der dritte Parameter ist ein Boolescher Wert. Er wird daher ohne Anführungszeichen übergeben und lautet entweder `true` oder `false`.

- ▶ `false`: Scriptausführung wird angehalten, bis der Server die Daten zurückliefert (synchron).
- ▶ `true`: Scriptausführung läuft weiter; HTTP-Anfrage wird im Hintergrund ausgeführt (asynchron).

Eine synchrone Abfrage wählt man nur, wenn die angeforderten Daten für das korrekte Weiterverarbeiten der Seiten unerlässlich sind – dies kann (eher selten) bei Formularverarbeitung erforderlich sein. Für eine asynchrone Abfrage (die Regel) steht daher `true`:

```
xhr.open("POST", "beispiel.html", true);
```

Das Problem, auf das man nun stößt, liegt auf der Hand: Das Eintreffen der Daten nach erfolgreicher Anfrage muss natürlich erkannt werden – und dies, obwohl das Script gleichzeitig weiterläuft. Glücklicherweise besitzt das XHR-Objekt eine Eigenschaft, die hierfür abgefragt werden kann: `xhr.readyState`.

readyState	Bedeutung	Erläuterung
0	Nicht initialisiert (<i>unsent</i>)	Das XHR-Objekt existiert, aber noch war nichts los.
1	Initialisiert (<i>opened</i>)	<code>open()</code> -Methode wurde erfolgreich aufgerufen.
2	Kontaktiert (<i>headers received</i>)	Server hat mit allen erforderlichen HTTP-Headern geantwortet.
3	Daten kommen (<i>loading</i>)	Antwort wird geladen.
4	Fertig (<i>done</i>)	Laden der Antwort wurde erfolgreich abgeschlossen (bzw. durch Abbruch beendet – in diesem Fall wurde ein Error-Flag gesetzt).

Tabelle 4.63 Die vier Zustände von `xhr.readyState`

Den Zustandswechsel von `readystatechange` müssen Sie erkennen und behandeln (d.h., eine Funktion wird gestartet). Hierzu können Sie den Event Handler `onreadystatechange` einsetzen – und zwar ebenfalls am XHR-Objekt (als Event Listener):

```
xhr.onreadystatechange = handler; // -> Handler referenziert
```

Damit die Funktion `handler()` gestartet wird, müssen Sie diese an den Zustandswechsel binden. In diesem Fall als Referenz, weswegen *keine* Funktionsklammern verwendet werden dürfen.

Schritt 3: Daten abschicken

Die `send()`-Methode des Objekts sendet die Daten an den Server. Sie erwartet bei POST einen Parameter, nämlich die `name-value`-Pärchen der Daten (als Query-string; beispielsweise: `"wert1=xxx&wert1=yyy"`):

```
xhr.send(); // keine Parameter übergeben (leerer String)
```

Aufruf von `send()` bei GET-Übertragung

Achtung – wenn Daten per GET übertragen werden, muss der `send()`-Methode der Wert `null` übergeben werden!

```
var xhr = null;          // erstmal Variable erzeugen
if (window.XMLHttpRequest) {
    xhr = new XMLHttpRequest();
} else if (window.ActiveXObject) {
    xhr = new ActiveXObject("MSXML2.XmlHttp.3.0");
}
if (xhr != null)         // ist auch nichts schiefgegangen?
{
    xhr.open("POST", "beispiel.html", true);
    xhr.onreadystatechange = auswerten; // Funktion binden
    xhr.send();           // und los (keine Daten diesmal)
}
```

Die Funktion `auswerten()` überprüft bei *jedem* Statuswechsel, ob der erreichte Zustand den Wert 4 hat:

```
function auswerten() {
    // es interessiert jedoch nur Zustand 4:
    if (xhr.readyState == 4) {
        // wenn die Daten da sind, was machen...
        alert("Hey, es sind frische Daten da!");
    }
}
```

Schritt 4 – Wo sind die Daten?

Die empfangenen HTML-Daten befinden sich ebenfalls im XHR-Objekt, und zwar in den meisten Fällen als Textstring in das Property `responseText` gespeichert. Sie können sie »provisorisch« einfach per `alert()` ausgeben:

```
function auswerten() {
    // sind wir "fertig"?
    if (xhr.readyState == 4) {
        // zusätzlich wird der Response-Status geprüft:
        if( xhr.status >= 200 && xhr.status < 300) {
            // wenn also brauchbare Daten da sind, reagieren:
            alert(xhr.responseText);
        }
        else {
            // es ist was schiefgegangen
        }
    }
}
```

Vom Server bekommt das XHR-Objekt auch eine Statusmeldung, die in dessen Eigenschaft `status` landet und ausdrückt, ob die Anfrage letztendlich erfolgreich war oder nicht: Es hilft wenig, wenn das `responseText`-Property lediglich eine Fehlermeldung enthält (deshalb sagt es nichts aus, wenn es »nicht leer« ist).

Statusmeldung des Servers – Erfolg oder Fehler?

Eine *HTTP-Statusmeldung* wird vom Server in Form einer dreistelligen Ziffer gemeldet. Generell lässt sich sagen, dass eine Zahl zwischen 200 und 299 als *Erfolg* sowie alle Zahlen ab einschließlich 300 als *Fehler* zu werten sind. Am bekanntesten sind die Werte 200 für »Erfolg« und 404 für »Resource nicht gefunden«. Jedoch ist es sicherer, anstatt auf exakte Werte auf Bereiche zu prüfen (es sei denn, Sie kennen Ihren Server persönlich).

Per `alert()` geben Sie die Daten natürlich nicht aus. Sie sollen ja in die HTML-Seite geschrieben werden. Hierfür bereiten Sie z. B. einen Container vor, der ein ID erhält:

```
<div id="ausgabe">
</div>
```

Dieser kann nun über DOM angesprochen werden

```
document.getElementById("ausgabe") ...
```

und seinen Inhalt verändern

```
document.getElementById("ausgabe").innerHTML ...
```

indem der Dateninhalt des XHR-Objekts zugewiesen wird

```
document.getElementById("ausgabe").innerHTML = xhr.responseText;
```

Das Ergebnis sieht so aus:

```
<div id="ausgabe">
<h2>Dies ist ein AJAX-Test</h2>
<p>Dieser Text steht in der angeforderten Datei.</p>
<p>Es könnte natürlich auch eine Datenbankabfrage stattfinden...</p>
</div>
```

Offizielle Ressourcen:

The XMLHttpRequest Object (W3C Working Draft 19. November 2009):

www.w3.org/TR/XMLHttpRequest

4.12.2 Daten und Datentypen für Ajax

Grundsätzlich geht es bei einer Ajax-Anfrage darum, dass der Server den Client mit Daten versorgt. Der Datentyp kann intern explizit festgelegt werden, was über die anschließende Verarbeitung entscheidet. Zu diesem Zweck existiert eine Eigenschaft `dataType`, die in jQuery bei der Konfiguration des Requests gesetzt werden kann (dazu kommen wir gleich). Sie kann die Werte »text«, »html«, »script«, »json«, »jsonp« oder »xml« annehmen, wobei nur XML gesondert behandelt wird. Wird kein Datentyp festgelegt oder vom Server mitgeteilt, erschließt das XHR-Objekt den Typ aus den empfangenen Daten (»educated guess«) – es existiert für `dataType` daher kein fester Defaultwert.

Textdaten

Der einfachste Fall ist, dass die Daten einfach in Form einer Zeichenkette übertragen werden. Diese kann beispielsweise anschließend als Inhalt in irgendeinen Tag-Container im Dokument geschrieben werden. Die innere Struktur der Daten ist entweder nicht relevant, oder es existiert gar keine. Solche Daten landen generell in der `responseText`-Eigenschaft des XHR-Objekts.

HTML

Eine Variante von Textdaten sind HTML-Daten. Eigentlich wird auch hier nur eine Zeichenkette übertragen, die allerdings aus HTML-Quellcode besteht (in Form eines HTML-Fragments oder eines mit HTML angereicherten Textstrings). Auch HTML-Daten landen in `responseText` (es wird also im Grunde nicht zwischen HTML und Text unterschieden) und können, entweder über die Eigen-

schaft `innerHTML` oder die jQuery-Methode `.html()`, in ein Element geschrieben werden.

Anschließend sorgt der Browser dann für die Interpretation des Codes. Es ist wichtig, zu wissen, dass in den HTML-String eingebettete JavaScript-Befehle (in Form von Scriptblöcken) ausgeführt werden, *bevor* der HTML-Inhalt eingefügt wird.

JavaScript

Auch JavaScript-Daten können als Serverantwort geschickt werden. Ihnen wird der `dataType` »script« zugewiesen. JavaScript-Daten werden nach Eintreffen interpretiert und dann in Textform in `responseText` abgelegt.

JSON

Das JSON-Format erfreut sich aufgrund seiner Kompaktheit in Zusammenhang mit Ajax inzwischen großer Beliebtheit. Das Akronym steht für »JavaScript Object Notation«. JSON verpackt strukturierte Datenbeschreibungen in Objektliterale, die als Zeichenketten übertragen werden.

Diese Zeichenketten (es muss natürlich bekannt sein, dass es sich um JSON handelt) müssen nach Empfang noch geparkt werden, um in echte JavaScript-Objekte umgewandelt werden zu können. Hierfür existieren als *JSON-Parser* bezeichnete Hilfsanwendungen, die auch in jQuery implementiert sind. Das Format kann daher in jQuery-gestützten Anwendungen ganz unbefangen eingesetzt werden. Auch JSON-Daten werden in das `responseText`-Property abgelegt.

JSONP

Bei JSONP (»JSON with Padding«) handelt es sich um ein erweitertes JSON-Format, bei dem dem Server über einen GET-Querystring eine Callback-Funktion mitgeteilt wird, die als Wrapper von JSON-Daten verwendet werden soll. Hintergrund dieses Formats ist die Überwindung der Cross-Domain-Schranke, die auftritt, sobald Daten in Form von JavaScript von einem weiteren Server bezogen werden müssen. Das Prinzip von JSONP besteht darin, dem JSON-String durch den Server einen beliebigen Textstring voranzustellen und ihn in runde Klammern setzen zu lassen. Auf diese Weise kann er clientseitig als Funktionsaufruf mit einem Argument, nämlich dem JSON-String, interpretiert werden.

XML

Das Datenformat, dem Ajax zum Teil seinen Namen verdankt, ist XML. Grundsätzlich ist XML konzipiert, um strukturierte Daten zu beschreiben. Damit ist es vergleichbar mit JSON – allerdings besitzen beide Formate Stärken und Schwä-

chen. Die Stärke von XML besteht in der beliebig komplexen Natur der Daten. Erschwert wird die Nutzung von XML-Daten durch die Art, wie die Information aus dem XML extrahiert wird. XML muss nämlich erst in einen *DOM-Baum* umgewandelt werden, während geparstes JSON unmittelbar als zugängliches JavaScript-Objekt vorliegt.

Praktischerweise werden XML-Daten gleich als fertiger DOM-Baum in der Eigenschaft `responseXML` abgelegt. Dies erfordert jedoch, dass der Server beim Datenversand über den MIME-Typ (beispielsweise `text/xml`) darauf hinweist, dass es sich um XML-Daten handelt. Um das XML-DOM auszuwerten, können anschließend sämtliche jQuery-Selektormethoden eingesetzt werden.

4.12.3 jQuery und Ajax

Das jQuery Framework bündelt einerseits – wie nicht anders zu erwarten – alle Unterschiede zwischen den Ajax-Implementierungen aus. Zum anderen stellt es eine Fülle an Utility- und Convenience-Methoden zur Verfügung, die es einfach machen, Ajax-Anwendungen zu schreiben. So genügt beispielsweise der Aufruf `.load(url)` als Methode des jQuery-Objekts, um eine externe Ressource per Ajax anzusprechen und in ein Objekt zu laden. Dies wird noch im Praxisteil zur Genüge durchexerziert werden. An dieser Stelle soll deshalb eine Übersicht über diejenigen Methoden gegeben werden, die sonst zu kurz kämen, und natürlich zudem eine Grundlage für späteres Verständnis geschaffen werden.

4.12.4 Low-level Ajax Utilitys

Obwohl die Dinge durchaus einfach sein können – manchmal sind sie es eben doch nicht. Das heißt, man muss sich neben allen bequemen Abkürzungen auch den »dornigen Weg« offenhalten, der bedeutet, alles in Einzelheiten selbst festzulegen. Hierzu dienen die Low-level Ajax Utilities und Methoden, die jQuery in Ajax-Belangen zur Verfügung stellt. Sie sind zwar umständlicher in der Handhabung als die später vorgestellten Convenience-Methoden, bieten aber den Vorteil, alle Parameter explizit setzen zu können.

Utility	Erläuterung
<code>jQuery.ajax(obj)</code>	Startet allgemein einen Ajax-Request, dessen Parameter über das Konfigurationsobjekt <code>obj</code> bestimmt werden.
<code>jQuery.ajaxSetup(obj)</code>	Nimmt ein Objekt <code>obj</code> entgegen, das als Defaultkonfiguration für später vorzunehmende Ajax-Requests <code>jQuery.ajax(obj)</code> dient.

Tabelle 4.64 Low-level Utilities für Ajax Requests

Die Low-level-Methode \$.ajax()

Ein vollständig parametrisierbarer Ajax-Request kann über die Low-level-Methode `jQuery.ajax()` bzw. `$.ajax()` gestartet werden. In diesem Fall ist dem Aufruf eine Parameterliste in Form eines JavaScript-Objekts zu übergeben, das den Request konfiguriert. Die hier festzulegenden Eigenschaften dieses Konfigurationsobjekts sind in den folgenden Tabellen beschrieben – sie sind sämtlich optional (gegebenenfalls existieren Defaultwerte), können also beliebig zusammengestellt werden. Minimal muss mit der Eigenschaft `url` der URL einer Ressource (hier eine HTML-Seite) angegeben werden (sofern Sie nicht mit einem externen Setup über `$.ajaxSetup()` arbeiten – doch dazu gleich mehr):

```
$.ajax( { url: "beispiel.html" } );
```

Normalerweise werden jedoch mehr Angaben eingesetzt. Hier wird zusätzlich zur Ressource der Kontext (`context`) auf den Body-Tag des ausführenden Dokuments gelegt. Auf diesen Kontext beziehen sich die Callback-Funktionen des Ajax-Requests. In diesem Fall zeigt das `this` in der an `success` gebundenen Funktion daher auf den `<body>`, an den bei erfolgreichem Abschluss eine CSS-Klasse gebunden wird:

```
$.ajax( { url: "beispiel.html",
          success: function(){
              $(this).addClass("fertig");
          },
          context: document.body
        } );
```

Warum ein Objekt zum Festlegen der Konfiguration?

Der Vorteil an der Übergabe eines Konfigurationsobjekts gegenüber einem Array oder einer Werteliste besteht darin, dass jeder Wert als Objekteigenschaft einen Namen besitzt. Sie brauchen daher weder auf eine Reihenfolge zu achten, noch müssen Sie zwangsweise nicht benötigte Werte setzen.

Defaultkonfiguration mit \$.ajaxSetup()

In der Regel muss eine ganze Reihe von Parametern beim Request mit `$.ajax()` explizit gesetzt werden. Müssen mehrere oder viele ähnliche Requests erfolgen, ist es praktisch, bestimmte konstant zu setzende Werte (für Defaultwerte natürlich überflüssig) zentral festzulegen, damit dies nicht – fehlerträchtig – bei jedem Einzel-Request aufs Neue geschehen muss. Hierzu dient das `$.ajaxSetup()`-Utility, das gleichfalls ein Konfigurationsobjekt entgegennimmt:

```
$.ajaxSetup( { url: "adresses.php",
               success: function(){
```

```

        $(this).addClass("fertig");
    },
    context: document.body,
    type: "POST"
});

```

Ein Aufruf von `$.ajax()` bindet automatisch das in `$.ajaxSetup()` definierte Konfigurationsobjekt ein und fügt die ihm unmittelbar übergebenen Eigenschaften hinzu. In diesem Fall sind es Daten, die für diesen speziellen Aufruf übertragen werden sollen:

```
$.ajax( { data: { 'vorname': 'Peter', 'nachname': 'Panter' } } );
```

Konfigurationsobjekt für allgemeine Ajax-Requests – Basic Options

Von besonderem Interesse in Zusammenhang mit den Low-level-Methoden sind natürlich das *Konfigurationsobjekt* und seine Eigenschaften. Da die Optionen zahlreich sind, wurden sie für diese Übersicht auf mehrere Tabellen verteilt. Betrachten wir zunächst die wichtigsten Grundeinstellungen, die vorgenommen werden können, von denen `url` und `context` bereits in den vorangegangenen Beispielen erläutert wurden.

Eigenschaft	Defaultwert	Typ
<code>type</code>	'GET'	string
<code>url</code>	URL der aktuellen Seite	string
<code>contentType</code>	'application/x-www-form-urlencoded'	string
<code>context</code>	–	object

Tabelle 4.65 Konfigurationsobjekt für `jQuery.ajax()` – Basic Options

type-Eigenschaft

Grundsätzlich erfolgt ein Request als GET-Request, solange nichts anderes festgelegt wird. Wird Datenversand mit POST gewünscht, muss die `type`-Eigenschaft entsprechend gesetzt werden. POST-Daten werden als UTF-8-kodiert an den Server geschickt.

url-Eigenschaft

Die Adresse der Ressource, von der die Daten als URI-String bezogen werden. Wird kein `url`-Wert übergeben, dient Adresse des Dokuments als Defaultziel.

contentType-Eigenschaft

Der MIME-Type, der vom Server als Antwort auf einen Request im HTTP-Header mitgesendet wird. Per Default ist dies, wie auch bei Formulardaten üblich, »application/x-www-form-urlencoded«.

context-Eigenschaft

Legt per Objektreferenz einen DOM-Knoten als Kontext fest, der für die Callback-Funktionen des Requests verwendet werden soll.

Konfigurationsobjekt für allgemeine Ajax-Requests – Basic Options (Data)

Drei weitere Eigenschaften aus der Basic-Gruppe sind für das Datenhandling des Requests zuständig.

Eigenschaft	Defaultwert	Typ
data	–	object string
dataType	xml json script html	string
processData	true	boolean

Tabelle 4.66 Konfigurationsobjekt für jQuery.ajax() – Basic Options (Data)

data-Eigenschaft

Diese Eigenschaft enthält die Daten, die im Rahmen des Requests an den Server geschickt werden. Dies kann eine Zeichenkette in Form eines Querystrings sein oder ein JavaScript-Objekt, das die `key-value`-Pärchen als Properties enthält (dies wird als *Map* bezeichnet). Liegt der Wert in `data` als Map vor, wird er vor Versenden der Daten in einen Querystring umgewandelt, sofern die Eigenschaft `processData` nicht Gegenteiliges anordnet.

dataType-Eigenschaft

Die `dataType`-Eigenschaft bezeichnet den Typ der vom Server geschickten Daten als »text«, »html«, »script«, »json«, »jsonp« oder »xml«. Für die Typen »text« und »xml« werden die Daten unverändert aus den Properties `responseText` bzw. `responseXML` ausgelesen und der Funktion übergeben, die als `success`-Handler festgelegt ist.

processData-Eigenschaft

Mit `processData` wird das Verhalten gegenüber Daten-Maps im `data`-Property bestimmt. Normalerweise werden Daten-Maps vor Versenden in einen Querystring umgewandelt. Ist dies nicht erwünscht, setzt man `processData` auf `false`.

Konfigurationsobjekt für allgemeine Ajax-Requests – Callbacks

Auch Callbacks können im Konfigurationsobjekt festgelegt werden. Hier handelt es sich im Prinzip um Funktionsobjekte, die an eine bestimmte Phase während des Requests gebunden sind und dann ausgeführt werden. An das Funktionsobjekt werden Daten als Parameter übergeben (siehe die Funktionssignaturen in der folgenden Tabelle).

In den meisten Fällen ist dies u. a. eine Referenz auf das XHR-Objekt, das den Request repräsentiert. Das bedeutet, dass die Callback-Funktionen beispielsweise Zugriff auf dessen `responseText` oder `responseXML` haben, also auf die vom Server geschickten Daten.

Callback-Property	Signatur des Funktionsobjekts
<code>beforeSend</code>	<code>function(XHR)</code>
<code>error</code>	<code>function(XHR, status, errorThrown)</code>
<code>dataFilter</code>	<code>function(data, type)</code>
<code>success</code>	<code>function(data, status, XHR)</code>
<code>complete</code>	<code>function(XHR, status)</code>

Tabelle 4.67 Konfigurationsobjekt für `jQuery.ajax()` – Bindung von Callbacks

Der Aufruf der Callbacks der vorangegangenen Tabelle wird bei der Verarbeitung eines individuellen XHR-Requests in folgender Reihenfolge eingetaktet.

beforeSend-Callback

Diese Funktion ist in der Lage, ein XHR-Objekt vor der Auslösung seiner `send()`-Methode noch kurzfristig zu modifizieren, wie beispielsweise den Request mit einem anderen Header auszustatten. Hierfür bekommt die Callback-Funktion das XHR-Objekt als (einziges) Argument übergeben. Über den Rückgabewert `false` ist es dem Callback möglich, den Request notfalls zu canceln:

```
$.ajax({
  beforeSend: function(xhr){
    // alles, was vor Abschicken des Requests zu tun ist
  });
```

error-Callback

Schlägt ein Request fehl, wird das Error-Ereignis ausgelöst, das diesen Callback triggert. Die Funktion erhält das XHR-Objekt als ersten Parameter sowie einen String, der die Art des Fehlers bezeichnet. Wird eine Exception geworfen, steht als dritter Parameter ein entsprechendes Objekt zur Verfügung:

```
$.ajax({
  error: function(xhr, error){
    // alles, was im Fehlerfall zu tun ist
  });
```

dataFilter-Callback

Eine Sonderstellung nimmt der `dataFilter`-Callback ein, der beim Eintritt des `Success`-Ereignisses ausgelöst wird (jedoch vor dem `success`-Callback selbst!). Hier handelt es sich um eine bei Bedarf zu definierende Callback-Funktion, die die Daten vorbereitet, die im XHR-Objekt bei Erfolg in `responseText` abgelegt wurden, um sie anschließend an den `success`-Callback weiterzureichen. Die Funktion kann beispielsweise dafür sorgen, dass die Daten tatsächlich dem als zweiten Parameter übergebenen Typ entsprechen. Die vorbereiteten (und gegebenenfalls veränderten) Daten müssen von der Funktion als Rückgabewert wieder zur Verfügung gestellt werden:

```
$.ajax({
  dataFilter: function(data, status){
    // Daten für success-Callback vorbereiten
    // ... und verarbeitet zurückgeben:
    return data;
  });
```

Definieren Sie kein `dataFilter`-Callback (dies ist durchaus kein Muss), werden die Daten unmittelbar an den `success`-Callback weitergegeben.

success-Callback

Die Funktion des `success`-Callbacks wird bei Erfolg des Requests ausgelöst (wobei vorher, falls definiert, gegebenenfalls der `dataFilter`-Callback abgearbeitet wird). Die Funktion nimmt als ersten Parameter die beim Request erhaltenen Daten entgegen, als weitere Argumente stehen der Success-Code (als String) und das XHR-Objekt selbst zur Verfügung:

```
$.ajax({
  success: function(data, success, xhr){
    // alles, was bei Erfolg des Requests zu tun ist
  });
```

complete-Callback

Der `complete`-Callback wird aufgerufen, wenn der XHR-Request abgeschlossen ist. Dies findet also im Anschluss an den Aufruf der `success`- und `error`-Callbacks statt (sofern diese definiert sind). Der Callback-Funktion wird als erstes Argument

das XHR-Objekt übergeben. Als zweites Argument dient ein String, der den Status des Requests beschreibt:

```
$.ajax ({
  complete: function(xhr, status){
    // alles, was nach Abschluss des Requests zu tun ist
  });
```

Konfigurationsobjekt für allgemeine Ajax-Requests – JSONP-Aspekte

Für den Datentyp JSONP gibt es zwei Werte im Konfigurationsobjekt.

Eigenschaft	Defaultwert	Typ
jsonp	'callback'	string
jsonpCallback	(generierter Bezeichner)	string

Tabelle 4.68 Konfigurationsobjekt für jQuery.ajax() – JSONP-Aspekte

jsonp-Eigenschaft

Die Eigenschaft `jsonp` erhält eine Zeichenkette, die den Bezeichner der Callback-Funktion darstellt, der im Rahmen eines Requests mit JSONP-Daten verwendet werden soll. Defaultwert ist der String `callback`, was aus dem Teilstring `'callback=?'` innerhalb des Querystrings des URL resultiert.

Soll die Callback-Funktion einen anderen Namen erhalten, ist er hier zu nennen. Übergibt man an `jsonp` beispielsweise den Wert `'onJsonPLoad'`, ergibt dies im URL den Teilstring `'onJsonPLoad=?'`.

jsonpCallback-Eigenschaft

Im Unterschied zur Eigenschaft `jsonp`, die nur den String nennt, der im URL als Callback-Bezeichner eingesetzt werden soll, nennt `jsonpCallback` den wirklichen Bezeichner der Callback-Funktion, die für den aktuellen *jsonp*-Request eingesetzt werden soll. Dies kann das Cache-Verhalten des Browsers für derartige Requests verbessern, wenn sie per GET erfolgen. Ansonsten generiert jQuery selbstständig einen Bezeichner, was in den meisten Fällen die bessere (und einfachere) Lösung darstellt.

Konfigurationsobjekt für allgemeine Ajax-Requests – Advanced Options

Weitere Eigenschaften des Konfigurationsobjekts werden als *Advanced Options* bezeichnet. Sie werden entsprechend seltener benötigt.

Eigenschaft	Defaultwert	Typ
<code>async</code>	<code>true</code>	boolean
<code>cache</code>	<code>true</code>	boolean
<code>global</code>	<code>true</code>	boolean
<code>ifModified</code>	<code>false</code>	boolean
<code>password</code>	–	string
<code>timeout</code>	–	num
<code>username</code>	–	string
<code>scriptCharset</code>	–	string

Tabelle 4.69 Konfigurationsobjekt für `jQuery.ajax()` – Advanced Options

async-Eigenschaft

Die Eigenschaft `async` legt fest, ob der betreffende Request synchron (`false`) oder asynchron (`true`, Default) erfolgen soll. Den Request synchron verlaufen zu lassen, bedeutet, dass alle anderen Scripte während der Verarbeitung des Requests angehalten werden. Dies kann erforderlich sein, falls ohne die Daten, die der Request liefern soll, kein sinnvolles Fortfahren möglich ist (beispielsweise wenn es darum geht, ein Formular während einer Eingabe zu modifizieren). Die Gefahr besteht allerdings, dass der Request kein Ergebnis bringt und so die Scriptverarbeitung insgesamt abbricht.

cache-Eigenschaft

Die `cache`-Eigenschaft steuert das Verhalten des Browsers gegenüber der angefragten Ressource. Für den Defaultwert `true` wird diese durch den Browser gecacht und im Falle der Wiederholung des Requests aus dem Cache statt von der Quelle bezogen. Möchten Sie dies nicht, setzen Sie diesen Wert auf `false`.

Die Eigenschaft steht standardmäßig auf `true`, außer für Daten von Typ JSONP und Script, die grundsätzlich nicht gecacht werden (hier immer `false`).

global-Eigenschaft

Mit den `global`-Eigenschaften bestimmen Sie, welche Callbacks durch den betreffenden Request zu verwenden sind. Im Defaultfall (`true`) werden die globalen Request Handler eingesetzt. Möchten Sie den Request bezüglich Callbacks auf die lokal vereinbarten Funktionen beschränken und die globalen Handler übergehen, setzen Sie `global` auf `false`. Siehe auch Abschnitt 4.12.6, »Globale Handler-Methoden«

ifModified-Eigenschaft

Der Server schickt einer Ressource stets einen Header *lastModified* voraus, der aussagt, wann die Ressource das letzte Mal modifiziert wurde. Wird der *ifModified*-Eigenschaft der Wert *true* zugewiesen, legt dies fest, dass ein Request nur dann als erfolgreich gewertet wird, wenn die angefragte Ressource seit dem letzten auf sie erfolgten Request erneuert (d.h. geändert) wurde. Der Defaultwert der Eigenschaft ist *false*, sodass es im Allgemeinen irrelevant ist, ob eine angefragte Ressource verändert wurde.

username-Eigenschaft, password-Eigenschaft

Mancher HTTP-Request ist auf eine passwortgeschützte Ressource gerichtet. In diesem Fall wird der Username des zu nutzenden Accounts in der *username*-Eigenschaft, das zu verwendende Passwort in der *password*-Eigenschaft abgelegt. Einen Defaultwert gibt es für beide Eigenschaften nicht.

timeout-Eigenschaft

Mit der *timeout*-Eigenschaft kann eigens für den konfigurierten Request ein lokaler Timeout eingerichtet werden. Anderenfalls wird (sofern festgelegt) der globale Timeout verwendet, der in `$.ajaxSetup` vereinbart sein könnte. Der übergebene Zahlenwert wird als Zeitdauer in Millisekunden interpretiert. Liefert der Request während der Timeout-Periode kein Ergebnis, wird er mit Fehlermeldung abgebrochen. Einen Defaultwert für diese Eigenschaft gibt es nicht.

scriptCharset-Eigenschaft

Die *scriptCharset*-Eigenschaft wird nur für Requests benötigt, die ein Ergebnis vom Datentyp »jsonp« oder »script« und per GET vorgenommen werden. In diesem Fall dient die Eigenschaft dazu, dass das Ergebnis der Anfrage als in dem übergebenen Zeichensatz vorliegend interpretiert wird. Dies ist jedoch nur dann zwingend erforderlich, wenn dieser Zeichensatz sich von dem der einbettenden Ressource unterscheidet.

Callback	Erläuterung
xhr	Callbackfunktion zur Erstellung des xhr Objekts des aktuellen Requests

Tabelle 4.70 Konfigurationsobjekt für `jQuery.ajax()` – XHR-Callback

xhr-Eigenschaft

An diese Eigenschaft kann eine Callback-Funktion gebunden werden, die die Erstellung des XHR-Objekts vornehmen soll. Dies ist nur sinnvoll, wenn Sie die eigene Funktion von `jQuery` zur Erstellung des XHR-Objekts nicht nutzen möchten,

weil Sie eigene Wege gehen wollen. Ansonsten wird per Default ein XMLHttpRequest-Objekt erzeugt, in dem dies implementiert ist, anderenfalls (dies betrifft einige Versionen des Internet Explorers) ein ActiveXObject.

4.12.5 Utilities und Convenience-Methoden

Bequemlichkeit ist Trumpf, auch für Programmierer. Daher freut man sich, für bestimmte Standard-Requests auf eigens definierte Methoden zurückgreifen zu können, die quasi vorkonfigurierten `$.ajax()`-Aufrufen entsprechen. Einige dieser Methoden werden direkt vom `$`-Objekt (bzw. dem `jQuery`-Objekt) zur Verfügung gestellt, können also als *globale Utilities* bezeichnet werden.

Methode	Erläuterung
<code>jQuery.get()</code>	Setzt einen HTTP-GET-Request an den Server ab.
<code>jQuery.post()</code>	Setzt einen HTTP-POST-Request an den Server ab.
<code>jQuerygetJSON()</code>	Holt JSON-encodierte Daten über einen GET-Request.
<code>jQuery.getScript()</code>	Holt ein JavaScript über einen GET-Request und führt es aus.
<code>jQuery.param()</code>	Serialisiert ein übergebenes Objekt oder Array für die Verwendung als URL-Parameter oder im Rahmen eines Ajax-Requests.

Tabelle 4.71 jQuery-Utility-Methoden für HTTP-Requests

jQuery.get()

Die Methode `jQuery.get()`, meist kürzer als `$.get()` geschrieben, holt eine externe Ressource per GET-Request. Sie benötigt mindestens den URL der anzusprechenden Ressource als Argument. Optionale weitere Parameter bestehen in einem Datenobjekt und einer Callback-Funktion, die im Erfolgsfall (success) ausgelöst werden soll. Als vierter, ebenfalls optionaler Wert kann eine Angabe des Datentyps übergeben werden, der für das Ergebnis erwartet wird.

Eine Anfrage mit `$.get()` kann exemplarisch wie folgt aussehen (hier wurde auf den zweiten Parameter verzichtet, nämlich die Nutzdaten, die der Server beim Request übermittelt bekommt):

```
$.get('beispiel.html', function(ergebnis) {
    // hat der Server Daten geliefert?
    if (ergebnis) {
        // dann ins Dokument einfügen:
        $('#ausgabe').html(ergebnis);
    }
});
```

Wie Sie an dem Beispiel ablesen können, erhält die Callback-Funktion die vom Server übermittelten Daten übergeben. Hier werden sie in einem Parameter `ergebnis` aufgefangen und anschließend mittels der jQuery-Methode `.html()` in einen Container `#ausgabe` eingefügt. Über zwei weitere (hier nicht verwendete) Argumente erhält die Callback-Funktion Zugriff auf den Statusstring und das XMLHttpRequest selbst.

Vielleicht einmal als Vergleich – hätten Sie den gleichen Request mittels der Basisroutine `$.ajax()` durchgeführt, hätte der Code so gelautet:

```
$.ajax({
  url: 'beispiel.html',
  success: function(ergebnis) {
    if (ergebnis) {$('#ausgabe').html(ergebnis);}
  }
});
```

Das ist zwar nicht nennenswert länger, aber, wegen der Notwendigkeit der Objektschreibweise, etwas unhandlicher.

Sie können `$.get()` auch gut in Zusammenhang mit Formularen einsetzen, die in diesem Fall als Datenlieferanten dienen. Allerdings müssen die Formulardaten in eine für Ajax verwendbare Form gebracht werden. Hierzu dient die Methode `.serialize()`, die bereits in Zusammenhang mit Formularen vorgestellt wurde. Gehen wir von einem einfachen Formular aus, das in einem `<div>`-Container liegt, das auch gleich für die Ausgabe des Ergebnisses eingesetzt wird:

```
<div id="umfrage">
  <form id="meinFormular" action="umfrage.php">
    <label for="fw">Ihr Lieblingsframework?</label>
    <input type="text" name="fw" id="fw" />
    <input type="submit" value="Umfrage" />
  </form>
</div>
```

Der User soll wie gewohnt auf den SUBMIT-Button klicken können. Statt das Formular zu verschicken, wird dabei jedoch ein Ajax-Request angestoßen. Hierfür wird `$.get()` verwendet. Zunächst muss jedoch verhindert werden, dass die Formulardaten den normalen Weg gehen – hierfür wird mit `.preventDefault()` der Submit unterbunden. Dazu verwenden Sie das Eventobjekt, das die Callback-Funktion von `.submit()` übergeben bekommt (hier mit `e` bezeichnet). Anschließend werden aus dem Formular die nötigen Daten geholt, die die anzusprechende Ressource und ein Callback für die Weiterverarbeitung der geholten Daten festlegen. Letztere besteht darin, dass diese Daten per `.html()` ins Dokument geschrieben werden:

```

$('#meinFormular').submit( function(e) {
    // Daten nicht auf normalem Weg abschicken:
    e.preventDefault();

    // $(this) ist das Formular:
    var url = $(this).attr('action');

    // Daten serialisieren für Ajax-Verwendung:
    var meineDaten = $(this).serialize();

    // vom Server wird HTML erwartet, also:
    var datentyp = 'html';

    // die Ajax-Antwort ins Dokument einbauen
    var verarbeiten = function(antwort){
        // Formular löschen, Antwort in Container schreiben:
        $('#umfrage').html(antwort)
    };

    // den GET-Request absetzen:
    $.get(url, meineDaten, verarbeiten, datentyp);
});

```

Die `$.get()`-Methode bekommt hier lediglich Variablen übergeben, was die Funktion universeller macht. In diesem Beispiel könnte die `url` auch direkt übergeben werden. Die Variable `meineDaten` wird über die `.serialize()`-Methode befüllt. Sie enthält dann einen Querystring, der beispielsweise (je nachdem, was Sie eingeben) die Form `fw=jQuery` haben kann. Ist das Formular komplexer, spiegelt sich das im Querystring entsprechend wider, wobei sich nur diejenigen Eingabefelder wiederfinden, die die `successful`-Kriterien erfüllen (mehr dazu erfahren Sie in Abschnitt 4.8, »Formulare verarbeiten mit jQuery«).

jQuery.post()

Die Methode `jQuery.post()` arbeitet analog zu `$.get()`, jedoch mit einem POST-Request. Auch hier ist der einzige obligatorisch zu übergebende Wert der URL der angefragten Ressource. Optional können als zweites Argument wiederum ein Datenobjekt, als drittes eine Callback-Funktion und drittens der erwartete Datentyp übergeben werden. Die Callback-Funktion erhält als ihre drei Eingangsparameter die Request-Daten, den Statusstring und das XHR-Objekt. Soweit nichts Neues.

Eine Anfrage mit `$.post()` könnte beispielsweise so aussehen:

```

$.post('beispiel.html', function(ergebnis) {
    // hat der Server Daten geliefert?

```

```

    if (ergebnis) {
        // dann ins Dokument einfügen:
        $('#ausgabe').html(ergebnis);
    }
});

```

Auch hier ein Blick auf einen analogen Request per `$.ajax()`. Hier muss, anders als bei der Nachbildung von `$.get()`, der Typ `type` des Requests explizit auf `POST` gesetzt werden (davor war dies nicht nötig, da ein Request automatisch mit `GET` erfolgt, solange nichts anderes gesagt wird):

```

$.ajax({
    type: 'POST',
    url: 'beispiel.html',
    success: function(ergebnis) {
        if (ergebnis) {$('#ausgabe').html(ergebnis);}
    }
});

```

Zwischen `$.get()` und `$.post()` gibt es im Handling, wie Sie sehen, für den jQuery-Programmierer keinen Unterschied. Da für `$.post()` eventuelle Nutzdaten per `POST` übertragen werden, ist die hier potenziell zu übergebende Datenmenge größer.

jQuerygetJSON()

Die globale Methode `$.getJSON()` offenbart ihren Zweck bereits im Namen – sie erzeugt einen `GET`-Request, auf den der Server mit Daten im `JSON`-Format antworten soll. Die Methode bringt bereits Mittel zur Verarbeitung der `JSON`-Daten mit. Wo diese herkommen, ist im Rahmen der Erläuterung des Prinzips eigentlich nicht weiter wichtig. Es kann sich um statische Dateien im `JSON`-Format handeln, oder um Ergebnisse einer Transaktion auf dem Server (wie eine Datenbankabfrage). Für dieses Beispiel wird ein einfaches statisches `JSON`-File eingesetzt (*blumen.json*):

```

{ "blumen1": "Rosen",
  "blumen2": "Tulpen",
  "blumen3": "Nelken" }

```

Achtung – die `JSON`-Syntax ist strikter als die von `JavaScript`

Entgegen den sonst in `JavaScript` üblichen Gepflogenheiten müssen Strings in `JSON`-Daten stets mit *doppelten Anführungszeichen* umgeben werden. Ärgerlicherweise schlägt `$.getJSON()` im Falle von Syntaxfehlern der Datenquelle fehl, ohne eine verwertbare Fehlermeldung auszugeben! Vorsicht also mit »von Hand geschriebenem« `JSON`.

Die Werte der Eigenschaften des JSON-Objekts sollen als Listenelemente in eine (derzeit noch leere) Liste mit ID "ausgabe" geladen werden:

```
<ul id="ausgabe"></ul>
```

Die bereits vorgestellte JSON-Datei wird nun über die Methode `$.getJSON()` asynchron geladen. Eine Callback-Funktion sorgt dafür, dass die Daten nach ihrem Eintreffen verarbeitet werden, und bekommt diese dafür als Argument übergeben. Als optional verwertbares zweites Argument können Sie auf den Statustext zugreifen (im Beispiel unterbleibt dies). Folgender Code steht im `ready()`-Block:

```
$.getJSON('blumen.json', function(ergebnis){
    $.each(ergebnis, function(i, val){
        $('<li>' + val + '</li>').appendTo('#ausgabe');
    });
});
```

Die Utility-Funktion `$.each()` iteriert durch das JSON-Objekt. Seine Callback-Funktion besitzt zwei Argumente – `i` nimmt den Namen des Objekt-Properties an (es wird in der Ausgabe nicht berücksichtigt), `val` enthält den Wert, der anschließend mit `<li>`-Tags umgeben und in die Ausgabe geschrieben wird.

jQuery.getScript()

Analog zu `$.getJSON()` dient `$.getScript()` dazu, per GET eine JavaScript-Datei zu holen, die anschließend unverzüglich ausgeführt wird. Die Datei kann beliebiges JavaScript enthalten, das auch Bezug auf jQuery-Funktionen nehmen kann. In folgendem Beispiel wird eine externe Datei dazu verwendet, auf die HTML-Seite einzuwirken. Dafür soll sie nicht sofort geladen werden – das wäre reizlos –, sondern erst nach Klick auf einen Button. Der relevante Ausschnitt der HTML-Seite sieht so aus:

```
<div>
    <button>Klicken, um Absätze klickbar zu machen</button>
</div>
<p>Diese Absätze sind nicht klickbar.</p>
<p>Diese Absätze sind nicht klickbar.</p>
<p>Diese Absätze sind nicht klickbar.</p>
```

Im lokalen Scriptblock stehen lediglich ein paar Zeilen, um dem Button den Click-Event zuzuweisen und bei Klick das externe Script zu laden. Die Beschriftung des Buttons wird geändert, sobald der User ihn betätigt hat:

```
$('button').click( function(){
    $.getScript('getScript.js');
```

```
$(this).text('Danke sehr!');
});
```

Das externe Script ändert zunächst die Hintergrundfarbe der Textabsätze von Grau auf Gelb. In einer `.each()`-Schleife wird der Text aller Absätze geändert und zum Schluss jedem Absatz ein Click-Handler zugewiesen, der die Hintergrundfarbe eines geklickten Absatzes auf Rot setzt:

```
/* ----- getScript.js ----- */
// alert("Externes Script geladen");

$('p').css({'backgroundColor':'#ff9'})
    .each( function() {
        $(this).text('Absatz kann jetzt geklickt werden.')
    })
    .click( function() {
        $(this).css({'backgroundColor':'#f99'});
    });
/* ----- Ende getScript.js ----- */
```

Dieses Manöver kann eingesetzt werden, um Scriptmodule nachträglich (oder nur bei Bedarf) zu laden. Die Scripte werden stets im globalen Kontext ausgeführt und können daher auf alle bereits in diesem Kontext vorhandenen Daten und Objekte zugreifen – namentlich das jQuery-Objekt selbst.

jQuery.param()

Das Utility `$.param()` nimmt ein Objekt entgegen und formt dieses in einen Querystring um. Vorsicht ist geboten, wenn das Objekt eine innere Hierarchie besitzt, d.h., wenn die Objekteigenschaften ihrerseits Objekte oder Arrays darstellen. Das Ergebnis der Serialisierung ist in diesem Fall nicht über alle Versionen von jQuery konstant.

Geht man von einem Objekt wie diesem aus

```
var obj = { "blumen1":"Rosen",
            "blumen2":"Tulpen",
            "blumen3":"Nelken" };
```

ergibt der Aufruf

```
var serialized = $.param(obj);
```

folgenden Querystring

```
blumen1=Rosen&blumen2=Tulpen&blumen3=Nelken
```


Die Daten werden automatisch URL-encoded, was deutlich wird, wenn man ein komplexeres Objekt übergibt. Hier besteht der Wert des einzigen Objekt-Properties aus einem Array:

```
var obj1 = {"blumen": [ "Rosen", "Tulpen", "Nelken" ]};
```

Dies ergibt URL-encoded

```
blumen%5B%5D=Rosen&blumen%5B%5D=Tulpen&blumen%5B%5D=Nelken
```

und entsprechend dekodiert

```
blumen[]=Rosen&blumen[]=Tulpen&blumen[]=Nelken
```

Eine Array-Eigenschaft wird also durch Wiederholung des Bezeichners mit angehängten (leeren) eckigen Klammern umgesetzt. Im Querystring müssen die Array-Klammern enkodiert werden. Bei Objekten, deren Eigenschaften wiederum Objekte sind, stellt sich die Lage wie folgt dar:

```
var obj2 = { "blumen1": {sorte: "Rosen", farbe: "rot"},
  "blumen2": {sorte: "Tulpen", farbe: "gelb"},
  "blumen3": {sorte: "Nelken", farbe: "weiss"} };
```

Das Objekt wird URL-encoded in dieser Weise serialisiert (die Zeilenumbrüche wurden der Lesbarkeit halber eingefügt):

```
blumen1%5Bsorte%5D=Rosen&blumen1%5Bfarbe%5D=rot
&blumen2%5Bsorte%5D=Tulpen&blumen2%5Bfarbe%5D=gelb
&blumen3%5Bsorte%5D=Nelken&blumen3%5Bfarbe%5D=weiss
```

Auch dies stellt sich dekodiert etwas lesbarer dar:

```
blumen1[sorte]=Rosen&blumen1[farbe]=rot
&blumen2[sorte]=Tulpen&blumen2[farbe]=gelb
&blumen3[sorte]=Nelken&blumen3[farbe]=weiss
```

Vorsicht – nicht abwärtskompatibel

Mit älteren Versionen von jQuery ergibt der Methodenaufruf zum Teil andere Serialisierungen. So erzeugt *jQuery 1.2.6* aus `obj1` folgenden String:

```
blumen=Rosen&blumen=Tulpen&blumen=Nelken
```

und aus `obj2` resultiert folgendes Ergebnis (dekodiert):

```
blumen1=[object+Object]
&blumen2=[object+Object]
&blumen3=[object+Object]
```

Die Kompatibilitätsprobleme werden derzeit in der Community diskutiert. Wir raten dazu, ausführlich zu testen, ob es wirklich unumgänglich ist, mit `$.param()` zu arbeiten, und ob die Scripts mit mehreren jQuery-Versionen lauffähig sein müssen.

Methode	Erläuterung
<code>.load(url)</code> <code>.load(url [,d] [,fn])</code>	Lädt die mit dem URL <code>url</code> bezeichnete Ressource per HTTP vom Server und fügt sie als HTML in das durch die Collection bestimmte Zielelement ein. (Nimmt optional weitere Parameter für Request-Daten <code>d</code> und eine Callback-Funktion <code>fn</code> entgegen.) Anmerkung: <code>.load()</code> (Ajax) und <code>.load()</code> (Event) werden nur anhand ihrer Parameter unterschieden.

Tabelle 4.72 jQuery-Convenience-Methoden für HTTP-Requests

Methode `.load()`

Sollen Daten vom Server geholt und einer Collection zur Verfügung gestellt werden, ist die `.load()`-Methode der einfachste Weg, dies zu bewerkstelligen. Sie arbeitet, anders als die bislang vorgestellten Ajax-Methoden, nicht im globalen Kontext. Unter den globalen Ajax-Utilities ist ihr die Methode `$.get()` am ähnlichsten, die allerdings (anders als `.load()`) eine explizite Callback-Funktion für den erfolgreichen Datentransferabschluss `success` aufruft. Die Methode `.load()` begnügt sich im Erfolgsfall hingegen damit, den HTML-Inhalt der Collection-Items stillschweigend durch die erhaltenen Daten zu ersetzen.

Um es einfach auszudrücken: Die Methode `.load()` holt eine externe Resource und lädt diese, sobald (und nur wenn) sie eingetroffen ist, beispielsweise in einen Container, der durch die Collection bestimmt wurde:

```
$('#ausgabe').load('beispiel.html');
```

Mehr als das braucht in vielen Fällen nicht geschrieben zu werden. Es ist nicht nötig, sich mit weiteren Details, wie dem XHR-Status oder Callbacks auseinanderzusetzen.

Same Domain Policy – kein Load externer Ressourcen

Beachten Sie, dass aufgrund der Sicherheitsbeschränkungen keine Dokumente aus fremden Domains per `load()` geladen werden können. Dies gilt grundsätzlich für Ajax-Requests. Liegt das anfragende Dokument in `www.meinedomain.de`, scheitert z. B. folgender Aufruf:

```
$('#ausgabe').load('http://www.anderswo.de/beispiel.html');
```

Eine Fehlermeldung erfolgt nicht – es werden lediglich keine Daten übermittelt.

Ein einfacher Fall besteht im Laden eines Textdokuments in einen Container. Es könnte sich auch um ein HTML-Fragment handeln, also Markups enthalten. Hier ein Ausschnitt aus dem HTML-Dokument, das die Inhalte lädt:

```
<div id="wrap">
  <p id="trigger">Bitte erzähl von Alice...</p>
  <div id="ausgabe"></div>
</div>
```

In den Container `div#ausgabe` sollen per `.load()` externe Inhalte geladen werden. Dies wird durch Klick auf den Textabsatz `p#trigger` veranlasst. Das Script hierzu ist sehr einfach:

```
$(document).ready(function() {
  $('#trigger').click(function(){
    $('#ausgabe').load('load_alice.txt');
  });
});
```

Ja reicht denn das? Tut es! Wirklich. Aber da ist noch mehr. Natürlich kann auch anstelle eines Textdokuments eine HTML-Seite adressiert werden. Nehmen wir an, diese sei wie folgt strukturiert (wobei hier nur ein Blick in den Body erfolgt):

```
<body>
  <div id="kap1">
    <p>Alice schloß die Thür auf ...</p>
  </div>
  <div id="kap2">
    <p>Es schien ihr ganz unnütz, ... </p>
  </div>
  <div id="kap3">
    <p>Denn sie hatte mehre hübsche Geschichten...</p>
  </div>
  ...
</body>
```

Jetzt wäre es natürlich schön, nicht stets das gesamte HTML-Dokument in den Ausgabecontainer pressen zu müssen. Wie wäre es, wenn man nur einen Abschnitt laden könnte? Das geht – und hier handelt es sich um ein »Extra-Feature« der `.load()`-Methode, die »regulären« Ajax-Requests nicht zur Verfügung steht.

Wie schon gezeigt, erwartet `.load()` als ersten Parameter (es gibt noch mehr) einen URL, die auf die Ressource zeigt. Diese wird als String übergeben. Nun erlaubt jQuery in diesem String zusätzlich auch die Übergabe eines Selektors, der ins Innere des Dokuments verweist.

Zwischen Dokument-URI und Selektor muss ein trennendes Leerzeichen stehen:

```
$('#ausgabe').load('url selector');
```

Vorsicht – vergessen Sie das Leerzeichen, wird der gesamte Inhalt des Body-Elements des externen Dokuments in den Ausgabecontainer eingefügt:

```
// Achtung: Dies funktioniert nicht (lädt ganzes Dokument):
$('#ausgabe').load('load_alice2.html#kap2');
```

Dies wäre korrekt, um über den Fragment Identifier den entsprechenden Absatz anzusprechen, taugt jedoch nicht als Filterkriterium für jQuery. Um nur den Textabsatz in `div#kap1` in die Ausgabe zu holen, schreiben Sie vielmehr korrekterweise:

```
// Dies lädt, wie gewünscht, nur den einen Containerinhalt:
$('#ausgabe').load('load_alice2.html #kap1');
```

Da bekannt ist, dass in diesem Dokument fünf entsprechende Abschnitte `#kap1`, `#kap2`, `#kap3` etc. sind, wird ein Zähler eingesetzt, um durch Klicken auf `p#trigger` zum nächsten Absatz weiterzugehen. (Bevor Sie fragen: Ja, das Argument von `.load()` kann tatsächlich auf diese Art dynamisch erstellt werden!)

```
$(document).ready(function() {
    var zaehler = 1;
    $('#trigger').click( function(){
        $('#ausgabe').load('load_alice2.html #kap' + zaehler);
        zaehler++;
        if (zaehler > 5) zaehler = 1;
    });
});
```

Was für ein Selektor kann übergeben werden?

Frei nach Radio Eriwan: Im Prinzip jeder formulierbare jQuery-Selektor. Wenn Sie möchten, können Sie im Rahmen des URL-Arguments von `.load()` also einen regelrechten Drilldown ins Dokument vornehmen. Angenommen, Sie wollten den Inhalt aller `<span>`-Elemente im vierten `<div>`-Container des Dokuments auslesen und in Ihre Ausgabe schreiben, ginge dies wie folgt:

```
$('#ausgabe').load('load_alice2.html div:eq(3) span');
```

Sie können also nach IDs, Klassen und Dokumentstruktur filtern, um per `.load()` eine HTML-Quelle anzuzapfen. Mehr darüber erfahren Sie im Praxisteil.

Die Methode `.load()` nimmt noch ein oder zwei weitere Parameter entgegen, die wahlweise übergeben werden. Der zweite Parameter ist ein JavaScript-Objekt, das dazu dient, dem Server im Rahmen des Requests Daten übergeben zu können. Dies kann der Fall sein, wenn über `.load()` eine Datenbankabfrage angestoßen werden soll:

```
// Prinzip einer load()-Anfrage mit Daten-Argument:
$('#out').load('db.php',{param1:'wert1',param2:'wert2'});
```

Das Datenargument kann ersatzlos entfallen (dies ist sogar eher die Regel). Dennoch kann auch in diesem Fall ein weiteres (eigentlich sonst drittes) Argument übergeben werden, da dieses sich vom Typ her unterscheidet und daher keine Verwechslungsgefahr besteht: Dieses dritte Argument ist ein Funktionsobjekt, das als Callback dient, wenn der Ladevorgang abgeschlossen ist.

Die Funktion kann beliebige Anweisungen enthalten, die nach Abschluss des Requests ausgeführt werden:

```
$('#ausgabe').load('load_alice.html', function() {
    alert("Alice fertig geladen!");
});
```

Nützlich ist jedoch, dass die Funktion stets auch Parameter übergeben bekommt. Es steht Ihnen frei, sie in Übergabeparametern »aufzufangen« oder nicht. Die Signatur lautet:

```
function(response, status, xhr) {...}
```

Dies bedeutet, dass die Callback-Funktion stets drei Argumente übergeben bekommt. Der erste Parameter `response` wird stets aus dem `responseText`-Property des Request-Objekts befüllt, sprich, er enthält den vom Server erhaltenen Antworttext. Der zweite Parameter `status` hat bei erfolgreicher Anfrage den Wert »success« oder »notmodified«, im Fehlerfall den Wert »error«.

Nur über Webserver

Obwohl `.load()` in Ansätzen auch aus dem File-System (über direkten Dateizugriff) funktioniert, muss das `status`-Property von einem Server geliefert werden. Um die Beispiele in ihrer ganzen Funktionalität nachzuvollziehen, müssen Sie sie über einen (auch lokalen) Webserver aufrufen.

Bliebe noch zu klären, wie jQuery damit umgeht, wenn das für `.load()` geforderte Dokument nicht existiert bzw. nicht ausgeliefert werden kann: In diesem Fall sendet der Server, der die Anfrage erhielt, eine Fehlermeldung, die über den `status`-Parameter der Callback-Funktion abgefragt werden kann. Dies geschieht folgendermaßen:

```
// das angeforderte Dokument existiert nicht:
$('#ausgabe').load('load_aliceXXX.html',
    function(response, status, xhr) {
        if (status == "error") {
            $('#ausgabe').html("Alice hat heute frei...");
        }
    });
```

4.12.6 Globale Handler-Methoden

Anstatt die Handler für bestimmte Punkte in der Ajax-Request-Kette im Konfigurationsobjekt des betreffenden Requests zu definieren, legt man auch gerne globale Handler an, die sich auf alle zur Laufzeit ausgeführten Requests beziehen. Diejenigen Requests, die mit Convenience-Methoden wie `.get()`, `.post()` oder `.load()` erzeugt werden, haben die Möglichkeiten eines Konfigurationsobjekts ohnehin nicht.

Methode	Erläuterung
<code>.ajaxStart(fn)</code>	Ajax-Ereignis; Legt eine Callback-Funktion <code>fn</code> für den Beginn des Ajax-Requests fest.
<code>.ajaxSend(fn)</code>	Ajax-Ereignis; Startet eine Callback-Funktion <code>fn</code> vor dem Absenden des Ajax-Requests.
<code>.ajaxError(fn)</code>	Ajax-Ereignis; Legt eine Callback-Funktion <code>fn</code> für den mit Fehlermeldung beendeten Ajax-Request fest.
<code>.ajaxSuccess(fn)</code>	Ajax-Ereignis; Startet eine Callback-Funktion <code>fn</code> , wenn der Ajax-Requests erfolgreich beendet wurde.
<code>.ajaxComplete(fn)</code>	Ajax-Ereignis; Legt eine Callback-Funktion <code>fn</code> für den erfolgreich beendeten Ajax-Request fest.
<code>.ajaxStop(fn)</code>	Ajax-Ereignis; Startet eine Callback-Funktion <code>fn</code> , sobald der letzte Ajax-Request beendet ist.

Tabelle 4.73 Globale jQuery-Handler für Ajax-Requests

Die Callbacks werden in der Reihenfolge der Tabelle getriggert, zuerst also `ajaxStart` und dann `ajaxSend`. Der `ajaxError`-Callback wird nur im Fehlerfall aufgerufen, wobei der Callback für `ajaxSuccess` unterbleibt. In jedem Fall wird anschließend `ajaxComplete` und zuletzt der `ajaxStop`-Callback aufgerufen.

.ajaxStart()

Der Callback für einen `ajaxStart`-Event wird getriggert, wenn das Script vor Beginn der Ausführung eines Requests »merkt«, dass aktuell keine weiteren Requests in Ausführung sind – mit anderen Worten wird der Callback nur vor dem allerersten Request aufgerufen bzw. wenn alle bisherigen Requests beendet wurden (also `.ajaxStop()` aufgerufen wurde). Von einer Gruppe von Requests triggert also nur der erste die Callbacks für `ajaxStart`.

.ajaxSend()

Ein `ajaxSend`-Ereignis findet individuell zu Beginn jedes Requests statt. Die entsprechenden Callbacks werden also durch jeden Request getriggert.

.ajaxSuccess()

Ist ein Request erfolgreich, hat also ein Ergebnis gebracht, tritt für diesen Request das `ajaxSuccess`-Ereignis ein, und die entsprechenden Callbacks werden getriggert.

.ajaxError()

Ist ein einzelner Request nicht erfolgreich, tritt für diesen anstelle seines *ajaxSuccess*-Ereignisses ein *ajaxError*-Ereignis ein und dessen Callbacks werden ausgelöst.

.ajaxComplete()

Nach Abschluss eines Requests, egal, ob erfolgreich oder nicht, wird ein `ajaxComplete`-Event getriggert und dessen Callbacks gestartet.

.ajaxStop()

Dieses Ereignis ist das Pendant zu `.ajaxStart()`. Es wird getriggert, wenn der letzte Request einer Gruppe beendet ist und keine weiteren Requests mehr laufen. Von einer Gruppe von Requests triggert der als Letzter beendete die Callbacks.

Um die Callbacks testweise aufzurufen, wird folgender HTML-Testaufbau eingesetzt, der aus einem leeren Div besteht, in das das Request-Ergebnis (in diesem Fall ein einfacher Text) und ein leeres `<ol>`-Element eingefügt wird:

```
<div id="ausgabe"></div>
<ol id="protokoll"></ol>
```

In Letzterem wird mittels der Ajax-Callbacks jeweils ein Listenelement eingefügt. Dies geschieht in der Reihenfolge der Callback-Aufrufe (siehe Abbildung 4.19). Folgender Code steht im `ready()`-Block:

```
// Binden der Callbackfunktion für ajaxStart:
$('#protokoll').ajaxStart(function(e, xhr) {
    $(this).append(
        '<li> Callback ajaxStart aufgerufen.</li>');
});

// Binden der Callbackfunktion für ajaxSend:
$('#protokoll').ajaxSend(function(e, xhr, settings) {
    $(this).append(
        '<li> Callback ajaxSend aufgerufen für "' +
        settings.url + '".</li>');
});
```

```
// Binden der Callbackfunktion für ajaxSend:
$('#protokoll').ajaxError(function(e, ajaxError) {
    $(this).append(
        '<li>Callback ajaxError aufgerufen.</li>');
});

// Binden der Callbackfunktion für ajaxSuccess:
$('#protokoll').ajaxSuccess(function(e, xhr, settings) {
    $(this).append(
        '<li> Callback ajaxSuccess aufgerufen für "' +
        settings.url + '".</li>');
});

// Binden der Callbackfunktion für ajaxComplete:
$('#protokoll').ajaxComplete(function(e, xhr, settings) {
    $(this).append(
        '<li>Callback ajaxComplete aufgerufen für "' +
        settings.url + '".</li>');
});

// Binden der Callbackfunktion für ajaxStop:
$('#protokoll').ajaxStop(function(e, xhr) {
    $(this).append(
        '<li>Callback ajaxStop aufgerufen.</li>');
});

// last, not least: der Ajax-Request
$.get('get_alice.txt', function(ergebnis){
    $('#ausgabe').html(ergebnis);
});
```

Listing 4.4 AJAX – jQuery Ausschnitt

Die Reihenfolge der Bindungen im Script spielt keine Rolle. Da die Methoden jeweils ein jQuery-Objekt zurückgeben, wären sie auch anreihbar. Sie könnten daher die hier als Einzelaufrufe geschriebenen Callback-Bindungen auch zusammenfassen (was im Endeffekt nicht unbedingt übersichtlicher wäre).

Der vereinbarte Callback für `.ajaxError()` wird in diesem Beispiel nie getriggert, da beim Holen der Ressource kein Fehler auftritt. Wenn Sie ihn bei der Arbeit sehen wollen, ändern Sie einfach den Ressourcenaufruf auf eine nicht existente Ressource.

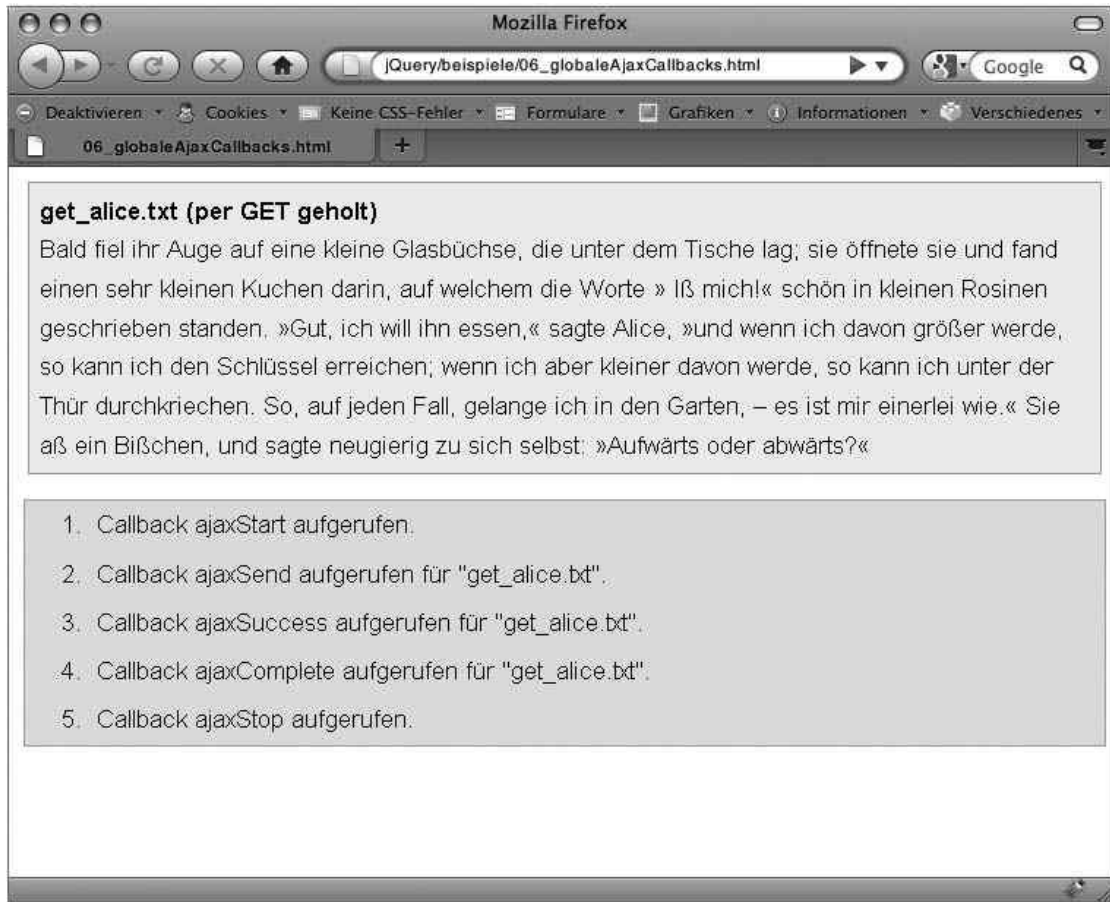


Abbildung 4.19 Globale Ajax-Callbacks im Testbett

4.13 Utilities – praktisches Dies und Das

Eine Reihe von Eigenschaften und Methoden – einige davon wurden bereits zu jeweils passendem Zeitpunkt vorgestellt – wird nicht auf Collections angewendet, sondern steht unabhängig von diesen direkt als »Angebot« des jQuery-Basisobjekts `$` zur Verfügung. Wir bezeichnen diese Eigenschaften und Methoden (vielleicht etwas unscharf) als »Utilities«, da sie eher Werkzeugcharakter besitzen.

Ein jQuery-Utility wird also allgemein wie folgt angesprochen:

```
jQuery.utility();
```

Da jQuery ein Alias für `$` ist, können Sie jederzeit auch kürzer schreiben:

```
$.utility();
```

Im Rahmen der Dokumentation haben wir uns entschieden (in den meisten Fällen jedenfalls), das explizitere `jQuery` anstelle von `$` einzusetzen. Sie sollten sich aber stets vergegenwärtigen, dass die andere Schreibweise ebenso gilt.

Utilities werden nicht auf Collections angewendet

Ist eine Methode oder Eigenschaft so definiert, dass sie direkt auf das jQuery-Basisobjekt angewendet werden soll, kann sie nicht für die Verarbeitung von Collections verwendet werden:

Korrekt ist `$.utility()` – aber nicht: `$(selector).utility()`.

In einigen Fällen existieren gleichnamige Methoden für beide Gebiete – wie im Fall von `jQuery.data()` und `.data()` –, die dann allerdings in verschiedenen Kontexten arbeiten. Die Namensgleichheit ist gerechtfertigt, da die Aufgaben identisch sind. Pauschal kann man sagen, dass die Collection-Methoden stets in einem Kontext arbeiten (dem ihrer Collection), während Utility-Methoden kontextfrei sind. Sie müssen also ihren Arbeitskontext entweder künstlich erhalten (als Argument) oder arbeiten im Kontext des `window`-Objekts.

4.13.1 Konfliktvermeidung mit anderen Frameworks

Der Fall ist denkbar, dass jQuery zusätzlich zu einem anderen Framework eingesetzt werden muss, das gleichfalls eine Funktion mit Namen `$()` definiert hat (dies trifft beispielsweise für »Prototype« zu).

Ohne vorbeugende Maßnahmen wird es JavaScript-Fehlermeldungen geben, da einerseits die jQuery-Methoden auf dieses »andere« `$()`-Objekt natürlich nicht anwendbar wären bzw. weil andererseits jQuery die fremde `$()`-Funktion eventuell wiederum mit seinem eigenen `$()`-Objekt überschreibt. jQuery stellt ein nützliches Utility zur Verfügung, das in dem einen oder anderen Fall Schaden abwendet.

Utility	Erläuterung
<code>jQuery.noConflict()</code>	Gibt den <code>\$</code> -Bezeichner frei (z. B. für die Benutzung durch ein anderes, parallel eingesetztes Framework)

Tabelle 4.74 jQuery-Utility zur Konfliktvermeidung

jQuery vor Fremdframework eingebunden

Ist jQuery als Framework zuerst eingebunden, braucht eigentlich nichts weiter unternommen werden. Sie müssen lediglich darauf achten, dass `$()` durch das Fremdframework überschrieben wurde, die jQuery-Methoden für jQuery also nur über das Alias `jQuery()` aufzurufen sind:

```
<!-- 1) jQuery definiert seine $()-Funktion -->
<script src="../../js/jquery-1.4.1.js"></script>
```

```
<!-- 2) Prototype überschreibt $() durch seine Version -->
<script src="../../pt/prototype.js"></script>
```

```
<script type="text/javascript">

    // Achtung: $() gehört hier Prototype!
    function absaetzeZaehlen() {
        // hier muss jQuery() statt $() stehen:
        alert(jQuery('p').length);
    }

    // Anweisungen, die nach dem Laden ausgeführt werden:
    jQuery(document).ready( function(){
        // auch hier gehört $() Prototype!
        alert(jQuery('p').length);
    });
</script>
```

Damit ließe sich leben. Sie können sich aber einiges an Umbenennungen ersparen und `$()` ad hoc zumindest innerhalb des `.ready()`-Blocks für jQuery »retten« – beispielsweise um anderswo bereits geschriebenen Code für diesen Bereich zu recyceln. Außerhalb von `.ready()` stünde `$()` aber noch immer ausschließlich für Prototype zur Verfügung.

Hintergrund ist, dass die anonyme Funktion in `.ready()` stets das jQuery-Objekt übergeben bekommt – es steht Ihnen frei, es in einem Parameter aufzufangen. Nennen Sie diesen `$`, kann das jQuery-Objekt wieder über `$()` adressiert werden – wie gesagt, das gilt dann nur innerhalb von `ready()`:

```
<!-- 1) jQuery definiert seine $()-Funktion -->
<script src="../../../js/jquery-1.4.1.js"></script>

<!-- 2) Prototype überschreibt $() durch seine Version -->
<script src="../../../pt/prototype.js"></script>

<script type="text/javascript">

    // Achtung: $() gehört hier Prototype!
    function absaetzeZaehlen() {
        // hier muss jQuery() statt $() stehen:
        alert(jQuery('p').length);
    }

    // Anweisungen, die nach dem Laden ausgeführt werden:
    jQuery(document).ready( function($){
        // jetzt gehört $() hier drinnen wieder jQuery!!
        alert($('p').length);
    });
</script>
```

Ein Konflikt zwischen dem »inneren« und dem »äußeren« \$ kann hierbei nicht auftreten – alles, was sich in der `.ready()`-Methode abspielt, befindet sich dank einer Closure in einem eigenen Gültigkeitsbereich. Ähnliches gilt aber auch für den Rest des Codes. Hierfür wird ein kleiner Trick verwendet, nämlich eine anonyme, »vor Ort« ausgeführte Funktion. Das Prinzip ist dieses:

```
(function(){
    // Code, der sofort ausgeführt wird
})();
```

Achten Sie auf die runden Klammern am Ende des Ausdrucks (die Parameterklammern) – diese bewirken die augenblickliche Ausführung. Das zweite Paar runder Klammern um das Funktionsobjekt selbst ist notwendig, da die Parameterklammern nicht unmittelbar auf die schließende geschweifte Klammer folgen dürfen (Syntaxfehler).

Übergeben Sie der Funktion einen Wert, sind Sie auch schon fertig – es bietet sich das Objekt namens `jQuery` an, das (unabhängig davon, was mit `$` passiert ist) in jedem Fall existiert. Das übergebene Objekt gelangt (wie voraussehbar!) über einen Parameter namens `$` ins Innere der Funktion:

```
(function($){
    // Code, der sofort ausgeführt wird
    // hier ist $ nun Synonym für jQuery
})(jQuery);
```

Schreiben Sie das vorangegangene Beispiel mit dieser Erweiterung neu, erhalten Sie:

```
<!-- 1) jQuery definiert seine $()-Funktion -->
<script src="../../js/jquery-1.4.1.js"></script>

<!-- 2) Prototype überschreibt $() durch seine Version -->
<script src="../../pt/prototype.js"></script>

<script type="text/javascript">

    // Achtung: $() gehört hier Prototype!

    // aber nicht in diesem Funktionsobjekt
    (function($){

        function absaetzeZaehlen() {
            // hier kann nun auch $() stehen:
            alert($('p').length);
        }
    })
```

```

})(jQuery);

// Anweisungen, die nach dem Laden ausgeführt werden:
jQuery(document).ready( function($){
    // jetzt gehört $() hier drinnen wieder jQuery!!
    alert($('p').length);
});
</script>

```

Im Grunde wird nun mit zwei Closures gearbeitet – die eine davon im `.ready()`-Block, die andere im anonymen Funktionsobjekt. Beide schirmen jQuerys `$` vor Fremdeinwirkungen ab und erlauben, dass der Bezeichner außerhalb der Closures durch ein anderes Framework verwendet wird. Beachten Sie jedoch, dass die Kommunikation zwischen beiden Closures nicht trivial ist – beispielsweise, wenn Sie eine Funktion, die im anonymen Closure-Block definiert ist, im `.ready()`-Block aufrufen möchten.

Bislang bestand also keine Notwendigkeit für weitere Maßnahmen. Der Fall kann aber auch anders liegen, wie jetzt geschildert werden soll.

Fremdframework vor jQuery eingebunden

Wird das fremde Framework vor jQuery eingebunden, überschreibt jQuery wiederum dessen `$()` durch sein eigenes. Anweisungen an das andere Framework werden unwirksam. Diesem Umstand gehen Sie aus dem Weg, indem Sie an den Anfang Ihres eigenen Scripts die Anweisung `jQuery.noConflict();` setzen. jQuery gibt nun `$()` für das Fremdframework frei und verwendet nur noch das Alias `jQuery()`:

```

<!-- 1) Prototype definiert seine $()-Funktion -->
<script src="../../pt/prototype.js"></script>

<!-- 2) jQuery überschreibt $() durch seine Version -->
<script src="../../js/jquery-1.4.1.js"></script>

<script type="text/javascript">
    // Prototypes $() wird reaktiviert:
    jQuery.noConflict();

    // Achtung: $() gehört hier Prototype!
    function absaetzeZaehlen() {
        // hier muss jQuery() statt $() stehen:
        alert(jQuery('p').length);
    }

```

```
// Anweisungen, die nach dem Laden ausgeführt werden:
jQuery(document).ready( function(){
    // auch hier gehört $() Prototype!
    alert(jQuery('p').length);
});
</script>
```

Beim diesem Einsatz von `jQuery.noConflict()` müssen Sie anschließend alle Verweise auf die jQuery-Funktion von `$(...)` zu `jQuery(...)` umbenennen. Betroffen sind logischerweise alle Aufrufe von `$()`, die außerhalb der `ready()`-Methode stattfinden, beispielsweise innerhalb globaler Funktionen.

Auch innerhalb seiner `ready()`-Methode erkennt jQuery die `$()`-Funktion zunächst nicht mehr. Über den Übergabeparameter lässt sich `$` jedoch, wie gehabt, für den Bereich reaktivieren:

```
<script src="../../pt/prototype.js"></script>
<script src="../../js/jquery-1.4.1.js"></script>
<script type="text/javascript">
    jQuery.noConflict();

    // Achtung: $() gehört hier Prototype!
    function absaetzeZaehlen() {
        // hier muss jQuery() statt $() stehen:
        alert(jQuery('p').length);
    }

    // Anweisungen, die nach dem Laden ausgeführt werden:
    jQuery(document).ready( function($){
        // jetzt gehört $() hier drinnen wieder jQuery!!
        alert($('p').length);
    });
</script>
```

Für den restlichen Bereich bleibt Ihnen natürlich auch hier die Option mit dem anonymen Funktionsobjekt, dem, wie im Vorfeld beschrieben, das jQuery-Objekt als Parameter übergeben wird.

Wie funktioniert `jQuery.noConflict()`?

Eigentlich ganz simpel: Beim Laden prüft jQuery, ob `$` bereits definiert ist. Ist dies der Fall, wird jenes `$` unter einem anderen Namen »beiseitegeschafft«, also gespeichert. Der Bezeichner `$` gehört damit nun jQuery. Die Instruktion `jQuery.noConflict()` veranlasst, dass jQuery's `$` im Anschluss wieder durch die »gebunkerte« Kopie des ehemaligen `$` überschrieben und so der ursprüngliche Zustand vor dem Laden von jQuery wiederhergestellt wird. Das Alias `jQuery` ist hiervon nicht betroffen und kann weiter verwendet werden.

Variante 2 – ein neues Alias für \$()

Des Weiteren kann `.noConflict()` ein weiteres Alias zu `$()` und `jQuery()` erzeugen – die Methode gibt nämlich das jQuery-Objekt zurück, das einfach in eine Variable gespeichert werden kann:

```
<script src="../../pt/prototype.js"></script>
<script src="../../js/jquery-1.4.1.js"></script>
<script type="text/javascript">
    var Galileo = jQuery.noConflict();

    // statt $() kann nun auch Galileo() stehen:
    function absaetzeZaehlen() {
        alert(Galileo('p').length);
    }

    // nach wie vor kann auch jQuery() geschrieben werden:
    jQuery(document).ready( function(){
        // hier darf Galileo() und jQuery() stehen
    });
</script>
```

Variante 3 – zusätzlich auch jQuery() substituieren

Für den seltenen Fall, dass selbst diese Konfliktlösung nicht ausreicht, bietet sich die *extreme*-Variante des Konfliktmanagements an, die sowohl `$()` als auch `jQuery()` durch ein Alias ersetzt. Dies ist nur dann erforderlich, wenn jQuery in einen Konflikt mit sich selbst steht, beispielsweise aufgrund von Versionsinkompatibilitäten. Dies ist aber selten der Fall.

Utility	Erläuterung
<code>jQuery.noConflict(true)</code>	Gibt den <code>\$</code> -Bezeichner <i>und</i> den <code>jQuery</code> -Bezeichner frei und erzeugt eine Referenz auf das ursprüngliche jQuery-Objekt.

Tabelle 4.75 jQuery-Utility zur Konfliktvermeidung (»extreme«-Variante des Aufrufs)

Der `.noConflict()`-Methode muss bei der Erzeugung des Alias lediglich der Boolesche Wert `true` übergeben werden, um den *extreme*-Modus der Konfliktvermeidung für jQuery in Kraft zu setzen. Sowohl `$()` als auch `jQuery()` sind nun nicht zugänglich:

```
<script type="text/javascript">
    var Galileo = jQuery.noConflict(true);

    // statt $() darf nur noch Galileo() stehen:
```

```
function absaetzeZaehlen() {
    alert(Galileo('p').length);
}

// auch hier darf nicht mehr jQuery() geschrieben werden:
Galileo(document).ready( function(){
    // hier darf nur noch Galileo() verwendet werden
});
</script>
```

Selbstverständlich könnte auch dieses Szenario durch die Übergabe von \$ als Parametername an die anonyme Funktion in `.ready()` »gemildert« werden.

Vorsicht mit dem extreme-Modus

Seien Sie mit dem *extreme*-Modus aber *extrem* vorsichtig, besonders, wenn Sie Erweiterungen einsetzen. Alle Plugins, die intern auf den Funktionsnamen `jQuery()` verweisen, werden nicht mehr funktionieren!

4.13.2 Browser und Feature Detection

Gelegentlich mag es von Interesse sein, herauszufinden, welche Eigenschaften die aktuelle Laufzeitumgebung hat, in der das Script ausgeführt wurde. Hierzu wurden früher Browsererkennungen vorgenommen, oder, weil dies der häufigste zu berücksichtigende Fall war, unmittelbar geprüft, welches CSS-Boxmodell aktuell eingesetzt wird. Beides kann direkt als jQuery-Eigenschaften `jQuery.browser` und `jQuery.boxModel` abgefragt werden. Diese Vorgehensweise gilt jedoch gegenüber einer Feature-Überprüfung mit `jQuery.support` als veraltet oder zumindest nicht unproblematisch. Speziell `jQuery.boxModel` ist lediglich aus historischen Gründen (noch) implementiert.

Utility	Erläuterung
<code>jQuery.boxModel</code>	Veraltet. Prüft, ob das aktuell im Browser verwendete Boxmodell dem W3C-Box-Modell entspricht. Anmerkung: Diese Eigenschaft ist aus Gründen der Abwärtskompatibilität in jQuery 1.4 enthalten. Es wird empfohlen, stattdessen auf Feature Detection mittels <code>jQuery.support</code> auszuweichen.

Tabelle 4.76 jQuery-Utilities zur Browser- und Feature Detection

Utility	Erläuterung
<code>jQuery.browser</code>	Enthält Flags (<code>true/false</code>) für erkannten Browser (unterschiedet Webkit/Safari, Opera, MSIE, Mozilla) und die jeweilige Versionsnummer. Anmerkung: Wenn möglich, auf <code>jQuery.support</code> ausweichen, da <code>jQuery.browser</code> anfällig gegen Spoofing oder Maskierung durch den User Agent ist. Die Eigenschaft wird jedoch langfristig zur Verfügung stehen.
<code>jQuery.support</code>	Enthält Properties, anhand derer der aktuelle Browser und seine Features erschlossen werden können.

Tabelle 4.76 jQuery-Utilities zur Browser- und Feature Detection (Forts.)

Eigenschaft `jQuery.support`

Bei der Eigenschaft `jQuery.support` handelt es sich um ein Objekt, aus dem eine Reihe interessanter Eigenschaften der Laufzeitumgebung ausgelesen werden können – u.a. auch das verwendete Boxmodell. Ein paar der Eigenschaften möchten wir hier auflisten (weitere sind eher für Plugin-Entwickler von Interesse als für den alltäglichen Gebrauch):

- ▶ `jQuery.support.boxModel`
Hat den Wert `true`, wenn das W3C-Boxmodell gilt (ist `false` in MSIE, sofern dieser im Quirks-Mode arbeitet).
- ▶ `jQuery.support.opacity`
Hat den Wert `true`, wenn die CSS-Eigenschaft `opacity` unterstützt wird (ist `false` in MSIE, bei dem stattdessen Alpha-Filter eingesetzt werden müssen).
- ▶ `jQuery.support.style`
Hat den Wert `true`, wenn (wie nach DOM Level 2) die Inline-Styleeigenschaften eines Elements über die `style`-Eigenschaft ausgelesen werden kann (`false` in MSIE, wo stattdessen die Eigenschaft `cssText` verwendet wird).

Den Browsertyp oder die Version direkt auszulesen wird als unnötig erachtet. Eine entsprechende Eigenschaft ist daher in `jQuery.support` nicht implementiert. Weichen Sie, wenn diese Information unbedingt erforderlich ist, auf `jQuery.browser` aus.

4.13.3 Utilities zur Array-Verarbeitung

Speziell für die Verarbeitung von Arrays (hier sind nicht jQuery-Collections gemeint, bei denen es sich technisch nicht um Arrays handelt) stellt jQuery dem Webprogrammierer eine Reihe nützlicher Tools an die Seite. Diese sind teilweise vergleichbar mit Iterations- und anderen Hilfsmethoden, wie sie in den neueren

Versionen von JavaScript (ab 1.6) bereits enthalten sind, aber noch nicht in allen Browsern implementiert wurden. Die jQuery-Methoden hingegen stehen browserübergreifend zur Verfügung und sind weitestgehend selbsterklärend.

Utility	Erläuterung
<code>jQuery.each()</code>	Iteriert durch ein Array oder Objekt bzw. eine jQuery-Collection.
<code>jQuery.grep()</code>	Durchsucht ein Array anhand eines Filterkriteriums, ohne das Originalarray zu beeinträchtigen.
<code>jQuery.inArray()</code>	Gibt die Indexposition eines Elements in einem Array zurück.
<code>jQuery.makeArray()</code>	Wandelt ein »arrayähnliches Objekt« in ein echtes JavaScript-Array.
<code>jQuery.map()</code>	Schreibt den Inhalt eines Arrays oder arrayähnlichen Objekts in ein neues Array.
<code>jQuery.merge()</code>	Verbindet die Inhalte zweier übergebener Arrays in das erste der übergebenen Arrays.
<code>jQuery.unique()</code>	Erstellt eine Kopie eines übergebenen Arrays <code>arr</code> aus dem DOM-Knoten, aus dem aber doppelt auftretende Knoten entfernt wurden. Achtung: Es muss sich um ein DOM-Array handeln.

Tabelle 4.77 jQuery-Utilities für Array-Verarbeitung

jQuery.each()

Diese Utility-Methode `$.each()` arbeitet analog zur jQuery-Methode `.each()`, wobei sie nicht an eine Collection angereicht wird, sondern ihr eine Collection als Argument übergeben werden muss. Außer Collections nimmt `$.each()` aber auch »normale« Arrays oder JavaScript-Objekte entgegen und arbeitet diese in einer Schleife ab. Man könnte also vereinfachend sagen, dass `$.each()` der Anwendung einer `for`-Schleife auf ein Array bzw. der einer `for-in`-Schleife auf ein Objekt gleicht:

```
var arr = ['Montag',
'Dienstag',
'Mittwoch',
'Donnerstag',
'Freitag',
'Samstag',
'Sonntag']
// Inhalte der Arrayfächer landen als Absätze im Dokument:
$.each(arr, function(i, val){
    $('<p>' + (i + 1) + '. ' + val + '</p>').appendTo('body');
});
```

Hierbei bekommt die Callback-Funktion einen Parameter `i` übergeben, in den der Index des aktuell verarbeiteten Array-Items abgelegt wird. Im Inneren der Funktion dient der Index zum Nummerieren der Ausgabe. Der zweite nutzbare Parameter `val` der Callback-Funktion enthält den Wert des verarbeiteten Items als String (der alternativ über das Keyword `this` ebenfalls zugänglich wäre).

Dasselbe geht analog auch mit einem Objekt. In diesem Fall nimmt ein hier ebenfalls übergebenes `i` keinen numerischen Indexwert an, sondern den Property-Namen der aktuell verarbeiteten Eigenschaft des Eingabeobjekts. In `val` wird wieder der Wert des Items abgelegt:

```
var obj = { Fach1:'Biologie',
            Fach2:'Physik',
            Fach3:'Geschichte' };
$.each(obj, function(i, val){
    $('<p>' + i + ': ' + val + '</p>').appendTo('body');
});
```

jQuery.merge()

Die Utility-Methode `$.merge()` verschmilzt zwei übergebene Arrays und gibt ein Ergebnisarray zurück. Bei der Verarbeitung wird das erste übergebene Array durch das Ergebnis überschrieben – die Methode arbeitet also *destruktiv*. Verwenden Sie ein leeres Array-Literal als erstes Argument, kann das Utility auch zum Kopieren eines Arrays eingesetzt werden:

```
// Grundrezept und zwei Zutatenlisten :
var grundrezept = ['Zucker','Eier','Mehl','Salz'];
var zutaten1 = ['Hefe','Wasser'];
var zutaten2 = ['Backpulver','Wasser'];
// Grundrezept + erste Zutatenliste :
$.merge(grundrezept, zutaten1);
// Array grundrezept wurde überschrieben:
$('<p>' + grundrezept.toString() + '<p>').appendTo('body');
```

Ein Array wird kopiert, indem ein leeres Array als erstes Argument übergeben und der Rückgabewert der Methode in einer Variablen aufgefangen wird:

```
var rezeptkopie = $.merge([], grundrezept);
```

Auf gleichem Weg kann auch ein temporäres erstes Eingabeargument für einen Merge-Prozess erzeugt werden, wodurch das Originalarray erhalten bleibt.

Hier werden aus dem Grundrezept und den Zutatenlisten zwei neue Rezeptarrays erzeugt. Das Array `grundrezept` bleibt dabei unverändert:

```
var rezept1 = $.merge( $.merge([], grundrezept), zutaten1);
var rezept2 = $.merge( $.merge([], grundrezept), zutaten2);
$('<p>' + rezept1.toString() + '<p>').appendTo('body');
$('<p>' + rezept2.toString() + '<p>').appendTo('body');
```

jQuery.unique()

Die Utility-Methode `$.unique()` arbeitet ausschließlich mit DOM-Arrays (auch bekannt als *HTML-Collections*) und filtert aus diesen doppelt referenzierte Knoten aus. Die Methode arbeitet *destruktiv*, überschreibt also das Eingabearray. (Das veränderte Array ist gleichfalls als Rückgabewert verfügbar.)

Für das Beispiel gehen wir von einem Dokument aus, das fünf Textabsätze enthält, von denen zwei über einen ID verfügen:

```
<p id="p1">...</p>
<p id="p2">...</p>
<p>...</p>
<p>...</p>
<p>...</p>
```

Der Weg zu einem DOM-Array, das Duplikate enthält, erscheint etwas forciert – vergleichbare Array können in der Praxis aber durchaus spontan auftreten, wonach es dann möglicherweise Duplikate zu entfernen gilt. Zunächst wird ein DOM-Array aufgrund des Tag-Namens erstellt, anschließend eines auf Grundlage der IDs. Beide Arrays werden über `$.merge()` verschmolzen:

```
var domArray = $('p').get();
var domArray2 = $('#p1, #p2').get();
domArray = $.merge(domArray, domArray2);
```

Ein kleiner Test zeigt, dass das Array nun tatsächlich sieben Knotenreferenzen enthält, also zwei Duplikate (wie wir wissen, da wir den Quelltext kennen):

```
alert("Vor $.unique: " + domArray.length); //-> 7
```

Nun werden die Duplikate mittels `$.unique()` ausgefiltert:

```
$.unique(domArray);
```

Eine neue Längenprüfung zeigt, dass die Duplikate entfernt wurden:

```
alert("Nach $.unique: " + domArray.length); // -> 5
```

jQuery.grep()

Das Utility `$.grep()` nimmt als erstes Argument ein Array entgegen und gibt ein Array bestehend aus all denjenigen Items zurück, die den Kriterien einer als zwei-

tes Argument übergebenen Filterfunktion gerecht werden. Die Methode arbeitet *nicht destruktiv*, erhält also das Originalarray der Eingabe. Die Filterfunktion erhält den Wert `val` und den Index `i` des betrachteten Items übergeben:

```
var zahlen = [1,2,4,5,6,12,15,17,22,23,24,30,31];
var zahlen_odd = $.grep(zahlen, function(val,i){
    return (val%2 == 1); // true für ungerade Zahlen
});
// zahlen_odd: [1,5,15,17,23,31]
```

Aus dem Eingabearray sucht `$.grep()` die ungeraden Zahlen heraus, für die die Filterfunktion den Wert `true` zurückgibt. Das Array `zahlen` wird nicht angetastet und kann als Argument eines weiteren Greps verwendet werden, der diesmal ein Array aus den geraden Zahlen erzeugt:

```
var zahlen_even = $.grep(zahlen, function(val,i){
    return (val%2 == 0); // true für gerade Zahlen
});
// zahlen_even: [2,4,6,12,22,24,30]
```

Ein optionaler dritter Parameter (`invert`) von `$.grep()` kehrt die Wirkung des Filters um, wenn für ihn `true` übergeben wird. Beachten Sie, dass die Filterfunktion nun das Gegenteil von dem tut, was das Ergebnis erwarten lässt:

```
var zahlen_odd2 = $.grep(zahlen, function(val,i){
    return (val%2 == 0); // true für gerade Zahlen
}, true);
// zahlen_odd2: [1,5,15,17,23,31]
```

jQuery.map()

Die Utility-Methode `$.map()` iteriert durch das Eingabearray und gibt jedes Item nach Verarbeitung durch die als zweites Argument übergebene Funktion in ein neues Array aus. Die Methode arbeitet *nicht destruktiv*, belässt also das Eingabearray, wie es ist:

```
var zahlen = [1,2,4,5,6,12,15,17];
var zahlen_mod = $.map(zahlen, function(val,i){
    return (val%2 == 1);
});
// zahlen_mod: [true,false,false,true,false,false,true,true]
```

Die Callback-Funktion erhält zwei Argumente, nämlich den Wert `val` und den Index `i` des verarbeiteten Items des Eingabearrays:

```
var zahlen_doppelt = $.map(zahlen, function(val,i){
    return val * 2;
```

```
});
// zahlen_doppelt: [2,4,8,10,12,24,30,34]
```

Der Index kann in die Berechnung des Ergebniswerts mit einbezogen werden:

```
var zahlen_index = $.map(zahlen, function(val,i){
    return val * i;
});
// zahlen_index: [0,2,8,15,24,60,90,119]
```

jQuery.inArray()

Die Methode `$.inArray()` überprüft, ob ein Wert, der der Methode als erstes Argument übergeben wird, dem eines Items des als zweites Argument übergeben Arrays entspricht. Ist dies der Fall, wird der Index `i` des betreffenden Items zurückgegeben, für das eine Übereinstimmung auftritt. Wird keine Übereinstimmung gefunden, gibt die Methode den Wert `-1` zurück. Hierin verhält sie sich analog zur JavaScript-Methode `indexOf()`:

```
var werte = ['Peter','Kurt',7,100,'77'];
```

Die Zahl 7 wird an dritter Stelle gefunden:

```
var erg1 = $.inArray(7,werte);
// erg1: 2
```

Der String »Peter« wird am Anfang des Arrays (Position 0) gefunden:

```
var erg2 = $.inArray('Peter',werte);
// erg2: 0
```

Der String »Pete« wird nicht gefunden. Teilübereinstimmungen werden von `$.inArray()` also nicht erkannt:

```
var erg3 = $.inArray('Pete',werte);
// erg3: -1
```

4.13.4 Utility zur Stringbearbeitung

Das Stringobjekt von JavaScript bietet zwar von Haus aus so allerhand – ein Utility zur Normalisierung bzw. zum Entfernen von vorausgehenden und folgenden Whitespace-Zeichen aus einer Zeichenkette fehlt jedoch.

Utility	Erläuterung
<code>jQuery.trim()</code>	Entfernt Whitespace von Anfang und Ende eines übergebenen Strings <code>str</code> .

Tabelle 4.78 jQuery-Utilities zur Stringbearbeitung

jQuery.trim()

Für diesen Zweck bietet jQuery das Utility `jQuery.trim()` an, das störende Leerzeichen von Anfang und Ende eines Strings entfernen kann:

```
$(document).ready(function() {
    // String mit überflüssigem Leerraum:
    var string1 = "    viel Nichts um Lärm    ";
    alert(string1.length); //-> 27
    string1 = jQuery.trim(string1);
    alert(string1.length); //-> 19

    // String mit internem Leerraum:
    var string2 = "viel    Nichts    im Lärm";
    alert(string2.length); //-> 27
    string2 = jQuery.trim(string2);
    alert(string2.length); //-> 27
});
```

Es findet dabei jedoch ausschließlich eine Entfernung von »leading«- und »trailing«-Whitespace statt (siehe `string1`). Eine echte Normalisierung, d.h. das Zusammenfassen mehrerer aufeinanderfolgender Leerzeichen, ist mit `trim()` nicht möglich (siehe `string2`). Auch verändert die Funktion nicht den Originalstring, sondern gibt eine bearbeitete Kopie zurück (dies wird hier dazu verwendet, um das Original zu überschreiben).

4.13.5 Utilities für DOM-Knotenverarbeitung

Spezielle Utility-Methoden von jQuery nehmen DOM-Knoten als Argumente entgegen. Dies kann im Rahmen einer Strukturprüfung erfolgen (»Enthält Knoten A den Knoten B?«) oder um die übergebenen Knoten als jQuery-Objekt weiterverarbeiten zu können. Es sind dies `$.contains()` und `$.pushStack()`.

Utility	Erläuterung
<code>jQuery.contains(elem1, elem2)</code>	Prüft, ob ein DOM-Knoten <code>elem1</code> einen DOM-Knoten <code>elem2</code> enthält.
<code>jQuery.pushStack(elems)</code>	Legt eine Sammlung von DOM-Elementen auf den jQuery-Verarbeitungsstack.

Tabelle 4.79 jQuery-Utilities für DOM-Knoten

jQuery.contains()

Die Methode `$.contains()` prüft, ob zwei DOM-Knoten in Relation zueinander stehen, indem der zweite übergebene Knoten Teil des Elementinhalts des ersten

ist. In folgendem HTML-Fragment befindet sich `p#p2` im Inneren von `div#box1`, der Container `div#box2` dagegen nicht:

```
<div id="box1">
  <p id="p1">Eins.</p>
  <p id="p2">Zwei.</p>
  <p id="p3">Drei.</p>
  <p id="p4">Vier.</p>
</div>
<div id="box2">Box2</div>
```

Dies kann nun auch geprüft werden:

```
$(document).ready(function() {
  var box1 = document.getElementById('box1');
  var p2 = document.getElementById('p2');
  var box2 = document.getElementById('box2');

  if(jQuery.contains(box1,p2)) {
    alert("p2 ist in box1");          // Ausgabe
  }
  else {
    alert("p2 ist nicht in box1");
  }

  if(jQuery.contains(box1,box2)) {
    alert("box2 ist in box1");
  }
  else {
    alert("box2 ist nicht in box1"); // Ausgabe
  }
});
```

Beachten Sie, dass der Methode *zwei DOM-Knoten* übergeben werden müssen. Mit jQuery-Objekten funktioniert der Vergleich nicht.

jQuery.pushStack()

Die `$.pushStack()`-Methode nimmt DOM-Knoten als Array entgegen und bildet aus diesen ein neues jQuery-Objekt, das weiterverarbeitet werden soll. Das Prinzip ist wie folgt

```
var col = jQuery([]).pushStack(DOM-Array);
```

oder, mit `$()` anstelle von `jQuery()` geschrieben

```
var col = $([]).pushStack(DOM-Array);
```


Zunächst muss über die jQuery-Factory-Funktion eine leere Collection gebildet werden, auf die die `jQuery.pushStack()`-Methode angewendet wird. Hierzu wird ihr ein leeres Array-Literal übergeben (ein direkter Aufruf als `jQuery.pushStack()` oder `$.pushStack()` scheitert!): `jQuery([])`. Die `jQuery.pushStack()`-Methode bekommt ein Array aus DOM-Knoten übergeben. Dies kann direkt aus einer DOM-Methode wie `getElementsByTagName()` stammen oder künstlich als Array-Literal erstellt werden. Betrachten wir nun im Folgenden beide Möglichkeiten.

Aus dem bereits vorhin verwendeten Quelltext

```
<div id="box1">
  <p id="p1">Eins.</p>
  <p id="p2">Zwei.</p>
  <p id="p3">Drei.</p>
  <p id="p4">Vier.</p>
</div>
<div id="box2">Box2</div>
```

werden ein Array aus allen `<p>`-Containern sowie Einzelreferenzen auf `p#p1`, `p#p2` und `div#box2` gebildet. Sie sollen über `jQuery.pushStack()` in eine Collection `col` überführt und klickbar gemacht werden. Zunächst für das DOM-Array:

```
$(document).ready(function() {
  var alleP = document.getElementsByTagName('p');

  var col = jQuery([]).pushStack(alleP);
  col.click(function(){
    $(this).css({border:'1px solid red'})
  });
});
```

Die `<p>`-Container sind klickbar und erhalten bei Klick einen roten Rahmen.

Nun soll dasselbe nur für die ersten beiden Absätze, aber auch für den folgenden `<div>`-Container geschehen. Es werden diesmal Referenzen auf Einzelknoten erzeugt, die dann als Array-Literal übergeben werden müssen:

```
$(document).ready(function() {
  var p1 = document.getElementById('p1');
  var p2 = document.getElementById('p2');
  var b2 = document.getElementById('box2');

  var col = jQuery([]).pushStack([p1,p2,b2]);
  col.click(function(){
    $(this).css({border:'1px solid red'})
  });
});
```

```
});
});
```

Schwieriger wird es, wenn das DOM-Array zusammen mit einer oder mehreren Einzelreferenzen verwendet werden soll, da die Methode `jQuery.pushStack()` nur ein Array und keine gemischten Argumente entgegennimmt. Aufrufe wie `jQuery([]).pushStack(alleP,[b2])` oder Ähnliches scheitern.

Hilfe bringt das jQuery-Utility `$.merge()`, das zwei Arrays zusammenfasst und so das Argument in die benötigte Form bringt:

```
$(document).ready(function() {
    var b2 = document.getElementById('box2');
    var alleP = document.getElementsByTagName('p');

    var col = jQuery([]).pushStack(jQuery.merge(alleP,[b2]));
    col.click(function(){
        $(this).css({border:'1px solid red'})
    });
});
```

Anschließend sind alle Textabsätze und auch der `<div>`-Container klickbar.

4.13.6 Utilities für Funktionsaufrufe

Einige Utilities von jQuery erleichtern die Lösung von Standardproblemen im Zusammenhang mit Funktionen, Scripts und Scopes. Beispielsweise erzeugt `jQuery.noop` ein leeres Funktionsobjekt, das eingesetzt werden kann, wenn ein funktionsloser Platzhalter für eine Callback-Funktion benötigt wird.

Utility	Erläuterung
<code>jQuery.globalEval()</code>	Führt übergebenen JavaScript-Quellcode im globalen Scope aus.
<code>jQuery.noop</code>	Ein leeres Funktionsobjekt
<code>jQuery.proxy()</code>	Nimmt ein Funktionsobjekt entgegen und gibt es mit dem gewünschten Scope zurück.

Tabelle 4.80 jQuery-Utilities für Funktionsobjekte

jQuery.globalEval(code)

Die Hilfsmethode `jQuery.globalEval()` führt übergebenen Code im globalen Scope aus. Dies geschieht, entgegen dem Namen, nicht über die `eval()`-Routine von JavaScript, sondern vielmehr, indem als erstes Kindelement des `<head>` ein `<script>`-Block erzeugt und sofort ausgeführt wird. Als Demonstration soll hier

eine globale Variable erzeugt und von einer globalen Funktion aus angesprochen werden (dies klappt!). Eine im `.ready()`-Block definierte Variable hingegen ist in einer Closure verborgen und kann nicht angesprochen werden:

```
$(document).ready(function() {
    var p1 = document.getElementById('p1');
    var test1 = "Unerreichbar für externe Funktion";

    jQuery.globalEval('var test2 = "Erreichbar
        für externe Funktion"');

    $('#p1').click(function(){
        test();
    });
});

function test() {
    // test1 ist hier 'undefined':
    if ((typeof test1)!='undefined') {
        alert(test1);
    } else {
        alert("test1 ist nicht erreichbar!");
    }
    // test2 ist global deklariert:
    if((typeof test2)!='undefined'){
        alert(test2);
    } else {
        alert("test2 ist nicht erreichbar!");
    }
}
```

jQuery.proxy()

Das Utility `$.proxy()` verschiebt den Kontext, in dessen Zusammenhang eine Funktion ausgeführt wird, auf ein beliebiges anderes Objekt. Hier wird bei Klick auf einen Textabsatz ein anderer Textabsatz mit einem CSS-Style versehen (zugegeben, hierfür gäbe es andere, näherliegende Lösungen):

```
$(document).ready(function() {
    var p1 = document.getElementById('p1');
    var p2 = document.getElementById('p2');

    $('#p1').click( $.proxy( function() {
        // meldet: 'p2':
        alert(this.id);
        // färbt Text in p2 rot:
        $(this).css({color:'red'})
    }, p2));
});
```

```

    }, p2)
  );
});

```

Die Methode erhält als erstes Argument eine Callback-Funktion, die in neuem Kontext auszuführen ist, und als zweites Argument eine Referenz auf den Kontext, in dem dies zu geschehen hat (hier ein DOM-Knoten in der Variable `p2`). Das zugrunde liegende HTML sieht (Ausschnitt) wie folgt aus:

```

<div id="box1">
  <p id="p1">Klick für proxy-Test</p>
  <p id="p2">Dieser Absatz färbt sich rot.</p>
</div>

```

4.13.7 Objektverarbeitung und Erweiterung von jQuery

Im Rahmen der jQuery-Utilities findet sich auch die Low-level-Methode `$.extend()`. Die Methode dient, je nach Art und Zahl der übergebenen Argumente, verschiedenen Zwecken. Zum einen kann `$.extend()` zwei oder mehr ihr übergebene Objekte in ein Objekt zusammenführen. Dieses »Zielobjekt« (`target`) wird hierbei um die Eigenschaften eines weiteren »Quellobjekts« (oder weiterer Quellobjekte) erweitert, das als zweites Argument übergeben wird (bzw. die in weiteren Argumenten übergeben werden).

Vorsicht

Hierbei werden die Property-Werte des Targets bei gleichem Property-Namen durch die Werte der Quellobjekte ersetzt!

jQuery.extend(target, obj)

Die Methode wird, wenn sie Objekte verschmelzen soll, mit zwei oder mehr Argumenten angewendet. Zwei übergebene Objekte werden verschmolzen, wobei das `target`-Objekt überschrieben wird. Ist bei (mindestens) drei Argumenten das erste ein Boolescher Wert, findet die Verschmelzung der als Argument zwei und drei übergebenen Objekte rekursiv statt (als Deep-Copy), sofern der Wert `true` ist.

Utility	Erläuterung
<code>jQuery.extend(target, obj)</code>	Vereinigt den Inhalt eines Target-Objekts <code>target</code> mit dem Inhalt eines zweiten Objekts <code>obj</code> (oder mehrerer übergebener Objekte <code>obj1, ..., objN</code>).

Tabelle 4.81 Utility zum Merging von Objekten

Utility	Erläuterung
<code>jQuery.extend(deep, target, obj)</code>	Vereinigt den Inhalt eines Target-Objekts <code>target</code> mit dem Inhalt eines zweiten Objekts <code>obj</code> (oder mehrerer übergebener Objekte <code>obj1, ..., objN</code>). Führt dies als Deep-Copy aus, wenn für <code>deep</code> der Wert <code>true</code> übergeben wird.

Tabelle 4.81 Utility zum Merging von Objekten (Forts.)

Sind beide Objekte nicht hierarchisch, funktioniert die Verschmelzung problemlos. Die Werte gleichnamiger Properties werden überschrieben, Properties aus den Source-Objekten, die im Target nicht existieren, werden dort eingefügt:

```
$(document).ready(function() {
    // Objekt 1 (target):
    var obj1 = {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'};
    // Objekt 2 (source):
    var obj2 = {b2: 'Lilien', b4: 'Petunien'};

    alert(obj1.toSource());
    // -> {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'}

    jQuery.extend(obj1, obj2);

    alert(obj1.toSource());
    // -> ({b1: 'Rosen', b2: 'Lilien', b3: 'Nelken', b4: 'Petunien'})
});
```

Die Lage verändert sich, sobald die Objekteigenschaften selbst Objekte sind. In diesem Fall verhält sich die Methode zunächst gleich, ersetzt also gleichnamige Properties in Targets durch jene aus Source:

```
$(document).ready(function() {
    // Objekt 1 (target)
    var obj1 = {strauss: {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'}};
    // Objekt 2 (source)
    var obj2 = {strauss: {b2: 'Lilien', b4: 'Petunien'}};
    alert(obj1.toSource());
    // -> {strauss: {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'}}

    jQuery.extend(obj1, obj2);

    alert(obj1.toSource());
    // -> {strauss: {b2: 'Lilien', b4: 'Petunien'}}
});
```

Es findet also *kein* rekursives Verschmelzen von Unterobjekten statt – stattdessen wird ein `target`-Unterobjekt durch ein gleichnamiges `source`-Unterobjekt einfach überschrieben. Wollen Sie ein rekursives Merging auf allen Unterebenen erreichen, müssen Sie der Methode `jQuery.extend()` als erstes Argument den Wert `true` (für Rekursivität, also Deep-Copy) übergeben:

```
$(document).ready(function() {
    // Objekt 1 (target)
    var obj1 = {strauss: {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'}};
    // Objekt 2 (source)
    var obj2 = {strauss: {b2: 'Lilien', b4: 'Petunien'}};
    alert(obj1.toSource());
    // -> {strauss: {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'}}

    jQuery.extend(true, obj1, obj2);

    alert(obj1.toSource());
    // -> {strauss: {b1: "Rosen", b2: "Lilien", b3: "Nelken",
        b4: "Petunien"}}
});
```

Die Methode `jQuery.extend()` überschreibt stets das Target-Objekt, arbeitet also *destruktiv*. Wollen Sie dies nicht bzw. sollen beide ursprünglichen Objekte `obj1` und `obj2` nach der Operation unverändert weiter bestehen, verwenden Sie ein leeres Objekt als Target.

Da die Methode das erweiterte Objekt als Rückgabewert ausgibt, kann es einfach in eine neue Variable `obj3` abgelegt werden:

```
$(document).ready(function() {
    // Objekt 1 (jetzt ebenfalls source!)
    var obj1 = {strauss: {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'}};
    // Objekt 2 (zweite source)
    var obj2 = {strauss: {b2: 'Lilien', b4: 'Petunien'}};
    alert(obj1.toSource());
    // -> {strauss: {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'}};

    var obj3 = jQuery.extend(true, {}, obj1, obj2);

    // bleibt unverändert:
    alert(obj1.toSource());
    // -> {strauss: {b1: 'Rosen', b2: 'Tulpen', b3: 'Nelken'}}
    alert(obj3.toSource());
    // -> {strauss: {b1: "Rosen", b2: "Lilien", b3: "Nelken",
```

```

        b4: "Petunien" }}
    );

```

Okay, es gilt: Nach wie vor wird das Target überschrieben. Da es sich jetzt jedoch um ein leeres »Wegwerfobjekt« handelt, macht dies nichts. Die restlichen Source-Objekte bleiben (wie bislang auch immer) unverändert.

jQuery.extend(obj)

Mit anderer Signatur erweitert `jQuery.extend()` direkt das jQuery-Objekt: Wird nur ein Argument (ein Objekt) übergeben, nimmt jQuery an, dass das `target`-Argument entfallen ist und wendet die Methode auf das jQuery-Objekt selbst an, das nun als Target dient.

Utility	Erläuterung
<code>jQuery.extend(obj)</code>	Erweitert, wenn nur ein Argument <code>obj</code> übergeben wird, das jQuery-Objekt um die Eigenschaften des übergebenen Objekts.

Tabelle 4.82 Utility zur Erweiterung von jQuery (Teil 1)

Die Methode kann mit dieser Signatur eingesetzt werden, um Funktionen in den jQuery-Namensraum einzufügen, und so das jQuery-Objekt selbst erweitern.

Im folgenden Beispiel wird der Methode `jQuery.extend()` ein Objekt übergeben, in dem eine anonyme Funktion an eine Eigenschaft `isEven` gebunden wird. Dieses Objekt wird durch `extend()` in das jQuery-Objekt integriert, dem `isEven` anschließend als Methode zur Verfügung steht:

```

jQuery.extend({
    isEven: function(m) {
        var b = false;
        if ((m % 2) == 0) {b = true; }
        return b; }
});

```

Diese neue jQuery-Methode – im Grunde handelt es sich trotz aller Kürze um eine »Erweiterung« – lässt sich nun einfach über `jQuery.isEven(int)` oder `$.isEven(int)` aufrufen.

jQuery.fn.extend(obj)

Vergleichbar arbeitet auch `jQuery.fn.extend()`. Diese Methode eignet sich besonders, um Plugins zu programmieren.

Utility	Erläuterung
<code>jQuery.fn.extend(obj)</code>	Erweitert das jQuery-Objekt über die Prototype-Kette um die Eigenschaften des übergebenen Objekts <code>obj</code> .

Tabelle 4.83 Utilities zur Erweiterung von jQuery (Teil 2)

Plugins zu erstellen bedeutet, eigene Methoden zu definieren, die später einfach an ein jQuery-Objekt angehängt werden können. Die Anwendung geschieht nach folgendem Muster (`.galileo()` und `.design()` sind hierbei Plugin-Methoden):

```
$("#container").galileo(true).design("#009900");
```

Der Code des Plugins verwendet `jQuery.fn.extend()`, um die neuen Methoden an das jQuery-Objekt zu binden. Dies geschieht über die Prototype-Chain (hierbei ist `jQuery.fn` nur ein »handlicheres« Alias für `jQuery.prototype`):

```
jQuery.fn.extend({
  galileo: function(bool) {
    if (bool) {
      this.append("Galileo würde jQuery mögen");
    } else {
      this.append("Leider kein Kommentar");
    }
    return this;
  },
  design: function(farbe) {
    if (farbe) {
      $(this).css({ color: farbe });
    }
    return this;
  }
});
```

Der Plugin-Code wird zur Wiederverwendbarkeit in einer JS-Datei gespeichert und (hier als `plugin.js` in einem Ordner *plug01*) in das HTML-Dokument eingebunden, in dem die Plugin-Funktionalität benötigt wird. Achtung – dies muss stets *nach* dem Einbinden des Frameworks selbst geschehen:

```
...
<script src="../../js/jquery-1.4.js"></script>
<script src="../../js/plug01/plugin.js"></script>
<script type="text/javascript">
  $(document).ready( function(){
    // Aufruf von Plugin-Methoden:
    $("#container").galileo(true).design("#009900");
  });
```



```

</script>
</head>
<body>
<div id="container"></div>
</body>
...

```

Mehr als Bezeichner und eventuelle Argumente des Methodenaufrufs braucht der Anwender eines Plugins nicht zu wissen. Hier also: Hinter dem Aufruf `.gallileo(bln)` verbirgt sich die Anweisung, dass eine bestimmte Zeichenkette in ein Element geschrieben werden soll. Die Methode `.design(farbe)` hingegen setzt die Schriftfarbe des selektierten Elements auf die übergebene Farbe. Um die eigentlichen internen Abläufe muss sich der Anwender nicht kümmern.

4.13.8 Test-Utilities

Die Möglichkeiten, die in JavaScript implementiert sind, um Objekte und ihre genauen Eigenschaften einfach zu erkennen, sind in vielen Fällen unzureichend. Der hierfür vorgesehene Operator `typeof` gibt sehr gerne die pauschale Information `Object` zurück, wo man es gerne etwas genauer hätte – ist der betrachtete Gegenstand ein Array, eine Funktion oder ein »gewöhnliches« Objekt?

jQuery kann hier weiterhelfen, indem es spezialisierte Test-Utilities zur Verfügung stellt. Diese bekommen das zu untersuchende Objekt übergeben und geben daraufhin einen Booleschen Wert zurück.

Utility	Erläuterung
<code>jQuery.isArray()</code>	Prüft, ob es sich beim Eingabeargument um ein Array handelt.
<code>jQuery.isEmptyObject()</code>	Prüft, ob das übergebene Objekt leer ist.
<code>jQuery.isFunction()</code>	Prüft, ob das übergebene Objekt ein Funktionsobjekt ist.
<code>jQuery.isPlainObject()</code>	Prüft, ob das übergebene Objekt ein echtes Objekt ist (als Literal <code>{}</code> oder über Konstruktor <code>new Object</code> erzeugt).
<code>jQuery.isXMLDoc()</code>	Prüft, ob ein DOM-Knoten Teil eines XML-Dokuments ist (oder ob es sich um den Document Node eines XML-Dokuments handelt).

Tabelle 4.84 jQuery-Utilities für Tests von Daten und Objekten

Methode jQuery.isArray()

Die Methode untersucht ein übergebenes Objekt und gibt `true` zurück, wenn es sich um ein JavaScript-Array handelt. Ist es »nur« ein arrayähnliches Objekt (beispielsweise eine jQuery-Collection), gibt die Methode `false` zurück:

```
$(document).ready(function() {
    // jQuery-Collection (arrayähnlich):
    var col = $("#box p");
    alert(col.length);           // -> 4

    // echtes Array:
    var arr = ['Rosen', 'Tulpen', 'Nelken'];
    alert(arr.length);          // -> 3

    alert(jQuery.isArray(col)); // -> false
    alert(jQuery.isArray(arr)); // -> true
});
```

Methode Query.isEmptyObject()

Die Testmethode `Query.isEmptyObject()` erkennt, ob ein ihr übergebenes Objekt Eigenschaften besitzt oder nicht. Sie reagiert allerdings »falsch positiv« auch auf andere leere Eingangsgrößen, wie den Booleschen Wert `false`, leere Strings und die Zahl 0. Auch der Wert `null` oder der leere Rückgabewert eines ohne Erfolg angewendeten DOM-Knoten-Accessors wie `getElementById()` ergibt `true`. Leere Collections hingegen werden als `false` erkannt (ein jQuery-Objekt besitzt ja stets Eigenschaften):

```
$(document).ready(function() {
    var num = 0;           // Zahl 0
    var nll = null;        // Wert null
    var str = "";          // leerer String
    var col = $("#unbekannt"); // leere Collection

    // nicht-leeres Objekt:
    var obj1 = {'b1': 'Rosen', 'b2': 'Tulpen', 'b3': 'Nelken'};
    // leeres Objekt:
    var obj2 = {};

    // nichtexistentes DOM-Element:
    var elem = document.getElementById('unbekannt');

    alert(jQuery.isEmptyObject(num)); // -> true
    alert(jQuery.isEmptyObject(nll)); // -> true
    alert(jQuery.isEmptyObject(str)); // -> true
```

```

alert(jQuery.isEmptyObject(col)); //-> false
alert(jQuery.isEmptyObject(obj1)); //-> false
alert(jQuery.isEmptyObject(obj2)); //-> true
alert(jQuery.isEmptyObject(elem)); //-> true
});

```

Methode `jQuery.isPlainObject()`

Mit der Utility-Methode `jQuery.isPlainObject()` können »echte« Objekte, d.h. solche, die regulär entweder als *Literal* vorliegen oder mittels Konstruktor erzeugt wurden, von »objektähnlichen« Gegenständen, wie jQuery-Collections oder DOM-Knoten, unterschieden werden. Auch andere an die Testroutine übergebene Typen, wie Zahlen, Strings oder Boolesche Werte, geben `false` zurück:

```

$(document).ready(function() {
    // nicht-leerer String
    var str = "Ich bin kein Objekt!";

    // nicht-leere Collection:
    var col = $("#box p");

    // echtes Objekt:
    var obj = {'b1': 'Rosen', 'b2': 'Tulpen', 'b3': 'Nelken'};

    // existierendes DOM-Element:
    var elem = document.getElementById('box');

    alert(jQuery.isPlainObject(str)); //-> false
    alert(jQuery.isPlainObject(col)); //-> false
    alert(jQuery.isPlainObject(obj)); //-> true
    alert(jQuery.isPlainObject(elem)); //-> false
});

```

Methode `jQuery.isFunction()`

Die Utility-Methode `jQuery.isFunction()` überprüft, ob es sich bei dem ihr übergebenen Argument um ein Funktionsobjekt handelt. Sie gibt entsprechend `true` zurück, wenn dies der Fall ist, ansonsten `false`. Praktisch kann dieses Utility sein, wenn es festzustellen gilt, ob es sich bei einem Argument wirklich um eine Call-back-Funktion handelt, bevor dieses übergeben wird:

```

var meineFunktion = function(){
    alert("Ich bin eine Funktion!");
}

function zweiteFunktion() {
    alert("Ich bin auch eine Funktion!");
}

```

```
}
var keineFunktion = ['nur','ein','Array'];
$(document).ready(function() {
  // direkt übergebenes Funktionsobjekt: true
  alert(jQuery.isFunction( function(){
    alert('Nun?')
  })); //-> true

  // Variable mit Funktionsreferenz: true
  alert(jQuery.isFunction(meineFunktion)); //-> true

  // übergebener Funktionsname: true
  alert(jQuery.isFunction(zweiteFunktion)); //-> true

  // kein Funktionsobjekt, sondern Array: false
  alert(jQuery.isFunction(keineFunktion)); //-> true
});
```


»As someone with a background in computer science, I find it quite surprising that so many designers and nonprogrammers find jQuery to be compelling.« – John Resig

5 jQuery – der Praxiseinsatz

Willkommen im Praxisteil! Im Folgenden werden Sie den im vorangegangenen Abschnitt vorgestellten Sprachschatz, der in Kapitel 4, »jQuery – die Übersicht«, detailliert aufgearbeitet wurde, in komplexen Zusammenhängen einsetzen. Sie werden sehen, was Sie mit jQuery in der Praxis alles anstellen können, wie jQuery das Entwicklerleben erleichtern bzw. das Designerleben vielfältiger gestalten kann.

Vielleicht haben Sie sich entschlossen, den Referenzteil zu überspringen und direkt an dieser Stelle des Buchs anzusetzen. Gut so, Sie werden trotzdem zurechtkommen. Möglicherweise werden Sie an bestimmten Stellen zurückblättern, um Hintergrundinformationen zu den angewendeten Methoden zu erhalten. Auf entsprechende Erklärungen wurde hier weitestgehend verzichtet, um den Informationsfluss nicht zu stören. Eine gewisse Redundanz mancher Passagen ließ sich dennoch nicht vermeiden – wir bitten dies schon jetzt zu entschuldigen.

Über die vorgestellten Beispiele

Viele der Beispiele sind auf das Notwendige reduziert. Einerseits sollen diese Bausteine die Herangehensweise verdeutlichen, wie jQuery-Anwendungen aufgebaut werden können, andererseits sollen sie dazu anregen, eigene Skripte zu generieren, sie weiter zu verfeinern und mit eigenen Ideen zu erweitern. Für viele der angerissenen Anwendungen gibt es bis ins Detail ausgearbeitete Plugins, weitere finden sich im Web mit einem anderen Lösungsweg.

Manchmal schlagen Entwickler einen komplizierten Weg ein, versehen eine Anwendung mit einem Funktionsumfang, den Sie im Moment gar nicht benötigen, obwohl der Kern sehr klar überlegt ist. Auch dann ist es oft einfacher, sich an ein eigenes Konzept zu wagen, als ein vorhandenes Script zu vereinfachen.

In diesem Teil des Buches werden Sie einen Einblick gewinnen, wie die Logik von jQuery-Skripten aufgebaut ist. Auf viele dieser Funktionalitäten sind Sie bereits

als Anwender im Netz gestoßen und haben sich vielleicht gefragt, wie das eine oder andere aufgebaut ist.

Ziel ist es, in diesem Teil einmal die Motorhaube aufzuklappen, den Motor auszubauen und zu betrachten, um zu erkennen, wie solche Scripte grundsätzlich aufgebaut sind, mit welcher Denkweise Sie ein Script angehen können. Es obliegt Ihrer Kreativität, diese Einzelteile zu einem Eigenen auszubauen, zu verfeinern oder als Anregung zu verwenden, um einen vielleicht ganz anderen Programmieransatz zu wählen.

5.1 Schönere Navigationen

Als Erstes werden Sie sich dem Aufbau von Navigationen widmen. Noch vor wenigen Jahren waren FlyOut Navigationen, Akkordeon Navigationen oder allgemein gesprochen dynamische Navigationen ein Abenteuer für die Betreiber von Webseiten. Gut gemeinte dynamische Scripte, ambitioniert eingesetzt, wurden schnell wieder zurückgezogen, da sie nur unzuverlässig und nicht browserübergreifend funktioniert haben, geschweige denn barrierefrei waren.

Allerdings – gerade in den letzten Jahren hat die Entwicklung der Browserhersteller enorme Fortschritte gemacht. Selbst der Hersteller aus Redmond verspricht, sich zukünftig an Standards zu halten, und seit der Version 8 des Internet Explorers hat sich einiges getan – die nächste Version 9 lässt sich sehr vielversprechend an. So können Webentwickler guten Gewissens Funktionen implementieren, die vor einigen Jahren noch undenkbar waren.

Selbstredend war die Mozilla Foundation mit Firefox immer daran interessiert, die Standards, die das W3C, respektive die HTML Working Group, vorschlägt, genau umzusetzen. Dazu gehören auch Projekte wie Webkit oder der Hersteller des Opera Browsers. Heute können wir uns darauf verlassen, dass wenigstens die wichtigsten Spezifikationen für CSS 2 in allen modernen Browsern eingehalten werden, das hilft gerade bei dynamischen Navigationen enorm viel.

Einschränkend muss betont werden, dass gerade Unternehmen den Umstieg auf neue Systeme scheuen, nicht zuletzt aus Kostengründen. Es bleibt der bittere Beigeschmack, dass Webentwickler sich zähneknirschend noch um ältere Browser kümmern müssen. Es geistern noch immer Versionen bestimmter Browser durch die Weblandschaft, und Entwickler befinden sich daher in der misslichen Lage, zu entscheiden, ob sie diese noch zu berücksichtigen gedenken oder nicht. Selbst wenn eine plattform- und browserübergreifende Bibliothek wie jQuery eingesetzt wird.

Sie als Designer oder Entwickler möchten mit der Entwicklung von Websites nicht Kunden mittels raffiniert ausgetüftelter, aber nur eingeschränkt nutzbarer Technologie verschrecken, sondern Sie wollen Ihren Besuchern eine intuitive Weboberfläche kreieren, mit der Sie durch die Technologie einen Mehrwert für den Nutzer schaffen. Dabei helfen Ihnen einige Grundkonzepte, die Sie bei der Entwicklung moderner Sites einhalten sollten. Sie werden diese Konzepte, namentlich »unobtrusive«, also »unaufdringliches« JavaScript und »progressive Erweiterung«, anhand von anschaulichen Beispielen kennenlernen.

5.1.1 Die FlyOut-Navigation

Die Anforderung ist, eine Navigation über zwei Ebenen zu realisieren, von der zunächst nur die erste Ebene angezeigt wird. Alle Menüpunkte der ersten Ebene werden horizontal angeordnet, die Menüpunkte der zweiten Ebene sind zunächst verborgen. Mittels Mouseover öffnet sich die zweite Ebene und schiebt sich dabei über einen möglicherweise darunterliegenden Inhalt.



Abbildung 5.1 FlyOut-Navigation

Diskussion

Natürlich soll diese Navigation auch in anderen Ausgabemedien funktionieren – möglicherweise mit einem komplett anderen Stylesheet, soll vielleicht sogar von einem Screenreader verstanden werden. Um dieses Ziel zu erreichen, muss die HTML-Auszeichnung semantisch korrekt umgesetzt werden. Das ist vielleicht nicht das ganze Geheimnis, aber »die halbe Miete«.

Semantisch in diesem Zusammenhang meint, Sie verwenden HTML-Elemente, die der Bedeutung des Inhalts entsprechen. Ein Absatz ist ein Absatz, ein `<p>`-Element. Eine Überschrift ist eine Überschrift, beispielsweise ein `H1`-Element. Es hat sich in der Praxis etabliert, Navigationen semantisch als unsortierte Liste zu verstehen, also `<ul>` und `<li>` zu verwenden. Allgemein formuliert: Navigationen sind nichts anderes als Listen aus Menüpunkten.

Schalten Sie versuchsweise das Stylesheet dieses Beispiels aus, um zu sehen, wie die Seite ohne Layout funktioniert.

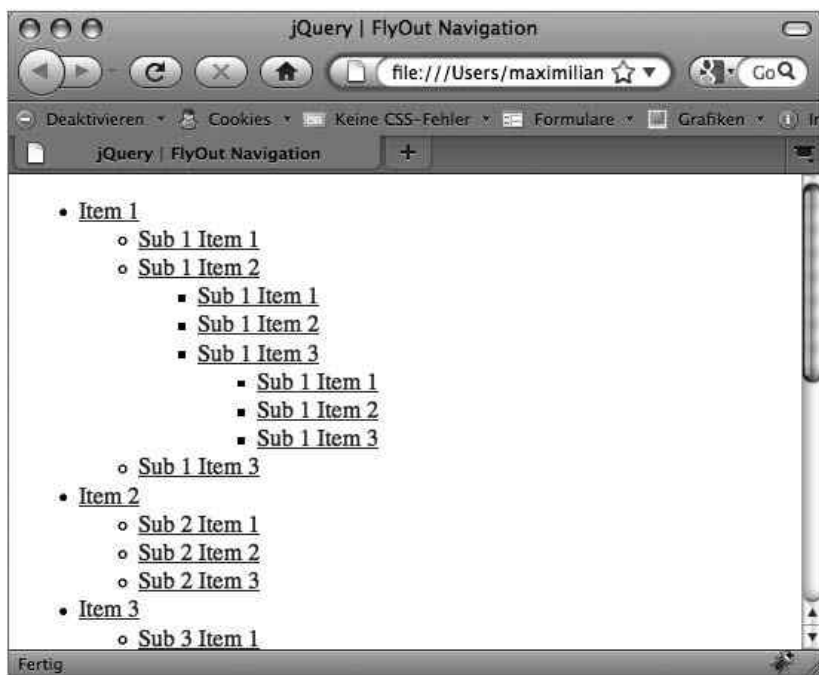


Abbildung 5.2 FlyOut-Navigation ohne CSS

Sie sehen eine zweistufige Navigation. Beide Ebenen sind sichtbar, die zweite Ebene ist eingerückt. Auf jeden Fall können Sie die Seite auch ohne Styles benutzen. Schön sieht sie zwar nicht aus, aber aufgeräumt und gut strukturiert.

Schalten Sie die Stylesheets wieder ein. Sie sehen, dass das Menü auch allein mit CSS, also ohne JavaScript, respektive jQuery, funktioniert. Grundlage ist die Möglichkeit, an jedes HTML-Element eine Pseudo-Klasse `:hover` binden zu können.

Früher konnte man nur dem `<a>`-Element erfolgreich ein `a:hover` zuweisen. Dies war vielfach bewährt, um für Links beim Mouseover eine Änderung der Linkfarbe zu bewirken. In der Spezifikation für CSS war `:hover` zwar für alle HTML-Elemente zugelassen, allerdings beherrschte dies leider der seinerzeit am meisten verbreitete Browser (der Internet Explorer 6) nicht. Das `:hover` blieb in der Praxis auf `<a>`-Elemente beschränkt. In allen modernen Browsern können Sie diese

Pseudo-Klasse auch einem LI-Element zuweisen und so ein FlyOut-Menü auch ohne JavaScript realisieren. Was soll das? Warum man dann noch jQuery benötigt, werden Sie fragen.

Erstens, um auch dem Internet Explorer 6, in dem die reine CSS-Lösung nicht funktioniert, diese Dynamik beizubringen. Zweitens, um das FlyOut-Menü in der bestmöglichen Eleganz strahlen zu lassen: Mittels CSS klappt das Untermenü knallhart auf und auch wieder zu. Mit jQuery können Sie es sanft einblenden oder aus- und einfahren. Der Fantasie sind hier keine Grenzen gesetzt. Auch Farbüberblendungen sind möglich, und Plugins wie jQuery UI ermöglichen sogar noch weitere Effekte. In diesem Beispiel beschränken wir uns jedoch aufs Wesentliche: Aus- und Einfahren und Ein- und Ausblenden.

Der Internet Explorer 6 und das FlyOut-Menü

Die Verbreitung des Internet Explorers 6 beträgt zur Zeit (Mitte 2010) etwa 7 bis 16 %. So manche Website muss diesen Browser noch unterstützen, so sehr es den Webentwicklern auch aufstoßen mag. Das im Anschluss betrachtete FlyOut-Menü wird im Internet Explorer 6 zwar nur mit JavaScript funktionieren. Die verschiedenen Browserwatch-Dienste erheben jedoch, dass nur 0,3 bis 5 % aller Internetnutzer JavaScript abgeschaltet haben. Unter diesen wohl auch einige Internet Explorer 6-Nutzer.

Jeder Webentwickler muss an dieser Stelle entscheiden, ob er dies verantworten kann oder nicht, dass jene Internet Explorer 6-Nutzer die Navigation nicht benutzen können. Es hängt von individuellen Faktoren eines bestimmten Projekts ab, ob im Einzelfall von solchen Scripts Abstand genommen werden muss.

Das Beispiel

Beginnen Sie als Erstes damit, das HTML-Gerüst aufzubauen. Hier die funktionsfähige gesamte Notation:

```
<!DOCTYPE html>
<html>
<head>
  <style type="text/css">
    @import url("flyout.css");
  </style>
  <script src="jquery-1.4.2.js"></script>
  <script src="flyout.js"></script>
</head>
<body>
  <ul>
    <li><a href="#">Item 1</a>
      <ul>
        <li><a href="#">Sub 1 Item 1</a></li>
      </ul>
    </li>
  </ul>
</body>
</html>
```

```

        <li><a href="#">Sub 1 Item 2</a></li>
        <li><a href="#">Sub 1 Item 3</a></li>
    </ul>
</li>
<li><a href="#">Item 2</a>
    <ul>
        <li><a href="#">Sub 1 Item 1</a></li>
        <li><a href="#">Sub 1 Item 2</a></li>
        <li><a href="#">Sub 1 Item 3</a></li>
    </ul>
</li>
<li><a href="#">Item 3</a></li>
<li><a href="#">Item 4</a></li>
</ul>
</body>
</html>

```

Listing 5.1 HTML-Gerüst – FlyOut Navigation

Sie können jeder Zeit dieses Beispiel um eigene Menüs und Unterebenen erweitern, für unser Szenario reicht dieser Testaufbau.

Nun müssen wir einen kleinen Ausflug in die Welt der Cascading Style Sheets machen, um zu verstehen, wie das FlyOut-Menü funktioniert. Sie haben im `<head>` des HTML-Dokuments bereits eine Stylesheet-Datei (*flyout.css*) eingebunden. Betrachten wir nun die für dieses Beispiel wichtigsten CSS-Anweisungen. Anweisungen, die das Layout betreffen, sind hier weitgehend weggelassen (Sie finden Sie aber in den mit dem Buch mitgelieferten Beispieldateien):

```

* {
    padding:0;
    margin:0;
}

```

Zuerst setzen Sie global alle Außenabstände und Innenabstände auf 0 zurück. Dieser Ansatz mit dem Universalselektor reicht für dieses Beispiel:

```

ul {
    list-style-type:none;
    margin:10px;
}

```

Sie schalten mit `list-style-type:none`; für die unsortierte Liste, und alle darin verschachtelten Listen die störenden Bulletpunkte aus. Es handelt sich, wie beim 10px-Margin, um eine rein kosmetische Operation.

```
ul ul {
  display:none;
  position:absolute;
  width:150px;
  margin:0;
  left:0px;
  top:26px;
}
```

Als Nächstes entfernen Sie mit `display:none`; das Untermenü der zweiten Ebene. Sie machen es nicht nur unsichtbar, sondern entfernen es komplett aus dem Anzeigebereich. Es nimmt in der Darstellung des Dokuments keinen Platz mehr ein. Damit sich diese Unterebene über darunterliegende Inhalte legt, »heben« Sie es mit `position:absolute` aus dem Dokumentfluss heraus. Mit `top:26px` ordnen Sie es direkt unterhalb der ersten Ebene an. Voraussetzung ist, dass die erste Ebene auch exakt diese 26px hoch ist. (Gegebenenfalls müssen Sie in Ihrem selbst gebauten Beispiel diesen Wert anpassen.)

```
li {
  float:left;
  position:relative;
  padding:5px 15px 5px 5px;
  background-color:#EEEEEE;
}
```

Die einzelnen Listenelemente, vor allem die der ersten Ebene, positionieren Sie mit `position:relative` als Ankerpunkt für die verschachtelten `<ul>`-Elemente. Die Hintergrundfarbe setzen Sie, damit Sie sehen, wie sich die Navigation vom Rest des Dokuments abhebt. Fühlen Sie sich frei, hier mit Rahmen oder anderen Hintergrundfarben zu experimentieren. Mit `float:left` sorgen Sie dafür, dass sich die Menüpunkte der ersten Ebene horizontal anordnen.

Bezugspunkt setzen für `position:absolute`

Wenn Sie probeweise die Zeile mit `position:relative` weglassen oder auskommentieren, werden Sie sehen, dass sich das mit `position:absolute` versehene Untermenü komplett unabhängig von seinem Elternelement ganz an den oberen linken Rand des Browserfensters hängt. Erst durch den Status POSITIONIERT dient das List-Item als Bezugspunkt für »seine« Unterliste:

```
li li {
  float:none;
  background-color:#CCCCCC;
}
```

Für die Listenelemente der zweiten Ebene setzen Sie `float:none`. Das bedeutet nicht etwa, dass die Float-Umgebung aufgehoben wird. Sie sind damit jedoch sicher, dass für die untergeordneten Listenelemente kein Float gesetzt ist. Diese sollen sich ja vertikal anordnen.

Die eigentliche Dynamik steckt hinter diesem Selektor:

```
li:hover ul {
    display:block;
}
```

Hier soll erreicht werden, dass einem in einem LI-Listenpunkt verschachtelten `<ul>`-Block (die zweite Ebene des Menüs) bei der Mausektion `:hover` die Eigenschaft `display:block` zugewiesen wird. Zur Erinnerung – zuvor noch hatten Sie diese zweite Ebene auf `display:none` gesetzt.

Nun setzen Sie die innere Liste auf die Eigenschaft `BLOCKELEMENT`. Damit wird sie sichtbar und nimmt Raum im Dokument ein. Dies aber nicht im Rahmen des Dokumentflusses, was dazu führt, dass Inhalte, die »im Fluss« sind, gegebenenfalls verdeckt werden.

Nun wird das Ganze mit jQuery gewürzt. Es folgt der komplette jQuery-Code:

```
function menu(obj){
    if (!obj.length) return;
    $(obj).find("ul").css({display: "none"});
    $(obj).hover(function(){
        $(this).find('ul').slideDown(300);
    },function(){
        $(this).find('ul').slideUp(300);
    });
}

$(document).ready(function(){
    menu($(".ul li"));
});
```

Listing 5.2 JavaScript-Code – FlyOut-Navigation

Die paar Zeilen reichen? Ja, mehr brauchen Sie auch nicht, um der Navigation die nötige Eleganz zu verleihen und dem Sorgenkind aller derzeit gebräuchlichen Browser auch auf die Beine zu helfen.

Im `$(document).ready(fn)`-Abschnitt rufen Sie nun die Funktion `menu()` mit dem Parameter des jQuery-Objekts auf, das Sie als Navigation manipulieren wollen.

Die Anweisung `if (!obj.length) return` stellt sicher, dass die Funktion nur ausgeführt wird, wenn das jQuery-Objekt Elemente gespeichert hat und nicht unnötige Rechenleistung verschlingt, wenn kein Element gefunden wurde.

Die Anweisung `$(obj).find("ul").css({display:"none"});` setzt die zweite Ebene zurück. Augenscheinlich unsinnig, da Sie das bereits im Stylesheet getan haben. Kommentieren Sie die Zeile probeweise aus, und Sie werden feststellen, dass in vielen Browsern der erste Aufruf des Submenüs nicht wie gewünscht erfolgt, sondern dass das Submenü beim ersten Mal hart einblendet. Also: Sie benötigen diese Anweisung, um das Script nach dem Laden der Seite zurückzusetzen.

Als Nächstes folgt der eigentliche Effekt mit dem Event Handler: `$(obj).hover()` erwartet zwei Funktionen als Argumente, eine für *Mausdrüber* und eine für *Mausraus*. In der ersten Funktion gehen Sie von dem Element aus, auf dem sich die Maus gerade befindet. Der Ausdruck `$(this)` bezieht sich auf das gerade überflogene LI-Element, mit `.find("ul")` wird das unsichtbare, verschachtelte `<ul>`-Element gefunden und mit dem Effekt `.slideDown()` nach unten gefahren. Die zweite Funktion `.slideUp()` übernimmt demgemäß die Aufgabe, den Submenüblock wieder nach oben zu ziehen.

Das war's ... – noch nicht ganz! Wenn Sie die Navigation ausprobieren und ein wenig wild mehrfach über die erste Ebene hin- und herfahren, werden Sie bemerken, dass das Script sich genau »merkt«, dass Sie bereits zwanzigmal darübergewischt haben. Es fährt, wie ihm geheißen, zwanzig Mal auf und ab. Das sieht lustig aus, ist aber nicht zweckmäßig, da Sie vielleicht weiter navigieren wollen, während die Navigation fröhlich zu Ende zappelt. Das verdanken Sie der Eigenheit, dass jQuery für Effekte eine Warteschlange, die Queue, abarbeitet, »first in, first out«, bis alle gespeicherten Aufrufe durchlaufen sind.

Als Abhilfe modifizieren Sie das Script wie folgt

```
...
$(this).find('ul').stop(true,true).slideDown(300);
...
$(this).find('ul').stop(true,true).slideUp(300);
...
```

und fügen die fett hervorgehobene Methode `stop()` für beide Anweisung hinzu.

Die beiden Argumente bedeuten `clearQueue=true` und `jumpToEnd=true`. Sie geben also die Anweisung, bei jedem erneuten Hover zuerst die laufende Animation anzuhalten, die Queue zu löschen und zum Ende der Animation zu springen. Damit unterbrechen Sie den unerwünschten Nebeneffekt und stellen sicher, dass die Animation nur ausgeführt wird, wenn der Benutzer sie benötigt.

Sie können sich noch ein wenig mit dem Beispiel aufhalten und ausprobieren, wie das Menü mit einem anderen Effekt funktioniert:

```
...
$(this).find('ul').stop(true,true).fadeIn(300);
...
$(this).find('ul').stop(true,true).fadeOut(300);
...
```

Das wären 300 Millisekunden für den FadeIn. Um das Menü hingegen einfach nur hart einzublenden, brauchen Sie das Script lediglich für den Internet Explorer 6. Sie können das Script so ändern:

```
...
$(this).find('ul').fadeIn(1);
...
$(this).find('ul').hide();
...
```

Wenn nur der Internet Explorer 6 bedient werden soll, setzen Sie die Script-Tags in *Conditional Comments*. Natürlich nur, falls Sie jQuery nicht noch für andere Zwecke verwenden wollen:

```
...
<!--[if IE 6]>
  <script src="jquery-1.4.2.js"></script>
  <script src="flyout.js"></script>
<![endif]-->
...
```

Zur Erläuterung – Conditional Comments (CC) sind eine Erweiterung der HTML-Kommentarsyntax, die ausschließlich der Internet Explorer ab der Version 5 unterstützt. Es sind, streng genommen, gewöhnliche HTML-Kommentare, nur dass sie Steuerzeichen enthalten, die dem Internet Explorer sagen, was er zu tun hat. HTML-Kommentare zeichnen sich eigentlich dadurch aus, dass sie eben nicht ausgewertet werden. Dies ist bei Conditional Comments anders, jedoch nur im Internet Explorer: Andere Browser behandeln sie wie normale Kommentare.

Die CC besitzen ein übersichtliches Vokabular, wenn man denn überhaupt von Vokabular sprechen kann. Hier ein Beispiel:

```
<!--[if IE6]>
irgendein Text oder HTML-Elemente
<![endif]-->
```

Was sind Conditional Comments?

So schwierig sich der Internet Explorer manchmal gebärdet, so leicht lässt sich über dieses Werkzeug ein Sonderweg für diesen widerspenstigen Browser eröffnen. Die HTML-Auszeichnung bleibt dabei valide. Es ist nach der Spezifikation ein sauberer HTML-Kommentar und somit eine praktische Browserweiche. Es hat sich durchgesetzt, über diesen Weg Scripte oder Stylesheets einzubinden, die dann nur für den besagten Browser zuständig sind. Das funktioniert stabil und zuverlässig. Ein Beispiel für einen sinnvollen CC wäre:

```
<!--[if lt IE7]>
<link rel="stylesheet" type="text/css" href="ie6.css" />
<[/endif]>
```

Das Stylesheet mit dem Dateinamen *ie6.css* wird nur geladen, wenn der anfordernde Browser (bei dem es sich um einen Internet Explorer handeln muss, da er sonst die Anweisung nicht »sehen« würde) eine Versionsnummer niedriger (*lt* steht für »kleiner als«) als 7 besitzt. Weitere Operatoren sind *!* für »nicht«, *gt* für »größer als«, *lte* für »kleiner gleich« und *gte* für größer gleich«.

Navigation um zusätzliche Ebenen erweitern

Damit haben Sie einige Möglichkeiten durchgespielt, die sich Ihnen mit diesem Menü bieten. Was passiert aber, wenn Sie das Menü noch um zusätzliche Ebenen erweitern wollen? Schauen wir uns das einfach mal an.

Lassen Sie uns die Navigation um eine, vielleicht noch zwei weitere Ebenen erweitern. Dazu sind im Wesentlichen nur ein paar CSS-Anweisungen hinzuzufügen. Auch das Script muss um eine Methode ergänzt werden:

```
...
$(this).find('ul').first().stop(true,true).fadeIn(300);
...
$(this).find('ul').first().stop(true,true).fadeOut(300);
...
```

Damit beim `.hover()` nicht alle Untermenüs aufklappen, schränken Sie das `.find("ul")` auf das erste gefundene Element mit `.first()` ein. Nun müssen Sie per CSS noch bestimmen, dass die nächste Ebene nach *rechts* öffnet und nicht nach *unten*:

```
ul ul ul {
    left:150px;
    top:0;
}

li li li {
    float:left;
    background-color:#CCCCCC;
}
```


Eines dürfen Sie nicht übersehen: Das Menü muss auch ohne JavaScript funktionieren. Daher müssen Sie noch die Pseudo-Klassen `:hover` anpassen:

```
li:hover ul ul {
    display:none;
}
li:hover ul {
    display:block;
}
li li:hover ul {
    display:block;
}
```



Abbildung 5.3 FlyOut-Navigation mit einer weiteren Ebene

Zur Übersicht hier noch mal das gesamte Beispiel, beginnend mit der CSS-Datei namens *flyout.css*. Am HTML-Gerüst haben Sie keine Veränderung vorgenommen, die wird an dieser Stelle übergangen. Ebenso werden allgemeine Stilangaben ausgelassen, die nichts mit der Funktionalität zu tun haben:

```
...
ul {
    width:450px;
    list-style-type:none;
    margin:10px;
}
ul ul {
    display:none;
```

```

    position:absolute;
    width:150px;
    margin:0;
    left:0px;
    top:26px;
}
ul ul ul {
    left:150px;
    top:0;
}
li {
    float:left;
    position:relative;
    padding:5px 15px 5px 5px;
    background-color:#EEEEFF;
}
li li {
    float:none;
    background-color:#CCCCCC;
    padding:0;
}
li li li {
    background-color:#CCCCCC;
}
li li li a {
    background-color:#666666;
}
li li a {
    width:147px;
    margin:0 0 0 0;
    padding:3px 0 3px 3px;
    line-height:20px;
    display:block;
}
li:hover ul ul {
    display:none;
}
li:hover ul {
    display:block;
}
li li:hover ul {
    display:block;
}

```

Listing 5.3 CSS Styles – FlyOut Navigation

Zum Abschluss noch der Inhalt der JavaScript-Datei `flyout.js`:

```
function menu(obj){
    if (!obj.length) return;
    $(obj).find("ul").css({display: "none"});
    $(obj).hover(function(){
        $(this).find('ul').first().stop(true, true).slideDown(300);
    },function(){
        $(this).find('ul').first().stop(true, true).slideUp(300);
    });
}
$(document).ready(function(){
    menu($(".ul li"));
});
```

Listing 5.4 JavaScript-Code – FlyOut Navigation

5.1.2 Die Tabs: Karteireiter

Gerade auf Websites der großen Tageszeitungen können Sie oft eine *Karteireiter-Navigation* antreffen: sogenannte *Tabs*, in der bestimmte Schwerpunktthemen herausgehoben werden. Der Vorteil liegt darin, dass Sie mit einem Klick auf einen der Reiter keine neue Seite aufrufen, sondern nur die Ansicht zwischen einzelnen Blöcken wechseln. Jene Blöcke wurden bereits als `<div>`-Elemente geladen und mit `display:none` aus dem Dokumentfluss entfernt und versteckt.

So könnten Tabs aussehen, hier eine einfache Version:

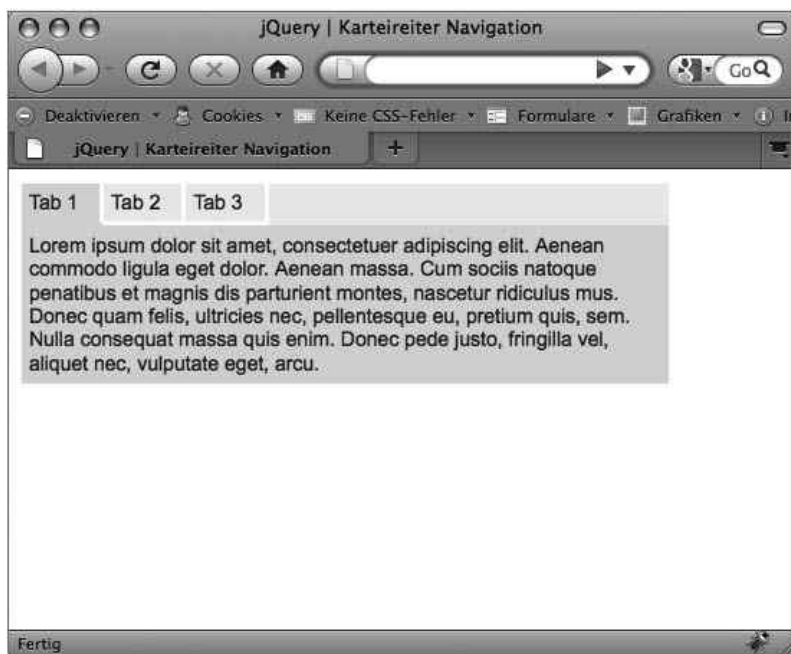


Abbildung 5.4 Tab-Navigation

Später werden Sie sehen, dass es für diese Funktionalität ein eigenes Plugin gibt. Manchmal reicht aber so eine einfache Variante, zumal Sie sie über CSS gut noch weiter gestalten können.

Der Leitgedanke ist wieder der, dass Sie »unaufdringlich« arbeiten. Diese Seite soll auch ohne JavaScript funktionieren. Obendrein soll das Script nicht in das Markup eingreifen und ungültigen Code erzeugen.

Das Gerüst wird lediglich drei Inhaltsblöcke zeigen, der Rest wird dynamisch erzeugt. Fangen Sie auch hier wieder mit dem HTML-Gerüst an:

```
<!DOCTYPE html>
<html>
<head>
<title>jQuery | Karteireiter Navigation</title>
<link rel="stylesheet" href="tabs.css" type="text/css" />
<script type="text/javascript" src="jquery-1.4.2.js"></script>
<script type="text/javascript" src="tabs-advanced.js"></script>
</head>
<body>
  <div id="tab-1" class="tabs" title="Tab 1">
    ... Inhalt hier ...
  </div>
  <div id="tab-1" class="tabs" title="Tab 2">
    ... Inhalt hier ...
  </div>
  <div id="tab-1" class="tabs" title="Tab 3">
    ... Inhalt hier ...
  </div>
</body>
</html>
```

Listing 5.5 HTML-Gerüst – Karteireiter (Tabs)

Sie setzen drei `<div>`-Blöcke untereinander und füllen sie mit Inhalt. Betrachten Sie das Ergebnis ohne Scripteinbindung, sehen Sie nichts anderes als drei Textblöcke.

Doch zurück zu den Vorarbeiten. Die `<ul>`-Listen können Sie über CSS formatieren und zusätzliche Styles für die Tab-Navigation vorbereiten:

```
ul {
  width:450px;
  list-style-type:none;
  margin:10px 0 0 10px;
  background-color:#EEEEEE;
```

```
    overflow:hidden;
}
```

Da Sie die Menüpunkte horizontal anordnen wollen, geben Sie den Listeneinträgen ein `float:left` mit auf den Weg. Sie verzichten bei diesem Beispiel auf einen `<a>`-Tag, der standardmäßig den Mauszeiger auf `pointer` setzt. Aus diesem Grund fügen Sie dem LI-Element den Style `cursor:pointer` hinzu:

```
li {
    float:left;
    cursor:pointer;
}
```

Hier setzen Sie für einen Status `CURRENT`, also für das gerade angeklickte Element einen hervorgehobenen Style:

```
li.current {
    background-color:#DDDDDD;
    border-bottom:3px solid #DDDDDD;
}
```

Für die Standardansicht, also ohne JavaScript, setzen Sie die Breite und Außenabstände:

```
div.tabs {
    width:450px;
    margin: 0 0 10px 10px;
}
```

Wenn Sie gleich mittels jQuery die Tabs aktivieren, werden Sie die Klasse `dyn-tabs` dynamisch hinzufügen:

```
div.dyn-tabs {
    width:440px;
    padding:5px;
    height:100px;
    overflow:auto;
    background-color:#DDDDDD;
    display:none;
}
```

Die Tabs werden ein anderes Aussehen bekommen. Der Hintergrund wird eingefärbt, und es werden erst einmal alle Inhaltscontainer auf `display:none` gesetzt.

Das nun folgende Script ist zwar nicht sehr aufwendig, generiert aber immerhin sogar die Beschriftung der Tab-Navigation (also die Menüpunkte TAB 1, TAB 2 etc.):

```

function tabs(pages) {
  if (!pages.length) return;
  pages.addClass("dyn-tabs");
  pages.first().show();
  var tabNavigation =
    $('<ul id="tabs" />').insertBefore(pages.first());
  pages.each(function() {
    var listElement = $("<li />");
    var label = $(this).attr("title")
      ? $(this).attr("title")
      : "Kein Label";
    listElement.text(label);
    tabNavigation.append(listElement);
  });
  var items = tabNavigation.find("li");
  items.first().addClass("current");
  items.click(function() {
    items.removeClass("current");
    $(this).addClass("current");
    pages.hide();
    pages.eq($(this).index()).fadeIn("slow");
  });
}

$(document).ready(function(){
  tabs($(".div.tabs"));
});

```

Listing 5.6 JavaScript-Code – Karteireiter (Tabs)

Die erste Anweisung innerhalb der Funktion stellt die Frage, ob keine Textblöcke mit der Klasse `pages` gefunden wurden (`pages.length` ist dann 0). In diesem Fall wird das Script hier beendet. Das Argument `pages` übergibt ein jQuery-Objekt. In ihm sind alle Textblöcke gespeichert. Der Selektor `"div.tabs"` selektiert alle entsprechenden Elemente.

Anschließend folgt die Initialisierung, es wird als Erstes die Klasse `"dyn-tabs"` hinzugefügt und der erste `<div>`-Block mit `pages.first().show()`; sichtbar gemacht.

Mit der Zeile

```

var tabNavigation =
  $('<ul id="tabs" />').insertBefore(pages.first());

```

wird einerseits vor dem ersten Textblock ein `<ul>`-Element erzeugt und andererseits dieses Element als jQuery-Objekt in der Variablen `tabNavigation` gespeichert.

Im Folgenden wird über das jQuery-Objekt `pages` mit `.each(fn)` iteriert und für jeden Textblock, der im Dokument gefunden wurde, ein Listenelement LI angelegt (`listElement = $("<li />");`) und in das `<ul>`-Element angehängt (`tabNavigation.append(listElement);`). Die Bezeichner, die Label der Tab-Navigation, wurden im HTML-Gerüst als `title`-Attribut angelegt. Falls Sie vergessen sollten, das `title`-Attribut anzugeben, existiert eine Fallback-Lösung: Es wird der String »kein Label« angehängt.

In der Variablen `item` werden alle Listenelemente in einem jQuery-Objekt abgelegt und ein Event Handler `.click(fn)` hinzugefügt.

Was bei jedem Mausklick passiert:

- ▶ Es werden alle Klassen `current` entfernt.
- ▶ Es wird beim angeklickten Element die Klasse `current` hinzugefügt.
- ▶ Es werden alle Textblöcke ausgeblendet.
- ▶ Es wird der zum angeklickten LI-Element gehörige Textblock eingeblendet.

Dazu ist zu sagen, das Script liest den zum LI-Element gehörenden `.index()` aus, also die Position, an der dieses Element in der Liste gehört, und benutzt es, um mit `pages.eq($(this.index()))` den entsprechenden Textblock auszuwählen und mit `.fadeIn()` anzuzeigen. Sie können selbstredend auch `.slideDown` oder `.show()` verwenden.

Es wurde zwar die ganze Zeit von Textblöcken geredet, aber Sie können natürlich Inhalte jeglicher Art einfügen – durchaus auch Bilder oder Flash-Filme. Ebenfalls vorstellbar ist es, beim Klick auf einen Tab Inhalte per Ajax nachzuladen. Wie das geht, werden Sie noch erfahren. Versprochen!

5.1.3 Das Akkordeon

Nein, wir machen jetzt keine Musik! Für eine bestimmte Art der Navigation hat sich der Begriff *Akkordeon-Navigation* etabliert. Hier werden Querbalken gezeigt, die eine Überschrift oder einfach einen Begriff zeigen, der beim Anklicken gleich einer Ziehharmonika Inhalt freigibt. Klicken Sie einen anderen Balken an, schließt sich der erste, und der neue Inhalt öffnet sich.

Der Grundgedanke ist wieder der, dass diese Seite auch ohne JavaScript funktionieren kann. Hat der Benutzer das Scripting abgeschaltet, soll eine scheinbar einfache Seite erscheinen, die nur aus Überschriften und Textblöcken besteht.

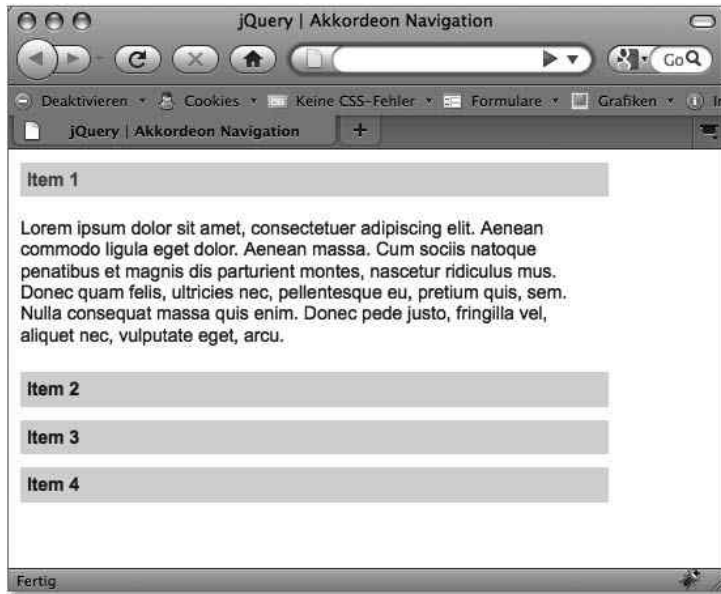


Abbildung 5.5 Akkordeon-Navigation

Dies wird deutlich, wenn wir uns das HTML-Gerüst ansehen:

```
<!DOCTYPE html>
<html>
<head>
<title>jQuery | Akkordeon Navigation</title>
<link rel="stylesheet" href="bandoneon.css" type="text/css" />
<script type="text/javascript" src="jquery-1.4.2.js"></script>
<script type="text/javascript" src="bandoneon.js"></script>
</head>
<body>
<h1 class="bar">Item 1</h3>
<div class="content">
  <p>...Inhalt kommt hier ...</p>
</div>
<h1 class="bar">Item 2</h3>
<div class="content">
  <p>...Inhalt kommt hier ...</p>
</div>
<h1 class="bar">Item 2</h3>
<div class="content">
  <p>...Inhalt kommt hier ...</p>
</div>
...
</body>
</html>
```

Listing 5.7 HTML-Gerüst – Akkordeon Navigation

Einfache H1-Überschriften (es könnten aber auch beliebige andere Elemente sein) werden gefolgt von `<div>`-Containern, in denen »irgendetwas« enthalten ist, sei es Text, Bilder, Tabellen oder etwas anderes. Beachten Sie die Klassennamen `bar` und `content`, mittels derer von jQuery aus die HTML-Elemente gesteuert werden (Sie können natürlich auch andere Klassennamen verwenden).

Bei den CSS-Stilen gibt es keine Besonderheiten zu beachten. Werfen Sie einen kurzen Blick auf die wichtigsten Klassen:

```
.bar {
  background-color:#DDDDDD;
  cursor:pointer;
}
```

Den Klickbalken wird eine Hintergrundfarbe mit der Klasse `.bar` gegeben. Da kein `<a>`-Tag verwendet wird, um einen klickbaren Bereich zu erzeugen, fügen Sie noch `cursor:pointer` hinzu:

```
.current {
  color:#CC0000;
}
```

Die Klasse `.current` wird später dynamisch hinzugefügt, um den Status ANGEKLIKT zu markieren. Den Content gestalten Sie so, wie Sie wollen – dies sei hier nur mit dem Innenabstand angedeutet:

```
.content {
  padding:10px;
}
```

Und nun das Script:

```
function bandoneon(content,bar) {
  if (!content.length && !bar.length) return;
  content.hide();
  bar.click( function() {
    bar.removeClass("current");
    content.not(":hidden").slideUp('slow');
    var current = $(this);
    $(this).next()
      .not(":visible")
      .slideDown('slow',function() {
        current.addClass("current");
      });
  });
}
$(document).ready( function() {
```

```
bandoneon( $(".content"), $(".bar") );
});
```

Listing 5.8 JavaScript-Code – Akkordeon Navigation

Sie erzeugen (im `.ready()`-Block) als Argumente innerhalb des Funktionsaufrufs `bandoneon()` zwei jQuery-Objekte: einmal für alle Content Container `$(".content")` und einmal für alle Balken `$(".bar")`.

In der Funktion `bandoneon` wird die Seite zunächst initialisiert, indem alle Content Container unsichtbar geschaltet werden (`content.hide()`). Zu sehen sind nun nur die Balken, sonst nichts.

Als Nächstes fügen Sie den Event Handler `.click()` zu den Balken `.bar` hinzu. Es folgen zwei Anweisungen, mit denen Sie vorhergehende Zustände zurücksetzen, also eine Art *Reset* durchführen: Das Script löscht alle Klassen `.current`. Diese markieren, wie bereits erwähnt, den angeklickten Zustand. Danach fährt der offene Container zu, indem alle Container durchlaufen werden. Es wird dann nur der offene nach oben geschoben:

```
content.not(":hidden").slideUp('slow');
```

Der Aufruf `$(this).next()` führt zum direkt unterhalb des angeklickten Balkens liegenden Content Container, dieser wird animiert und ausgefahren.

Die Methode mit dem Argument `.not(":visible")` schließt diejenigen Container aus, die bereits geöffnet sind. Das ist dann der Fall, wenn ein Balken bereits angeklickt wurde und der Benutzer ein zweites Mal darauf klickt. Dem Effekt `.slideDown()` geben Sie noch eine Callback-Funktion mit auf den Weg, ihr fügen Sie die schon erwähnte Klasse `.current` hinzu.

Es erscheint eleganter, wenn die Schrift im Balken erst am Ende der Animation den Status ändert. Die Variable `current` haben Sie außerhalb der Callback-Funktion angelegt. Sie wird benötigt, um auf den Wert von `$(this)` innerhalb der Funktion zuzugreifen (`$(this)` würde sich innerhalb der Funktion auf den selektierten Container beziehen und nicht auf den angeklickten Balken!).

Wer dies nicht mag, kann folgende Alternative verwenden:

```
function bandoneon(content,bar) {
  if (!content.length && !bar.length) return;
  content.hide();
  bar.click( function() {
    bar.removeClass("current");
    content.not(":hidden").slideUp('slow');
    $(this).next()
```

```

        .not(":visible")
        .slideDown('slow')
        .prev()
        .addClass("current");
    });
});
}
$(document).ready( function() {
    bandoneon($(".content"), $(".bar"));
});

```

Listing 5.9 Alternativer JavaScript-Code – Akkordeon Navigation

Es wird an die Kette nach `.slideDown()` noch die Methode `.prev()` angehängt, und die Callback-Funktion verschwindet. Wobei die Variable `.current` wegfällt.

Alternative: ein jQuery UI Widget

Später wird gezeigt, dass es ein jQuery UI Widget für die Akkordeon-Funktionalität gibt, das auf einfache Weise einzubinden ist. Es ist mit reichlich Optionen ausgestattet und kann für viele Zwecke angepasst werden. Oftmals bieten solche Plugins weit mehr Funktionalitäten, als Sie benötigen, abgesehen von der zusätzlichen Ladezeit. So werden Sie erwägen, eigene einfachere Scripte zu erstellen, die nur das machen, was sie sollen, nicht mehr und nicht weniger.

5.1.4 Die Spaltennavigation

Im nächsten Beispiel können Sie sich dem Beispiel der Spaltennavigation zuwenden. Beim Start der Seite sehen Sie eine Spalte mit den Listenelementen dieser Spalte, alle anderen Ebenen sind verdeckt. Klicken Sie ein Element an, blendet sich die nächste Ebene ein. Jede Ebene wird nebeneinander geöffnet.

Sie definieren eine maximale Höhe der Navigation. Wenn Sie mehr Navigationseinträge zeigen wollen, als es die definierte Gesamthöhe zulässt, erscheint für die jeweilige Spalte ein Scrollbalken. Bei einem einfachen Klick öffnet sich die nächste Ebene, mit einem Doppelklick öffnet sich der Link, der hinter jedem Listenelement liegt. Die Spaltennavigation eignet sich dann, wenn sehr viele Ebenen und Elemente dargestellt werden müssen, vielleicht findet sie sogar Anwendung in einer Art Sitemap.

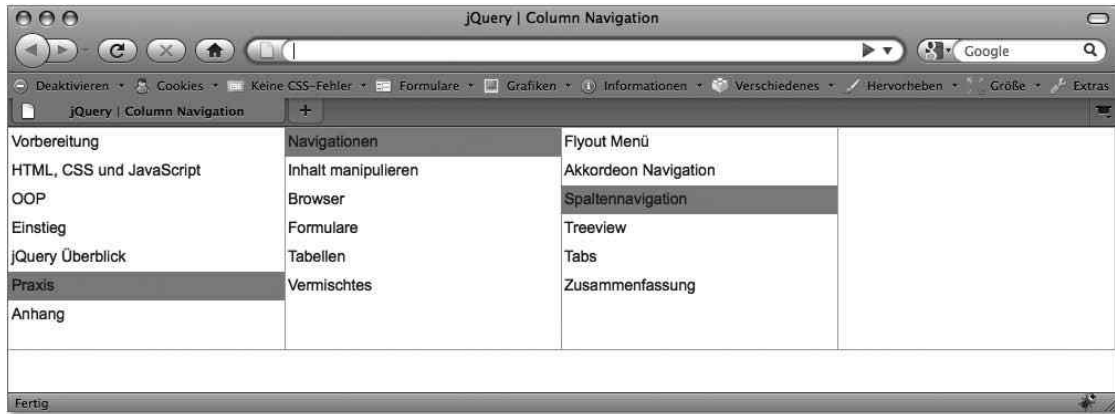


Abbildung 5.6 Spaltennavigation

Ausgang ist wieder eine verschachtelte `<ul>`-Liste:

```
<!DOCTYPE html>
<html>
<head>
<title>jQuery | Column Navigation</title>
<link rel="stylesheet" href="columnnav.css" type="text/css" />
<script type="text/javascript" src="jquery-1.4.2.js"></script>
<script type="text/javascript" src="jquery.replace-1.0.js">
</script>
<script type="text/javascript" src="columnnav.js"></script>
</head>
<body>
<ul id="menu">
  <li><a href="#">Vorbereitung</a>
    <ul>
      ...
    </ul>
  </li>
  <li><a href="#">HTML, CSS und JavaScript</a>
    <ul>
      ...
    </ul>
  </li>
  <li><a href="#">jQuery Überblick</a>
    <ul>
      <li><a href="#">jQuery Core</a>
        <ul>
          ...
        </ul>
      </li>
      <li><a href="#">Selektoren</a>
```

```

        <ul>
            ...
        </ul>
    </li>
</ul>
</li>
    ...
    <li><a href="#">Anhang</a></li>
</ul>
</body>
</html>

```

Listing 5.10 HTML-Gerüst – Spaltennavigation

Im `<head>` des Dokuments sehen sie neben der Einbindung des jQuery Frameworks noch die Datei `jquery.replace-1.0.js`. Dieses Plugin benötigen Sie, um bestimmte Elemente durch andere zu ersetzen. Es beinhaltet die Methode `.replaceAllElements()`, die Sie gleich kennenlernen werden. Zur Benutzung von Plugins schlagen Sie in Kapitel 4, »jQuery – Überblick«, nach, aber im Grunde genommen reicht es, zu wissen, dass Sie im `<head>` eine Datei einbinden und die Methoden und Eigenschaften, die das Plugin bereitstellt, benutzen können. Eigentlich ist das der gleiche Weg, den Sie mit dem jQuery Framework selbst beschreiten, nur, dass ein Plugin die Funktionalitäten erweitert, ohne dass Sie den jQuery-Namensraum verlassen. Das Plugin finden Sie auf dem Begleitmedium und in Abschnitt, 5.6 »Plugin-Entwicklung«.

Nun zum HTML-Code des Body. Die `<ul>`-Liste kann, ja soll sogar richtig umfangreich sein, sie sollte bis zu vier Ebenen in die Tiefe gehen und viele Listenelemente beinhalten, damit Sie genug Futter zum Testen haben.

Sie sollten wieder den Gedanken im Hinterkopf haben, was passieren soll, wenn ein Nutzer JavaScript ausgeschaltet haben sollte. In diesem Fall wird einfach eine unsortierte Liste angezeigt.

Um eines vorwegzunehmen, diese Liste, so wie wir sie erstellt haben, wird nicht als Spaltennavigation darstellbar sein, Sie müssen um eine verschachtelte `<ul>`-Liste noch ein Element umschließen, das die Gesamthöhe begrenzt und bei Bedarf einen Scrollbalken zeigt. Die Auszeichnung sähe dann etwa so aus:

```

...
<ul>
    <div>
        <li>Item
            <ul>
                <div>

```

```

        <li>Item
        ...
      </li>
    </div>
  </ul>
</li>
</div>
</ul>
...

```

Dieser Code, um es klar zu sagen, ist nicht valide. In `<ul>`-Listen sind nur und ausschließlich `<li>`-Elemente erlaubt, keine `<div>`-Container. Dieser Aufbau würde zwar in dem einen oder anderen Browser funktionieren, vielleicht aber in zukünftig auf dem Markt erscheinenden Browsern nicht mehr. Sie würden sich in höchst unberechenbarem Gefilde bewegen. Diesen Weg werden Sie nicht beschreiten. Sie sollten sich stets an ein sauberes und valides Markup halten. Ohne Wenn und Aber. Welche Lösungen bieten sich an dieser Stelle aber denn nun eigentlich an?

Sie könnten den HTML-Baum als valide `<ul>`-Liste aufbauen und mit jQuery `<div>`-Container mit `.wrap()` umschließen. Der dynamisch generierte Code wäre dann wiederum nicht valide. Nächste Möglichkeit: Sie könnten von Hause aus keine Listenelemente verwenden, sondern verschachtelte `<div>`-, `<p>`- und `<span>`-Container, dann sähe das Gerüst so aus:

```

...
<div>
  <p>Item</p>
  <span>
    <div>
      <p>Item</p>
      ...
    </div>
  </span>
</div>
...

```

Das hat den Vorteil, dass der Code valide ist, er wird uneingeschränkt funktionieren, aber aus Sicht der semantischen Auszeichnung ist das nicht befriedigend. `<ul>`- und `<li>`-Elemente entsprechen der Bedeutung der Inhalte. Was tun? Sie nehmen das Beste beider Welten und verwenden für die HTML-Auszeichnung Listenelemente. Mit jQuery werden Sie anschließend jedes einzelne `<li>`-Element durch ein `<div>` ersetzen, `<li>`-Elemente werden zu `<p>`. Nach der Ersetzung fügen Sie noch die benötigten `<span>`-Elemente hinzu, und so können Sie

sich alle Wünsche erfüllen: Der Aufbau ist valide, semantisch korrekt, der durch jQuery dynamisch erzeugte Umbau der Elemente funktioniert einwandfrei.

Also, es gilt weiterhin der eingangs gezeigt Code, ein tief verschachtelter Baum, bestehend aus Listenelementen. Für diese Liste benötigen Sie gesonderte CSS Styles für die Noscript-Benutzerfraktion, die Sie an dieser Stelle unberücksichtigt lassen, bis auf eine Bemerkung, die noch gestattet sei: Sie tun sich in diesem Fall leicht, Stile für beide Fälle in einem Stylesheet festzulegen, da Sie alle Elemente mithilfe von jQuery ersetzen. Sie fügen die Styles für `<ul id="menu">` wie für `<div id="menu">` ein, Sie benötigen keine zusätzlichen Noscript-Abfragen. Einen Einwand könnten Sie noch hegen, man darf einen ID nur ein einziges Mal innerhalb eines Dokuments verwenden, trifft das nicht zu? Doch, es wird nur das Element ersetzt, der DOM-Baum wird verändert, so bleibt `id="menu"` weiterhin eindeutig, nur im Stylesheet wird sie zweimal definiert, da ist nichts dagegen zu sagen.

Um zu verdeutlichen, wie die einzelnen Container positioniert werden, müssen Sie sich vorstellen, wie die Boxen angeordnet werden, was es bedeutet, wenn verschachtelte Container länger sind als der sichtbare Bereich:

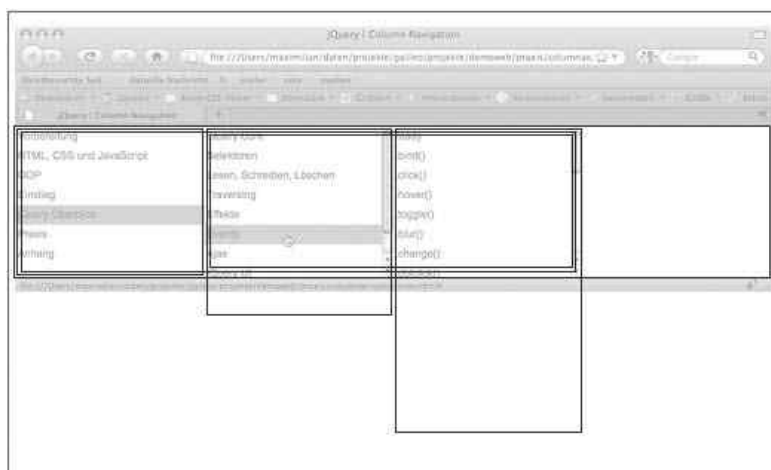


Abbildung 5.7 Spaltennavigation – verschachtelte Container

Damit diese Grafik nicht unübersichtlich wird, wurde ein Container unterschlagen, und zwar derjenige, der entscheidet, ob der Scrollbalken generiert wird: `div#wrapper span`. Merken Sie sich das einfach, jede Spalte besitzt noch einen umschließenden Container, der steuert, ob ein Scrollbalken generiert wird oder nicht.

Wenden Sie sich nun den für unsere Spaltenansicht benötigten Stilen zu:

```
...
#wrapper {
    position:relative;
```

```

width:1001px;
height:200px;
border:1px solid #B2B2B2;
overflow-x:auto;
overflow-y:hidden;
}
...

```

Sie positionieren den äußersten Container `<div id="wrapper">` relativ, damit darin verschachtelte Container mit einer absoluten Positionierung den wrapper als »Anker« verwenden. Also wird dieser Container zum Bezugspunkt gemacht, von dem aus sich darin verschachtelte Boxen aus dem Dokumentfluss herausheben können, ihre Position aber über das Elternelement finden. Mit `overflow-x:auto` wird bestimmt, dass im Falle, dass die Liste an Elementen länger wird, als es der sichtbare Container zulässt, ein Scrollbalken gezeigt wird. Einen waagerechten Scrollbalken unterbinden Sie mit `overflow-y:hidden`:

```

...
div#wrapper div {
  position:absolute;
  width:250px;
  height:100%;
  border-right:1px solid #B2B2B2;
}
...

```

Für die einzelnen Ebenen definieren Sie eine Breite von 250 px, die Höhe müssen Sie auf 100 % setzen, diese Höhe bezieht sich auf das Elternelement, hier ist es `div#wrapper`:

```

...
div#wrapper #menu div {
  left:250px;
  top:0;
  display:none;
}
...

```

Für alle `<div>`-Elemente ab der zweiten Ebene definieren Sie einen Abstand von links und von oben. Jeder untergeordnete Container wird mit 250 px Abstand neben seinem Elternelement geöffnet, alle Boxen werden auf einer Schulterlinie bei 0 px aufgereiht:

```

...
div#wrapper span {
  display:block;

```



```

    height:100%;
    overflow-x:hidden;
    overflow-y:auto;
}
...

```

Hier folgt das zuvor bereits erwähnte `<span>`-Element, die Höhe soll 100% der Gesamthöhe des äußeren Containers `<div id="wrapper">` betragen, ein senkrechter Scrollbalken soll bei Bedarf angezeigt werden:

```

...
p:hover {
    background-color:#EFEFEF;
}
...

```

Zur Verfeinerung verpassen Sie jedem Absatz eine Hintergrundfarbe, wenn der Benutzer mit der Maus über einen Menüpunkt fährt:

```

...
div#wrapper a {
    color:#323232;
    outline:none;
    width:100%;
    padding:3px 3px 5px 3px;
    display:block;
}
...

```

Und jeder A-Link innerhalb von `div#wrapper` erhält ein Styling:

```

...
a.select {
    color:#FFFFFF;
    background-color:#999999;
}
...

```

Die Klasse `a.select` wird dynamisch bei jedem Klick mittels jQuery eingefügt, um die aktuell ausgewählte Ebene zu markieren, und sie wird auch wieder entfernt, wenn ein anderer Listenpunkt ausgewählt wird. Damit wären wir beim eigentlichen Script:

```

function columnnav(obj) {
    if (!obj.length) return;
    obj.wrap('<div id="wrapper"></div>');
    var wrapper = obj.parent();

```

```

$(wrapper).find("div").wrapInner("<span></span>");
$(wrapper).find("a").click( function(e){
    e.preventDefault();
    var parent = $(this).parent();
    parent.siblings().find("div").fadeOut("slow");
    parent.siblings().find("a").removeClass("select");
    parent.find("div:first")
        .fadeIn("slow")
        .find("span")
        .animate({scrollTop:0}, 400);
    $(this).addClass("select");
});
$(wrapper).find("a").dblclick( function() {
    window.location = $(this).attr("href");
});
}
$(document).ready(function(){
    $("#menu").replaceElements({},function() {
        columnnav($("#menu"));
    });
});

```

Listing 5.11 JavaScript-Code – Spaltennavigation

Ganz unten beginnt das Script zu laufen, `$(document).ready()` wartet wieder, bis der DOM-Baum geladen ist, erst in diesem Moment folgt der Aufruf des Plugins. Sie werden das Beispiel in Abschnitt 5.6, »Plugin-Entwicklung« näher kennenlernen. Die Methode `.replaceElements()` ist ein kleines Tool, das die gewünschten DOM-Elemente ersetzt, so wie es bereits besprochen wurde. Als Callback-Argument wird eine anonyme Funktion mit dem Aufruf der Funktion `columnnav()` übergeben. Sie wird erst dann ausgeführt, wenn die Anweisungen dieses Plugins `.replaceElements()` beendet ist, also erst dann, wenn alle `<ul>`- und `<li>`-Elemente in `<div>` und `<p>` umgewandelt wurden.

Die Funktion `columnnav()`

Die Methoden `.wrap()` und `.wrapInner()` fügen die im CSS-Teil besprochenen Elemente `<div id="wrapper">` und `<span>` in den Navigationsbaum ein.

Es folgen zwei Event Handler, der erste steuert mit `.click()` den gesamten Navigationsbaum, der zweite, `.dblclick()`, liest das `href`-Attribut aus und übergibt es an die JavaScript-Eigenschaft `window.location`. Der Link wird daraufhin geöffnet.

Wenden Sie jetzt den Blick zum `.click()`-Event. Jedem `<a>`-Element innerhalb der gesamten Navigation wird dieser Event zugewiesen. Die anonyme Funktion übergibt Ihnen das Eventobjekt `e`. Es wird benötigt, um die ursprüngliche Ausführung des `<a href="#">`-Klicks mit `e.preventDefault()` zu unterbinden.

Es stellt sich jetzt die Frage, welche Funktionalitäten Sie bei der Auslösung des Click-Events benötigen.

Es sind sowohl das Ein- und Ausblenden der jeweils nächsten Ebene als auch die Hervorhebung des angeklickten `<a>`-Elements.

Zunächst müssen Sie, ausgehend vom getriggerten `<a>`-Element, die nicht benötigten Ebenen und Markierungen aufheben:

```
...
parent.siblings().find("div").fadeOut("slow");
parent.siblings().find("a").removeClass("select");
...
```

Danach wird die Markierung gesetzt, indem eine Klasse `select` zum A-Element hinzugefügt wird und die nächste Unterebene, sofern vorhanden, einblendet wird:

```
...
parent.find("div:first")
    .fadeIn("slow")
    ....
    $(this).addClass("select");
});
...
```

Das war's eigentlich. Wenn der User auf einer Unterebene durch eine lange Liste scrollt, anschließend an eine andere Stelle navigiert, um später wieder an die Liste mit dem gescrolltem Inhalt zu stoßen, wird er feststellen, dass diese Position im Hintergrund gespeichert wurde mit:

```
...
.find("span").animate({scrollTop:0}, 400);
...
```

Sorgen Sie dafür, dass die Scrollposition stets zurückgesetzt wird, und zwar mit einer Animation. Wenn Ihnen das nicht gefällt, können Sie die Methode `.scrollTop()` auch direkt aufrufen:

```
...
.find("span").scrollTop(0);
...
```

5.1.5 Von der Spaltennavigation zum Drill-Menü

Stellen Sie sich vor, Sie sehen nicht alle Spalten der Spaltennavigation nebeneinander, sondern stets nur die aktuelle Ebene. Wählt der User eine andere Ebene aus, fährt eine Unterebene von rechts nach links in den sichtbaren Bereich. Die Elternebene bewegt sich zugleich von rechts nach links aus dem sichtbaren »Fenster« heraus.

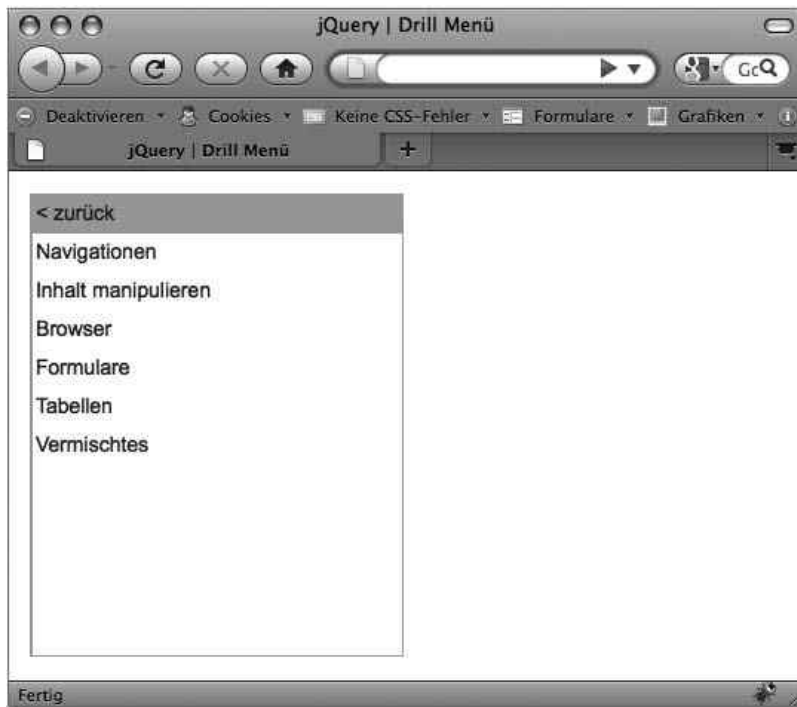


Abbildung 5.8 Drill-Menü

Dazu müssen Sie nur das Menü aus dem vorigen Beispiel etwas modifizieren. Im Prinzip passiert nichts weiter, als dass Sie den äußeren Container `#wrapper` in der Gesamtbreite auf eine Spaltenbreite reduzieren, in diesem Fall auf 250 px. Die dahinter oder besser darin verschachtelten Container werden als Ganzes nach links bewegt, wenn der User eine Ebene tiefer navigiert, und nach rechts, wenn er eine Ebene höher navigiert. Für die Elternebene wird noch ein Menüpunkt ZURÜCK eingefügt.

Gegenüber dem letzten Beispiel ändert sich im Wesentlichen nur der Block für das Div-Element mit dem ID `wrapper`:

```
#wrapper {
  position:relative;
  width:250px;
  height:310px;
```

```

    overflow: hidden;
}

```

Und dem ZURÜCK-Menüpunkt können Sie auch noch eine CSS-Klasse verpassen:

```

a.back {
    background-color: #B2B2B2;
}

```

Alle anderen Anweisungen bleiben unverändert. Blicken Sie nun auf die Script-datei:

```

function drillmenu(obj) {
    if (!obj.length) return;
    obj.wrap('<div id="wrapper"></div>');
    var wrapper = obj.parent();
    $(wrapper).find("div").wrapInner("<span />");
    var str = '<p><a class="back" '
        + 'href="#">&lt; zurück</a></p>';
    obj.find("span").not(":eq(0)").prepend(str);
    var distance = 0;
    obj.find("a").click( function(e){
        e.preventDefault();
        var parent = $(this).parent();
        parent.siblings().find("div").hide();
        obj.find("a").removeClass("select");
        parent.find("div:first").show().find("span").scrollTop(0);
        $(this).not(".back").addClass("select");
        if ($(this).parent().index() != 0) {
            if (!$ (this).parent().find("div").length) return;
            distance += -250;
        }
        else {
            if ($(this)
                .parent().parent().parent()
                .attr("id") == obj.attr("id")) return;
            distance += 250;
        }
        obj.animate({left: distance + "px"});
    });
    $(wrapper).find("a").dblclick( function() {
        window.location = $(this).attr("href");
    });
}
$(document).ready(function(){
    $("#menu").replaceAllElements({},function() {

```

```

        drillmenu($("#menu"));
    });
});

```

Listing 5.12 JavaScript-Code – Drill-Menü

Zunächst müssen Sie den ZURÜCK-Menüpunkt in jeder Ebene (außer der obersten) einfügen, dies geschieht mit der Methode `.prepend()`.

Wie im Spaltennavigationsmenü müssen alle nicht benötigten Ebenen aus der Anzeige entfernt werden. Sie werden mit der Methode `.hide()` ausgeschaltet.

Die `if`-Anweisung `$(this).parent().index() != 0` stellt sicher, dass der erste Menüpunkt ausgeschlossen wird, er wird nur benötigt, wenn der User wieder eine Ebene höher navigieren will, dies passiert in der `else`-Anweisung. Die nächste `if`-Anweisung stellt sicher, dass keine Verschiebung nach links stattfindet, wenn er auf der untersten Ebene angelangt ist. Wenn alle Bedingungen zutreffen, wird der Abstand zur Nullkoordinate um 250 px dem negativen Bereich hinzuaddiert. Das heißt, der ganze Container wird jeweils um 250 px nach links verschoben, wenn Sie auf eine Unterebene navigieren, und um 250 px nach rechts, wenn Sie sich wieder Richtung Root bewegen. Die Animation erfolgt weiter unten mit `obj.animate({left: distance + "px"});`.

Bleibe nur noch die Abgrenzung auf der ersten Ebene, dass das Menü nicht aus dem Bild läuft, wenn Sie auf den ersten Menüpunkt klicken. Das bewältigen Sie mit der `if`-Anweisung innerhalb der `else`-Anweisung, sie fragt ab, ob sich der User in der Ebene mit dem ID des Root-Knotens befindet: `obj.attr("id")`.

Nicht zu vergessen der Event Handler `.dblclick()`, der entweder Inhalte lädt oder auf eine neue Seite führt. Aber das kennen Sie bereits aus dem Beispiel der Spaltennavigation.

5.1.6 Das Tree Menu

Im vorletzten Beispiel wurden die Mac-Freunde mit der Finder-ähnlichen Spaltennavigation bevorzugt, jetzt soll eine einfache Version eines Tree Menus ausprobiert werden, wie man es aus der Windows-Welt kennen. So wird das Beispiel aussehen (siehe Abbildung 5.9).

Beim Klick auf einen Menüpunkt wird die Unterebene freigegeben, die darunterliegenden Menüpunkte verschieben sich nach unten. Das Plus-Icon verwandelt sich in ein Minus-Symbol, als Zeichen, dass das Menü geöffnet ist. Befinden sich keine Kindelemente mehr an einzelnen Menüpunkten, wird kein Icon gezeigt. Beim Klick auf einen geöffneten Teilbaum schließt er sich wieder.

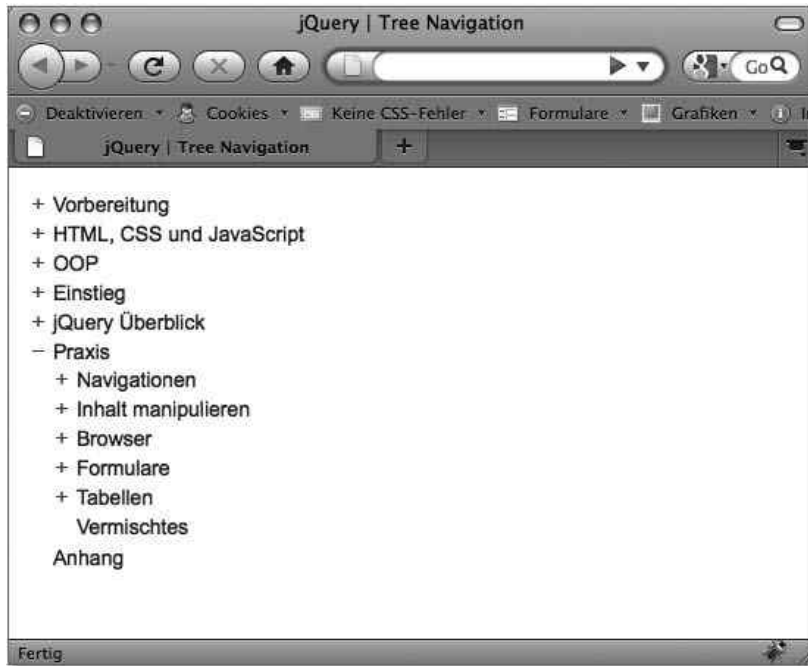


Abbildung 5.9 Tree-Navigation

Beim Doppelklick wird eine Aktion ausgeführt. Das kann der Verweis auf eine neue Seite sein oder das Laden neuer Inhalte via Ajax.

Sie können auch denselben HTML-Baum verwenden, den Sie im letzten Beispiel verwendet hatten, Sie müssen auch keine andere Klassen hinzufügen, Sie müssen einfach nur ein anderes Script einhängen und eine andere CSS-Datei einbinden:

```
...
<title>jQuery | Tree Navigation</title>
<link rel="stylesheet" href="treemenu.css" type="text/css" />
<script type="text/javascript" src="jquery-1.4.2.js"></script>
<script type="text/javascript" src="treemenu.js"></script>
...
```

Auch die Elementersetzung mit `.replaceAllElements()` müssen Sie nicht vornehmen, der Baum ist so, wie er ist, geeignet. Fangen Sie mit dem Stylesheet an:

```
* {
  padding:0;
  margin:0;
  font-family:Arial, Helvetica, sans-serif;
  color:#333333;
  font-size:14px;
  line-height:18px;
}
#menu a {
  display:block;
```

```

padding:0 0 0 15px;
outline:none;
text-decoration:none;
background-image:url(images/icon-plus.png);
background-repeat:no-repeat;
background-position:0 4px;
}
#menu li {
margin:0;
display:block;
border:1px solid #FFFFFF;
}
#menu a.selected {
background-image:url(images/icon-minus.png);
}
#menu a.leaf {
background-image:none;
}
#menu {
width:250px;
margin:15px 0 0 15px;
list-style-type:none;
}
#menu ul {
padding:0 0 0 15px;
list-style-type:none;
}

```

Listing 5.13 CSS Styles – Tree-Navigation

Das Stylesheet ist vergleichsweise einfach aufgebaut, die Unterebenen `#menu ul` werden von links jeweils um 15 px eingerückt, damit die Unterebenen gut strukturiert erkennbar sind. Später werden Sie beim `.click()` bzw. `.toggle()`-Event die Klasse `.selected` hinzufügen, die ein Icon für den Status GEÖFFNET als Hintergrundbild zeigt. Und Sie benötigen noch die Klasse `.leaf`, um die `<a>`-Elemente, die am Ende des Baumes stehen, also bildlich gesprochen die Blätter des virtuellen Baumes bilden, mit dieser Klasse zu versehen. Sie sorgt dafür, dass kein Icon angezeigt wird. Nun können Sie zur Scriptdatei übergehen:

```

function treemenu(obj) {
  if (!obj.length) return;
  obj.find("ul").each(function() {
    $(this).css("display", "none");
  });
  $("li:not(:has(ul))").find("a").addClass("leaf");
  obj.find("a").toggle(function(e) {

```



```

        e.preventDefault();
        $(this).parent().find("ul:first").slideDown(400);
        classManager($(this));
    },function(e) {
        e.preventDefault();
        $(this).parent().find("ul:first").slideUp(400);
        classManager($(this));
    });
    obj.find("a").dblclick( function() {
        window.location = $(this).attr("href");
    });
}
function classManager(obj) {
    if (!obj.hasClass("leaf")) {
        obj.toggleClass("selected");
    }
}
$(document).ready(function() {
    treemenu($("#menu"));
});

```

Listing 5.14 JavaScript-Code – Tree Navigation

Zuerst werden alle `<ul>`-Elemente unterhalb des Root-Elements mit dem ID `#menu` auf `display:none` geschaltet. Nur die erste Ebene ist sichtbar. Als Nächstes werden alle Knoten gesucht, die das Ende des Baumes bilden, und es wird die Klasse `leaf` hinzugefügt. Mit der Anweisung `$("li:not(:has(ul))")` wird befohlen: »Suche alle LI-Elemente, die keine Kindelemente mit Elementnamen `<ul>` haben.«

An diese Aktion werden Events an jedes `<a>`-Element gebunden. Die Methode `.toggle()` ermöglicht ein Hin- und Herschalten der `selected`-Klassen und das Ein- und Ausfahren der `<ul>`-Kindelemente. Im Einzelnen:

- ▶ `e.preventDefault()` verhindert, dass die standardmäßige Aktion eines `<a href="#">` ausgeführt wird, der Verweis auf eine andere Seite. Dies wollen wir in diesem Beispiel durch den Event Handler `.dblclick()` erreichen.
- ▶ `$(this).parent().find("ul:first").slideDown(400)` sorgt dafür, dass ausgehend von `$(this)`, also dem angeklickten `<a>`-Element, das `<ul>`-Kindelement vom `<li>`-Elternelement ausgefahren wird.
- ▶ `classManager($(this))` ruft eine kleine Hilfsfunktion auf, die die Klasse `selected` hinzufügt, wenn das gefundene Element kein `leaf` ist. Hintergrund ist der, dass die Grafiken mit den Plus- und Minuszeichen am Ende des Baumes nichts zu suchen haben.

5.1.7 Kleines Helferlein: dynamische Sitemap

Manchmal werden Sitemaps einfach nur lang. Es werden über Content-Management-Systeme Seiten über Seiten erzeugt, manchmal wünscht man sich, dass eine Sitemap eine Ebene weniger anzeigt, manchmal wünscht man sich auch den Gesamtüberblick über alle Seiten. Mit jQuery können Sie beides erreichen. Sie können einen Button mit einem Click-Event versehen, bei dem die dritte Ebene aus- und wieder eingeblendet werden kann.

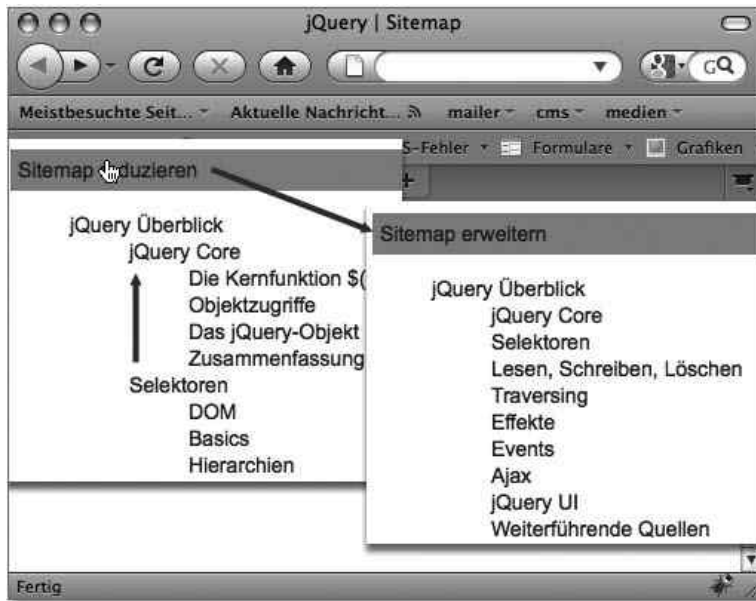


Abbildung 5.10 Dynamische Sitemap

Für das folgende kleine Beispiel verwenden Sie wieder die bereits zum Einsatz gekommene `<ul>`-Liste. Hier auf die Schnelle der jQuery-Code:

```
function sitemap(obj) {
    if (!obj.target.length) return;
    $("body").prepend('<a id="toggle-sitemap" href="#">'
        + obj.textMapOpen + '</a>');
    $("#toggle-sitemap").toggle(function() {
        obj.target.find("ul:visible").slideUp("slow");
        $(this).html(obj.textMapClosed);
    }, function() {
        obj.target.find("ul:hidden").slideDown("slow");
        $(this).html(obj.textMapOpen);
    });
}
jQuery(document).ready(function(){
    var obj = {
        target : $("ul#menu ul"),
```

```

        textMapClosed : "Sitemap erweitern",
        textMapOpen : "Sitemap reduzieren"
    };
    sitemap(obj);
});

```

Listing 5.15 JavaScript-Code – dynamische Sitemap

Dem Event Handler `.toggle()` werden zwei Funktionen zugewiesen, eine für das Ausfahren, eine für das Zusammenfahren. Entscheidend sind dabei die Zeilen

```

...
obj.target.find("ul:visible").slideUp("slow");
$(this).html(obj.textMapClosed);
...

```

und

```

...
obj.target.find("ul:hidden").slideDown("slow");
$(this).html(obj.textMapOpen);
...

```

Der Selektor wählt die jeweils dritte Ebene aus, diese lassen Sie animieren. Mit der Methode `.html()` übergeben Sie einen Inhalt, der innerhalb des `<a>`-Tags eingefügt wird, einmal »Sitemap reduzieren«, einmal »Sitemap erweitern«, je nach Zustand wird dieser Linktext ausgewechselt. Sie gehen wieder vollkommen *unobtrusive* vor, bei der Initialisierung des Scripts wird der gesamten Link (auch ein Button-Element wäre möglich) über die Methode `.prepend()` erzeugt. Falls JavaScript deaktiviert sein sollte, wird die Sitemap statisch angezeigt, nur der Button zum Ein- und Ausklappen wird in diesem Fall gar nicht gezeigt:

```

...
$("body").prepend('<a id="toggle-sitemap" href="#">'
    + obj.textMapOpen + '</a>');
...

```

5.1.8 Zusammenfassung

Alle Beispiele dieses Kapitels bauen auf unsortierten Listen, `<ul>`- und `<li>`-Elementen auf. Sie haben hier auf bemerkenswerte Weise sehen können, dass ein und derselbe HTML-Baum vollkommen unterschiedliche Darstellungen annehmen kann. Zum einen wird das grundsätzliche Design mittels Stylesheets festgelegt, zum anderen bietet jQuery die Freiheit, die Darstellung zu manipulieren, zu animieren, oder vollkommen umzubauen, ohne die Grundsätze von validem

HTML-Code, Trennung von Layout und Struktur, unaufdringlichem Scripting und semantischer Auszeichnung zu verletzen.

5.2 Von Tooltips bis Sprites

5.2.1 Tooltips

Tooltips kennt man, wenn `<a>`-Elemente oder Imagemaps das `title`-Attribut tragen, werden die meisten Browsertypen gelbe Fähnchen zeigen, die Bedienhilfen oder Zusatzinfos beinhalten. Schade nur, dass man das Layout dieser gut gemeinten Browserhilfen nicht beeinflussen kann. Mit jQuery können nun eigene Tooltips aus diesen `title`-Attributen generiert werden, die Sie wunderschön mittels CSS gestalten können, wie Sie wollen. Außerdem können Sie ihr Erscheinen nach Belieben gestalten. Einblenden, einfahren – die ganze Palette an jQuery-Möglichkeiten steht Ihnen zur Verfügung.

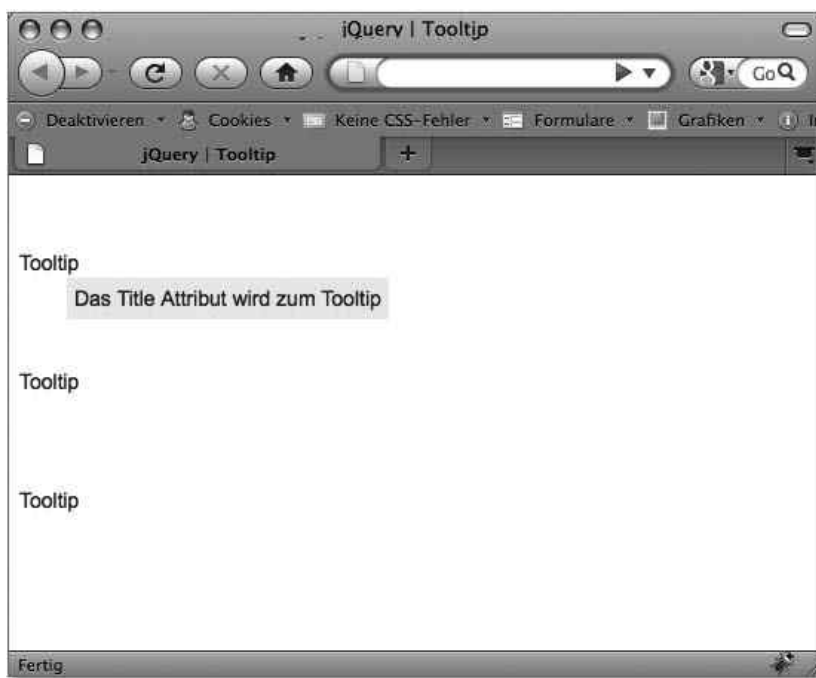


Abbildung 5.11 Tooltip

Blicken Sie zunächst einmal wieder auf den HTML-Baum, Sie binden im Kopf des Dokuments die externen Dateien ein, im Body können Sie mehrere `<a>`-Tags anlegen. Wir begnügen uns im Moment mit einem einzigen:

```
<!DOCTYPE html>
<html>
<head>
```

```

<title>jQuery | Tooltip</title>
<link rel="stylesheet" href="tooltip.css" type="text/css" />
<script type="text/javascript" src="jquery-1.4.2.js"></script>
<script type="text/javascript" src="tooltip.js"></script>
</head>
<body>
  <p>
    <a href="#"
      class="tooltip"
      title="Das Title Attribut wird zum Tooltip">Tooltip
    </a>
  </p>
  ...
</body>
</html>

```

Listing 5.16 HTML-Gerüst – Tooltips

Die Idee ist, dass jedem `<a>`-Tag, das einen Custom-Tooltip erhalten soll, eine Klasse mit auf den Weg gegeben wird. Anhand der Klasse wird selektiert, und danach wird mit `$(".tooltip")` ein jQuery-Objekt erzeugt. Zudem wird später in der Funktion dynamisch ein `<div>`-Container generiert, der absolut positioniert und mittels Abfrage der Mausposition stets als Tooltip an die gewünschte Stelle beordert wird. Dem `<div>`-Container müssen Sie einige CSS-Anweisungen zuordnen:

```

div#tooltip {
  display:none;
  position:absolute;
  z-index:1000;
}

```

Nach den wenigen Vorarbeiten können Sie geschwind zum jQuery-Teil übergehen:

```

function tooltip(obj) {
  if (!obj.length) return;
  $("body").append('<div id="tooltip" />');
  var tooltip = $("#tooltip");
  var title;
  obj.hover(function() {
    title = $(this).attr("title") ?
      $(this).attr("title") :
      "No Title";
    $(this).attr("title","");
    tooltip.html(title);
    tooltip.stop(true,true)
      .delay(50)

```

```

        .fadeIn("slow")
        .dequeue();
    },function() {
        $(this).attr("title",title);
        tooltip.stop(true,true).fadeOut("slow");
    }).mousemove(function(e) {
        tooltip.animate({
            top:e.pageY + 10,
            left:e.pageX + 10
        },200);
    });
}
jQuery(document).ready(function(){
    tooltip($(".tooltip"));
});

```

Listing 5.17 JavaScript-Code – Tooltips

Mit der Methode `.append()` hängen Sie den Tooltip-Container an das Body-Element an. Per CSS haben Sie zuvor bestimmt, dass dieses Element zunächst nicht dargestellt wird. Als Nächstes wird mit `.hover()` der Event Handler initialisiert. Er wird an jedes selektierte Element angehängt. Der anonymen Funktion wird das Objekt übergeben, mit dem alle Tooltip-Klassen abgefragt werden. Sie müssen sich darüber im Klaren sein, dass jQuery das gesamte HTML-Dokument auswertet, da Sie über eine Klasse selektieren und nicht über einen ID. Das kostet bei aufwendigen HTML-Seiten Rechenzeit. Sie können den Selektor ein wenig optimieren, wenn Sie ihn spezifizieren, beispielsweise über `$( "a.tooltip" )`. In modernen Browsern verhält sich dieser Selektor performanter. Allerdings schließen Sie Tooltips in anderen Elementen aus, es könnte ja schließlich notwendig sein, auch ein LI-Element mit einem Tooltip zu versehen.

Im nächsten Schritt wird der Wert des `title`-Attributs in der Variablen `title` gespeichert. Es wird über den Conditional Operator abgefragt, ob ein `title`-Attribut vorhanden ist. Wenn nicht, wird der Defaultwert »No Title« gespeichert. Die Variable wird umgehend mit `tooltip.html(title)` im Tooltip-Container abgelegt.

Jetzt wird der Effekt `.fadeIn()` gesetzt, allerdings wird die `.stop()`-Methode vorangestellt, die bei mehrmaligem Mouseover verhindert, dass der Tooltip wild herumflakkert. Interessant ist zudem noch die Methode `.dequeue()`, im Folgenden werden Sie sehen, dass noch eine Methode `.animate()` eingesetzt wird. Es soll aber auf jeden Fall verhindert werden, dass diese Methoden nacheinander ausgeführt werden, sie sollen zwar versetzt durch das `.delay()` ablaufen, dies

aber in paralleler Weise. Es soll erreicht werden, dass der Tooltip zum einen eingeblendet wird, zum anderen soll eine sanfte Bewegung zur Mausposition stattfinden.

Der zweite Event `.mousemove()` wird benötigt, um die Mausposition bei jeder Bewegung abzufragen und damit den Tooltip an der richtigen Stelle zu animieren. Dieser Event wird nur ausgeführt, wenn sich die Maus über einem Element mit der Klasse `tooltip` bewegt, ansonsten lauscht dieser Event, ob seine Dienste gefragt sind.

5.2.2 Links sammeln, im Footer ausgeben

Stellen Sie sich vor, Sie haben eine Seite mit vielen Quellenangaben in einem Fließtext, die Quellenangaben verweisen auf andere Seiten im Netz. Nun wollen Sie mittels CSS eine Druckansicht anbieten. Die Links sind in der Druckausgabe normalerweise nicht sichtbar. Sie können aber mittels jQuery alle Links in einer Seite sammeln und deren URL am Ende der Seite im Footer auflisten:

```
function collectLinks(obj) {
    if (!obj.length) return;
    var label = "<h4>Verwendete Links:</h4> \n";
    var content = label;

    var dateAndTime = new Date();
    var day = formatDate(dateAndTime.getDate());
    var month = formatDate(dateAndTime.getMonth() + 1);
    var year = dateAndTime.getFullYear();
    var hours = formatDate(dateAndTime.getHours());
    var minutes = formatDate(dateAndTime.getMinutes());
    var dateString = "<p>&nbsp;</p><p>Erzeugt am: "
        + day+"."+month+"."+year+" um "
        + hours + ":"+minutes+"</p> \n";

    var links = obj.find("a");
    links.each(function() {
        content += "<p><strong>"
            + $(this).text()
            + " :</strong>"
            + $(this).attr("href")
            + "</p> \n";
    });
    $("#footer").html(content);
    content +=dateString;
```

```

    }
    $(document).ready(function() {
        collectLinks($("#content"));
    });
    function formatDate(d) {
        if (d < 10)d = "0" + String(d);
        return d;
    }

```

Listing 5.18 JavaScript-Code – Links sammeln und ausgeben

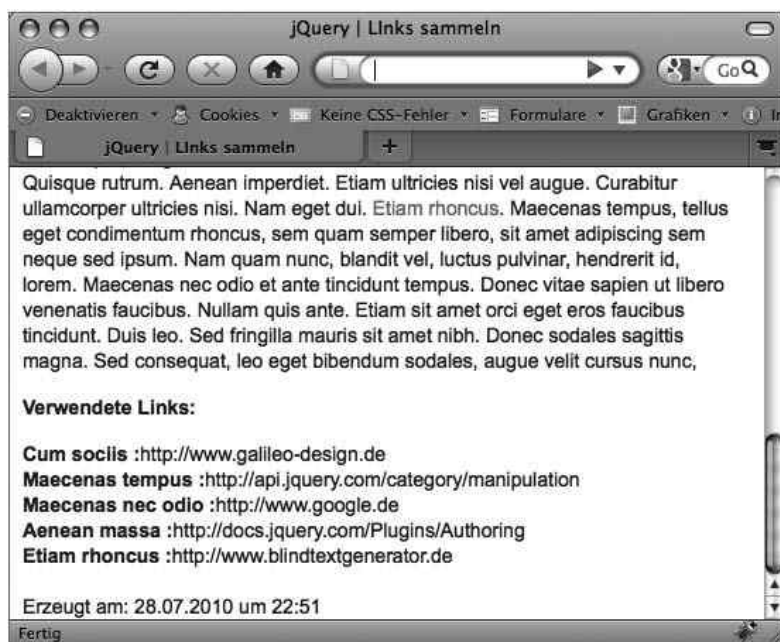


Abbildung 5.12 Links am Seitenende anzeigen

Die Zeile »Erzeugt am: ...« wird mit dem JavaScript-Date()-Objekt generiert. Es ist pures JavaScript. jQuery wird für das Selektieren der <a>-Elemente verwendet. Sind sie erst einmal gesammelt, werden sie über .each() iteriert und mit der Methode .html() an das Seitenende an das Element mit dem ID footer dynamisch angefügt.

5.2.3 Die Koblde auf meinem Bildschirm

Sprites werden in der Computerwelt schon lange als Elemente bezeichnet, die frei über den Bildschirm schweben. Im Webdesign bezeichnet es das Verfahren, dass per CSS ein Ausschnitt definiert wird und ein dahinterliegendes Bild, das größer als der sichtbare Ausschnitt ist, per CSS-Angaben verschoben wird. Dadurch können einfach Animationen ablaufen, vorausgesetzt, die Objekte werden per JavaScript verschoben. Diesem Verfahren werden Sie später in Abschnitt 5.7, »jQuery

UI«, noch einmal begegnen. Dort werden die Bilder aber nicht animiert. In diesem Beispiel soll es darum gehen, Grafikelemente zu animieren und quer über den Bildschirm zu schieben. Das erinnert an frühe Versionen von Flash-Animationen, es ist aber nach wie vor eine nette Spielerei, vor allem weil es dank jQuery ein Leichtes ist, solche Experimente browserübergreifend umzusetzen. Es gibt ein reizendes jQuery-Plugin, das für Webentwickler die Aufgabe der aufwendigen Animation übernimmt, es soll Ihnen also hier nicht vorenthalten bleiben.

Sie können das Plugin unter <http://spritely.net/> ausprobieren und herunterladen. Es befindet sich auch auf der Begleit-DVD dieses Buches.



Abbildung 5.13 Startseite von spritely.net

Das Beispiel auf der Website der Entwickler zeigt eine Idylle vom Land. Zwei Vögel bewegen ihre Flügel, und per Zufallsgenerator fliegen sie an verschiedenen Positionen auf der Bühne vorbei. Der Hintergrund wird einer Theaterkulisse gleich von rechts nach links animiert. Er besteht aus mehreren Ebenen, aus zwei Hügелеbenen und einer Wolkenenebene. Das Flugziel der Vögel können Sie mit Mausklicks verändern. Wenn Sie irgendwo innerhalb des Browserfenster klicken, wird Ihnen einer der beiden Vögel folgen.

Die Animation erfolgt dadurch, dass der Vogel als Hintergrundbild mit den drei Flügelschlagphasen, in einem einzigen Bild zusammengefasst, mittels JavaScript von einem Frame zum nächsten verschoben wird.



Abbildung 5.14 Sprites – drei Phasen einer Animation

Was wäre dieses Buch, wenn Sie Beispiele, die mit diesem Plugin mitgeliefert werden, einfach nur nachstellen. Nein, Sie werden im Folgenden ein eigenes Beispiel kennenlernen. Zugegebenermaßen ein recht sinnfreies. Wenn Sie aber experimentierfreudig sind und Ihren Spieltrieb kultivieren wollen, kommen Sie sicher auf Ihre Kosten.

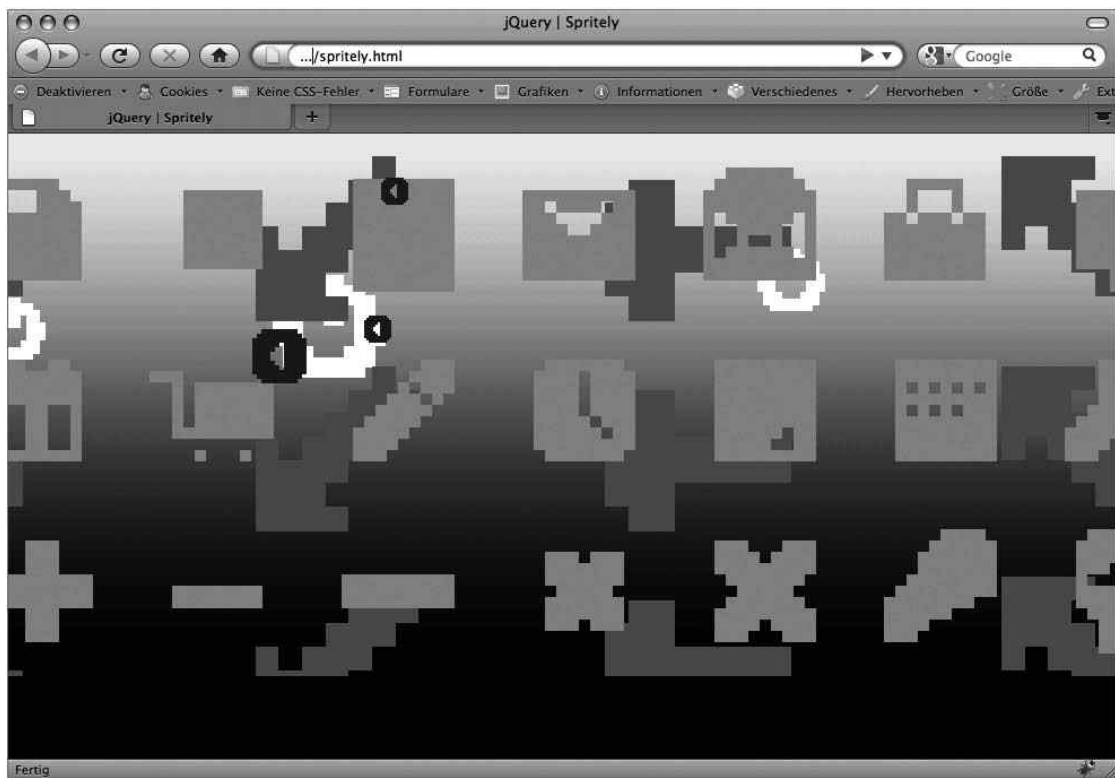


Abbildung 5.15 jQuery-Icons vergrößert und animiert

Dieses vogelwilde Durcheinander sind Icons, die dem jQuery UI CSS Framework entnommen wurden. Sie sind mittels Bildbearbeitung vergrößert und farblich verändert worden. Lassen Sie uns mit dem Vordergrund beginnen. Statt Vögel wird hier eine Reihe von Icons animiert, die nacheinander durchwechseln, die Bilddatei sieht folgendermaßen aus:



Abbildung 5.16 Sprites – Icons als einzelne Phasen einer Animation

Die Hintergründe bestehen aus vier Ebenen; ganz hinten ein Verlauf – er wird nicht animiert – und drei Ebenen mit vergrößerten Icons, die in verschiedenen Richtungen und verschiedenen Geschwindigkeiten laufen. Dies geschieht in einer Endlosschleife, d.h., sie wird automatisch wiederholt. Die Geschwindigkeit wird über die Mausposition gesteuert. Befindet sich die Maus links, laufen die zwei vorderen Hintergrundebenen langsamer, bewegen Sie die Maus nach rechts, werden sie schneller. Im Vordergrund schwirren drei Sprites, zwei kleinere, ein größeres. Sie können die Sprites per Drag & Drop über den Bildschirm schieben oder irgendwo innerhalb des Fensters klicken, dann wird eines der Sprites dem Klick folgen.

Setzen Sie nun das HTML-Gerüst auf:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery | Spritely</title>
  <link rel="stylesheet"
        type="text/css" href="css/spritely.css" />
  <!--[if lte IE 6]>
    <link rel="stylesheet" type="text/css" href="css/ie6.css" />
  <![endif]-->
  <script src="js/jquery-1.4.2.js" type="text/javascript">
  </script>
  <script src="js/jquery-ui-1.8.2.custom.js"
        type="text/javascript">
  </script>
  <script src="js/jquery.spritely-0.2.js" type="text/JavaScript">
  </script>
  <script src="js/init-spritely.js" type="text/javascript">
  </script>
</head>
<body>
  <div id="container">
    <div id="stage" class="stage">
      <div id="bg" class="stage"></div>
      <div id="level1" class="stage"></div>
      <div id="level2" class="stage"></div>
      <div id="level3" class="stage"></div>
    </div>
  </div>
</body>
</html>
```

```

    </div>
    <div id="icons1"></div>
    <div id="icons2"></div>
    <div id="icons3"></div>
  </div>
</body>
</html>

```

Listing 5.19 HTML-Gerüst – Spritely-Animation

Sie benötigen zusätzlich zum Spritely-Plugin noch das jQuery UI Framework. Wie Sie das zusammenstellen und herunterladen, erfahren Sie in Abschnitt 5.7, »jQuery UI«. Um Ihnen einen Schnelleinstieg an dieser Stelle zu ermöglichen: Auf der Begleit-DVD zu diesem Buch finden Sie im Ordner für dieses Beispiel auch eine Version des Plugins, das Sie einfach verwenden und einbinden können.

Sie müssen noch eine Browserweiche mittels Conditional Comments einrichten, um dem Internet Explorer 6 eine adäquate Darstellung zu ermöglichen. Sie werden für das Spritely-Experiment PNG-Grafiken mit Alpha-Transparenz verwenden. Das beherrscht der Internet Explorer 6 leider nicht, daher muss hier dem Browser eine andere Darstellung angeboten werden. Zusätzlich binden Sie noch die Datei *spritely.css* ein, in der Sie die notwendigen Styles notieren:

```

#stage {
  top: 0px;
  left: 0px;
  z-index: 100;
}
.stage {
  position: absolute;
  top: 0;
  left: 0;
  width: 100%;
  min-width: 900px;
  height: 480px;
  overflow: hidden;
}
#bg {
  background: #CCCCCC url(../images/background.png) 0 0
              repeat-x;
}
#level1 {
  background: transparent
              url(../images/level-1.png) 305px 102px repeat-x;
}

```

```

#level2 {
    background: transparent url(../images/level-2.png);
}
#level3 {
    background: transparent url(../images/level-3.png) 0 10px
        repeat-x;
}
#icons1 {
    background: transparent url(../images/sprite-1.png) 0 0
        no-repeat;
    position: absolute;
    top: 150px;
    left: 65px;
    width: 32px;
    height: 32px;
    z-index: 2000;
    cursor: pointer;
}
#icons2 {
    background: transparent url(../images/sprite-2.png) 0 0
        no-repeat;
    position: absolute;
    top: 20px;
    left: 150px;
    width: 64px;
    height: 64px;
    z-index: 2002;
}
#icons3 {
    background: transparent url(../images/sprite-1.png) 0 0
        no-repeat;
    position: absolute;
    top: 200px;
    left: 300px;
    z-index: 2002;
}

```

Listing 5.20 CSS Styles – Spritely-Animation

In der Stylesheet-Datei werden die `<div>`-Container, soweit notwendig, absolut positioniert, damit die einzelnen Ebenen übereinandergestapelt werden. Für die Sprites werden Sie sehr hohe *z*-Indizes vergeben, um sicherzustellen, dass die Sprites auch an oberster Stelle zu liegen kommen, sie sind ja schließlich die Hauptdarsteller und sollen klar sichtbar sein. Woran erkennt eigentlich das Plug-

in, wie viele Einzelbilder solch ein Sprite besitzt? Sie definieren es mit den Höhen- bzw. Breitenangaben. Dadurch dass Sie dieses Sprite auf 32x32 px setzen, berechnet das Script, dass es das Hintergrundbild mit den 14 Icons in 32er-Schritten zum nächsten Bild bewegen muss. Das zweite Sprite, das Sie hier integrieren können, besitzt die Dimensionen 64x64 px.

Nun zum JavaScript-Code:

```
$(document).ready(function() {
  $('#icons1')
    .sprite({fps: 9, no_of_frames: 12})
    .spRandom({top: 50, bottom: 200, left: 300, right: 320})
    .isDraggable()
    .activeOnClick()
  $('#icons2')
    .sprite({fps: 9, no_of_frames: 12})
    .spRandom({top: 0, bottom: 400, left: 30, right: 620})
    .isDraggable()
    .activeOnClick()
    .active();
  $('#icons3')
    .sprite({fps: 9, no_of_frames: 6})
    .spRandom({top: 50, bottom: 20, left: 0, right: 620})
    .isDraggable()
    .activeOnClick()
  $('html').flyToTap();
  $('#level1')
    .pan({fps: 30, speed: 0.7, dir: 'left', depth: 10});
  $('#level2').pan({fps: 30, speed: 2, dir: 'left', depth: 30});
  $('#level3').pan({fps: 30, speed: 3, dir: 'left', depth: 70});
  $('#level2, #level3, #level1').spRelSpeed(8);
  $("#container").mousemove(function(event) {
    var newSpeed = Math.round(event.pageX / 100);
    $('#level2').spSpeed(newSpeed);
    $('#level3').spSpeed(newSpeed * -1);
  });
});
```

Listing 5.21 JavaScript-Code – Spritely-Animation

Im Grunde genommen gibt es für die Animationen nur zwei essenzielle Methoden, nämlich `.sprite()` und `.pan()`. Mit der Methode `.sprite()` lassen sich Objekte selbst animieren, mit der Methode `.pan()` animieren Sie Hintergründe. Der ersten Methode geben Sie die Option `fps` für »Frames per Second« mit, damit steuern Sie die Geschwindigkeit, mit der Option `no_of_frames` für »Number of

Frames« steuern Sie, wie viele Frames abgespielt werden sollen. Wenn ein Sprite nicht endlos animiert werden soll, können Sie mit der Option `play_frames` einen numerischen Wert für die Anzahl an abzuspielenden Frames angeben, danach stoppt das Sprite. Letztere Option wird in dem Beispiel nicht verwendet. Mit der Methode `.spRandom()` initiieren Sie, dass sich die Sprites nach einem Zufallsmuster, aber innerhalb bestimmter Grenzen bewegen sollen, die Grenzen legen Sie mit den Optionen `top`, `bottom`, `left` und `right` fest. Mit `.isDraggable()` initiieren Sie die Möglichkeit, das Sprite mit der Maus zu ziehen. Die Methode `.activeOnClick()` ermöglicht es, ein Sprite zu markieren, damit es aktiviert wird, um an die Stelle zu fliegen, die Sie innerhalb des Fenster anklicken. Mit `.active()` aktivieren Sie ein Sprite bei der Initialisierung. Sie können nur ein einziges Sprite aktivieren, aber alle Sprites auf `.activeOnClick()` setzen.

Die Methode `.pan()` besitzt die Optionen `fps` und `speed`, um die Geschwindigkeit zu regeln. Mit `.sprite()` können Sie die Option `speed` nicht verwenden. Sie können allerdings der Option `dir` die Werte `left` und `right` übergeben, damit bestimmen Sie die Laufrichtung der Animation. Mit `depth` legen Sie eine Art Tiefenwirkung fest, die bei der Animation die Geschwindigkeit angleicht, um einen räumlichen Effekt zu erzielen. Ähnliches erreicht die Methode `$('#level2, #level3, #level1').spRelSpeed(6)`, die die Geschwindigkeit relativ zu allen im Selektor angegebenen Elemente herstellt. Schließlich aktivieren Sie die Sprite-zu-Maus-Aktion mit der Methode `$('#html').flyToTap()`.

Innerhalb des `.mousemove()`-Events fragen Sie die aktuelle Mausposition ab, um die Geschwindigkeit zu regeln. Sie haben zwar bei der Initialisierung des Scripts jede Menge Geschwindigkeiten festgelegt, aber mit der Methode `.spSpeed(newSpeed)` können Sie sie jederzeit ändern. Sie tun das für die Elemente `#level-1` und `#level-2`, wobei Sie für `#level-2` die Richtung mit `* -1` umkehren.

Zum Abschluss noch ein Wort zur Performance, dieses Script beansprucht auf einem halbwegs modernen Rechner bereits einen beträchtlichen Anteil an Prozessorleistung. Bedenken Sie, dass die Animation nicht nur bei der Initialisierung eine Lastspitze erreicht. Die Rechenkerne werden kontinuierlich belastet. Sie sollten dieses Plugin mit Bedacht einsetzen, zum Experimentieren aber lädt es auf jeden Fall ein.

5.3 Spiel mit Bildern

Wenn Sie Bildergalerien mit mehreren oder vielen Bildern einrichten wollen, müssen Sie zuerst überwachen, ob die Bilder in den Browser geladen wurden, und dann die Bilder zeigen und vielleicht erst bei einem Status GELADEN eine Ani-

mation ablaufen lassen. Nichts einfacher als das, möchte man meinen. Schauen Sie sich dazu einen Scriptabschnitt an, der uns einfach nur logisch erscheint:

```
$(document).ready(function(){
    var images = $("div.load img");
    images.hide();
    images.parent().addClass("loading");
    images.each(function() {
        $(this).load(function() {
            $(this).parent().removeClass("loading");
            $(this).fadeIn(400);
        });
    });
});
```

Listing 5.22 JavaScript-Codebilder laden (nicht lauffähig)

Man kann diese Zeilen folgendermaßen übersetzen: »Finde alle IMG-Elemente, und verstecke sie. Anschließend blende die Bilder langsam ein, wenn sie mittels des Event Handlers `.load()` geladen wurden.« Wenn Sie noch dazu einen `<div>`-Container außerhalb des oder der Bilder mit einer festen Bildhöhe versehen müssen und diesen dynamisch aus der Höhe des Bildes auslesen wollen, werden Sie in einigen Browsern eine Überraschung erleben.

Wenn Sie dieses Script einsetzen, ohne diesen Text weiterzulesen, werden Sie wahrscheinlich mindestens eine schlaflose Nacht damit verbringen, herauszufinden, warum die Bilder beim ersten Aufruf erscheinen, wenn Sie die Seite allerdings erneut laden, die Callback-Funktion mit `.fadeIn()` nicht mehr ausgeführt wird.

Achtung

Wenn Bilder bereits im Cache des Browsers liegen, werden sie vom `.load()`-Event ignoriert. Das betrifft den Internet Explorer bis Version 8, Opera bis Version 10.x und den Firefox 3.x. Webkit bzw. Safari scheinen davon nicht betroffen zu sein.

Eine Möglichkeit, das Problem zu lösen, ist, diese Anweisungen mit folgendem Event Handler aufzurufen:

```
$(window).load(function() {
    var images = $("div.load img");
    if (!images.length) return;
    images.parent().addClass("loading");
    obj.hide();
    obj.each(function() {
        $(this).parent().removeClass("loading");
    });
});
```



```

        $(this).fadeIn(400);
    });
;
});

```

Listing 5.23 JavaScript-Code – Bilder laden

Wenn am `window`-Objekt der Event Handler `.load()` aufgerufen wird (dies ist ein Fallback auf den JavaScript-Event-Handler `window.onload`), wartet die Anwendung, bis alle abhängigen Dateien geladen wurden. Sie ersetzen also `$(document).ready()` durch `$(window).load()`.

Der Handler `.load()`, den Sie an die `IMG`-Objekte angehängt hatten, fällt weg, alle anderen Anweisungen aus dem vorangegangenen Beispiel bleiben. Das scheint in allen Browsern zu funktionieren.

Es gibt noch eine andere Lösung, die hier kurz vorgestellt wird, die ein paar Vorteile gegenüber der globalen Lösung hat. Wenn Sie Bilder dynamisch nachladen, bleibt `$(window).load()` wirkungslos. Zudem ist die im Folgenden vorgestellte Lösung universell einsetzbar. Da Sie nicht die Einzigen sind, die auf dieses Problem stoßen werden, und es schon vorher Webentwickler gab, die mit diesem Problem konfrontiert wurden, werden Sie im Internet schnell fündig. Es gibt zahlreiche Plugins, die dieses Manko beheben wollen. Paul Irish, ein prominentes Mitglied der jQuery Community, hat mit André Hansson ein Plugin veröffentlicht, das genau diese Schwäche behebt und den `load()`-Event mit einem Trick erweitert. Dieses Plugin macht nichts anderes, als die `src` eines Bildes, also die Referenz auf das Bild, zurückzusetzen und dann neu zu setzen. Anschließend wird eine Callback-Funktion ausgeführt.

Plugin für das Cached-Image-Problem

<http://github.com/paulirish/jquery.imgloaded>

Das Plugin finden Sie auch auf der Begleit-DVD zu diesem Buch.

Das Plugin wird folgendermaßen verwendet, zuerst inkludieren wir es im `<head>`-Element der Seite:

```

...
<script type='text/javascript' src='jquery-1.4.2.js'>
</script>
<script type='text/javascript' src='jquery.imagesloaded.js'>
</script>
<script type='text/javascript' src='load.js'></script>
...

```

Im Scriptteil rufen wir die Methode auf:

```
$(document).ready(function(){
  var images = $(".load img");
  images.parent().addClass("loading");
  images.hide();
  images.imagesLoaded(function() {
    $(this).each(function() {
      $(this).fadeIn(400,function() {
        $(this).parent().removeClass("loading");
      });
    });
  },true);
});
```

Listing 5.24 JavaScript-Code – Bilder laden mit Plugin

Es wird ein neues jQuery-Objekt mit dem Selektor `'.load img'` instantiiert, und das Plugin wird mit der Methode `.imagesLoaded()` aufgerufen. Als Parameter geben wir eine anonyme Funktion mit auf den Weg, die in unserem Fall die geladenen Bilder einblendet und die Klasse `.loading` entfernt. Als zusätzliches Argument können Sie noch angeben, ob die Callback-Funktion einmalig, nachdem alle Bilder geladen wurden, ausgeführt werden soll (`true`) oder bei jedem Bild (`false`). Dieses Plugin scheint im Moment die sicherste Methode, das `.load()`-Problem zu umgehen. Es mag sich mit einer neuen Version von jQuery ändern, aber zum jetzigen Stand Mitte 2010 können wir diese Methode empfehlen.

5.3.1 Galerie I: einfache Slideshow

Nachdem Sie das grundsätzliche Vorgehen des Ladeprozesses von Bildern mit Unterstützung von jQuery untersucht haben, können Sie dieses Wissen nutzen, um eine einfache Slideshow zu bauen. Die Diashow zeigt ein Bild, im Hintergrund, unsichtbar, sind weitere Bilder geladen und warten darauf, eingeblendet zu werden. Beim Start der Seite passiert das automatisch, zusätzlich haben Sie eine kleine Navigation zur Verfügung, mit der der Benutzer die Slideshow steuern kann. Die Funktionen sind Abspielen, Stoppen, ein Bild vorwärts, ein Bild rückwärts.

Sie müssen zunächst sicherstellen, dass alle Bilder geladen werden, um im Anschluss eine kleine Animation anzustoßen, die die einzelnen Bilder ein- und ausblendet.



Abbildung 5.17 Slideshow

Beginnen Sie zuerst wieder mit dem HTML-Teil:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery | Load Images</title>
  <style type="text/css" media="screen">
    @import url("slideshow.css");
  </style>
  <script type='text/javascript' src='jquery-1.4.2.js'>
  </script>
  <script type='text/javascript' src='jquery.imagesloaded.js'>
  </script>
  <script type='text/javascript' src='slideshow.js'>
  </script>
</head>
<body>
  <div id="container">
    <div class="slideshow">
      
      
      
    </div>
  </div>
</body>
</html>
```

```

        
    </div>
</div>
</body>
</html>

```

Listing 5.25 HTML-Gerüst – Slideshow

Der dazugehörige CSS-Teil:

```

div.slideshow {
    position:relative;
    margin:15px;
    width:500px;
    min-height:150px;
    padding:10px;
    background-color:#CCCCCC;
}
div.slideshow img {
    position:absolute;
    /* display:none; */
}
.loading {
    background-image:url(loading.gif);
    background-repeat:no-repeat;
    background-position:50% 50%;
}

```

Listing 5.26 CSS Styles – Slideshow

Die Idee ist, alle Bilder in einen Container zu legen und mit `position:absolute` übereinanderzustapeln. Mittels jQuery werden nach dem vollständigen Laden die einzelnen IMG-Elemente gesteuert.

Hier sei noch auf ein interessantes Phänomen hingewiesen: Wenn Sie ein IMG-Element im CSS-Teil auf `display:none` setzen, wird im Opera (Version 9.5) der `.load()`-Event-Handler nicht gefeuert. Sie dürfen nicht im CSS Stylesheet die Bilder verbergen, Sie müssen im Scriptteil mit der Methode `.hide()` alle Bilder verstecken, um sie nacheinander einzublenden:

```

$(document).ready(function(){
    var images = $(".slideshow img");
    images.parent().addClass("loading");
    images.hide();
    images.imagesLoaded(function() {
        slideshow($(this),3000);
    });
}

```

```

    },true);
  });

function slideshow(obj,speed) {
  if (obj.length <= 1) return;
  var array = [];
  var fn = {
    loop: function() {
      fn.blend();
      setTimeout(fn.loop, speed );
    },
    blend: function() {
      var current = obj.filter(".current")
        .fadeOut("slow")
        .removeClass("current");
      current = (current.next().length) ?
        current.next() : obj.first();
      current.fadeIn("slow").addClass("current");
    },
    getLargest: function(array) {
      return Math.max.apply( Math, array );
    }
  }
  obj.hide();
  obj.each(function() {
    array.push($(this).height());
  });
  obj.parent().animate({height:fn.getLargest(array)},"fast");
  obj.parent().removeClass("loading");
  obj.first().fadeIn("slow",function() {
    setTimeout(fn.loop, speed );
  }).addClass("current");
}

```

Listing 5.27 JavaScript-Code – Slideshow

Im Grunde genommen ist dieses Beispiel eine Erweiterung des letzten Scripts. Sie lagern den relevanten Code in eine neue Funktion namens `slideshow` aus.

Zuerst fügen Sie wieder die Klasse `loading` hinzu, verbergen alle Bilder mit `hide()` und rufen anschließend die Methode `.imagesLoaded()` des Plugins aus dem letzten Beispiel auf. Darin finden Sie als Callback den Funktionsaufruf `slideshow()`.

Blicken wir nun auf die Funktion `slideshow()`. Da die Bilder übereinandergelegt werden (Sie erinnern sich, mittels CSS haben Sie sie innerhalb des Selektors

`div.slideshow img` mit `position: absolute` versehen), nehmen Sie sie aus dem Dokumentfluss heraus. Ein nachfolgender Textcontainer würde von unseren absolut positionierten Bildern überdeckt werden, Sie müssen dem umgebenden Container eine Höhe mit auf den Weg geben. Sie könnten natürlich bei einem kleinen statischen Projekt die Höhe fest vergeben, was passiert aber, wenn Sie mit einem Content-Management-System arbeiten? Hier wissen Sie unter Umständen die Höhe erst beim Laden der Bilder. Sie ermitteln das Bild mit der größten Höhe. Sie legen dazu die einzelnen Height-Werte in einem Array ab, und mit der Funktion ermitteln Sie den Maximalwert:

```
getLargest: function (array){
    return Math.max.apply( Math, array );
}
```

Anschließend setzen Sie entweder mit der Methode `.css()` diesen Maximalwert oder wie in unserem Beispiel mit `.animate({height:fn.getLargest(array)}, "fast")`.

Da alle Bilder geladen sind, kann nun die Klasse `loading` entfernt und das erste Bild der Reihe eingeblendet werden:

```
obj.first().fadeIn("slow",function() {
    setTimeout(fn.loop, speed );
}).addClass("current");
```

Als Callback-Funktion wird die Funktion `loop()` aufgerufen, allerdings mit einer zeitlichen Verzögerung.

Die eigentliche Schleife zur Animation der Bilder wird mit der Funktion `fn.loop()` erzeugt:

```
loop: function() {
    fn.blend();
    setTimeout(fn.loop, speed );
},
```

Diese Funktion ist für das automatische Aufrufen der Funktion `fn.blend()` zuständig. Die Funktion `fn.loop()` wird zeitverzögert mit `.setTimeout()` rekursiv aufgerufen.

Der eigentliche Mechanismus läuft in der Funktion `fn.blend()` ab. Mit der folgenden Zeile wird das Element, das die Klasse `current` enthält, gefiltert:

```
var current = obj.filter(".current")
    .fadeOut("slow")
    .removeClass("current");
```

Die CSS-Klasse wird innerhalb einer Reihe von Bildern nur einmal vergeben. Das so gefilterte jQuery-Objekt blendet die Elemente aus und entfernt die Klasse `current`.

Die Variable `current` wird mit folgender Struktur verändert:

```
current = (current.next().length) ?
    current.next() : obj.first();
```

Es wird abgefragt, ob es ein IMG-Element `current.next()` gibt. Wenn diese Bedingung `true` zurückgibt, wird die Variable `current` auf das nächste Element in die Reihe von Bildern gesetzt, und wenn die Bedingung `false` zurückgibt, `.next()` also beim letzten Element angelangt sind, wird das erste IMG-Element in der Variablen gespeichert. Anschließend wird das nächste Bild (oder das erste) eingeblendet. Wird diese Funktion in Schleifen aufgerufen, erhalten Sie eine Loop, in der ein Bild nach dem anderen aufgerufen wird.

Fehlt noch eine Steuerung, bisher läuft die Animation automatisch ab. Sie werden im nächsten Schritt einen STOP-/PLAY-Button sowie VOR- und ZURÜCK-Buttons einbauen.

Dazu müssen Sie den HTML-Teil erweitern, Sie könnten diese Elemente auch via jQuery erzeugen, das wäre weit mehr unobtrusive, aber der Einfachheit halber fügen Sie die Schaltelemente direkt in den HTML-Code, und zwar hinter den Slideshow-Container:

```
...
<div class="controls">
  <a href="#" class="back">&lt;</a>
  <a href="#" class="start-stop">Stop</a>
  <a href="#" class="forward">&gt;</a>
</div>
...
```

Sie können jetzt das Script erweitern (die neu hinzugefügten Zeilen sind wieder fett hervorgehoben):

```
$(document).ready(function(){
//keine Veraenderung
...
});
function slideShow(obj,speed) {
  if (obj.length <= 1) return;
  var array = [];
  var auto = true;
  var direction = "forward";
```

```

var fn = {
  loop: function() {
    if (!auto) return;
    fn.blend();
    setTimeout(fn.loop, speed );
  },
  blend: function() {
    var current = obj.filter(".current")
      .fadeOut("slow")
      .removeClass("current");

    if (direction == "forward") {
      current = (current.next().length) ?
        current.next() :
        obj.first();
    }

    if (direction == "back") {
      current = (current.prev().length) ?
        current.prev() :
        obj.last();
    }

    current.fadeIn("slow").addClass("current");
  },
  getLargest: function(array){
    return Math.max.apply( Math, array );
  }
}

obj.hide();
obj.each(function() {
  array.push($(this).height());
});
obj.parent().animate({height:fn.getLargest(array)},"fast");
obj.parent().removeClass("loading");
obj.first().fadeIn("slow",function() {
  setTimeout(fn.loop, speed );
}).addClass("current");
$( ".controls a.start-stop").click(function(e) {
  e.preventDefault();
  obj.stop(true,true);
  auto = auto ? false : true;
  direction = "forward";
  if (auto) {fn.loop(); $(this).html("Stop"); }
  else { $(this).html("Play"); }
});
$( ".controls a.forward").click(function(e) {
  e.preventDefault();

```



```

        obj.stop(true,true);
        auto = false;
        direction = "forward";
        $(this).parent().find("a.start-stop").html("Play");
        fn.blend();
    });
    $(".controls a.back").click(function(e) {
        e.preventDefault();
        obj.stop(true,true);
        auto = false;
        direction = "back";
        $(this).parent().find("a.start-stop").html("Play");
        fn.blend();
    });
}

```

Listing 5.28 JavaScript-Code – Slideshow erweitert

Sie fügen im Wesentlichen in den schon bekannten Code eine Variable `auto` ein. Wenn sie auf `true` gesetzt ist, wird die Funktion `fn.loop()` rekursiv ausgeführt, wenn die Variable auf `false` gesetzt ist, wird die Schleife unterbrochen. Fertig ist zumindest schon einmal der STOP-Button. Mit der Variablen `direction` steuern Sie die Laufrichtung, Sie wollen schließlich auch einen ZURÜCK-Button einbauen. Wenn die Variable `direction` auf »back« gesetzt wurde, ändert sich die Selektion des nächsten Elements, nun wird nach dem vorhergehenden Element `.prev()` gesucht:

```

current = (current.prev().length) ?
           current.prev() :
           obj.last();

```

Wenn die Bedingung `current.prev()` `true` zurückgibt, gehen Sie zum vorhergehenden IMG-Element, wenn nicht, dann selektieren Sie das letzte Element der Bilderreihe.

Blieben nur noch die Schaltflächen, Buttons bzw. die Event Handler, mit denen Sie die Aktionen START/STOP, VOR und ZURÜCK aufrufen. Eigentlich sind es keine Buttons, sondern `<a>`-Elemente, sie seien hier der Einfachheit halber aber generell Buttons genannt.

Alle `.click()`-Events versehen Sie mit `e.preventDefault()` und verhindern dadurch, dass die dem `<a>`-Element zugrunde liegende Aktion `href` ausgeführt wird. Ebenso soll bei allen Handlern erreicht werden, dass die Animation zunächst auf `.stop()` gesetzt wird. Die Parameter, die übergeben werden, bewirken erstens, dass die `.queue()` geleert wird (erstes Argument), und zweitens, dass die Animation zum Endzustand springt.

Es wird damit verhindert, dass die Animation in Zwischenzuständen stehen bleibt. Indem sie zum Endpunkt springt, ist gewährleistet, dass die Deckkraft der Bilder bei jeder Stopp-Aktion 100 % beträgt. An dieser Stelle enden die Gemeinsamkeiten aller Buttons. Betrachten Sie nun den START-/STOP-Button, mit der Zeile:

```
auto = auto ? false : true;
```

Mit jedem Klick wechselt die Variable `auto` den Wert von `true` auf `false` und umgekehrt. Enthält die Variable `auto` `false`, wird die `fn.loop()` unterbrochen, und die Bedingung `if (auto) fn.loop()` ist nicht erfüllt. Wird beim nächsten Klick `auto` wieder auf `true` gesetzt, wird `fn.loop()` gestartet. Daneben gibt es noch die kosmetischen Anweisungen, dem Button entweder den Text »Play« oder »Stop« hinzuzufügen: `$(this).html("Play");` Um die Richtung festzulegen, wird dem Event Handler noch die richtige *direction* mit auf den Weg gegeben. Den VOR- und ZURÜCK-Buttons geben Sie ebenfalls die entsprechende *direction* mit auf den Weg und rufen mit `fn.blend()` die nächste bzw. vorherige Seite in Einzelschritten auf.

5.3.2 Galerie II: Imagebox

Mittlerweile kennen die meisten Webnutzer die berühmte Lightbox, vor ein paar Jahren hat sie ein findiger Webentwickler, Lokesh Dhakar, ganz ohne JavaScript Framework aufgebaut. Sie können das Originalscript unter

<http://www.huddletogether.com/projects/lightbox2/>

herunterladen und ausprobieren. Die Lightbox findet Verwendung in unzähligen Projekten, sie wurde als Standardmodul in viele CMS integriert, es gibt Plugins für WordPress, Typo3, und es gibt unzählige Nachbauten. Es gibt Greyboxen, Shadowboxen, Thickboxen, Fancyboxen und unzählige weitere Varianten.

Allen gemeinsam ist, nach dem Klick auf eine Bildvorschau legt sich ein abgedunkelter Layer über die Seite, und darüber wird eine Box mit der Vergrößerung des Vorschaubilds angezeigt. Meist kann man mehrere Bilder zusammenfassen, und mittels VOR- und ZURÜCK-Buttons zwischen ihnen hin- und herschalten.

Es existieren ziemlich ausgearbeitete Versionen der Lightbox. Seien Sie sich darüber im Klaren, dass das Script, das hier vorgestellt wird, vergleichsweise wenig Funktionsumfang und wenige Parameter zur Konfiguration besitzt. Es wird Ihnen überlassen bleiben, damit herumzuxperimentieren und es auszubauen. Sie werden im weiteren Verlauf auch ein bekanntes jQuery-Plugin kennenlernen, das die hier vorgestellten Funktionen abbildet, aber zuerst wird hier ein eigenes einfaches Script vorgestellt.



Abbildung 5.18 Eine Art Lightbox

Zur Funktionsweise: Das Script sucht sich sämtliche `<a>`-Elemente, die einen bestimmten Klassennamen, beispielsweise `.lightbox`, enthalten, und fügt sie zu einer Gesamtgalerie zusammen. Klicken Sie ein Vorschaubild, öffnet sich die Vollansicht des Bildes, mit den VOR- und ZURÜCKbuttons navigieren Sie zwischen mehreren Bildern. Wollen Sie die Box schließen, klicken Sie in den grauen Abdecker. Wollen Sie eine andere Navigation realisieren, sei Ihnen das überlassen, so könnten Sie in die Fußzeile der Imagebox noch einen SCHLIESSEN-Button integrieren, Metainformationen einblenden oder eine Slideshow ablaufen lassen, analog zum letzten Beispiel. Es gibt auch noch keine Abfrage in diesem Script, die VOR- und ZURÜCK-Buttons auszublenden, wenn nur ein einziges Bild gefunden wurde. Wird JavaScript deaktiviert, wird über das `href`-Attribut des Thumbnails das Vollbild geladen.

Für unsere Zwecke soll dieser Versuchsaufbau genügen. Fangen wir mit der Basis an, dem HTML-Gerüst:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery Galerie</title>
  <link rel="stylesheet" href="galerie.css" media="all"
    type="text/css" />
  <!--[if lte IE 6]>
    <link href="galerie-ie6.css" rel="stylesheet"
```

```

    type="text/css" media="all" />
<![endif]-->
<script type="text/javascript" src="jquery-1.4.2.js"></script>
<script type='text/javascript' src='jquery.imagesloaded.js'>
</script>
<script type="text/javascript" src="galerie.js"></script>
</head>
<body>
  <div id="overlay"></div>
  <div id="imgContainer">
    <div id="imgBox"></div>
    <div id="controls">
      <a href="#" class="back">&lt;</a>
      <a href="#" class="forward">&gt;</a>
    </div>
  </div>
  <h1>Galerie mit jQuery</h1>
  <div class="thumbs">
    <a class="lightpopup" href="images/bild-1.jpg">
      
    </a>
    <!-- weitere Bilder -->
    ...
  </div>
</body>
</html>

```

Listing 5.29 HTML-Gerüst – LightPopUp

Der fett hervorgehobene Teil des Listings sind die Elemente, die Sie für den Abdecker `#overlay`, die eigentliche Box für das große Bild `#imgContainer` bzw. `#imgBox` und den Container `#controls` für VOR- und ZURÜCK-Buttons benötigen. Wenn Sie das Script zur Perfektion treiben wollen, lassen Sie diese Elemente später mittels jQuery erzeugen, aber um das Script so übersichtlich wie möglich zu halten, soll es genügen, die paar Zeilen in das HTML-Dokument zu notieren.

Der untere Block, der auch fett hervorgehoben ist, beherbergt die Vorschaubilder, die Thumbnail-Bilder, Sie werden im jQuery-Teil mit einem Selektor alle Bilder selektieren, die die Klasse `.lightpopup` besitzen, und das Attribut `href` als Quelle für das Hauptbild auslesen. Mehr passiert hier nicht.

Der CSS-Teil enthält eine kleine Besonderheit, es soll erreicht werden, dass der Abdecker, unser Div-Element mit dem ID `overlay`, über das gesamte Browserfenster reicht und mit der Eigenschaft `position:fixed` immer über der gesamten

HTML-Seite zu schweben scheint. Ebenso das Bild für die Vollansicht. Selbst wenn sie die darunterliegende Seite scrollen, soll das Fenster immer im Vordergrund fixiert stehen bleiben. Ganz einfach, sagen Sie? Nein, nicht ganz. Der Internet Explorer 6 weiß mit dem Wert `position:fixed` nämlich nichts anzufangen. Hier können Sie sich eines Tricks bedienen und mittels Conditional Comments diesem Browser ein eigenes Stylesheet verpassen:

```
html {
    overflow-y: hidden;
}
body {
    height: 100%;
    overflow: auto;
}
div#overlay {
    position:absolute;
}
div#imgContainer {
    position:absolute;
}
```

Es wird unterhalb des Gesamt-Stylesheets eingebunden und überschreibt die für uns benötigten Stellen. Die Anweisung `overflow-y: hidden` im HTML-Element bewirkt, dass sich kein zweiter Scrollbalken zeigt, da für das Body-Element `overflow:auto` gesetzt wurde. Bleibt noch, die Höhe auf 100 % zu setzen, damit, falls das Dokument länger als das Browserfenster wird, ein Scrollbalken erzwungen wird. Mit der absoluten Positionierung wird erreicht, dass der benötigte Container im Browserfenster fixiert wird. Sie können anhand dieses CSS-Tricks ein `position:fixed` für den Internet Explorer 6 erzwingen. Inwieweit dieser Trick mit anderen CSS-Stilen, beispielsweise `float`-Umgebungen, in besagtem Browser kollidiert, kann an dieser Stelle nicht hundertprozentig abgeschätzt werden. Bei unseren Tests hat dieser Trick tadellos funktioniert, aber bei solchen Hacks gilt immer: Testen Sie sehr ausführlich.

Doch nun zum Stylesheet für alle Browser. Dem `<div>`-Container `#overlay` wird die Eigenschaft `position:fixed` gesetzt sowie eine Hintergrundfarbe gegeben, deren Deckkraft später per Script ein wenig herabgesetzt wird:

```
div#overlay {
    display:none;
    position:fixed;
    top:0;
    left:0;
    width:100%;
```

```

height:100%;
background-color:#C1C1C1;
z-index:1000;
}

```

Der andere wichtige `<div>`-Container ist der mit dem ID `#imgContainer`, der später das Bild in der Hauptansicht halten wird:

```

div#imgContainer {
  display:none;
  position:fixed;
  overflow:hidden;
  z-index:1001;
}

```

Wichtig ist noch ein hoher `z`-Index, damit sichergestellt wird, dass die beiden Schichten immer ganz oben zu liegen kommen, wobei `#imgContainer` logischerweise noch etwas höher liegen muss. Sie sollten nicht vergessen, die innen liegende `IMG`-Elemente noch absolut zu positionieren:

```

#imgBox img {
  position:absolute;
}

```

Nun können Sie zum `jQuery`-Teil übergehen. Einige Erkenntnisse aus vorangegangenen Beispielen können Sie hier übernehmen. So ist auch das Plugin *jquery.imagesloaded.js* von Nutzen. Es stellt sicher, dass auch Bilder, die bereits im Browsercache liegen, den `.load()`-Event auslösen. Dieses Beispiel ist ein wenig umfangreicher ausgefallen, bleiben Sie aber dennoch am Ball:

```

function lightPopUp(obj) {

  // Wenn kein Vorschaubilder gefunden wurden
  if (!obj.length) return;

  //Default Werte
  var defaults = {
    slowSpeed:400,
    fastSpeed:200,
    fasterSpeed:100,
    maxOpacity:0.8 }

  // benoetigte jQuery-Objekte
  var objects = {
    imgContainer : $('#imgContainer'),
    imgBox : $('#imgBox'),

```

```

controls : $('#controls'),
overlay : $("#overlay")
}

// Variablen, deren Werte auch nach Ausführung der Funktion
//justifyImgBox erhalten bleiben
var imgHeight = 1;
var imgWidth = 1;
...

```

Listing 5.30 JavaScript-Code – LightPopUp (Teil 1)

Am Anfang werden Variablen bzw. Objekte angelegt, die im Verlauf der Funktion `lightPopUp()` noch benötigt werden. Die jQuery-Objekte werden hier einmalig als Eigenschaften des Objekts `objects` abgelegt. Dies wird deshalb gemacht, um nur ein einziges Mal alle notwendigen Elemente des DOM-Baums zu selektieren, zumindest die, die sich nicht verändern werden, um im weiteren Verlauf immer auf die in der Variablen gespeicherten Objekte zugreifen zu können. Das bedeutet einen klaren Performancevorteil. Auf die Variablen `imgHeight` und `imgWidth` wird noch im weiteren Verlauf eingegangen:

```

...
// Variablen, die margin, padding und border speichern
var diffWidth =
    objects.imgContainer.outerWidth(true) -
    objects.imgContainer.width();
var diffHeight =
    objects.imgContainer.outerHeight(true) -
    objects.imgContainer.height();
...

```

Listing 5.31 JavaScript-Code – LightPopUp (Teil 2)

Es soll erreicht werden, dass sich das Bild gleichmäßig zentriert im Browserfenster anordnet. Es gibt zwei unbekannte Werte, die nicht vorausberechnet werden können, zum einen die Bildgröße, sie muss nach dem Laden des Bildes dynamisch berechnet werden, zum anderen die Außen- und Innenabstände der Imagebox. Diese Werte sollen allein über CSS gesteuert werden. Deshalb wird die Breite `.width()` und Höhe `.height()` des noch leeren `#imgContainers` von der Höhe und Breite einschließlich der Abstände `.outerWidth()` und `.outerHeight()` abgezogen. Das Ergebnis wird gleich als Variable in die Positionierungsberechnungen mit einfließen:

```

...
// alle Funktionen in einem Objekt
var fn = {

    //loading lädt Bild und ruft PlugIn imagesLoaded auf
    //@ param String Bildquelle
    loadImg: function(src) {
        var findImg = objects.imgBox.find("img");
        (findImg.length) ?
        findImg.fadeOut(100,function() {loading()}) :
        loading();
        function loading() {
            var img = new Image();
            img.src = src;
            $(img).imagesLoaded(function() {
                fn.openImgBox($(this));
            },true);
        }
    },
    ...

```

Listing 5.32 JavaScript-Code – LightPopUp (Teil 3)

Alle Funktionen werden im Objekt `fn` gekapselt. Betrachten Sie jetzt die Funktion `fn.loadImg()`. Sie ist dafür zuständig, die Bilder zu laden und als Callback die Funktion `fn.openImgBox()` aufzurufen. Es wird an dieser Stelle das Plugin `.imagesLoaded()` aus vorigem Beispiel aufgerufen. An dieser Stelle könnten Sie eine Abfrage einbauen, ob das Plugin gefunden wurde, und eine dementsprechende Warnmeldung ausgeben. Es ist aber nicht zwingend notwendig. Betrachten Sie dies als Ihre Aufgabe. Es wird hier davon ausgegangen, dass das Plugin ordnungsgemäß eingebunden wurde:

```

...
//openImgBox oeffnet imgBox
openImgBox: function(obj) {
    objects.imgBox.html(obj);
    objects.imgBox.find("img").hide();
    objects.overlay
        .fadeTo(defaults.fastSpeed,
            defaults.maxOpacity,
            function() {
                fn.formatImgBox();
            });
},
...

```

Listing 5.33 JavaScript-Code – LightPopUp (Teil 4)

Die Funktion `fn.openImgBox()` speichert mit der Methode `.html()` das geladene Bild, verbirgt sich sofort mit der Methode `.hide()`. Anschließend wird der Container `#overlay` eingeblendet, und als Callback-Funktion wird `fn.formatImgBox()` aufgerufen:

```
...
// formatImgBox formatiert imgBox
formatImgBox: function() {
    objects.imgContainer.css({
        display:"block"});
    var locHeight = objects.imgBox.find("img").height();
    var locWidth = objects.imgBox.find("img").width();
    objects.imgBox
        .find("img")
        .css({height:imgHeight,width:imgWidth});
    imgHeight = locHeight;
    imgWidth = locWidth;
    objects.imgBox
        .find("img")
        .animate({
            opacity:1,
            height:imgHeight,
            width:imgWidth
        },defaults.slowSpeed);
    objects.imgBox
        .animate({
            height:imgHeight,
            width:imgWidth},
            defaults.slowSpeed
        );
    fn.justifyImgBox(true,defaults.slowSpeed);
},
...
```

Listing 5.34 JavaScript-Code – LightPopUp (Teil 5)

Betrachten Sie jetzt die Funktion `fn.formatImgBox()`. Zuerst setzen Sie die CSS-Eigenschaft `display:block` des IMG-Containers. Täten Sie das nicht, würden in einigen Browsern die `.height()`- und `.width()`-Angaben nicht korrekt ausgelesen werden.

In den Variablen `imgHeight` und `imgWidth`, die Sie außerhalb dieser Funktion aber innerhalb des Scopes der Funktion `lightPopUp()` deklariert haben, ist der jeweils vorhergehende Wert der Bildgröße gespeichert. Stellen Sie sich vor, Sie haben bereits ein Bild geladen und klicken den Button VORWÄRTS. Die Werte des

bereits geladenen Bildes verwenden Sie, um das neue Bild und damit automatisch den `#imgContainer` auf die Größe des vorhergehenden Bildes zu setzen. Anschließend werden die Größenwerte neu ausgelesen und in die lokalen Variablen `locHeight` und `locWidth` gespeichert. Mit diesen Werten wird das neu geladene Bild animiert, und es wird auf die tatsächlichen Dimensionen vergrößert oder verkleinert. Betrachten Sie das Ergebnis im Browser. Verwenden Sie Bilder mit unterschiedlichen Größen, und Sie werden sehen, dass das Bild nicht hart in die Fläche gesetzt wird, sondern sich von einer Größe X auf Größe Y sanft ändert. Zuerst wird das Bild animiert, anschließend der Container `#imgBox`:

```
...
// justifyImgBox richtet imgBox aus
//@ dist bestimmt, ob Bildgroesse mit berechnet wird
justifyImgBox: function(dist,speed) {
    var include = {
        height:0,
        width:0
    }
    if (dist) {
        include.height = imgHeight + diffHeight;
        include.width = imgWidth + diffWidth;
    }
    var top =
        Math.round(($ (window).height() - include.height) / 2 );
    var left =
        Math.round(($ (window).width() - include.width) / 2 );
    objects.imgContainer
        .animate({
            top:top,
            left:left
        },speed);
},
...
```

Listing 5.35 JavaScript-Code – LightPopUp (Teil 6)

Die Funktion `justifyImgBox` richtet den `#imgContainer`, respektive die `#imgBox`, neu aus. Das Bild mit den neuen Dimensionen wird innerhalb des Browserfensters neu ausgerichtet, und die `top`- und `left`-Werte für den `#imgContainer` werden neu gesetzt. Diese Funktion wird jedes Mal aufgerufen, wenn ein neues Bild geladen wird, aber auch, wenn sich die Fenstergröße ändert. Der Parameter `dist` bestimmt, ob Außenabstände, Ränder oder Innenabstände mit einbezogen werden sollen:

```

...
//changeImg aehnlich der Slideshow,
//sucht das naechste/vorherige Bild
// @param direction: Richtung, in die gesucht wird
changeImg: function(direction) {
    var current = obj.filter(".current");
    current.removeClass("current");
    if (direction == "forward") {
        current = current.next().length ?
            current.next() :
            obj.first();
    }
    if (direction == "back") {
        current = current.prev().length ?
            current.prev() :
            obj.last();
    }
    current.addClass("current");
    return current;
}
}
...

```

Listing 5.36 JavaScript-Code – LightPopUp (Teil 7)

Dem aufmerksamen Leser wird auffallen, dass die Logik der Funktion `fn.changeImg()` dem letzten Beispiel entnommen wurde. Sie geht stets von einem Element `current` aus und selektiert das nächste Element mit der Methode `.next()`, wenn eines vorhanden ist. Wenn nicht, wird das erste Element gewählt. Vorausgesetzt, die Richtung stimmt. Die hatten Sie in der Slideshow mit dem Parameter `direction` bestimmt, das tun wir auch hier. Soll es rückwärts gehen, wird das Element mit der Methode `.prev()` selektiert. Ist die Methode fündig geworden, weist sie die CSS-Klasse `current` zu.

Eine Anregung sei noch erlaubt: Es wurde gerade davon gesprochen, dass diese Logik dem Beispiel der Slideshow entnommen wurde. Versuchen Sie daher doch einmal, hier diesen automatischen Ablauf einzubauen. Alle Codeelemente, die Sie dazu benötigen, kennen Sie bereits, wenn Sie die vorhergehenden Beispiele angesehen haben. Eine schöne Aufgabe für einen verregneten Winterabend.

```

...
//Eventhandler
//
// Klick auf Thumb

```

```

// @param Event
obj.click(function(e) {
  e.preventDefault();
  var imgSrc = $(this).attr("href")
  $(this).addClass("current");
  fn.loadImg(imgSrc);
});

// Klick auf Abdecker
// @param Event
objects.overlay.click(function(e) {
  var obj = $(this);
  objects.imgContainer.fadeOut(defaults.fastSpeed,function() {
    obj.fadeOut(defaults.fastSpeed);
  });
});

//Klick auf grosses Bild, naechstes wird aufgerufen
objects.imgBox.click(function() {
  fn.loadImg(fn.changeImg("forward").attr("href"));
});

//Klick auf Button vorwaerts, naechstes wird aufgerufen
// @param Event
objects.controls.find("a.forward").click(function(e) {
  e.preventDefault();
  fn.loadImg(fn.changeImg("forward").attr("href"));
});
//Klick auf Button zureck, vorheriges wird aufgerufen
// @param Event
objects.controls.find("a.back").click(function(e) {
  e.preventDefault();
  fn.loadImg(fn.changeImg("back").attr("href"));
});
// bei jedem resize wird imgContainer ausgerichtet
$(window).resize(function() {
  if(!objects.imgContainer.filter(":visible").length) return;
  fn.justifyImgBox(true,defaults.fasterSpeed);
});
// justifyImgBox wir beim Start aufgerufen
fn.justifyImgBox(false,defaults.fastSpeed);
}
...

```

Listing 5.37 JavaScript-Code – LightPopUp (Teil 8)

Mit den Event Handlern steuert der Benutzer das Verhalten unserer Anwendung. Im Wesentlichen wird mit den Click-Events die Methode `fn.loadImg()` aufgerufen. Mit Klick auf das Objekt `objects.overlay` wird der graue Schleier geschlossen. Mit dem Funktionsaufruf `fn.justifyImgBox()` ganz am Ende wird die Anfangsposition des `#imgContainers` festgelegt, diese Box wird nicht nur bei jedem Aufruf dynamisch neu ausgerichtet, sondern auch bei jeder Änderung des Browserfensters mit dem Event Handler `$(window).resize()`. Damit der `#imgContainer` beim ersten Start nicht von `top:0, left:0`, also von oben links ins Bild fliegt, sondern sich schon beim Start der Seite mittig ausrichtet, werden als Letztes wiederum mit dem Aufruf `fn.justifyImgBox()` der `top`- und der `left`-Wert festgelegt:

```
...
// Wenn DOM geladen
$(document).ready(function() {
    // LightPopUp
    // @param jQuery-Objekt mit allen Vorschaubildern
    lightPopUp($("#a.lightpopup"));
});
```

Listing 5.38 JavaScript-Code – LightPopUp (Teil 9)

Zuletzt wird innerhalb des Handlers `$(document).ready()` die Funktion `lightPopUp()` aufgerufen. Das Verfahren kennen Sie bereits aus anderen Beispielen aus diesem Buch. Es wird noch ein jQuery-Objekt mit allen `<a>`-Elementen mit der CSS-Klasse `lightpopup` erzeugt und der Funktion als Argument übergeben.

Fügen Sie nun die hier abgebildeten JavaScript-Codeblöcke zu einem einzigen Script zusammen, und experimentieren Sie. Das vollständige Beispiel finden Sie, wie immer, auf der Begleit-DVD zu diesem Buch.

5.4 Ausgewählte Plugins

Dass Sie das Ziel auf einem leichteren Weg erreichen können, wurde weiter bereits angedeutet. Dieses Versprechen soll im Folgenden eingelöst werden.

Fancybox

Sie werden nun ein fertiges Lightbox-Derivat namens *Fancybox* kennenlernen. Es gibt viele andere Plugins, die ähnliche Funktionen haben, aber die Fancybox zeichnet sich durch eine umfangreiche API aus, viele Konfigurationsmöglichkeiten und ist unter der MIT- und GPL-Lizenz verfügbar. Laden Sie zuerst das Paket unter <http://fancybox.net> herunter. Der Autor Janis Skarnelis würde sich übrigens über eine kleine Spende freuen, wenn Sie das Script einsetzen.

Neben seinem eigenen Plugin verwendet das Script noch weitere Plugins, die ebenfalls im Downloadpaket enthalten sind. Wenn Sie das Downloadpaket entpacken, sehen Sie neben den Dateien *ajax.txt*, *index.html* und *styles.css* die Ordner *example* und *fancybox*. Kopieren Sie den Ordner *fancybox* in Ihr Projektverzeichnis. Legen Sie eine neue HTML-Seite an.

Verwenden Sie im Folgenden das leicht modifizierte HTML-Gerüst, das Sie bereits im vorangegangenen Beispiel kennengelernt haben, und binden Sie folgende Dateien ein:

```
...
<link
  rel="stylesheet"
  href="fancybox/jquery.fancybox-1.3.1.css" />
<script
  type="text/javascript" src="jquery-1.4.2.js">
</script>
<script
  type="text/javascript"
  src="fancybox/jquery.fancybox-1.3.1.js">
</script>
<script
  type="text/javascript"
  src="fancybox/jquery.easing-1.3.pack.js">
</script>
<script
  type="text/javascript"
  src="fancybox/jquery.mousewheel-3.0.2.pack.js">
</script>
<script
  type="text/javascript"
  src="fancybox.js">
</script>
...
```

Listing 5.39 HTML-Gerüst – Fancybox (Ausschnitt 1)

Das Body-Element enthält die Elemente:

```
...
<a rel="lightbox" href="images/bild-1.jpg" title="Titel...">
  
</a>
...
```

Listing 5.40 HTML-Gerüst – Fancybox (Ausschnitt 2)

Statt des Klassennamens wird hier das Attribut `rel` verwendet, wenn der Benutzer eine Gruppe von Bildern durchblättern will. Der Einfachheit halber ist hier nur ein Bild notiert, listen Sie am besten mehrere Bilder auf:

```
$(document).ready(function() {
function formatTitle(title,
                        currentArray,
                        currentIndex,
                        currentOpts) {
return '<div id="custom-title">'
    + (title && title.length ? '<b>'+title+'</b>' : '' )
    + '| Bild ' + (currentIndex + 1) + ' von '
    + currentArray.length + '</div>'; }
$("a[rel=lightbox]").fancybox( {
    'padding' : 0,
    'transitionIn':'elastic',
    'transitionOut':'elastic',
    'speedIn':600,
    'speedOut':200,
    'overlayShow':true,
    'overlayOpacity':0.8,
    'hideOnContentClick':false,
    'titleFormat':formatTitle
});
});
```

Listing 5.41 JavaScript-Code – Initialisierung Fancybox

Die Funktion `.formatTitle()` zeigt sehr schön, wie Sie die Fancybox individuell gestalten können. Mit dieser Funktion legen Sie fest, wie Sie eine Bildunterschrift, die Sie aus dem `title`-Attribut des `<a>`-Elements herauslesen, formatieren können und unterhalb des Bildes öffnen werden. Die Funktion wird im weiteren Verlauf mit `'titleFormat'` aufgerufen.

Der eigentliche Aufruf der Fancybox wird mit der Methode `$(selector).fancybox()` erledigt. Wie Sie sehen können, rufen wir sie mit ein paar Argumenten auf. Hier zeigt sich die Stärke des Plugins, nämlich, dass Sie eine ganze Reihe von Einstellungen vornehmen können.

Mehr über die Fancybox

Eine Dokumentation der API finden Sie auf der Website der Fancybox unter <http://fancybox.net/api>. Alle Optionen und Methoden hier abzubilden würde den Rahmen sprengen.

Wenn Sie, wie in diesem Beispiel das Plugin `jquery.mousewheel-3.0.2.pack.js` inkludiert haben, können Sie mit dem Mausekran zwischen den einzelnen Bildern blättern. Das Plugin `jquery.easing-1.3.pack.js` ist dafür zuständig, dass die Animationen eleganter ablaufen, dass die Animation beispielsweise langsam beginnt und am Ende mit einer Beschleunigung endet. Beachten Sie, dass dieses Plugin mittlerweile in das Core des jQuery UI Frameworks integriert wurde. Es wird aber als eigenes Plugin im Paket der Fancybox veröffentlicht.

Zusammenfassung

Eigentlich bemerkenswert – mit ungefähr hundert Zeilen JavaScript (und nicht zu vergessen: jQuery!) gelingt es Ihnen, selbst eine Art Lightbox von Grund auf aufzubauen. Mit fünf nicht allzu komplexen Funktionen, ein paar Event Handlern und einigen Variablen lässt sich zumindest der Grundstock einer anständigen Bildergalerie schreiben: Bild laden, öffnen, formatieren, ausrichten, nächstes Bild auswählen, anzeigen, schließen.

Sie können auf diese Basis aufbauen, beispielsweise die Navigation erweitern, `title`-Attribute aus dem Link des Vorschaubilds extrahieren und als Bildunterschrift für die Vollansicht ausgeben.

Wer einen größeren Funktionsumfang benötigt, wird auf eines der im Web verfügbaren Plugins zurückgreifen. Wenn diese allerdings ein Feature nicht zur Verfügung stellen, das in einem Projekt gefordert wird, mag es vielleicht eine Option sein, ein eigenes Plugin zu entwerfen, das genau diese Anforderungen erfüllt. Das ist manchmal leichter, als ein viele Hundert Zeilen umfassendes Script nachzuvollziehen und dafür eine Erweiterung zu schreiben.

5.5 Ajax mit jQuery einsetzen

Vor ein paar Jahren war Ajax das »Buzz Word«. Alle sprachen über »die Innovation, die User Interfaces revolutionieren« würde. Heute ist nur noch wenig von Ajax die Rede, nicht, weil es aus der Mode gekommen ist, sondern weil es Teil des Weballtags geworden ist. Selbst als Ajax in Mode kam, war es eigentlich nur dieses Akronym, was neu war. Die Technik gab es bereits Jahre zuvor. Das JavaScript-Objekt `XMLHttpRequest` wurde von Microsoft bereits in den Internet Explorer 5.0 integriert, andere Browser übernahmen dieses Konzept. Mittlerweile verstehen alle modernen Browser diese Schnittstelle.

Ajax steht für »Asynchronous JavaScript and XML«, also für die asynchrone Dateiübertragung zwischen Server und Browser. Asynchron bedeutet, dass eine Anfrage seitens des Browsers an den Server per Script gestellt wird und nicht auf

eine Antwort des Servers warten muss, sondern andere Anweisungen ausführen kann. In der Praxis bedeutet das, dass Sie eine HTML-Seite in dem Browser öffnen können, und in diese Seite werden weitere Daten nachgeladen. Diese Daten können aus kompletten HTML-Seiten oder aus HTML-Fragmenten ohne Head- und Body-Elemente oder auch aus sonstigen XML-Dokumenten bestehen. Auch JSON-Objekte können geladen werden. Zu JSON erhalten Sie im weiteren Verlauf noch mehr Informationen.

Lassen Sie uns als Erstes beim Thema »HTML nachladen« bleiben. Nun kann man nicht davon ausgehen, dass das XMLHttpRequest-Objekt in allen Browsern einheitlich umgesetzt wurde, es ist auch (noch) kein endgültig verabschiedeter Standard des W3C. Hier ist es wieder jQuery, das die Arbeit erleichtert und im Hintergrund die verschiedenen Implementierungen berücksichtigt sowie eine einfache Programmierschnittstelle zur Verfügung stellt.

jQuery Ajax-Datentypen

► HTML-Dokumente und HTML-Fragmente

HTML-Dokumente können als Gesamtes in eine bereits geladene Seite integriert werden, wobei man aus der Gesamtseite einzelne DOM-Knoten extrahieren kann, um sie in eine geladene Seite einzufügen. HTML-Fragmente sind der einfachste Weg, externe Daten zu laden. Kein DOM-Traversing ist notwendig, die Daten werden so, wie sie sind, in ein bereits geladenes DOM-Element eingehängt.

► Textdokumente

können als plain-Text geladen werden.

► XML

ist das mächtigste Austauschformat, zudem hat es sich als eine allgemein anerkannte und weit verbreitete Sprache etabliert. Allerdings kann die Verarbeitung der Daten langsamer vonstatten gehen als in anderen Formaten.

► JavaScript

JavaScript-Dateien können nachgeladen werden, dies macht in seltenen Fällen Sinn, wenn man bestimmte Funktionen nur in ausgewählten Anwendungsfällen benötigt.

► JSON (JavaScript Object Notation)

ist ein simples Austauschformat, das im Wesentlichen aus Name-Wert-Paaren besteht. Es ist einfach zu generieren und leicht zu verarbeiten.

► JSONP

ist eine Erweiterung von JSON, die es ermöglicht, über Domaingrenzen hinweg Daten zu laden. P steht hier für »Padding«.

5.5.1 Laden von HTML-Elementen

Stellen Sie sich vor, Sie wollen eine HTML-Seite erstellen. Die Navigation soll stets im Bild an derselben Stelle stehen, und sie soll nicht bei jedem Aufruf einer

neuen Seite neu geladen werden. Nur die Inhalte werden ersetzt. Der Status des angeklickten Menüpunkts soll mit einer CSS-Anweisung markiert werden.

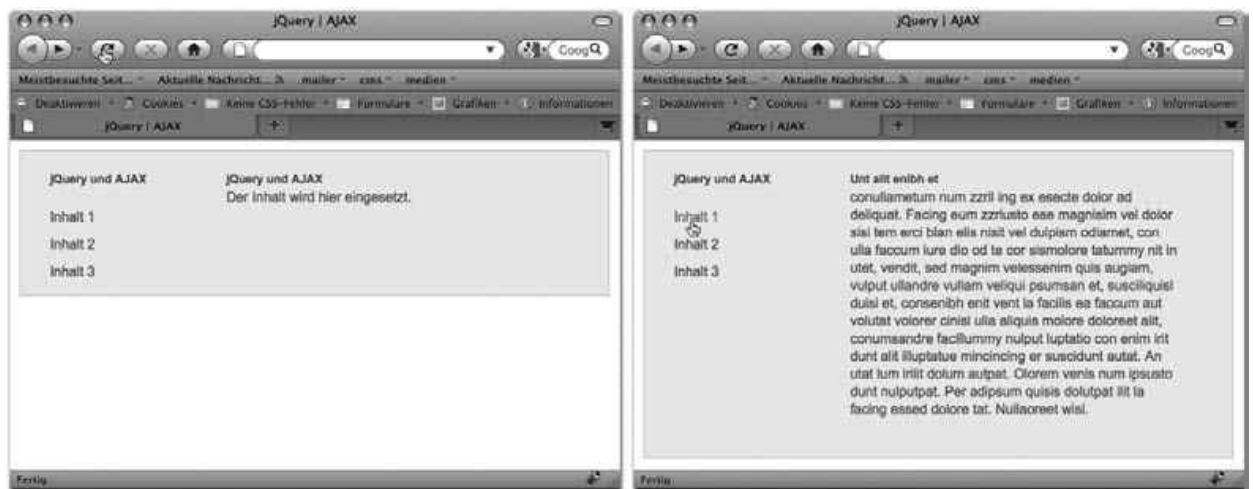


Abbildung 5.19 Laden von Inhalten mit Ajax

Was wird benötigt? Dazu zählt Folgendes:

- eine HTML-Seite
- mehrere HTML-Seiten mit Inhalten
- CSS- und JavaScript-Dateien

Als Erstes legen Sie die HTML-Startseite an:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>jQuery | AJAX</title>
<style type="text/css">
  @import url("ajax.css");
</style>
<script type='text/javascript' src='jquery-1.4.2.js'>
</script>
<script type='text/javascript' src='ajax.js'></script>
</head>
<body>
  <div id="container">
    <div id="col-1">
      <ul>
        <li><a href="content-1.html">Inhalt 1</a></li>
        <!-- weitere Listenelemente -->
        ...
      </ul>
    </div>
  </div>
```

```

        <div id="col-2">
            <p>Der Inhalt wird hier eingesetzt.</p>
        </div>
    </div>
</body>
</html>

```

Listing 5.42 HTML-Gerüst – HTML mit Ajax laden

Die CSS-Angaben beinhalten nur Standards, Fonts, Farben und Dimensionen, (Letztere werden an dieser Stelle übergangen). Betrachten Sie als Nächstes die HTML Snippets, die wir per Ajax in die Seite laden werden:

```

<div>
    <h1>Unt alit enibh et</h1>
    <p>
        conullametum num zzril ing ex esecte
        ...
    </p>
</div>

```

Listing 5.43 HTML-Gerüst – zu ladende HTML-Fragmente

Beachten Sie, dass wir lediglich einen HTML-Ausschnitt verwenden, das reicht für diese Zwecke vollkommen. Aus XML-Sicht ist dieses Dokument valide, Elemente werden geöffnet und wieder geschlossen. Ein XML-Parser wird an diesem Dokument nichts auszusetzen haben.

Das jQuery-Script ist sehr übersichtlich gehalten:

```

function loadPage(obj,target) {
    if (!obj.length && !target.length) return;
    obj.click(function(e) {
        e.preventDefault();
        var url = $(this).attr("href");
        obj.removeClass("current");
        $(this).addClass("current");
        target.fadeOut("slow",function() {
            target.load(url,function() {
                target.fadeIn("slow");
            });
        });
    });
}
// Wenn DOM geladen
$(document).ready(function() {

```

```
loadPage($("#url a"),$("#col-2"));
});
```

Listing 5.44 JavaScript-Code – HTML mit Ajax laden

Nachdem abgefragt wurde, ob die jQuery-Objekte auch wirklich vorhanden sind, wird der `.click()`-Event an die einzelnen Links der Navigation gebunden. Mit `$(this).attr("href")` wird die Referenz der zu ladenden Datei ausgelesen. Um den Menüpunkt als angeklickt zu markieren, wird bei jedem Klick eine Klasse `current` gesetzt. Zunächst werden aber erst einmal alle `current`-Klassen entfernt, die vielleicht vorher angeklickt wurden, die Navigation wird sozusagen von der Klasse `current` bereinigt. Sofort danach wird sie dann für den aktuell angeklickten Menüpunkt gesetzt.

Jetzt wird der eigentliche Ladeprozess eingeleitet, zuerst wird die Content-Spalte `#col-2` ausgeblendet, als Callback-Funktion wird die Methode `.load()` übergeben, die das HTML-Fragment in die Inhaltsspalte lädt. In diese Methode ist wiederum eine Callback-Funktion verschachtelt; wenn also das HTML-Fragment geladen wurde, wird die Content-Spalte `#col-2` wieder mit `.fadeIn()` eingeblendet.

Achtung

Die hier verwendete `.load()`-Methode ist nicht zu verwechseln mit der bereits beschriebenen gleichnamigen Methode `.load()`. Wenn man mindestens das Argument `.load(url)` verwendet, wird eine externe Datenquelle geladen, wenn man hingegen `.load(callbackFunktion)` verwendet, ist der Event Handler gemeint. Sie können selbstverständlich auch der Ajax-Methode `.load(url,callbackFunktion)` eine Callback-Funktion übergeben.

Die Methode `.load()` ist eine Kurzform des Interfaces `jQuery.ajax()`. In vielen Fällen reicht uns die Methode `.load()`.

Im nächsten Schritt wird das Beispiel um ein Feature erweitert. Es wird lediglich ein Bereich eines DOM-Baums geladen, und es soll jetzt eine komplette HTML-Seite und kein Fragment mehr geladen werden.

Bereiche einer HTML-Seite laden

Das kann durchaus sinnvoll sein. Wenn Sie die Prinzipien des unobtrusive JavaScript beibehalten wollen, könnte die Beispielseite auch ohne JavaScript funktionieren. Es werden schließlich `<a>`-Elemente mit `href`-Attributen verwendet, die ohne JavaScript wie gewohnt funktionieren würden:

```
<a href="content-1.html">Inhalt 1</a>
```

Wäre das Zieldokument ein vollständiges HTML-Dokument, hätten Sie das Ziel erreicht, und genau das wird in diesem Beispiel realisiert. Zusätzlich können Sie noch eine Navigation in *content-1.html* integrieren. Dann hätten Sie auch das Problem mit Google & Co. gelöst, denn schließlich könnten die Zieldokumente auch von Suchmaschinen indiziert werden und von ahnungslosen Besuchern gefunden werden. So können Sie auf jeder Unterseite eine Navigation anbieten. Egal, welche Unterseite vom User als Erstes angesteuert wird, er findet sich stets zurecht und kann navigieren. Wenn man diesen Gedanken zu Ende denkt, wird jede Seite, die zuerst aufgerufen wird, zur Hauptseite für alle anderen Seiten – je nachdem, welche man zuerst öffnet. Der Vorteil einer solchen Ajax-Lösung liegt auf der Hand. Die Navigation würde niemals neu geladen werden, und neue Inhalte könnten weich ein- und ausgeblendet werden. Das freut den Designer und sorgt für eine elegante Erscheinung.

Nun zu unserem Code. Die Codefragmente müssen Sie zu einem kompletten validen HTML-Dokument erweitern. Bis auf die unterschiedlichen Inhalte gibt es keinen Unterschied mehr zur Startseite. An der Startseite selbst müssen Sie keine Änderungen vornehmen, nur der jQuery-Code muss modifiziert werden, und zwar nur an einer einzigen Stelle:

```
target.load(url + " #col-2 > *",function(){
```

Es muss nur der String innerhalb der `.load()`-Methode geändert werden, der String, der in der Variablen `url` steckt, wird um [Leerzeichen][Selektor] erweitert. Der Selektor gibt die Anweisung, alle beliebigen Elemente innerhalb des Elements mit der ID `#col-2` an das Element zu hängen, das im jQuery-Objekt `target` selektiert wurde. Alle anderen Elemente werden ignoriert.

Wie kommt eine HTML-Seite in die Imagebox?

Versuchen Sie, Inhalte per Ajax zu laden und in die Imagebox aus dem Beispiel Galerie II einzufügen. Es fehlt nicht viel, es muss im Wesentlichen nur an einer Stelle eine Änderung des Scripts erfolgen.

Wie Sie sich denken können, hauptsächlich in der Funktion `loadImg()`. Zuerst deklarieren Sie in der Funktion `loadImg()` die Variable `ajax`, etwa in der Nähe der Variablen `imgWidth`:

```
...
var imgHeight = 1;
var imgWidth = 1;

var ajax = false;
...
```

Listing 5.45 JavaScript-Code – Imagebox und Ajax (Teil 1)

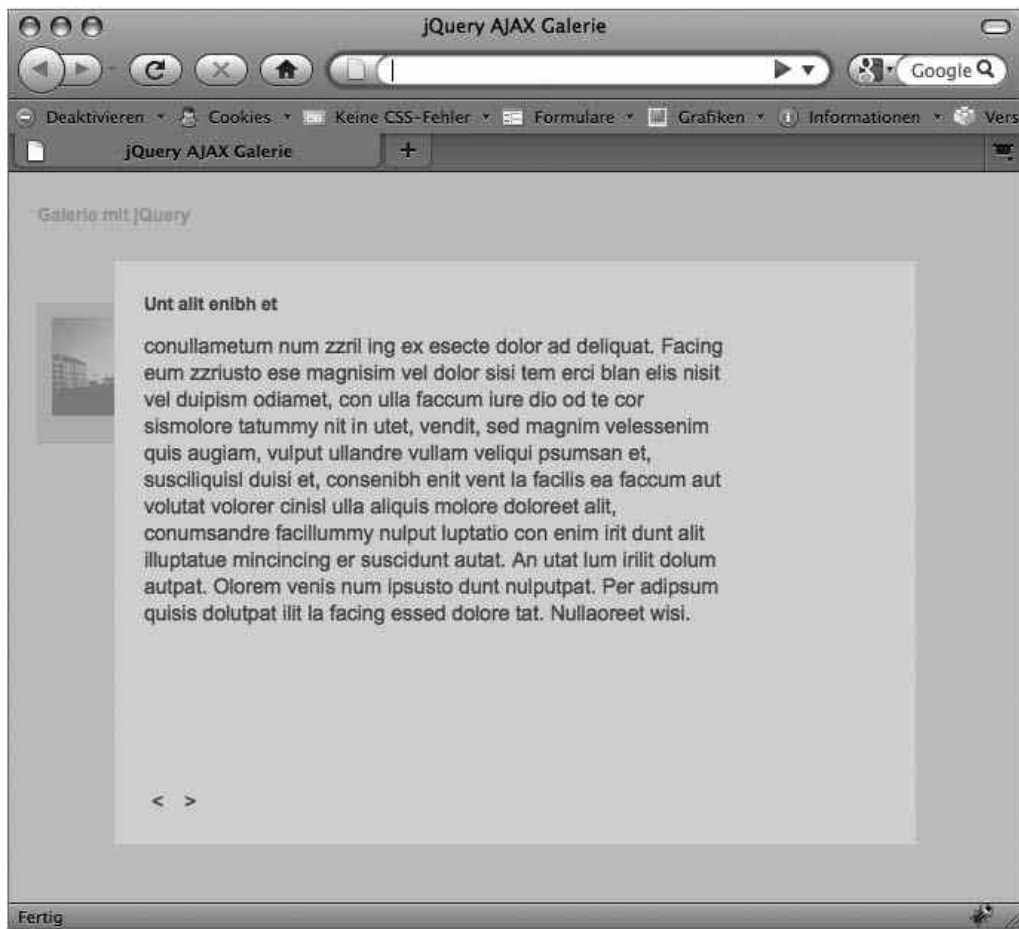


Abbildung 5.20 Laden von Text mit Ajax in die Imagegalerie

Um in der Funktion `loadImg` die hauptsächlichen Änderungen durchzuführen (Sie sollten an dieser Stelle überlegen, sie in `loadAssets` umzubenennen), führen Sie folgenden Code aus:

```
...
loadImg: function(src) {
    var types = ['xml','txt','html','htm'];
    ajax = false;
    var type = src.slice(src.indexOf(".")+1);
    for (i in types) {
        if (type == types[i])ajax = true;
    }
    // wenn es sich um HTML Dateien handelt
    if (ajax) {
        var url = $(this).attr("href");
        objects.imgBox.load(src + " #col-2 > *",function(){
            fn.openImgBox(false);
        });
    }
}
```

```

        return;
    }
    var findImg = objects.imgBox.find("img");
    (findImg.length) ?
    findImg.fadeOut(100,function() {loading()}) :
    loading();
    function loading() {
        var img = new Image();
        img.src = src;
        $(img).imagesLoaded(function() {
            fn.openImgBox($(this));
        },true);
    }
},
...

```

Listing 5.46 JavaScript-Code – Imagebox und Ajax (Teil 2)

Als Erstes definieren Sie die Dateiformate, die von der Applikation erwartet werden, in einem Array. Sie können dieses Array nach eigenen Bedürfnissen erweitern, wenn Sie weitere Formate laden wollen. Mit der JavaScript-Methode `.slice()` extrahieren Sie die Dateiendung. In der `for-in`-Schleife wird abgefragt, ob eine der Endungen in der zu ladenden Datei vorkommt, wenn dies der Fall sein sollte, wird die Variable `ajax` auf `true` gesetzt. In der nun folgenden `if`-Anweisung wird mit der Methode `.load()`, die Sie bereits in den vorigen Beispielen kennengelernt haben, die HTML-Datei geladen. Als Callback-Funktion wird die Funktion `fn.openImgBox(false)` aufgerufen:

```

...
openImgBox: function(obj) {
    if (obj.length) {
        objects.imgBox.html(obj);
        objects.imgBox.find("img").hide();
    }
    objects.overlay
        .fadeTo(defaults.fastSpeed,
            defaults.maxOpacity,function() {
                fn.formatImgBox();
            });
},
...

```

Listing 5.47 JavaScript-Code – Imagebox und Ajax (Teil 3)

Das Argument `false` übergeben wir, um der Funktion mitzuteilen, dass es sich nicht um ein Bild handelt, das wir geladen haben. Innerhalb der Funktion `open-ImgBox()` fragen wir ab, ob der Parameter ein jQuery-Objekt enthält, also ein IMG-Element. Wenn das nicht der Fall ist, wird die Methode `.html()` übersprungen, wir wollen ja nicht, dass die HTML-Datei, die wir mit `.load()` in die `#imgBox` geladen haben, mit etwas anderem überschrieben würde:

```
...
formatImgBox: function() {
    objects.imgContainer.css({
        display:"block"});
    var locHeight = objects.imgBox.find("img").height() ?
        objects.imgBox.find("img").height() :
        332;
    var locWidth = objects.imgBox.find("img").width() ?
        objects.imgBox.find("img").width() :
        500;
    objects.imgBox
        .find("img")
        .css({height:imgHeight,width:imgWidth});
    if (ajax) {
        objects.imgBox.css({height:imgHeight,width:imgWidth});
    }
    imgHeight = locHeight;
    imgWidth = locWidth;
    objects.imgBox
        .find("img")
        .animate({
            opacity:1,
            height:imgHeight,
            width:imgWidth
        },defaults.slowSpeed);
    objects.imgBox
        .animate({
            height:imgHeight,
            width:imgWidth
        },defaults.slowSpeed);
    fn.justifyImgBox(true,defaults.slowSpeed);
},
...
```

Listing 5.48 JavaScript-Code – Imagebox und Ajax (Teil 4)

Sie können mit der Methode `.find("img")` nicht die Dimensionen der HTML-Seite ermitteln, wenn diese Angaben also nicht möglich sind, setzen Sie feste

Werte, in diesem Fall 550 x 332 px. Sie können überlegen, die Dimensionen der HTML-Seite auszulesen, wenn diese vorhanden sind, und die Imagebox dementsprechend automatisch skalieren lassen, wenn es die Anforderungen erforderlich machen. In diesem Beispiel werden Sie das nicht erfahren, es soll sich auf das Wichtigste beschränken. Ein einsamer Winterabend sollte reichen, um diese Erweiterung zu schreiben.

Sie müssen aber überlegen, welche Konsequenzen das haben kann, wenn sich innerhalb einer HTML-Seite mit einer Breite 1.024 px eine per Ajax geladene Seite von ebenfalls 1.024 px öffnet. Ob es nicht besser ist, ein kleines Infofenster wie in unserem Fall mit einer festen Größe zu verwenden? Die `#imgBox` können Sie mit der CSS-Eigenschaft `overflow:auto` versehen. Wenn der Inhalt größer wird als die `#imgBox`, werden automatisch Scrollbalken angezeigt. Zum Abschluss das gesamte Script im Zusammenhang:

```
function lightPopUp(obj) {
    // Wenn kein Vorschaubilder gefunden wurden
    if (!obj.length) return;
    //Default Werte
    var defaults = {
        slowSpeed:400,
        fastSpeed:200,
        fasterSpeed:100,
        maxOpacity:0.8
    }
    // benoetigte jQuery-Objekte
    var objects = {
        imgContainer : $('#imgContainer'),
        imgBox : $('#imgBox'),
        controls : $('#controls'),
        overlay : $('#overlay')
    }
    // Variablen, deren Werte auch nach Ausführung
    // der Funktion justify... erhalten bleiben
    var imgHeight = 1;
    var imgWidth = 1;
    var ajax = false;
    // Variablen, die margin, padding und border speichern
    var diffWidth = objects.imgContainer.outerWidth(true) -
                    objects.imgContainer.width();
    var diffHeight = objects.imgContainer.outerHeight(true) -
                    objects.imgContainer.height();
    // alle Funktionen in einem Objekt
    var fn = {
```

```

//loading laedt Bild und ruft PlugIn imagesLoaded auf
//@ param String Bildquelle
loadImg: function(src) {
    var types = ['xml','txt','html','htm'];
    ajax = false;
    var type = src.slice(src.indexOf(".")+1);
    for (i in types) {
        if (type == types[i]) ajax = true;
    }
    // wenn ajax = true
    if (ajax) {
        var url = $(this).attr("href");
        objects.imgBox.load(src + " #col-2 > *",function(){
            fn.openImgBox(false);
        });
        return;
    }
    var findImg = objects.imgBox.find("img");
    (findImg.length) ?
        findImg.fadeOut(100,function() {loading() }) :
        loading();
    function loading() {
        var img = new Image();
        img.src = src;
        $(img).imagesLoaded(function() {
            fn.openImgBox($(this));
        },true);
    }
},
//openImgBox oeffnet imgBox
openImgBox: function(obj) {
    if (obj) {
        objects.imgBox.html(obj);
        objects.imgBox.find("img").hide(); //css({opacity:0.1});
    }
    objects.overlay.fadeTo(
        defaults.fastSpeed,
        defaults.maxOpacity,function() {
            fn.formatImgBox();
        });
},
// formatImgBox richtet imgBox aus
formatImgBox: function() {
    objects.imgContainer.css({display:"block"});
    var locHeight = objects.imgBox.find("img").height() ?

```

```

        objects.imgBox.find("img").height() : 332;
var locWidth = objects.imgBox.find("img").width() ?
    objects.imgBox.find("img").width() : 500;
objects.imgBox
    .find("img")
    .css({height:imgHeight,width:imgWidth});
if (ajax) {
    objects.imgBox.css({height:imgHeight,width:imgWidth});
}
imgHeight = locHeight;
imgWidth = locWidth;
objects.imgBox
    .find("img").animate({opacity:1,
                        height:imgHeight,
                        width:imgWidth},
                        defaults.slowSpeed);
objects.imgBox.animate({height:imgHeight,
                        width:imgWidth},
                        defaults.slowSpeed);
fn.justifyImgBox(true,defaults.slowSpeed);
},
// justifyImgBox richtet imgBox aus
//@ dist bestimmt, ob Bildgroesse mit berechnet wird
justifyImgBox: function(dist,speed) {
    var include = {height:0,width:0};
    if (dist) {
        include.height = imgHeight + diffHeight;
        include.width = imgWidth + diffWidth;
    }
    var top =
        Math.round(($ (window).height() - include.height) / 2 );
    var left =
        Math.round(($ (window).width() - include.width) / 2 );
    objects.imgContainer
        .animate({top:top,left:left },speed).dequeue();
},
//changeImg aehnlich der Slideshow,
//sucht das naechste/vorherige Bild
// @param direction: Richtung, in die gesucht wird
changeImg: function(direction) {
    var current = obj.filter(".current");
    current.removeClass("current");
    if (direction == "forward") {
        current =
            current.next().length ? current.next() : obj.first();
    }

```

```

    }
    if (direction == "back") {
        current =
            current.prev().length ? current.prev() : obj.last();
    }
    current.addClass("current");
    return current;
}
}
//Eventhandler
//
// Klick auf Thumb
// @param Event
obj.click(function(e) {
    e.preventDefault();
    var imgSrc = $(this).attr("href")
    $(this).addClass("current");
    fn.loadImg(imgSrc);
});
// Klick auf Abdecker
// @param Event
objects.overlay.click(function(e) {
    var obj = $(this);
    objects.imgContainer.fadeOut(defaults.fastSpeed,function() {
        obj.fadeOut(defaults.fastSpeed);
    });
});
//Klick auf grosses Bild, naechstes wird aufgerufen
objects.imgBox.click(function() {
    fn.loadImg(fn.changeImg("forward").attr("href"));
});
//Klick auf Button vorwaerts, naechstes wird aufgerufen
// @param Event
objects.controls.find("a.forward").click(function(e) {
    e.preventDefault();
    fn.loadImg(fn.changeImg("forward").attr("href"));
});
//Klick auf Button zureck, vorheriges wird aufgerufen
// @param Event
objects.controls.find("a.back").click(function(e) {
    e.preventDefault();
    fn.loadImg(fn.changeImg("back").attr("href"));
});
// bei jedem resize wird imgContainer ausgerichtet
$(window).resize(function() {

```

```

        if(!objects.imgContainer.filter(":visible").length) return;
        fn.justifyImgBox(true,defaults.fasterSpeed);
    });
    // justifyImgBox wir beim Start aufgerufen
    fn.justifyImgBox(false,defaults.fastSpeed);
}
// Wenn DOM geladen
$(document).ready(function() {
    // LightPopUp
    // @param jQuery-Objekt mit allen Vorschaubildern
    lightPopUp($(".lightpopup"));
});

```

Listing 5.49 JavaScript-Code – Imagebox und Ajax (das gesamte Script)

5.5.2 Laden von JSON

JSON (JavaScript Object Notation) ist ein Datenaustauschformat, von Mensch und Maschine lesbar, und wird in reiner Textform übermittelt. Im Gegensatz zu XML ist es ein sehr kompaktes Datenformat. Wie der Name bereits vermuten lässt, handelt es sich um valides JavaScript. Es ist nichts anderes als ein JavaScript-Objekt. Wenn keine komplizierte Datenstruktur vorliegt, ist es XML vorzuziehen. Es eignet sich besonders, um die zu übergebenden Daten serverseitig maschinell erzeugen zu lassen, beispielsweise per PHP oder Java. PHP und andere Programmiersprachen bringen von Hause aus Methoden zur Generierung von JSON-Objekten mit, im Falle von PHP sorgt die Methode `json_encode($array)` dafür, dass das JSON-Objekt automatisch erzeugt wird.

Im Folgenden wird ein Beispiel vorgestellt, das Daten per JSON lädt und als HTML-Tabelle darstellt.

Sie werden eine HTML-Seite andeuten, mit der Sie eine Tabelle mit jQuery-Methoden anzeigen lassen können. Mit den Links »Methoden A«, »Methoden ...« werden jeweils einzelne JSON-Dateien geladen und in einen `<div>`-Container mit der Methode `.append()` eingefügt. Man kann alle eingefügten Tabellen mit DATEN LÖSCHEN entfernen und anschließend wieder von vorn anfangen.

Zunächst soll das JSON-Format vorgestellt werden. Das JavaScript-Objekt wird mit den beiden geschweiften Klammern `{}` deklariert. In diese Klammern werden Schlüssel-Wert-Paare gebildet: `{"Schlüssel":"Wert","Schlüssel":"Wert", ...}`. Sie können als Werte auch Arrays einsetzen: `{"Schlüssel":["Wert1","Wert2", ...]}`.

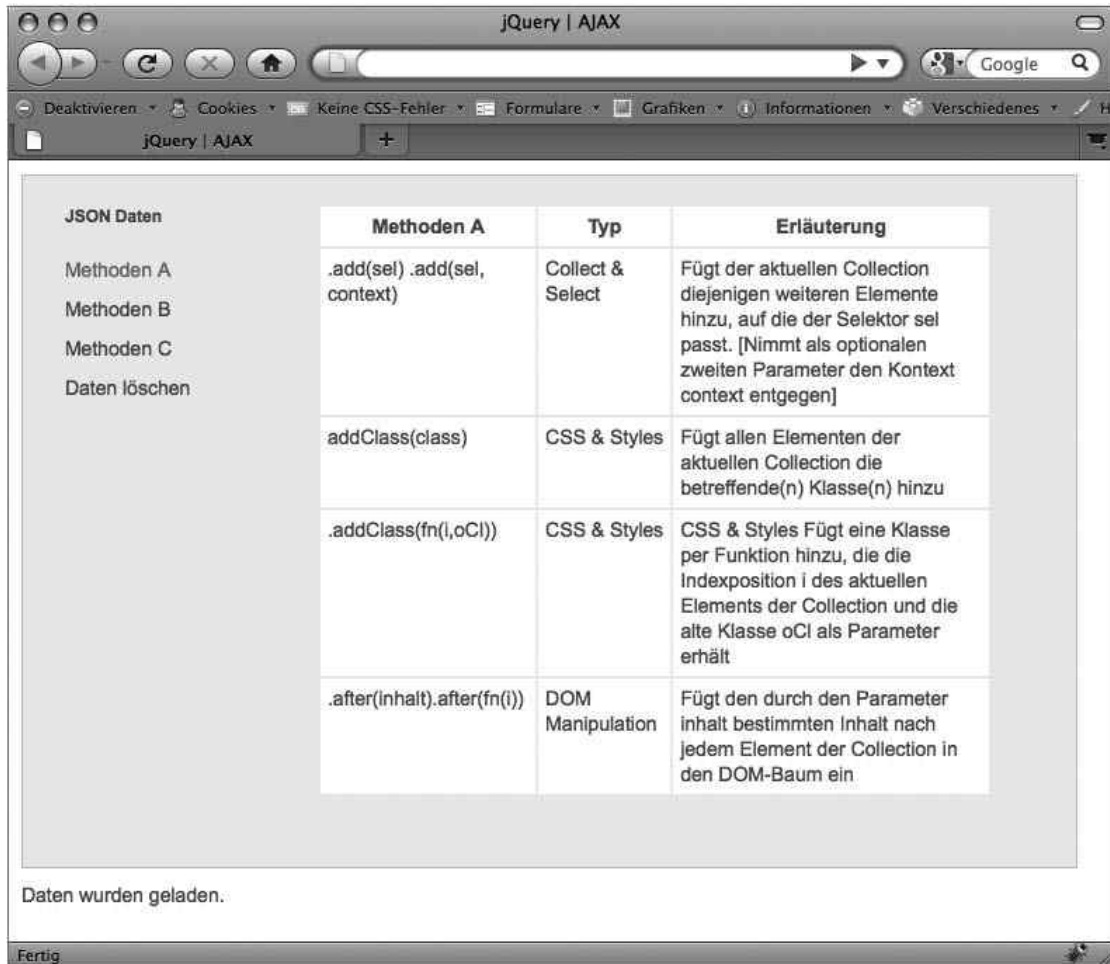


Abbildung 5.21 Laden von JSON-Daten in eine HTML-Tabelle

Wichtig

Seit der Version 1.4 verwendet jQuery dort, wo es in die Browser implementiert ist, die nativen JSON-Parser. Das bedeutet, dass die JSON-Dateien strikt die definierte Notation einhalten müssen. Versuchen Sie beispielsweise, 'Schlüssel': 'Wert'-Paare in einfachen Hochkommata einzuschließen, erhalten Sie eine Fehlermeldung. Sie müssen, um valides JSON zu erhalten, laut Definition doppelte Anführungszeichen verwenden. Mehr Informationen zu JSON erhalten Sie unter <http://www.json.org>. Um Ihren JSON-Notation zu validieren, können Sie Ihre Konstruktion unter <http://www.jsonlint.com> testen.

Neben Strings als Werten können Sie Integer, Boolesche Werte sowie `null` verwenden. Betrachten Sie einen Ausschnitt aus der hier verwendeten JSON-Datei:

```
{
  "header": [
    "Methoden A",
```

```

        "Typ",
        "Erläuterung"
    ]
],
"data": [
    [
        ".add(sel) .add(sel, context)",
        "Collect & Select",
        "Fügt der aktuellen ..."
    ],
    [
        ...
    ]
]
}

```

Listing 5.50 Beispiel JSON-Datei

Nun folgt der obligatorische Blick auf das HTML-Gerüst, das hier wieder verkürzt wiedergegeben wird:

```

<!DOCTYPE html>
<html>
<head>
<title>jQuery | AJAX</title>
<link rel="stylesheet" href="json.css" type="text/css" />
<script type="text/javascript" src="jquery-1.4.2.js">
</script>
<script type="text/javascript" src="json.js">
</script>
</head>
<body>
<div id="container">
<div id="col-1">
<ul>
<li><a class="load" href="data-1.json">Methoden A</a></li>
...
<a class="reset" href="data-3.json">Daten löschen</a></li>
</ul>
</div>
<div id="col-2"></div>
</div>
<div id="msg"></div>
</body>
</html>

```

Listing 5.51 HTML-Gerüst – Laden von JSON-Dateien

Sie versehen die `<a>`-Elemente, mit denen Sie JSON-Daten laden wollen, mit der CSS-Klasse `load`. Den Löschlink versehen Sie mit der Klasse `reset`. Über die Klassen wird später selektiert und der Event Handler an das entsprechende `<a>`-Element gebunden. Die geladenen Daten werden in eine HTML-Tabelle verpackt und in den Container `#col-2` eingesetzt.

Das Styling der Elemente wird an dieser Stelle vernachlässigt, betrachten Sie daher nun die Scriptdatei:

```
$(document).ready(function() {
    var $objects = {
        load: $("ul a.load"),
        reset: $("ul a.reset"),
        target: $("#col-2"),
        msg: $("#msg")
    }
    $objects.load.click(function(e) {
        e.preventDefault();
        if (!$$(this).not(".current").length) return;
        var lastTable = $objects.target.find("table").last();
        var url = $(this).attr("href");
        $(this).addClass("current");
        loadTableData($objects,url,function(){
            (lastTable.next().length) ?
                lastTable.next().hide().fadeIn("slow") :
                $objects.target.find("table").hide().fadeIn("slow");
        });
    });
    $objects.reset.click(function(e) {
        e.preventDefault();
        $objects.load.removeClass("current");
        $objects.msg.empty();
        $objects.target.empty();
    });
});
...
```

Listing 5.52 JavaScript-Code – Laden von JSON-Dateien (Teil 1)

Bis hierher sehen Sie die Event Handler. Der für dieses Script wichtigste Handler ist der `.click()`-Handler, der an alle `<a>`-Elemente gebunden ist, die die Klasse `load` besitzen: Im Objekt `$objects.load` sind sie alle über `$("ul a.load")` selektiert.

In diesem Handler wird im Wesentlichen die Funktion `loadTableData()` mit einem Callback aufgerufen. In ihr werden die JSON-Daten geladen, anschließend wird die neu erzeugte HTML-Tabelle mit `.fadeIn()` eingeblendet:

```
...
function loadTableData(obj,url,callback) {
    var Data = {
        dataToTable: function(obj, url) {
            var table = $('<table/>');
            $.getJSON( url, function( jsonObj ){
                if (jsonObj.header) {
                    $.each( jsonObj.header, function(i, data) {
                        Data.dataToRow(data,table,true);
                    });
                }
                if (jsonObj.data) {
                    $.each( jsonObj.data, function(i, data) {
                        Data.dataToRow( data, table,false);
                    });
                }
                obj.target.append( table );
                if($.isFunction(callback)){
                    callback.call(this);
                }
            });
        }
    };
    ...
}
```

Listing 5.53 JavaScript-Code – Laden von JSON-Dateien (Teil 2)

Hier werden die JSON-Dateien geladen. In der Anweisung `if (jsonObj.header)` wird abgefragt, ob ein Header-Objekt vorhanden ist. Wenn dies der Fall ist, soll ein HTML-Table-Header mit `<th>`-Elementen erzeugt werden. Diese Zeile wird die Tabellenüberschrift bilden. Innerhalb der Anweisung `if (jsonObj.data)` werden die eigentlichen Tabellendaten in eine HTML-Tabelle gepackt. Es wird jeweils die Funktion `Data.dataToRow()` aufgerufen. Diese Funktion generiert jeweils eine Tabellenzeile. Sind alle Daten durchlaufen und sämtliche Tabellenzeilen erzeugt, wird die gesamte Tabelle mit `.append()` an das Zielobjekt, hier den Container `#col-2`, angehängt. Mit der Callback-Funktion werden die erzeugten Tabelle eingeblendet:

```
...
obj.msg.ajaxError(function(e, xhr, settings, exception) {
    $(this).text("Fehler: " + exception);
});
```

```

    obj.msg.ajaxSuccess(function() {
        $(this).text('Daten wurden geladen.');
```

```
    });
},
dataToRow: function(data, table,isHeader) {
    var tr = $("<tr/>");
    var cell = isHeader ? "<th/>" : "<td/>";
    $.each(data, function(i) {
        var td = $(cell);
        td.append(data[i] );
        tr.append(td);
    });
    table.append(tr);
}
}
Data.dataToTable(obj,url);
}
```

Listing 5.54 JavaScript-Code – Laden von JSON-Dateien (Teil 3)

Die beiden Methoden `.ajaxError()` und `.ajaxSuccess()` sind optional, sie geben den jeweiligen Status wieder. Wichtiger ist die Funktion `dataToRow()`, in ihr werden die einzelnen Tabellenzeilen erzeugt.

Sie konnten sehen, dass in diesem Beispiel »pure« Daten geladen und in einer anderen Form wieder zusammengesetzt wurden. Sie sollten JSON-Daten maschinell erzeugen, beim manuellen Schreiben solcher Notationen sind die Fehlerquellen doch sehr erheblich. Mächtig ist dieses Format dann, wenn Sie eine große Menge Daten laden wollen. Mittels einer Logik müssen Sie anschließend die reinen Daten in eine Form bringen und in HTML-Elemente verpacken.

Cross-Domain-Bremse für JSON

Es gibt allerdings eine Einschränkung: Sie können nicht ohne Weiteres JSON-Daten über Domaingrenzen hinweg laden, Sie müssen sie über dieselbe Domain laden, über die auch die HTML-Seite geladen wurde. Aus Sicherheitsgründen wurde das XHR-Objekt so ausgelegt, dass das nicht möglich ist.

5.5.3 Laden von JSONP

Es gibt dennoch einen Weg, diese Grenze zu überschreiten. Textdateien, XML-Dateien oder HTML-Dateien dürfen zwar nicht über eine andere Domain geladen werden, JavaScript-Dateien hingegen schon.

Diesen Umstand nutzt die JSON-Erweiterung JSONP aus. Man kann sie auch als *Remote JSON* bezeichnen. Das P steht dabei für »Padding«. Wobei hier, im Gegensatz zur CSS-Eigenschaft »padding« (Innenabstand), eine »Umhüllung« oder »Einbettung« der JSON-Daten in eine Callback-Funktion gemeint ist.

Es wird also das JSON-Objekt in einen Funktionsaufruf eingebettet. Man kann auf diese Weise Daten mit einem Cross-Domain-Request laden – wie gesagt, bei Scriptdateien besteht die Sicherheitsbeschränkung nicht. Im Folgenden wird kurz ein JSONP-Beispiel angerissen.

Es soll eine JSONP-Datei von einer Domain geladen werden, die außerhalb der Domain liegt, von der die führende HTML-Seite geladen wurde. Die JSONP-Datei wird ein kleines PHP-Script sein, das den Parameter empfängt und als Funktionsaufruf zurückgibt. Sie benötigen also für dieses Beispiel Grundkenntnisse in PHP sowie eine Serverumgebung, die PHP ausführen kann. Sie können wie in Kapitel 1, »jQuery kennenlernen«, beschrieben, eine XAMPP- oder MAMP-Installation benutzen und über *localhost* auf die *jsonp.php*-Datei zuzugreifen.

Informationen zu PHP

Weitere Informationen zu PHP finden Sie auf dem englischsprachigen Portal <http://www.php.net/>, und die deutschsprachige Dokumentation unter <http://www.php.net/manual/de/> hält ebenfalls interessante Hinweise für Sie bereit.

```
...
<script type="text/javascript" src="jquery-1.4.2.js">
</script>
<script type="text/javascript">
$(document).ready(function() {
    $.ajax({
        dataType: 'jsonp',
        jsonp: 'overwriteCallback',
        url: 'http://www.domain.de/jsonp.php',
        success: function (data) {
            $("#msg").append(data.Kiwis);
        },
    });
});
</script>
...
```

Listing 5.55 Laden von JSONP-Daten (Teil 1)

Dazu verwenden Sie die Low-level-Methode `$.ajax()`. Als Datentyp wird `jsonp` eingesetzt, als `jsonp`-Parameter setzen Sie einen eigenen Begriff `overwriteCallback` ein und überschreiben den standardmäßig gesetzten Parameter `callback`. Sie könnten ebenso `.getJSON()` verwenden:

```
...
var url = 'http://www.domain.de/jsonp.php';
$.getJSON( url, 'callback=?', function( data ){
    $("#msg").append(data.Kiwis);
});
```

Listing 5.56 Laden von JSONP-Daten (Teil 2)

Der Parameter wird einfach als String als `data`-Argument übergeben. (Allerdings scheint es so, dass dieser Parameter nicht wie vorgesehen überschrieben wird. Sie müssen also wohl stets `callback=?` verwenden.)

Serverseitig wird ein PHP-Script verwendet, das das JSONP-Objekt erzeugt, das den Parameter als Funktionsaufruf zurückgibt. Zuerst wird eine Routine entwickelt, die die Daten enthält. In einer praktischen Anwendung würde hier sicherlich eine Datenbank als Datenquelle verwendet, in diesem Fall begnügen wir uns mit einem einfachen Array:

```
<?php
    $namen = array(
        'Birnen'    => 10,
        'Bananen'   => 20,
        'Kiwis'     => 30
    );
    $data = json_encode($namen);
    echo    $_GET['callback']
           . '(' . $data . ');';
    /* output: callback({
                                   "Birnen":10,
                                   "Bananen":20,
                                   "Kiwis":30
                               });
    */
?>
```

Listing 5.57 PHP-Code – Laden von JSONP-Daten

Der PHP-Kommentar gibt den Output wieder, den Sie erhalten, wenn das Script ausgeführt wird. Hier wird deutlich, dass die JSON-Daten in einen JavaScript-Funktionsaufruf gepackt werden.

5.5.4 Zusammenfassung

Mit Ajax können Sie nicht nur XML-Daten laden, sondern auch andere Datentypen wie JSON oder zusätzlichen JavaScript-Code. jQuery stellt den vollen Ajax-Funktionsumfang zur Verfügung. Mit den hier vorgestellten Funktionen und Methoden können Sie auf einfache Weise Daten nachladen, ohne die zuerst geladene HTML-Seite neu aufrufen zu müssen. Aus Sicherheitsgründen ist es in den Browsern implementiert, dass Sie Inhalte ausschließlich über die gleiche Domain wie die führende HTML-Seite laden können. Wenn Sie diese Restriktion umgehen müssen, verwenden Sie JSONP. Ajax bietet umfangreiche Möglichkeiten, innovative Webanwendungen zu entwickeln, die früher nur mithilfe von Browser-Plugins wie Adobe Flash umsetzbar waren.

5.6 Plugin-Entwicklung

Sie werden in diesem Praxisteil immer wieder Plugins einsetzen oder aus den besprochenen Beispielen Plugins erstellen wollen. Wie Sie Plugins einbinden, haben Sie bereits erfahren. Wie Sie selbst und mit welchem Muster Sie Plugins bereitstellen, erfahren Sie hier.

5.6.1 Eigene Plugins entwickeln

Das jQuery-Objekt kann um selbst definierte Methoden und Eigenschaften erweitert werden. Warum aber sollten Sie das tun? Sie könnten doch auch entsprechende Funktionen schreiben und diese Funktionen ausführen lassen, wie es bislang oft geschehen ist? Wollen Sie aber jQuery um häufig benötigte Aufgaben erweitern, ohne immer wieder in den Code hineinsehen zu müssen, sollten Sie für Ihren Zweck ein Plugin entwickeln.

- Die Wiederverwendbarkeit und ein modularer Aufbau sind der Garant dafür, um mit der Zeit ein umfangreiches Archiv mit verschiedensten Erweiterungen aufbauen zu können, das die Entwicklung von Websites enorm erleichtert. Sie sollten gleichfalls die Community unterstützen und Ihre selbst geschriebenen Plugins im jQuery Repository anbieten. Dieses Archiv erleichtert allen Webentwicklern das Leben, indem Lösungen zu stetig wiederkehrenden Aufgaben nicht immer neu erfunden werden müssen, sondern auf vorhandene Ressourcen zurückgegriffen werden kann.
- Man kapselt die einzelnen Anweisungen und kann dann über Methoden und Eigenschaften der Erweiterung zugreifen und muss sich nicht mehr mit den einzelnen Anweisungen herumschlagen.

- Jede Plugin-Methode sollte immer auch das jQuery-Objekt selbst zurückgeben, sodass eigene Methoden und fremde Methoden nahtlos aneinandergehängt werden können (Chaining). Die eigenen Methoden werden so ein erweiterter Bestandteil des Frameworks.
- Plugins sollten so weit parametrisiert werden, dass die Erweiterung möglichst universell einsetzbar ist.

Im Folgenden werden Sie ein Plugin entwickeln, das Elemente in einem HTML-Baum ersetzt. Angenommen, Sie haben eine unsortierte Liste `<ul>`- mit `<li>`-Listenelementen. Und nehmen wir einmal an, aus irgendeinem Grund wollen Sie diese Elemente in `<div>`- und `<p>`-Elemente umwandeln.

Aus der Liste

```
<ul>
  <li><a href="#">Item 1</a></li>
  ...
</ul>
```

wird folgende Verschachtelung

```
<div>
  <p><a href="#">Item 1</a></p>
  ...
</div>
```

Als Erstes müssen Sie eine neue JavaScript-Datei anlegen. Sie sollte genauso heißen, wie die Methode, die Sie im nächsten Schritt definieren werden. Nach folgendem Muster wird die Datei abgespeichert:

```
jquery.namedermethode[-Version].js
```

In unserem Beispiel sieht das Verlinken des neuen Plugins folgendermaßen aus:

```
<script type="text/javascript"
      src="jquery-1.4.2.js"></script>
<script type="text/javascript"
      src="jquery.replaceelements-1.0.js">
</script>
```

Der Kern eines Plugins ist die Methode, um die das jQuery-Objekt erweitert wird:

```
$.fn.replaceElements = function() {
  // Anweisungen
};
```

Der Webdeveloper, der das Plugin nutzt, wird das Plugin mit folgendem Aufruf einsetzen:

```
$(document).ready(function(){
    $('#menu').replaceAll();
});
```

Sie müssen daran denken, dass das jQuery-Objekt n Elemente enthalten kann: keines, eines oder mehrere. Sie können nicht erwarten, dass es immer nur ein Element zurückgibt; es besteht durchaus die Möglichkeit, dass Sie es mit einer größeren Menge an Elementen zu tun haben. Deshalb wird als Erstes das jQuery-Objekt selbst mit `.each()` iteriert:

```
$.fn.replaceAll = function() {
    return this.each(function(){
        //Anweisungen
    });
};
```

Als Rückgabewert der Iteration werden Sie wiederum ein jQuery-Objekt mit `return this.each()` erhalten. `this` ist das an das Plugin übergebene jQuery-Objekt. Damit stellen Sie sicher, dass Sie weitere Methoden an unsere Methode anfügen können und unser Plugin nahtlos in das Framework integriert wird.

Im nächsten Schritt werden die standardmäßigen Optionen, die `defaults`, bestimmt. Das Plugin wird dadurch auch funktionieren, wenn keine Optionen nach dem Muster

```
$('#menu').replaceAll('{li:span}');
```

übergeben wurden. Sie legen dazu die standardmäßigen Werte im Plugin selbst fest:

```
var defaults = {
    ul: 'div',
    li: 'p'
};
```

Das Plugin selbst verändern Sie, indem Sie das Argument `arg` angeben und im weiteren Verlauf die Variable `options`:

```
$.fn.replaceAll = function(arg) {
    var defaults = {
        ul: 'div',
        li: 'p'
    };
    var options = $.extend(defaults, arg);
    return this.each(function(){
        //Anweisungen z.B.:
        $("body").prepend("<p>options: " + options.ul + "</p>");
    });
};
```

```
});
}
```

In diesem Beispiel wird unsere Variable `defaults` mit den Werten des Arguments `arg` überschrieben, der Aufruf sieht so aus:

```
$(document).ready(function(){
    $("#menu").replaceElements({ul:"span"});
});
```

Jetzt können Sie die Variable `defaults` noch etwas verbessern, Sie werden aus der Variablen eine Eigenschaft von `replaceElements` machen:

```
$.fn.replaceElements.defaults = {
    ul: 'div',
    li: 'p'
};
```

Diese Eigenschaft ist öffentlich, »public«, dadurch kann man von außerhalb der Funktion `replaceElements` aufrufen. Der Aufruf von außen würde dann folgendermaßen aussehen:

```
$(document).ready(function(){
    ...
    $.fn.replaceElements.defaults.ul = 'span';
    ...
});
```

Außerdem müssen Sie noch die Argumente der Methode `$.extend()` ändern:

```
var options = $.extend({}, $.fn.replaceElements.defaults, arg);
```

Was auffällt, ist, dass Sie nicht nur die Eigenschaft `$.fn.replaceElements.defaults` verwenden können, Sie übergeben als ersten Parameter ein leeres Objekt `{}`. Täten Sie das nicht, würde bei einem neuerlichen Aufruf der Methode `.replaceElements()` der veränderte Wert ausgegeben werden. Hier ein Beispiel:

```
$(document).ready(function(){
    $("#menu")
        .replaceElements({ul:"span"})
        .replaceElements();
});
```

Sie sollen ja beliebige Methoden miteinander verketteten können, auch zwei oder mehrmals dieselbe Methode. Das Ergebnis sähe bei

```
var options = $.extend($.fn.replaceElements.defaults, arg);
```


folgendermaßen aus – betrachten Sie den generierten Code:

```
...
<body>
  <p>options: span</p>
  <p>options: span</p>
...
```

Im Falle von

```
var options = $.extend({}, $.fn.replaceElements.defaults, arg);
```

würde der generierte Code hingegen so aussehen:

```
...
<body>
  <p>options: div</p>
  <p>options: span</p>
...
</body>
...
```

Tipp:

Übergeben Sie `$.extend()` als erstes Argument ein leeres Objekt `{}`, bleibt der Defaultwert erhalten. Wird ein Objekt beim ersten Aufruf überschrieben, wird bei einem neuerlichen Aufruf ohne Optionen der Defaultwert verwendet.

Es kann sehr hilfreich sein, eine Funktion vor der Außenwelt zu schützen. Sie dient ausschließlich der internen Verarbeitung, um unser Plugin zu strukturieren, Redundanzen zu vermeiden oder um eine Funktion rekursiv aufzurufen. Innerhalb des Namensraums `replaceElements` platzieren Sie einfach eine Funktion:

```
$.fn.replaceElements = function() {
  ...
  function private() {
    // Amweisung
  };
};
```

Zu guter Letzt werden Sie die Plugin-Methode noch in eine Closure verpacken, um Kollisionen mit anderen Methoden oder anderen Frameworks zu vermeiden:

```
;(function($) {
  $.fn.replaceElements = function() {
    return this.each(function(){
      //Anweisungen
    });
  });
```

```
};
})(jQuery);
```

Sie umschließen die Methode mit einer anonymen Funktion. Mit ihr übergeben Sie eine Variable, in dem Fall \$. Eigentlich sollten Sie bei der Plugin-Entwicklung auf die Variable \$ verzichten, da andere Frameworks diese ebenso verwenden und es daher zu Konflikten kommen könnte. Aber durch die Closure stellen Sie sicher, dass diese Variable außerhalb des Scopes der anonymen Funktion nicht sichtbar ist. Wenn Sie mit dem Konzept der Closures nicht vertraut sind, werfen Sie einen Blick in Kapitel 1, »jQuery kennenlernen«, dort wird das Thema ange-rissen. Nun aber zurück zu dem Beispiel.

Hinweis

Ihnen ist vielleicht das *Semikolon* am Anfang des Scripts aufgefallen. Dieser Trick (eine leere Anweisung) kann verwendet werden, um Scriptfehler, die außerhalb unseres Plugins entstanden sind, an dieser Stelle abubrechen. Wenn Sie ein Plugin als »minified«-Version ausliefern wollen, ist dieses Verfahren sogar unerlässlich, um JavaScript-Fehler-meldungen zu vermeiden.

Sie können auf diese Weise problemlos die Variable \$ für Plugin-Entwicklung verwenden und so diese Sprachkonventionen übernehmen. In der letzten Zeile, unmittelbar nach der anonymen Funktion, wird selbige umgehend mit dem Ausdruck (jQuery) aufgerufen. Auf diese Weise erreichen Sie einen gut und sauber strukturierten Aufbau eines Plugins. Sie sollten sich hier und zukünftig an dieses Entwicklungsmuster halten, um effiziente konfliktfreie Erweiterungen für jQuery zu schreiben.

Minified jQuery

jQuery wird seit der Version 1.4 mit dem Tool Google Closure Compiler optimiert. Diese Java-Applikation löscht beispielsweise Leerzeichen, Zeilenumbrüche und verkürzt Variablennamen aus JavaScript-Dateien, um so die Ladezeiten von JavaScript-Dateien zu optimieren. Es wird der komplette Quelltext analysiert und umgeschrieben. Für Webentwickler ist der generierte Code nicht mehr nachzuvollziehen. Der Output sind weiterhin Textdateien, aber auf ein Höchstmaß durch viele Algorithmen optimiert. Das jQuery Framework (1.4.2) in lesbarer Form besitzt eine Größe von 164 KB, die durch Google Closure Compiler optimierte »minified«-Version ist nur 72 KB groß.

Der Google Closure Compiler ist eine Java-Anwendung, die über die Kommandozeile ausgeführt wird. Dadurch dass es sich um ein Java-Programm handelt, kann es auf Linux, Mac OS X und Windows ausgeführt werden.

Sie finden das Programm unter <http://code.google.com/intl/de-DE/closure/compiler/>. Dort gibt es auch eine Dokumentation. Wenn Sie das Plugin in diesem Kapitel verkleinern wollen, sieht der Kommandozeilenaufwurf folgendermaßen aus:

```
java -jar compiler.jar --js jquery.replace-1.0.js --js_output_file
jquery.replace-1.0.min.js
```

Der Nutzen bei einem solch kleinen Plugin ist das Verhältnis 2.825 Byte zu 917 Byte zugunsten der verkleinerten Version.

Das Kommandozeilenfenster, um das Kommando einzugeben, finden Sie unter Windows, indem Sie im Startmenü auf den Befehl AUSFÜHREN gehen und dann »cmd« eingeben.

Unter Mac OS X finden Sie das sogenannte *Terminalprogramm* unter PROGRAMME/DIENSTPROGRAMME.

Linux-User wissen, was zu tun ist, sie kennen nichts anderes als die Shell.

Im Terminalfenster müssen Sie sich mit dem Befehl `cd` (change directory) bis zum Verzeichnis, in dem Sie `compiler.jar`, sowie die zu verkleinernde Datei abgelegt haben, durchhangeln, und den soeben zitierten Befehl ausführen.

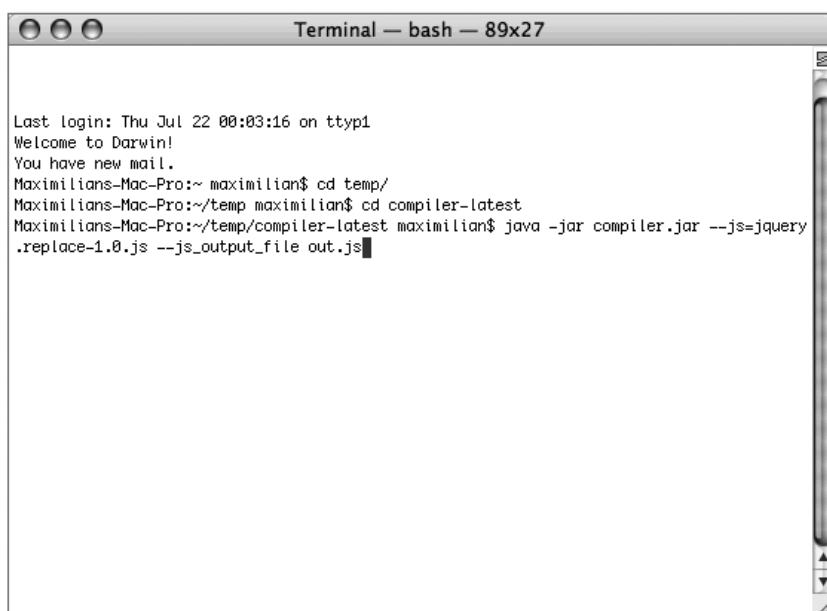


Abbildung 5.22 Erzeugen einer verkleinerten Version eines Plugins

Noch sind wir nicht ganz fertig mit unserer Aufgabe, ein Plugin zu schreiben, das `<ul>`- und `<li>`-Elemente ersetzt. Nachdem Sie nun grundsätzliche Bausteine der Plugin-Entwicklung kennengelernt haben, betrachten Sie abschließend das komplettierte Beispiel:

```
;(function($) {
  jQuery.fn.replaceElements = function(arg,callback) {
    var options = $.extend({},
      $.fn.replaceElements.defaults,
      arg,
      {callback:callback});
    return this.each(function() {
```

```

        replace(options,$(this).parent());
    });
};
function replace(opts,obj) {
    for (var i in opts) {
        $(obj).find(i).each(function() {
            var thisElement = $(this);
            thisElement.wrapInner("<"+ opts[i] +" />");
            if(thisElement.attr("id")) {
                thisElement.children()
                    .attr("id",thisElement.attr("id"));
            }
            thisElement.children().unwrap();
        });
    }
    callFn(opts.callback);
};
function callFn(callback) {
    if($.isFunction(callback)){
        callback.call(this);
    }
};
$.fn.replaceElements.defaults = {
    ul: 'div',
    li: 'p'
};
})(jQuery);

```

Listing 5.58 JavaScript-Code – ReplaceElements Plugin

Das eigentlich Neue findet in dem fett hervorgehobenen Abschnitt statt. Innerhalb der Methode `.each()` wird lediglich die private Funktion `replace()` aufgerufen, in der die Rechenoperationen erfolgen.

Die Idee ist, dass zuerst mit `.wrapInner()` innerhalb der im HTML-Baum vorhandenen Elemente das neue Element hinzugefügt wird und anschließend das neue Element mit `.children()` selektiert sowie das alte, nun äußere Element mit `.unwrap()` entfernt wird. Mit `.each()` werden alle Elemente innerhalb der Selektion durchforstet und nehmen die Ersetzung vor.

Nun gibt es noch Attribute. Die werden bei dieser Ersetzungsoperation automatisch entfernt. Es muss ein Weg gefunden werden, sie zu retten und wieder in die neu hinzugefügten Elemente einzusetzen. Um es einfach zu halten, werden in diesem Beispiel lediglich die Attribute `id` gerettet. Mit der Anweisung

```
thisElement.children().attr("id",thisElement.attr("id"))
```

werden sie übernommen. Für unsere Zwecke ist das vollkommen ausreichend.

Tipp

An dieser Stelle könnten Sie das Plugin aufbohren und sämtliche Attribute finden und ersetzen. Hierzu gibt es bereits ein fertiges Plugin im offiziellen jQuery-Verzeichnis, das Sie hier mit verwenden könnten, um ein möglichst generisches Plugin zu schreiben:

<http://plugins.jquery.com/project/getAttributes>

Laden Sie das Plugin herunter und binden es in die HTML-Seite ein:

```
<script type="text/javascript" src="jquery-1.4.2.js"></script>
<script type="text/javascript" src="jquery.getAttributes.js">
</script>
<script type="text/javascript" src="jquery.replace-1.0.js">
</script>
```

Das zusätzliche Plugin würden Sie folgendermaßen benutzen, statt

```
...
    if(thisElement.attr("id")) {
        thisElement.children().attr("id",thisElement.attr("id"));
    }
...
notieren Sie
```

```
...
    var attr = jQuery.getAttributes(thisElement,true);
    if(attr) {
        thisElement.children().attr(attr);
    }
...

```

Damit würden zuerst alle Attribute, unabhängig davon, um welche es sich handelt, in das neue Element übernommen werden. Hier sehen Sie einen wichtigen Vorteil von Plugins in der Praxis: Sie können das Plugin blind verwenden und sich (in den meisten Fällen) darauf verlassen, dass es das tut, was von ihm verlangt wird. Sie müssen sich nicht mit seinem Innenleben auseinandersetzen, sondern nur damit, die richtigen Optionen zu setzen.

Zu guter Letzt betrachten wir noch das Argument `callback`. Sie wollen dieses Plugin so einsetzen, dass direkt nach dem Laden einer Seite die Ersetzung stattfindet. Anschließend soll irgendetwas ausgelöst werden, beispielsweise Events an die neu erzeugten Elemente gebunden werden. Das können wir jedoch erst vollziehen, wenn die Ersetzung vorgenommen wurde. Dazu lassen wir eine Funktion erst dann starten, sobald die Ersetzung abgeschlossen ist.

Mit folgendem Aufruf

```
$(document).ready(function(){
    $("#menu").replaceAll({},function() {
```

```

        aFunction($("#menu"));
    });
});

```

übergeben wir als Callback eine anonyme Funktion und rufen darin eine beliebige Funktion auf.

Innerhalb des Plugins wird dieses Argument an die Funktion `replace` weitergereicht. Nachdem alle Objekte innerhalb der `for-in`-Schleife abgearbeitet wurden, erfolgt der Aufruf an die Funktion

```
callFn(opts.callback);
```

Innerhalb der Funktion stellen Sie sicher, dass es sich bei dem Argument um eine Funktion handelt, wenn ja, wird die Callback-Funktion ausgeführt:

```

function callFn(callback) {
    if($.isFunction(callback)) {
        callback.call(this);
    }
};

```

Dieses hier vorgestellte Plugin verwenden Sie in Abschnitt 5.1, »Schönere Navigationen«, dort müssen Sie in der Tat `<ul>`- und `<li>`-Elemente ersetzen, damit das HTML-Gerüst valide, semantisch korrekt und unobtrusive bleibt.

Einen kleinen Nachteil hat dieses Plugin: Wenn die Zahl der zu ersetzenden Elemente größer wird, ist die Ausführungszeit recht hoch. Das Script verlangt Rechenleistung. Die Prozessorauslastung sollten Sie daher im Auge behalten, sofern Sie das Plugin für einen weit verzweigten und elementreichen HTML-Baum einsetzen.

5.6.2 Allgemeines Muster eines jQuery-Plugins

Als Resümee werden im Folgenden die Syntaxteile zusammengefasst, die das Grundgerüst eines jQuery-Plugins formen. Dieses Grundgerüst können Sie für ziemlich jede Ihrer Plugin-Entwicklungen einsetzen:

```

;(function($) {
    jQuery.fn.nameOfPlugIn = function(arg) {
        var options = $.extend({}, $.fn.nameOfPlugIn.defaults, arg);
        return this.each(function() {
            //Anweisungen hier z.B.:
            $(this).html(options.nameN);
            private();
        });
    };
});

```

```

    };
    function private() {
        // Anweisungen hier
    };
    $.fn.nameOfPlugIn.defaults = {
        name1: 'value1',
        name2: 'value2',
        nameN: 'valueN'
    };
})(jQuery);

```

Listing 5.59 JavaScript-Code – allgemeines Muster eines jQuery-Plugins

Allgemeine Regeln zum Erstellen eines Plugins

- ▶ Zwar sollte man nie das Dollarzeichen als Namespace für Plugins verwenden, wenn Sie aber über eine anonyme Funktion eine Closure konstruieren, können Sie das \$ als Shortcut für das jQuery-Objekt der inneren Methoden und Eigenschaften übergeben.
- ▶ Die Datei sollte einem festen Namensschema folgen: *jquery.namedesplugins-1.0.js*. Der Name entspricht dem Namensraum der Methode. Auch die öffentlichen Eigenschaften sollten dem entsprechen.
- ▶ Verwenden Sie Optionen, damit der Nutzer des Plugins das Verhalten ändern kann.
- ▶ Definieren Sie ein Defaultobjekt für die Konfiguration, in dem Verhalten vordefiniert werden kann, über Optionen kann es überschrieben werden.
- ▶ Alle Methoden und Funktionen sollten mit einem Semikolon (;) beendet werden. Bei der Komprimierung des Codes könnten sonst Laufzeitfehler auftreten. Außerdem können Sie am Anfang des Plugins ein Semikolon setzen, um auf alle Fälle an dieser Stelle fehlerhafte Scriptanweisungen zu beenden.
- ▶ Das Plugin sollte stets ein jQuery-Objekt zurückgeben.
- ▶ Es sollte die `each()`-Methode verwendet werden, um sicherzustellen, dass alle selektierten Elemente durchlaufen werden.
- ▶ Alle Methoden, die Sie neu schreiben, werden dem `jQuery.fn`-Objekt hinzugefügt, und alle Funktionen dem jQuery-Objekt.
- ▶ Innerhalb von Methoden sollte `this` als Referenz auf das aktuelle jQuery-Objekt verwendet werden.

5.7 jQuery UI

Das jQuery UI steht für jQuery User Interface. Es ist eine Erweiterung des jQuery Frameworks, das einerseits erweiterte Interaktion wie Effekte, Drag & Drop, Easing und Farbanimationen bereitstellt, andererseits komplexe Oberflächen-Controls (Widgets) wie DatePicker, Navigationen und Dialogboxen zur Auswahl bietet.

jQuery UI umfasst eine Sammlung von leicht individualisierbarer, einfach zu konfigurierender Plugins. Im Einzelnen unterscheidet man folgende Bereiche.

Interaktion	Drag & Drop, Größenänderungen, Auswahlen, Sortierungen
Widgets	Akkordeon, Autovervollständigung, Button, Datepicker, Dialog, Fortschrittsbalken, Schieberegler, Tabs
Utilities	Positionierung
Effekte	Die Methoden <code>effect()</code> , Farbanimationen mit <code>animate()</code> , <code>switchClass()</code> , Erweiterungen für Methoden <code>show()</code> , <code>hide()</code> , <code>toggle()</code> , <code>addClass()</code> , <code>removeClass()</code> , <code>toggleClass()</code> .

Tabelle 5.1 jQuery UI – Übersicht

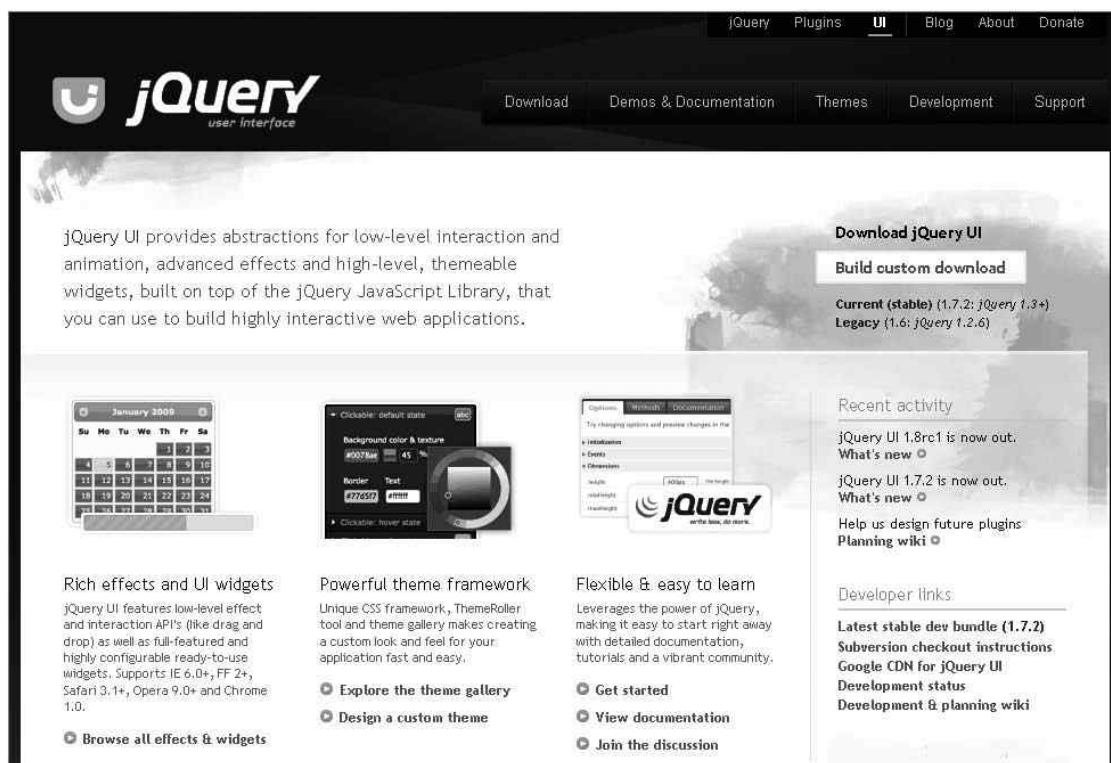


Abbildung 5.23 Website von jQuery UI

Download von jQuery UI

Sie können die aktuelle Version des Pakets jQuery UI als Zip-Archiv unter dieser Adresse downloaden:

► <http://jqueryui.com/download/>

Sie finden es in der Version 1.8.0 ebenfalls auf der Begleit-DVD zu diesem Buch.

Sie können in Form einer einzigen JavaScript-Datei alle UI-Funktionen »en bloc« einbinden, wenn Sie dies wünschen. Allerdings handelt es sich hier um eine größere Datei, sodass Sie dies eher für Testzwecke in Betracht ziehen sollten als für eine Produktionsumgebung, bei der es auf Performance ankommt. Hierfür bietet das jQuery UI die Option, gezielt ein oder mehrere Pakete zusammenzustellen, das nur aus den wirklich benötigten UI-Elementen besteht. Zunächst soll das komplette UI-Paket eingebunden und mit ihm ein paar einfache Grundübungen vollzogen werden.

Wenn Sie Ihren Download konfiguriert und heruntergeladen haben, finden Sie alle Dateien in einer *.zip-Datei verpackt auf Ihrer Festplatte vor. Der Dateiname lautet, abhängig von Ihrer Version, die Sie heruntergeladen haben *jquery-ui-1.8.2.custom.zip*.

Betrachten Sie nun bitte die Struktur des Pakets. Wenn Sie das Downloadpaket öffnen bzw. entpacken, sehen Sie drei Ordner:

- ▶ *[jquery-ui-ordner]/development-bundle/* beinhaltet Anwendungsbeispiele, zusätzliche Plugins, die Dokumentation, Lizenzbestimmungen und Themes. Zu Themes werden Sie im Folgenden noch mehr erfahren. Die für uns wichtigen Verzeichnisse sind *css* und *js*.
- ▶ In *[jquery-ui-ordner]/js/* befinden sich die Dateien des jQuery Frameworks sowie die jQuery UI-Plugins, die in einer einzigen Datei zusammengefasst sind. Im Gegensatz zum development-bundle, dort liegen alle Plugins in Einzeldateien vor.
- ▶ Im Verzeichnis *[jquery-ui-ordner]/css/* finden Sie alle Dateien, die Sie benötigen, um den Widgets ein Layout zu verleihen oder das vorgegebene Aussehen zu verändern.

5.7.1 Themeroller

Entweder wagen Sie sich vor und ändern die Stylesheets sowie die grafischen Elemente per Hand, oder Sie verwenden den Themeroller, mit dem Sie Stileigenschaften, Farben und Abstände über einen Konfigurator leicht ändern können. Den Themeroller finden Sie unter

- ▶ <http://jqueryui.com/themeroller/>

Bevor Sie also die Komponenten, die Sie für Ihr Projekt benötigen, herunterladen, können Sie hier auch das Aussehen in einem Theme zusammenpacken und mit der gesamten UI-Bibliothek herunterladen.

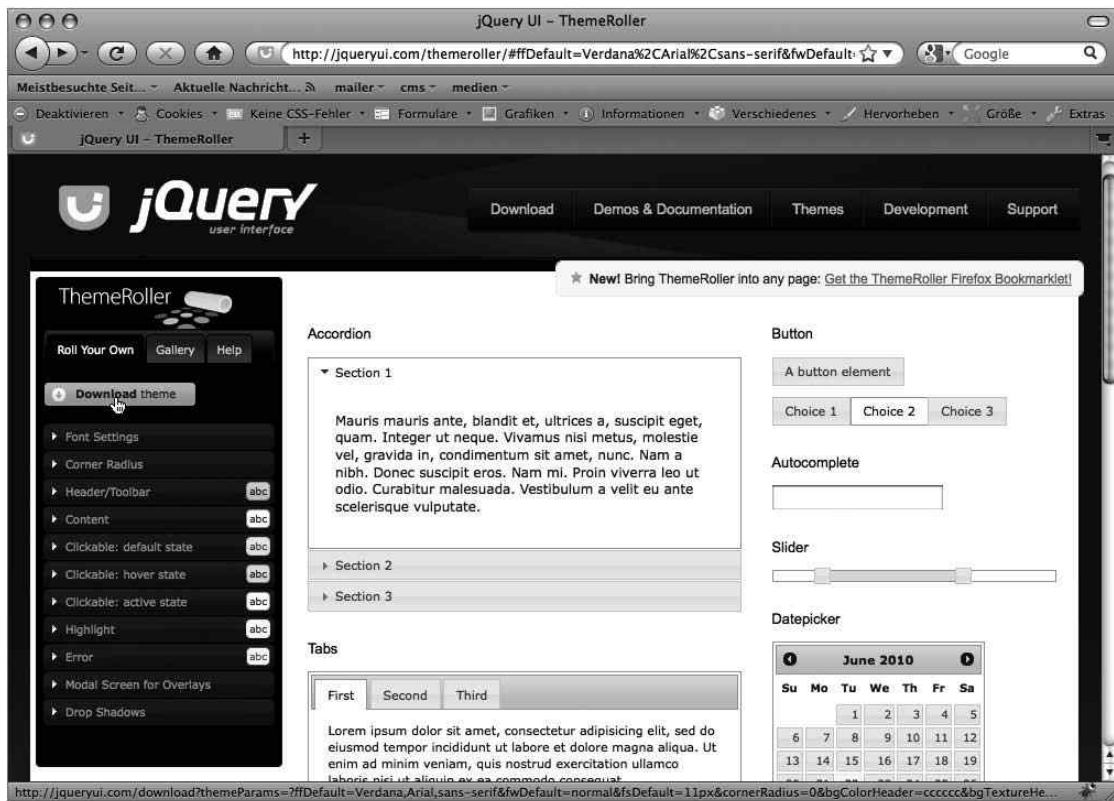


Abbildung 5.24 jQuery UI Themeroller

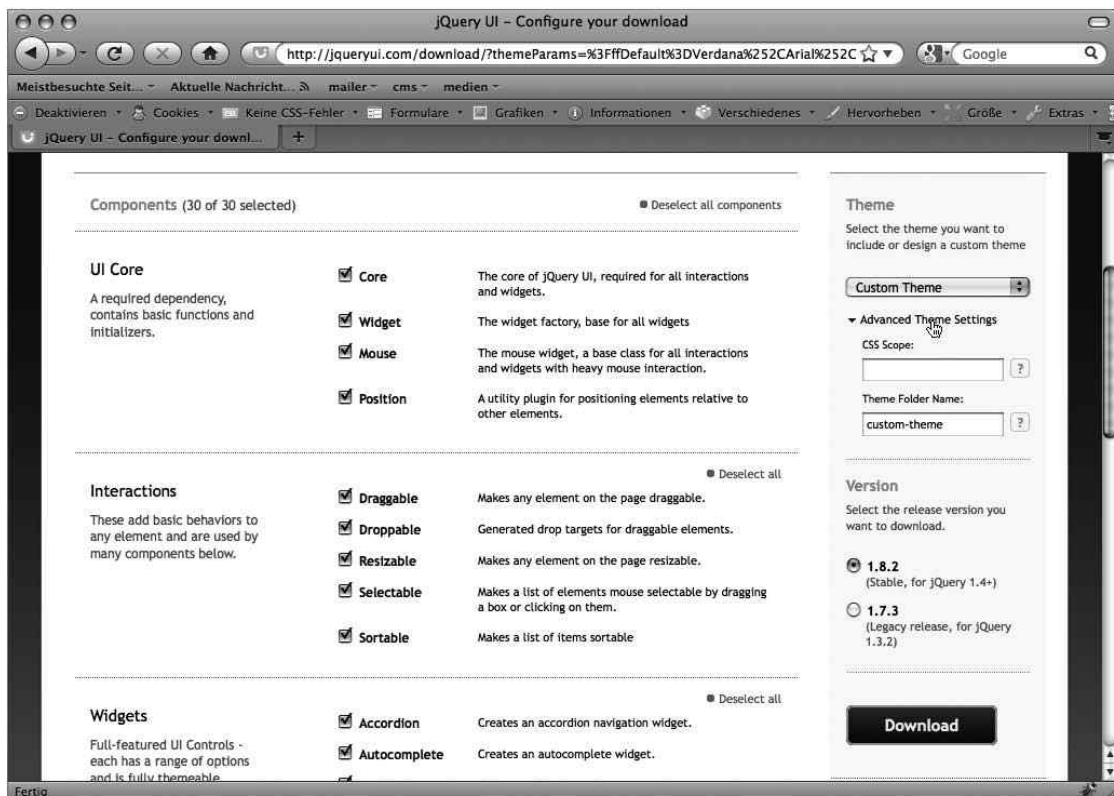


Abbildung 5.25 jQuery UI Download-Konfigurator

Neben eigenen Themes, die Sie selbst erstellen können, gibt es noch eine Reihe vorgefertigter Themes, die Sie auf der Downloadseite auswählen können.

Tipp

Falls Sie ein Theme über den Themeroller erstellt haben, können Sie es auch nachträglich noch einmal durch den Themeroller verändern. Im oberen Teil des CSS Stylesheets am Ort `[jquery-ui-ordner]/css/jquery-ui-1.8.2.custom.css` unterhalb des Copyright-Vermerks finden Sie einen URL mit recht vielen und langen Parametern. Er führt direkt zu Ihrem angepassten Theme. Sie können das Theme modifizieren und anschließend erneut herunterladen. Allerdings wird jedes Mal das Gesamtpaket aus ausgewählten Plugins und Theme zusammengepackt, Sie müssen nach dem Download das Theme manuell an den richtigen Ort kopieren.

5.7.2 Theme manuell bearbeiten

Die Einstellungen sind sehr vielfältig, die Sie für die UI Widgets vornehmen können. Es gibt allerdings auch Anforderungen, gerade wenn es darum geht, ein bestimmtes Corporate Design für ein bestimmtes Unternehmen webgerecht umzusetzen, für die das jQuery-Standardaussehen nicht ausreicht. Gute Designer sind (zu Recht) oft hartnäckig, was die genaue Umsetzung eines Designs anbelangt. Oft sind vorgefertigte Designs zu starr angelegt, um eine bestimmte Designanforderung realisieren zu können. Manches Unternehmen hat seine eigene Icon-Sprache entwickelt, die auf die Widgets übertragen werden muss. Die Lösung liegt darin, das Stylesheet und die Icons manuell umzuschreiben. Die Styles liegen zentral in einer einzigen CSS-Datei, in diesem Fall in `[jquery-ui-ordner]/css/jquery-ui-1.8.2.custom.css`.

Wenn Sie Icons modifizieren wollen, geht das auf denkbar einfachste Weise. Innerhalb des `css`-Verzeichnisses bzw. innerhalb des `theme`-Verzeichnisses finden Sie alle Icons in einer einzigen PNG-Datei zusammengefasst. Allerdings liegt diese Icon-Datei in vier verschiedenen Farbvarianten vor. Sie müssen nun einfach an den richtigen Positionen die Icons in dem Bildbearbeitungstool Ihrer Wahl verändern und an diesen Ort kopieren. Betrachten Sie nun eine Variante der Icon-Datei; verwenden Sie dazu die Datei `ui-icons_222222_256x240.png`, wobei 222222 für die Farbe der Icons steht:

Sie werden per CSS-Angaben an die richtige Stelle verschoben. Der Vorteil ist klar erkennbar, Sie müssen die Icons nur an einer zentralen Stelle bearbeiten. Dieses Verfahren nennt man auch CSS Sprites. Abbildung 5.27 zeigt einen Ausschnitt aus der CSS-Datei.

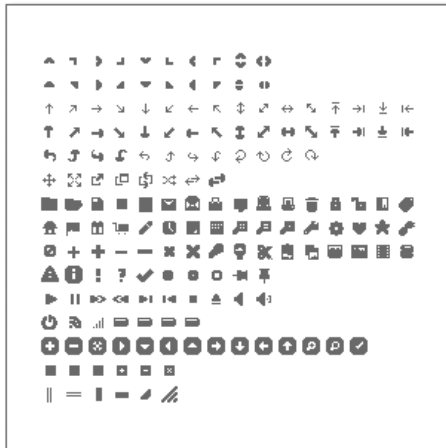


Abbildung 5.26 jQuery UI CSS Framework: Icons als CSS Sprites

```

/* positioning */
.ui-icon-carat-1-n { background-position: 0 0; }
.ui-icon-carat-1-ne { background-position: -16px 0; }
.ui-icon-carat-1-e { background-position: -32px 0; }
.ui-icon-carat-1-se { background-position: -48px 0; }
.ui-icon-carat-1-s { background-position: -64px 0; }
.ui-icon-carat-1-sw { background-position: -80px 0; }
.ui-icon-carat-1-w { background-position: -96px 0; }
.ui-icon-carat-1-nw { background-position: -112px 0; }
.ui-icon-carat-2-n-s { background-position: -128px 0; }
.ui-icon-carat-2-e-w { background-position: -144px 0; }
.ui-icon-triangle-1-n { background-position: 0 -16px; }
.ui-icon-triangle-1-ne { background-position: -16px -16px; }
.ui-icon-triangle-1-e { background-position: -32px -16px; }
.ui-icon-triangle-1-se { background-position: -48px -16px; }
.ui-icon-triangle-1-s { background-position: -64px -16px; }
.ui-icon-triangle-1-sw { background-position: -80px -16px; }
.ui-icon-triangle-1-w { background-position: -96px -16px; }
.ui-icon-triangle-1-nw { background-position: -112px -16px; }
.ui-icon-triangle-2-n-s { background-position: -128px -16px; }
.ui-icon-triangle-2-e-w { background-position: -144px -16px; }
.ui-icon-arrow-1-n { background-position: 0 -32px; }
.ui-icon-arrow-1-ne { background-position: -16px -32px; }
.ui-icon-arrow-1-e { background-position: -32px -32px; }
.ui-icon-arrow-1-se { background-position: -48px -32px; }
.ui-icon-arrow-1-s { background-position: -64px -32px; }
.ui-icon-arrow-1-sw { background-position: -80px -32px; }
.ui-icon-arrow-1-w { background-position: -96px -32px; }

```

Abbildung 5.27 jQuery UI CSS Framework – Ausschnitt der CSS-Datei

Die Icon-Datei wird insgesamt in den Hintergrund eines Elements gelegt. Für jedes Icon wird die Position des Backgrounds um den entsprechenden Wert verschoben. So ist jeweils nur das gewünschte Icon sichtbar. Wenn Sie Ihre Modifikationen innerhalb dieses Rasters vornehmen, können Sie die Positionswerte der CSS-Angaben belassen. Sie können die Icons und das jQuery UI CSS Framework übrigens auch ausschließlich ohne sonstige JavaScript-Nutzung verwenden:

```

<!DOCTYPE html>
<html>
<head>
  <title>jQuery | UI Beispiele</title>
  <link type="text/css"
        href="css/ui-lightness/jquery-ui-1.8.2.custom.css"
        rel="stylesheet" />
  <style type="text/css">

```

```

    #icons {margin: 0; padding: 0;}
    #icons a {margin: 2px; padding: 4px 0; float: left;}
    #icons span.ui-icon {float: left; margin: 0 4px;}
  </style>
</head>
<body>
  <div id="icons" class="ui-widget ui-helper-clearfix">
    <a href="#" class="ui-state-highlight ui-corner-all">
      <span class="ui-icon ui-icon-carat-1-n"></span>
    </a>
    ...
  </div>
</body>
</html>

```

Listing 5.60 HTML-Gerüst – jQuery UI CSS Framework

5.7.3 Ein kleines Beispiel: Hintergrundfarbe animieren

Im Folgenden werden Sie weder den Quelltext der UI-Bibliothek analysieren, noch alle Widgets und Effekte vorgestellt bekommen. Sie werden einige Plugins kennenlernen und deren Konfigurationsmöglichkeiten ausloten und versetzen sich so in die Lage, die Vorgehensweise auf alle Widgets zu übertragen.

Das erste Beispiel zeigt, wie Sie die Hintergrundfarbe einer HTML-Seite in einer Schleife langsam und kontinuierlich von einer Farbe in eine andere übergehen lassen. Beispielsweise von der Farbe #484848 zu #FFFFFF und wieder zurück.

Der HTML-Code enthält hauptsächlich die Einbindung der jQuery-Dateien:

```

<!DOCTYPE>
<html>
  <head>
    <title>Farben animieren</title>
    <script type="text/javascript" src="jquery-1.4.2.js">
    </script>
    <script type="text/javascript"
      src="jquery-ui-1.8.2.custom.js">
    </script>
    <script type="text/JavaScript">
      $(document).ready(function(){
        function changeBGcolor() {
          var obj = $("body");
          $(obj).animate({ backgroundColor: "#48484C" },
            15000)
            .animate({ backgroundColor: "#FFFFFF" },

```

```

15000,
function() {
    changeBGcolor();
});
    }
    changeBGcolor();
});
</script>
</head>
<body>
</body>
</html>

```

Listing 5.61 Ein erstes jQuery UI-Beispiel – Farbe animieren

Sie sehen in diesem Beispiel eine rekursive Funktion, also eine Funktion die sich selbst erneut aufruft. Im Grunde genommen wird die Ihnen bereits bekannte Methode `.animate()` mit den Argumenten für die Farbänderung und der Dauer von 15.000 Millisekunden aufgerufen. In der zweiten Methode wird ein Callback mit der Referenz auf sich selbst erzeugt. Das Ergebnis ist eine Endlosschleife. Alle Elemente dürften Ihnen bekannt vorkommen, nur, wenn Sie die jQuery UI-Plugins nicht einbinden, werden Sie eine CSS-Warnung erhalten, dass der Wert für die CSS-Eigenschaft `background-color` nicht verarbeitet werden kann.

Wenn Sie wirklich nur die Farbanimation in einer Produktivumgebung nutzen wollen, reicht es, nur den Effects Core auszuwählen, herunterzuladen und einzubinden, hier benötigen Sie nicht das gesamte Set an Plugins.

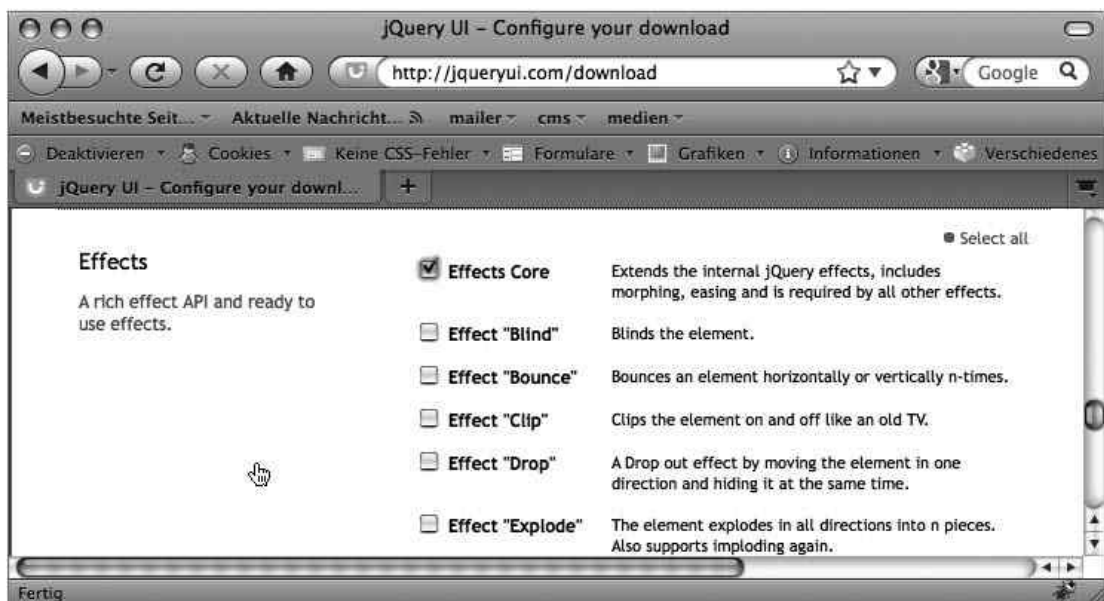


Abbildung 5.28 jQuery UI-Download des Effects Core

Neben Farbanimationen können Sie auch ganze CSS-Klassen animieren. Die zu animierende Klasse kann auch verschiedene Farben beinhalten, genauso wie Abstände oder Breitenangaben. Darüber werden Sie im nächsten Beispiel noch mehr erfahren.

5.7.4 Klassen animieren

Im Folgenden werden wir einen Container, der simple Textbausteine enthält, der über verschiedene Eigenschaften wie Breite und Hintergrundfarbe verfügt, mit einer neuen Klasse überschreiben. Beim Überschreiben sollen die Eigenschaften nicht hart ausgetauscht werden, sondern ineinander übergehen. Betrachten Sie zuerst den HTML-Quelltext:

```
<!DOCTYPE>
<html>
<head>
  <title>jQuery UI | CSS Klasse animieren</title>
  <link type="text/css" href="css/classanimation.css"
        rel="stylesheet" />
  <script type="text/javascript" src="js/jquery-1.4.2.js">
  </script>
  <script type="text/javascript"
        src="js/jquery-ui-1.8.2.custom.js">
  </script>
  <script type="text/JavaScript">
    $(document).ready(function(){
      $("#container").click(function() {
        $(this).toggleClass('secondClass', 1000);
      });
    });
  </script>
</head>
<body>
  <div id="container" class="oneClass">
    <h1>Container Inhalt</h1>
    <p>Lorem ipsum...</p>
  </div>
</body>
</html>
```

Listing 5.62 jQuery UI-Beispiel – CSS-Klassen animieren

Der jQuery-Code wurde direkt in die HTML-Seite gepackt, da er wirklich übersichtlich ist, die CSS-Klassen wurden in der Datei *classanimation.css* verortet, hier sind die beiden Klassen `oneClass` und `secondClass` wichtig:

```

...
.oneClass {
    width:400px;
    padding:30px;
    background-color:#996600;
}
.secondClass {
    margin:100px;
    width:200px;
    padding:10px;
    background-color:#FF9900;
}

```

Listing 5.63 CSS Styles – CSS-Klassen animieren

An dieser kleinen Spielerei sehen Sie, dass Sie komplexe CSS mit nur einer Zeile jQuery(UI)-Code erzeugen können. Mit der Methode `.toggleClass()`, die Sie aus dem jQuery Framework kennen, schaffen Sie Übergänge zwischen verschiedensten Eigenschaften, sofern Sie das Argument `duration`, hier die Zahl 1.000, also einen Integerwert übergeben.

5.7.5 jQuery UI-Dialog

Im nächsten Beispiel werden Sie das User Interface für einen modalen Dialog kennenlernen. »Modal« bedeutet, dass der Benutzer auf das Dialogfenster reagieren muss, bevor er eine andere Aktion ausführt.

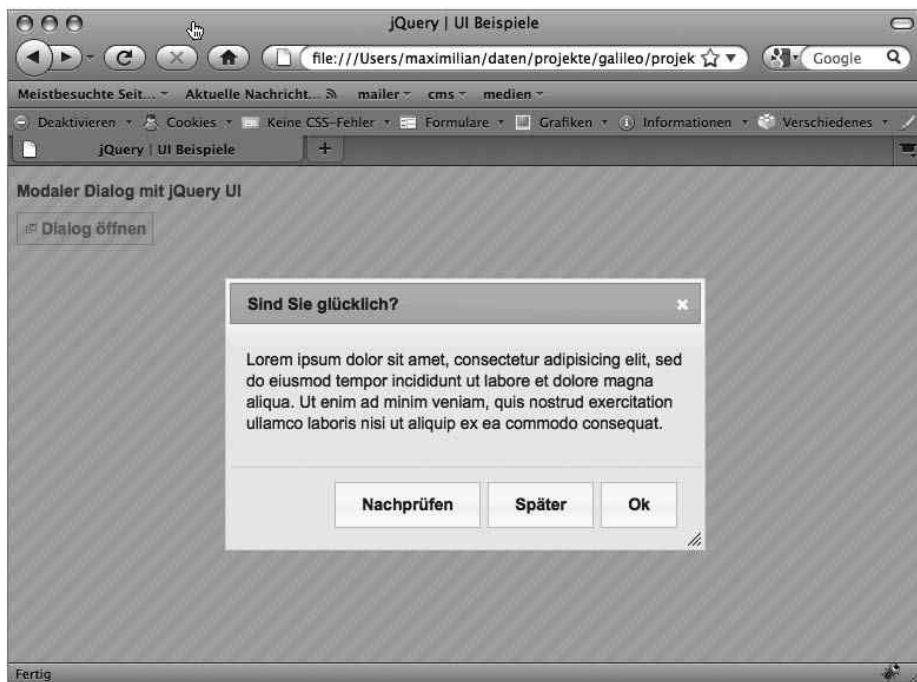


Abbildung 5.29 jQuery UI-Dialog

In diesem Beispiel verwenden Sie wie zuvor besprochen das gesamte jQuery UI-Paket. Während Sie sich im Entwicklungsprozess befinden, ist es einfacher mit dem gesamten Paket zu programmieren. Erst wenn Sie Ihr Projekt veröffentlichen, können Sie das Paket optimieren und nur die benötigten Komponenten zusammenfassen.

Laden Sie das Paket herunter, falls Sie das noch nicht getan haben, und legen Sie es in Ihrem Projektordner ab. Im Verzeichnis *js* sollten sich die jQuery UI-JavaScript-Datei befinden sowie die Datei des jQuery Frameworks: im Verzeichnis *css* zum einen eine CSS-Datei für dieses Projekt sowie zum anderen der Ordner des Themes, das Sie sich zusammengestellt oder ausgewählt haben. Zuerst betrachten Sie bitte wieder die HTML-Datei:

```
<!DOCTYPE>
<html>
<head>
  <title>jQuery | UI Beispiele</title>
  <link type="text/css"
        href="css/ui-lightness/jquery-ui-1.8.2.custom.css"
        rel="stylesheet" />
  <link type="text/css"
        href="css/jquery-ui-dialog.css"
        rel="stylesheet" />
  <script type="text/javascript"
        src="js/jquery-1.4.2.js"></script>
  <script type="text/javascript"
        src="js/jquery-ui-1.8.2.custom.js"></script>
  <script type="text/javascript"
        src="js/jquery-ui-dialog.js"></script>
</head>
<body>
  <h2 class="demoHeaders">Modaler Dialog mit jQuery UI</h2>
  <p>
    <a href="#" id="dialog_link"
      class="ui-state-default ui-corner-all">
      <span class="ui-icon ui-icon-newwin"></span>
      Dialog öffnen
    </a>
  </p>
  <div id="dialog" title="Sind Sie glücklich?">
    <p>Lorem ipsum...</p>
  </div>
</body>
</html>
```

Listing 5.64 HTML-Gerüst – jQuery UI-Dialog

Im Header binden Sie alle notwendigen Dateien ein, die Dateien *jquery-ui-dialog.css* und *jquery-ui-dialog.js* sind die Projektdateien, in denen Sie die CSS-Eigenschaften und den jQuery-Code für dieses Beispiel anlegen.

Im Body sehen einerseits das `<a>`-Element mit dem ID `dialog_link`, mit dem Sie den Dialog starten, zum anderen den `<div>`-Container mit dem ID `dialog`, der den Inhalt des Dialogs enthält. Das `title`-Attribut werden Sie verwenden, um eine Dialogüberschrift daraus zu generieren (siehe Abbildung 5.29). Die CSS-Datei enthält Standards zur Formatierung der HTML-Seite, die an dieser Stelle übersprungen werden können. Betrachten Sie die Scriptdatei *jquery-ui-dialog.js*:

```
$(document).ready(function() {
    // Dialog
    $('#dialog').dialog({
        autoOpen: false,
        width: 400,
        modal: true,
        buttons: {
            "Ok": function() {
                $(this).dialog("close");
            },
            "Nachprüfen": function() {
                location.href = "http://de.wikipedia.org/wiki/Glück";
            }
        }
    });
    // Dialog Link
    $('#dialog_link').click(function(e){
        e.preventDefault();
        $('#dialog').dialog('open');
    });
});
```

Listing 5.65 JavaScript-Code – jQuery UI-Dialog

Es wird zweimal die Methode `.dialog()` aufgerufen. Einmal werden dem jQuery-Objekt `$('#dialog')` die Optionen als Argumente übergeben. Neben `autoOpen`, `width` und `modal` die Optionen für die Dialogbuttons: Die Strings OK und NACHPRÜFEN legen die Labels der Buttons fest, die Callback-Funktion bestimmt, was passiert, wenn der Button mit der Maus gedrückt wird. Betrachten Sie noch einmal die Option `modal`: Sie legt mit dem Wert `true` fest, dass keine andere Aktion als das Drücken der beiden Buttons im Dialogfenster möglich ist. Außerdem wird ein Overlay generiert, das die gesamte HTML-Seite mit einem Layer bedeckt, in diesem Beispiel ein transparenter grauer Hintergrund.

Mit dem Event `.click()` wird erneut die Methode `.dialog()` mit dem Argument 'open' aufgerufen.

Sie sehen, mit wenigen Handgriffen können Sie mit jQuery UI ein Dialogfenster in Ihr Script integrieren, mit Optionen die Art und Weise der Funktionalität beeinflussen sowie die Gestaltung festlegen.

5.7.6 jQuery UI Accordion

Sie haben bis jetzt eine Erweiterung der Methode `.animate()` sowie ein Widget »Dialog« kennengelernt. Als letztes Beispiel in diesem Kapitel soll das Accordion Widget vorgestellt werden. Sie hatten bereits in Abschnitt 5.1, »Schönere Navigationen«, hierfür ein eigenes, einfaches Script erstellt.

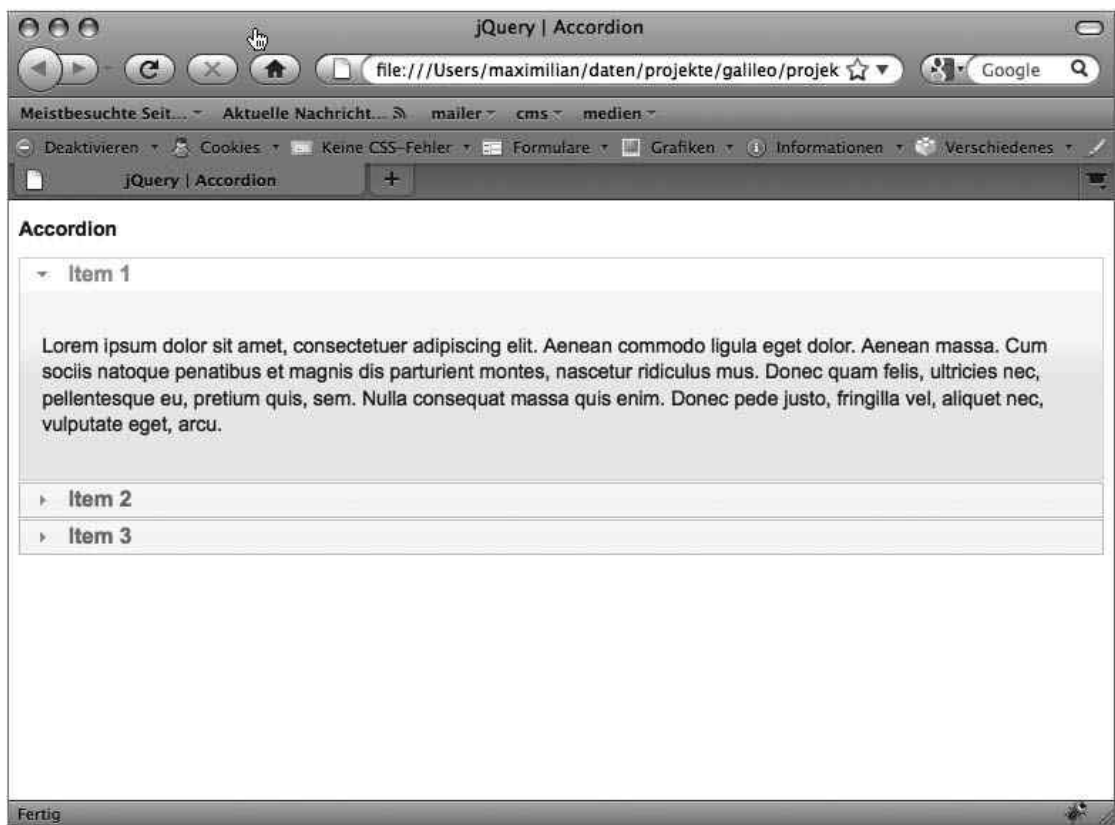


Abbildung 5.30 jQuery UI Accordion

Das Widget stellt wieder eine Reihe von Konfigurationsmöglichkeiten zur Verfügung. Wer ein einfaches Script bevorzugt, wird mit dem Beispiel aus »Navigationen« zufrieden sein. Wer aber Icons benötigt, die anzeigen, dass das Menü geöffnet ist, oder die CSS Styles der UI-Komponenten nutzen will, der greife auf dieses Widget zurück. Hier nun der HTML-Text:

```

<!doctype html>
<html>
<head>
<title>jQuery UI | Accordion</title>
  <link type="text/css"
        href="css/ui-lightness/jquery-ui-1.8.2.custom.css"
        rel="stylesheet" />
  <link type="text/css"
        href="css/jquery-ui-accordion.css" />
  <script type="text/javascript"
        src="js/jquery-1.4.2.js"></script>
  <script type="text/javascript"
        src="js/jquery-ui-1.8.2.custom.js"></script>
  <script type="text/javascript"
        src="js/jquery-ui-accordion.js"></script>
</head>
<body>
  <div id="accordion">
    <h3><a href="#">Item 1</a></h3>
    <div>
      <p>Lorem ipsum ...</p>
    </div>
    <h3><a href="#">Item 2</a></h3>
    <div>
      <p>Lorem ipsum ...</p>
    </div>
    ...
  </div>
</body>
</html>

```

Listing 5.66 HTML-Gerüst – jQuery UI Akkordion Navigation

Dieses Gerüst unterscheidet sich nur dadurch, dass das gesamte Accordion in einem Wrapper mit der ID `accordion` umschlossen wurde und dass die Überschriften noch einmal ein `<a>`-Element enthalten. Ansonsten ist es baugleich mit dem in Abschnitt 5.1, »Schönere Navigationen«, besprochenen Script, abgesehen von den im Header eingebundenen Dateien. Die CSS Styles werden wieder übergangen, betrachten Sie die Scriptdatei *jquery-ui-accordion.js*:

```

$(document).ready(function() {
  $("#accordion").accordion({
    header: "h3",
    // animated:"easeInOutExpo",
    animated:"bounceslide"
  });
});

```

```
});
});
```

Listing 5.67 JavaScript Code – jQuery UI Akkordion Navigation

Es würde noch kürzer gehen, die Option `header` benötigen Sie in diesem Fall nicht. Sie erwartet einen Selektor, mit dem Sie einschränken können, welche Elemente als Accordion Header verwendet werden sollen. Die Option `animated` bestimmt, wie ein Block eingeblendet werden soll, die Eigenschaft `bounceslide` fährt der Block mit dem »Bounce«-Effekt ein, und fährt es mit der Eigenschaft »swing« nach oben. Sie können neben `bounceslide` und `slide` jeden Effekt der Easing Equations verwenden.

5.7.7 Exkurs: Easing Equations

Von »Easing« war bereits in Kapitel 1, »jQuery kennenlernen«, die Rede. Hier kommen wir, weil es sich anbietet, kurz darauf zurück. Vor einigen Jahren hat sich u.a. Robert Penner Gedanken gemacht, damals noch für das Flash-Format, wie man mittels mathematischer Gleichungen lebendigere Animationen realisieren kann.

Das Ergebnis war sein Script »Easing Equations«, das findige jQuery-Entwickler erst als eigenständiges Plugin adaptierten und später in das jQuery UI Framework integrierten. Dieses Set an Gleichungen beschreibt verschiedene Abläufe von Bewegungen, anhand des »Quadratic Easing« soll das hier verdeutlicht werden. Das »EaseOutQuad« ist das Default-Easing innerhalb des jQuery UI-Plugins.

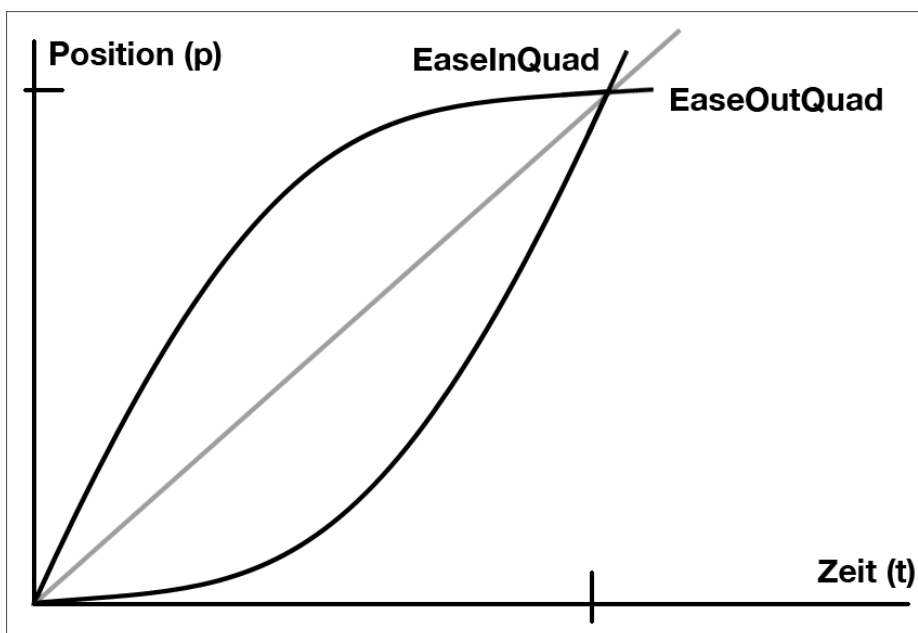


Abbildung 5.31 Quadratische Easing Equations

Einfach formuliert, beschreibt diese Kurve die quadratische Zu- oder Abnahme der Geschwindigkeit $p(t) = t^2$, wobei p die Position ist und t die Zeit. Grafisch dargestellt, beschreibt die Kurve `EaseOutQuad`, dass die Animation mit einer bestimmten Geschwindigkeit beginnt und zum Ende hin abgebremst wird.

Der umgekehrte Fall stellt die Kurve `EaseInQuad` dar, bei der eine höhere Anfangsgeschwindigkeit zum Ende hin abnimmt.

Zum Vergleich: Die graue Linie beschreibt eine Animation, bei der die Geschwindigkeit bis zum Erreichen einer Position gleich bleibt: $p(t) = t$.

Die Kurven `EaseInQuad` und `EaseOutQuad` sind die am einfachsten zu beschreibenden Kurven (abgesehen von der linearen Kurve). Wenn Sie sich mit dem mathematischen Hintergrund auseinandersetzen wollen, besuchen Sie <http://robertpenner.com/easing/>. Dort finden Sie ein Tutorial, das sämtliche von ihm entwickelten Easing Equations beschreibt, zwar für ActionScript, aber die Funktionen sind leicht für JavaScript übertragbar.

Neben den beiden hier beschriebenen Easing Equations gibt es noch eine Vielzahl weiterer. Am einfachsten ist es, den Quelltext der unkomprimierten Datei `jquery-ui-1.8.2.custom.js` zu öffnen, sofern Sie die »Effects Core« mit ausgewählt haben, und mittels Suchfunktion zu »jQuery Easing« springen. Probieren Sie anhand des bereits besprochenen Beispiels »accordion« die verschiedenen Easings aus. Eine Dokumentation der verfügbaren Easings sowie eine Demoanimation finden Sie unter

<http://gsgd.co.uk/sandbox/jquery/easing/>.

Dort ist auch das ursprüngliche Plugin `jquery.easing` zu finden, das, wie gesagt komplett in das jQuery UI gewandert ist.

5.7.8 Zusammenfassung

Sie haben in diesem Abschnitt noch lange nicht alle Möglichkeiten der jQuery UI Plugins kennengelernt. Weitere Beispiele werden Ihnen in nachfolgenden Abschnitten zu den Themen »Formulare« oder »Tabellen« begegnen. Auch haben Sie sich in diesem Abschnitt noch lange nicht alle Optionen erarbeitet, über die die hier besprochenen Widgets verfügen. Ziel war es ja auch nur, grundsätzlich zu erfahren, wie das jQuery UI Plugin die Grundfunktionen erweitert, und wie die Erweiterungen grundsätzlich anzuwenden sind.

Wenn Sie die WebSite des UI-Projekts besuchen, finden Sie zu jeder Funktion, jedem Effekt und zu jedem Widget Demos und ausführliche Beschreibungen der Optionen, bzw. der Argumente, die Sie den einzelnen UI-Methoden mit auf den

Weg geben können. Wenn Sie das UI-Plugin heruntergeladen haben, finden Sie kurze Demos zu Ihrem individuell zusammengestellten UI-Paket. Das Komplett-paket finden Sie in der Version 1.8.2 auf der Begleit-DVD zu diesem Buch.

5.8 Formulare beherrschen mit jQuery

Das Thema Formulare ist bereits innerhalb der offiziellen Dokumentation ein wichtiges Thema, so sind Webformulare eine wichtige Schnittstelle zwischen dem Anwender und der Maschine. Formulare werden nicht nur zur Kontaktaufnahme eingesetzt, sondern auch in den Interfaces komplexer interaktiver Webanwendungen wie Content-Management-Systemen oder Mail-Interfaces.

In Abschnitt 4.8, »Formulare beherrschen mit jQuery«, wurde das Thema bereits angerissen und die verfügbaren Methoden und Eigenschaften von jQuery vorgestellt. Hier werden Sie nun einige vertiefende Beispiele zum Umgang mit jQuery und Formularen kennenlernen. Zum einen kann auch die klassische Formularvalidierung, also die Prüfung auf Gültigkeit der Benutzereingaben, geprüft werden, zum anderen kann die Eingabe der Daten für den Benutzer stark vereinfacht werden, wie Sie am Beispiel der Datumseingabe, des Datepickers, sehen werden.

5.8.1 Formulare validieren

Ohne jQuery konnte man auch mit einfachem JavaScript-Formulare validieren, das war nie eine schwierige Aufgabe. Mit Unterstützung von jQuery können aber Formulareingaben komfortabler für den Entwickler und mit mehr Möglichkeiten ausgestattet realisiert werden. Es können Abhängigkeiten entwickelt werden – stellen Sie sich ein Kontaktformular vor, das eine Newsletter-Bestellfunktion beinhaltet.

So soll nicht einfach ein Newsletter bestellt werden können, der Nutzer soll die Möglichkeit haben, wahlweise zu einem oder mehreren Themen informiert zu werden. Jedoch, die Wahloptionen sollen erst eingeblendet werden, wenn auch brav die Checkbox für »Newsletter« angeklickt wurde. Also: Die Aufgabe lautet, ein Formular mit Pflichtfeldern und Abhängigkeiten zu kreieren.

Das folgende Script soll aber nicht von Grund auf neu entwickelt werden. Hier stellt die jQuery Community ein mächtiges Plugin zu Verfügung. Entwickelt von einem Core Member des jQuery Teams, von Jörn Zaefferer. Sie können sich diese und weitere Plugins unter folgender Adresse herunterladen:

<http://bassistance.de/jquery-plugins/jquery-plugin-validation/>



100

100

100

100


```

</script>
</head>
<body>
  <form id="form" method="post" action="">
    <fieldset>
      <legend>Formular mit Validierung</legend>
      <p>
        <label for="vorname">Vorname* </label>
        <input id="vorname" name="vorname" />
      </p>
      <p>
        <label for="name">Name* (mindestens 2 Zeichen)</label>
        <input id="name" name="name" />
      </p>
      <p>
        <label for="email">E-Mail*</label>
        <input id="email" name="email" />
      </p>
      <p>
        <label for="date">Wunschdatum</label>
        <input id="date" name="date" value="" />
      </p>
      <p>
        <label for="message">Mitteilung*</label>
        <textarea id="message" name="message"></textarea>
      </p>
      <p>
        <label for="newsletter">
          Ich möchte einen Newsletter erhalten
        </label>
        <input type="checkbox"
          id="newsletter" name="newsletter" />
      </p>
      <fieldset id="newsletter_themen">
        <legend>Themen</legend>
        <label for="themen_birnen">
          <input type="checkbox" value="birnen" name="themen"/>
          Birnen
        </label>
        <label for="themen_aepfel">
          <input type="checkbox" value="aepfel" name="themen"/>
          Äpfel
        </label>
        <label for="themen_bananen">
          <input type="checkbox" value="kiwi" name="themen"/>

```

```

        Kiwi
    </label>
    <label for="themen" class="error">
        Bitte wählen Sie mindestens zwei
        Newsletter Thema aus.
    </label>
</fieldset>
<p>
    <input class="submit" type="submit" value="Submit"/>
</p>
</fieldset>
</form>
<p>* Pflichtfelder</p>
</body>
</html>

```

Listing 5.68 HTML-Gerüst – Formular mit jQuery

Neben dem jQuery Framework und dem Plugin *jquery.validate.js* inkludieren Sie noch die Datei *messages_de.js*, sie dient dazu, die Fehlermeldungen und Warnungen in einer deutschen Lokalisation anzubieten. Selbstverständlich binden Sie noch die Projektdatei *jquery-form-validate.js* ein. Die Styleangaben übergehen wir größtenteils, allein auf ein Klasse soll hier hingewiesen werden:

```

...
#newsletter_themen label.error {
    display: none;
    width: 350px;
    margin-left: 10px;
}
...

```

Sie blendet die `error`-Klasse innerhalb des Elements mit der ID `#newsletter_themen` aus. Diese wird erst mittels JavaScript im Falle, dass keine Option ausgewählt wurde, hinzugefügt.

Als Nächstes betrachten Sie bitte die jQuery-Projektdatei:

```

$(document).ready(function() {
    // hier die Methode .validate()
    $("#form").validate({
        submitHandler: function() {
            alert("Simulation: abgeschickt");
        },
        rules: {
            vorname: "required",

```

```

        name: {
            required: true,
            minlength: 2
        },
        email: {
            required: true,
            email: true
        },
        themen: {
            required: "#newsletter:checked",
            minlength: 1
        },
        message: "required"
    }
});
// Optionen erst sichtbar, wenn Checkbox gechecked ist
var newsletter = $("#newsletter");
var isChecked = newsletter.is(":checked");
var themen = $("#newsletter_themen");
themen[isChecked ? "show" : "hide"]();
//var themaInputs = themen.find("input")
//                                     .attr("disabled", !isChecked);

newsletter.click(function() {
    themen[this.checked ? "fadeIn" : "fadeOut"](1000);
    //themaInputs.attr("disabled", !this.checked);
});
});

```

Listing 5.69 JavaScript-Code – Formular validieren mit jQuery

Die Validierung wird mit der Methode `.validate()` vorgenommen. Dieser Methode können Sie eine Reihe von Optionen übergeben. Als Erstes führen wir die den `submitHandler` ein, mit ihm bestimmen Sie, welche Aktion nach erfolgreicher Validierung durchgeführt werden soll. In unserem Fall simulieren wir der Einfachheit halber eine Aktion mit `alert()`, in einem realen Beispiel könnte der `submitHandler` so aussehen:

```

$("#form").validate({
    submitHandler: function(form) {
        form.submit();
    }
    ...
});

```

Achtung

Wobei Sie darauf achten sollten, ist, nicht `$(form).submit()` zu verwenden, sondern `form.submit()`. Erstere Anweisung ruft nach erfolgreicher Validierung erneut eine Validierung aus, Sie erhalten eine Endlosschleife, nämlich eine Funktion, die sich rekursiv selbst aufruft.

Die Methode `form.submit()` sendet also das Formular. Zur Erinnerung: Es wird an den URL gesendet, der im Form-Element im Attribut `action` angegeben wurde. In diesem Beispiel lassen wir es leer, da wir ein Absenden nur simulieren.

Eine weitere Option ist `rules`. Sie bestimmt die Regeln, nach denen ein Formelement validiert werden soll. Wenn der `vorname` den Wert `required` erhält, wird ein Input-Element mit dem Attribut `name="vorname"` ein Pflichtfeld. Neben `required` können Sie noch Bedingungen wie `minlength`, also Mindestlänge, oder `email` angeben.

Die Option `email` prüft, ob es sich um eine gültige E-Mail-Adresse handelt, also ob Buchstaben, `@`-Zeichen und ein Punkt vorkommen.

Mit der Option `messages` könnten Sie die Default-Fehlermeldungen mit eigenen Strings überschreiben. Sie haben aber gesehen, dass im Head des Dokuments die Datei `messages_de.js` eingebunden wurde. Diese enthält sämtliche Fehlermeldungen in deutscher Sprache, sodass wir die Option `messages` in diesem Beispiel gestrost übergehen können.

Interessant wird es nochmals mit den Bedingungen für das bzw. die Elemente mit dem `name`-Attribut `themen`. Mit dieser Option wird überprüft,

```
themen: {
  required: "#newsletter:checked",
  minlength: 1
},
```

ob das Element mit der ID `#newsletter` das Attribut mit dem Wert `checked` besitzt. Es handelt sich hier um die übergeordnete Checkbox `newsletter` und `themen` sind die Checkboxes, die sozusagen die Unterkategorien »Äpfel«, »Birnen«, »Bananen« enthalten. Es sind natürlich keine Unterkategorien im Sinne von Hierarchien, aber sie prüfen alle drei einen Bezug zum Element mit dem ID `#newsletter` ab. Darüber hinaus wird festgelegt, dass mindestens eine »Unterkategorie« angeklickt sein muss (`minlength:1`).

Bleibt noch die Funktionalität, dass die »Unterkategorien« nur angezeigt werden, wenn `#newsletter` angeklickt wurde. Die Antwort finden Sie im `.click()`-Event:

```
newsletter.click(function() {
    themen[this.checked ? "fadeIn" : "fadeOut"](1000);
})
```

Wenn das Element `#newsletter` den Wert des Attributs `checked=checked` enthält, soll das FIELDSET eingeblendet werden, anderenfalls soll es ausgeblendet werden. Interessant ist die Syntax:

Tipp

Sie können statt `$("#selector").fadeIn(1000)` auch die Schreibweise `$("#selector")["fadeIn"](1000)` verwenden. Dies ermöglicht Ihnen, einen String mittels einer Bedingung oder einen Methodennamen als String zu verwenden, das wird in diesem Beispiel von Vorteil, da wir mittels des ternären Operators entweder die eine oder die andere Methode auf ein jQuery-Objekt anwenden können.

Der gleiche Trick wird bereits ein paar Zeilen zuvor angewendet, bei der Initialisierung des Scripts, wenn es darum geht, nach dem Laden der HTML-Seite die `themen`-Checkboxes auszublenden.

Es gibt noch eine Alternative zum Ausblenden der Checkboxes, Sie können sie deaktivieren. Wenn Sie die Zeilen mit der Anweisung `themen[]...` auskommentieren und die Kommentare der darunterliegenden Zeilen entfernen

```
//themen[isChecked ? "show" : "hide"]();
var themaInputs = themen.find("input")
    .attr("disabled", !isChecked);
```

sowie die Kommentare in den Zeilen tauschen

```
//themen[this.checked ? "fadeIn" : "fadeOut"](1000);
themaInputs.attr("disabled", !this.checked);
```

bleiben die Checkboxes immer sichtbar, sind aber nur klickbar, wenn die Checkbox `#newsletter` angeklickt wurde.

5.8.2 Formulare senden mit Ajax

Ein Formular mithilfe der Ajax Technologie ist die elegante Art, ein Formular zu verarbeiten. nach dem Versenden der Formulardaten muss nicht die gesamte HTML-Seite neu geladen werden. Per Ajax werden die Daten an den Server geschickt, die Rückmeldung kann vom `XMLHttpRequest`-Objekt entgegengenommen und in einen `<div>`-Container ausgegeben werden. Es wird keine Antwortseite geladen, sondern der Server verarbeitet die empfangenen Daten und schickt eine entsprechende Erfolgs- oder Fehlermeldung, die in einen `<div>`-Container geladen wird.

Sie können das vorangegangene Beispiel verwenden, Sie müssen nur wenige Zeilen ändern, und zusätzlich eine PHP-Datei anlegen. Dazu gleich mehr, betrachten Sie als erstes die Änderungen der HTML-Seite. Zuerst müssen Sie noch ein Plugin inkludieren, das dem Validate-Paket beiliegt:

```
<script type="text/javascript"
    src="js/jquery.form.js"></script>
```

Das Plugin stellt die Methode `.ajaxSubmit()` zur Verfügung, damit Sie via Ajax Formulardaten senden können. Das Plugin wurde von Mike Alsup, einem Mitglied des jQuery Plugin Teams, bereitgestellt. Sie können es auch unter <http://jquery.malsup.com/form/> herunterladen.

Dem Form-Element müssen Sie nun einen Wert für das `action`-Attribut vergeben, es ist der Aufruf der PHP-Seite, die noch eingeführt werden wird:

```
...
<form id="form" method="post" action="form.php">
...
```

Des Weiteren benötigen Sie noch die Container, in denen die Ajax-Ergebnisse, bzw. Statusmeldungen angezeigt werden sollen:

```
...
<div id="msg">Bitte füllen Sie das Formular aus.</div>
...
<div id="log"> </div>
...
```

Vielleicht fügen Sie noch den Container `#msg` vor dem Formular ein und den `#log`-Container nach dem Formular.

Zum jQuery-Code müssen Sie im Wesentlichen den `submitHandler` aus dem letzten Beispiel verändern:

```
...
submitHandler: function(form) {
    $(form).ajaxSubmit({
        target: "#msg"
    });
},
...
```

Listing 5.70 JavaScript-Code – Formular mit jQuery und Ajax (Ausschnitt 1)

Sie fügen die Methode `.ajaxSubmit()` hinzu, übergeben das Argument `form` als Formular an das Objekt, und mit `target` bestimmen Sie, in welchen Container

die Serverantwort gesetzt werden soll. Wenn Sie dem Benutzer noch einen Status über den Ajax-Ladevorgang zeigen wollen, fügen Sie am Anfang des Scripts, aber innerhalb der `.ready()`-Methode, folgende Zeilen ein:

```
...
var msg = $("#msg");
var log = $("#log");
$(log).ajaxStart(function() {
    log.html("loading...");
}).ajaxStop(function() {
    log.html("loading beendet.");
}).ajaxError(function(a, b, e) {
    log.html(e);
});
...
```

Listing 5.71 JavaScript-Code – Formular mit jQuery und Ajax (Ausschnitt 2)

Wenn der Ladevorgang startet, wird eine entsprechende Meldung ausgegeben, wenn er abgeschlossen ist, ebenso. Ebenfalls eine Fehlermeldung würde mit der Methode `ajaxError()` ausgegeben werden.

Die PHP-Seite nimmt die gesendeten Daten entgegen und gibt eine Mitteilung zurück. Beachten Sie dabei, dass dieses Beispiel in einer Serverumgebung mit PHP Interpreter laufen muss, damit das PHP-Script ausgeführt werden kann.

Der PHP-Code könnte folgendermaßen aussehen:

```
<?php $name = $_REQUEST['name'];
    $vorname = $_REQUEST['vorname'];
    if($name && $vorname) {
        echo "Sehr geehrte(r) $vorname $name,"
            . "Ihre Daten wurden versendet";
    }
    else {
        echo "Die Daten wurden nicht an den Server gesendet.";
    }
?>
```

Listing 5.72 PHP-Code – Formular mit jQuery und Ajax

Der Server nimmt mit der Variablen `$_REQUEST` die Parameter entgegen, und bei Erfolg wird ein String mit der Methode `echo` zurückgegeben. Mehr soll an dieser Stelle nicht passieren. In einem realen Beispiel würden die Daten entweder in einer Datenbank gespeichert oder an eine E-Mail-Adresse weitergeleitet.

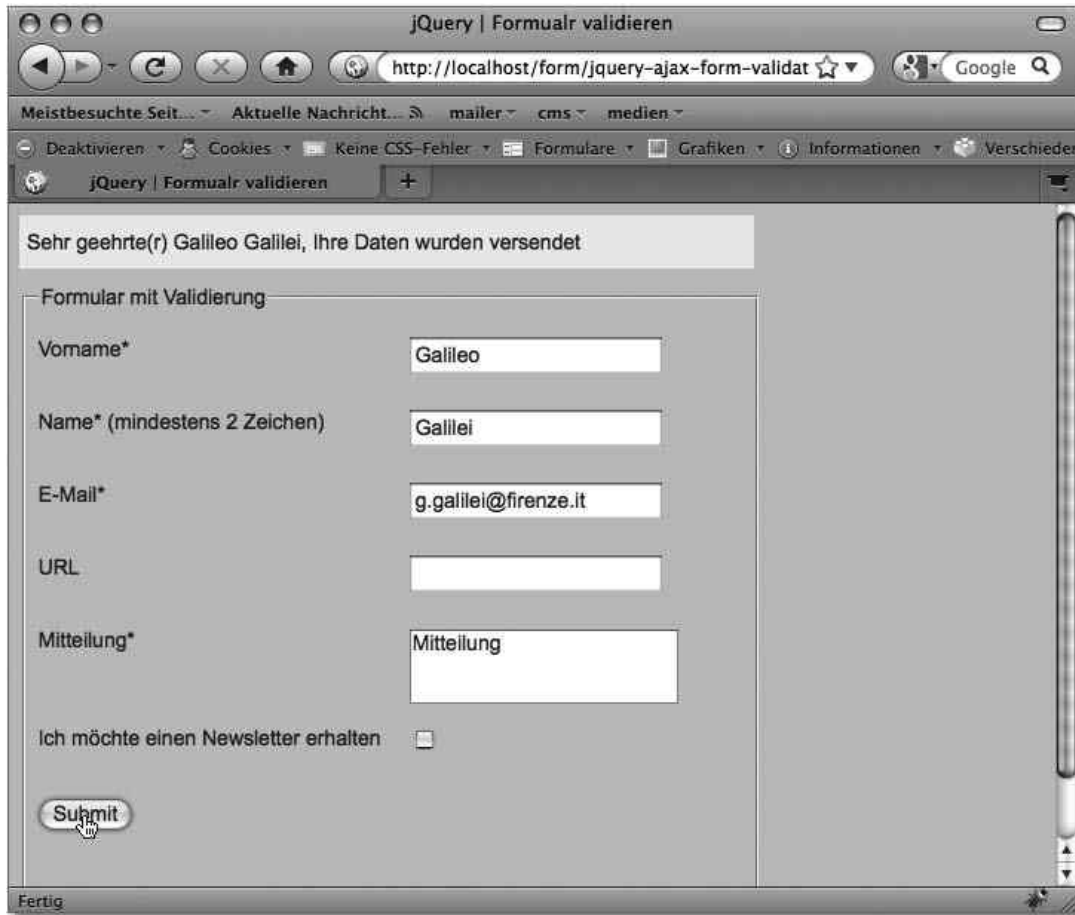


Abbildung 5.33 Formular senden mit Ajax

Im nächsten Schritt können Sie die Eingabe eines Datums bequemer gestalten. Sie werden dieses Formular um einen Datepicker erweitern.

5.8.3 Datepicker

Wie sieht eigentlich ein Datepicker aus? Es ist ein integrierter Kalender, der sich auf Wunsch öffnet, mit dem zwischen Monaten oder Jahren navigiert und ein Datum ausgewählt werden kann. Einerseits gestaltet ein Datepicker die Eingabe eines Datums wesentlich bequemer, zum anderen hilft er, sicherzustellen, dass die Eingabe des Datumsformats auch korrekt sein wird.

Dazu werden Sie wieder auf das jQuery UI-Plugin zurückgreifen. Binden Sie es in den Header der HTML-Seite ein.

Zuerst binden Sie das Theme ein, das Sie verwenden wollen. Das Theme finden Sie im Downloadpaket des jQuery UI-Plugins. Kopieren Sie das Theme in Ihren Projektordner.

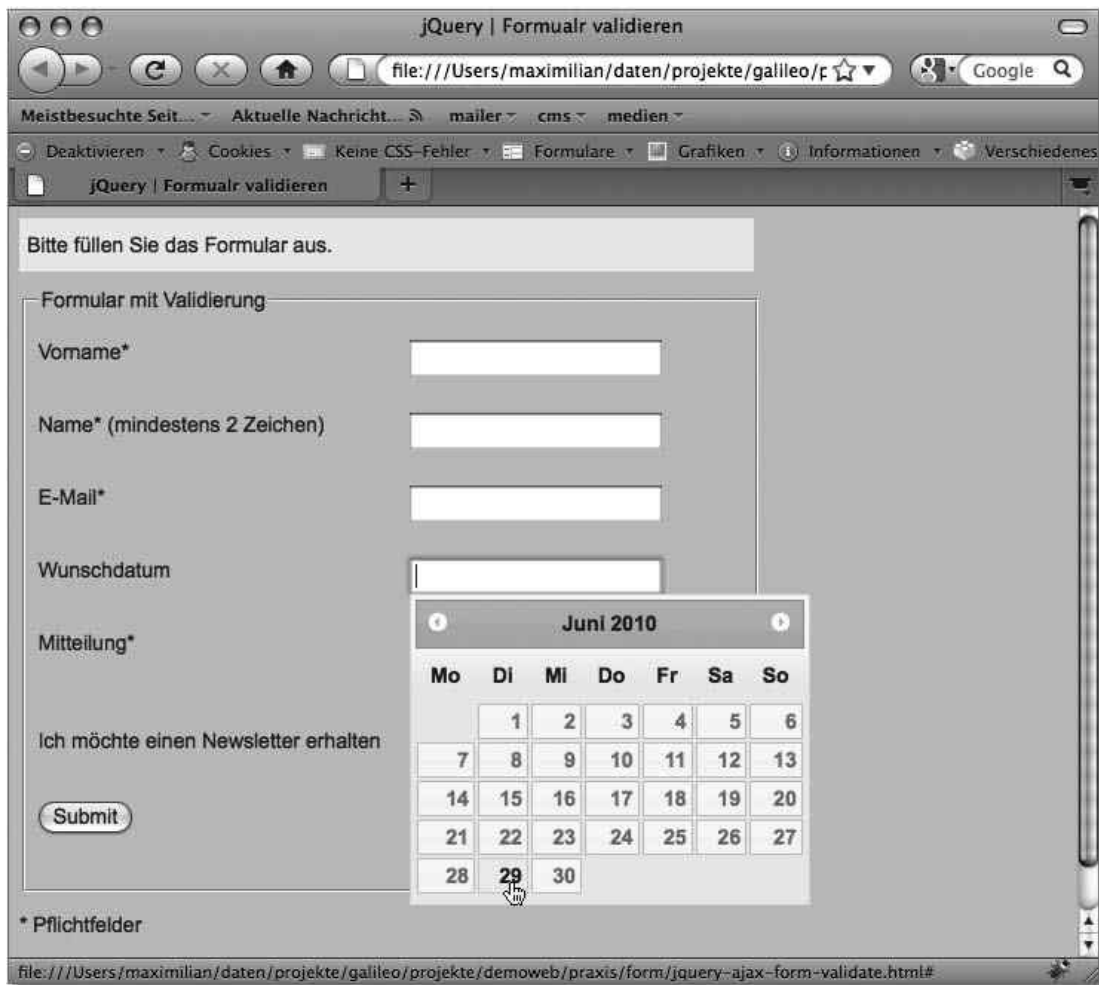


Abbildung 5.34 jQuery UI-Datepicker

Wie Sie Ihr Downloadpaket konfigurieren, erfahren Sie in Abschnitt 5.7, »jQuery UI«:

```
<link type="text/css"
      href="css/ui-lightness/jquery-ui-1.8.2.custom.css"
      rel="stylesheet" />
```

Außerdem soll erreicht werden, eine deutsche Lokalisierung einzubinden, damit die Monatsnamen, Wochentage in deutscher Sprache angezeigt werden. Möglicherweise möchten Sie alternativ auch eine französische Lokalisierung anbieten. Also werden Sie auch noch die Lokalisierungen für Deutschland und Frankreich einbinden. Sie finden die Dateien innerhalb des Downloadpakets des jQuery UI-Plugins im Verzeichnis *development-bundle/ui/i18n*. Legen Sie innerhalb Ihres Projektverzeichnisses das Verzeichnis *i18n* an, und kopieren Sie die gewünschten Sprachdateien dorthin:

```

<script type="text/javascript"
    src="js/jquery-ui-1.8.2.custom.js">
</script>
<script type="text/javascript"
    src="js/i18n/jquery.ui.datepicker-de.js">
</script>
<script type="text/javascript"
    src="js/i18n/jquery.ui.datepicker-fr.js">
</script>

```

Eine Kleinigkeit sollten Sie im HTML-Teil noch ändern, fügen Sie oberhalb des Input-Elements mit dem ID `#date` noch ein Select-Element hinzu:

```

<p>
  <label for="lang">Sprache</label>
  <select id="lang">
    <option value="">Default</option>
    <option value="fr">French (Français)</option>
    <option value="de" selected="selected">
      German (Deutsch)
    </option>
  </select>
</p>

```

Dieses Select-Element dient der Auswahl der anzuzeigenden Sprachdatei, die notwendige Logik findet sich wie immer in der JavaScript-Datei. Sie erweitern das vorhergehende Beispiel um einige Zeilen:

```

$("#date").datepicker($.datepicker.regional['de']);
//Event onChange
$("#lang").change(function() {
    $('#date').datepicker('option',
                          $.datepicker.regional[$(this).val()]);
});

```

Listing 5.73 JavaScript-Code – jQuery UI-Datepicker (Ausschnitt)

Mit `$("#date").datepicker()` initialisieren Sie den Datepicker und legen die Lokalisierung mit `$.datepicker.regional[]` fest. Der Selektor `$("#date")` bestimmt, für welches Input-Element der Datepicker initialisiert werden soll. Mit dem Event `.change()` wird der Datepicker neu mit einer anderen Lokalisierung initialisiert. Den Wert bezieht der Handler über `$(this).val`, also den Wert des ausgewählten Option-Elements. Mehr Konfiguration ist in diesem Beispiel nicht notwendig. Nichtsdestotrotz: Die Methode `.datepicker()` hält selbstverständlich viele Optionen bereit, um Datumsformate zu verwenden und deren Funktionalität zu beeinflussen. Auf der Seite <http://jqueryui.com/demos/datepicker/> wer-

den Sie fündig, wenn Sie weitere Optionen erkunden wollen. Zudem sind im Downloadpaket des jQuery UI-Plugins weitere Demos zu finden.

5.8.4 Nächste Ausbaustufe: Autocomplete

Jetzt werden Sie ein weiteres Widget aus dem Programm der jQuery UI-Plugins kennenlernen.

Wenn der Benutzer mindestens zwei Zeichen in ein Input-Feld `#city` eingibt, sollen über JSONP Daten geladen werden, die ihm eine Reihe von Datensätzen zeigen, die er für das gewünschte Ziel auswählen kann. Es öffnet sich unterhalb des Feldes `STADT` ein Auswahlménü mit Städtenamen, die der Benutzer übernehmen kann.

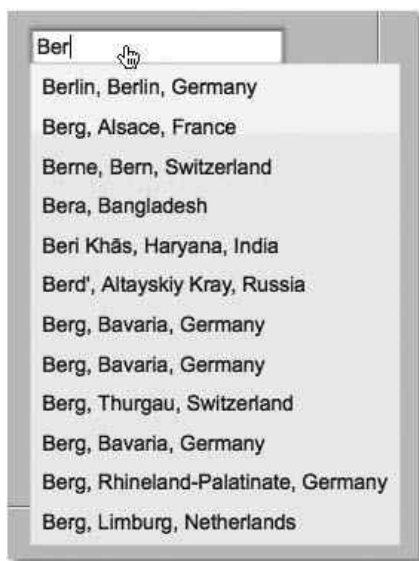


Abbildung 5.35 jQuery UI Autocomplete

Nun kommen die Daten nicht einfach aus dem Nichts. Es gibt einen Service, <http://www.geonames.org>, der kostenlos Geodaten zur Verfügung stellt. Die Datenbank umfasst ca. 8 Millionen Orte, zu jedem Ort können Sie Geocodes, Postleitzahlen, Einwohnerzahlen und viele andere Daten erfragen.

Geonames bieten auch Webservices an, mit denen Sie die Daten strukturiert im JSON-Format beziehen können. Der Aufruf sieht folgendermaßen aus:

```
http://ws.geonames.org/
searchJSON?featureClass=p&style=full&maxRows=12&name_startsWith=ber
```

Wobei `ber` der Suchstring ist, mit dem der Städtename beginnen soll. Diese Suchanfrage wird sowohl »Berlin« als auch »Bera« (in Bangladesh) zurückgeben, maximal aber nur 12 Orte (`maxRows`). Der Parameter `featureClass=P` gibt an, dass Angaben zu bewohnten Orten ausgegeben werden sollen (»Populated Place

Features«). Der Parameter `style` gibt an, in welchem Umfang die Daten ausgegeben werden sollen, hier wird `full` benötigt.

Eine *Dokumentation* des hier verwendeten Webservices finden Sie unter

<http://www.geonames.org/export/geonames-search.html>.

Ein Auszug einer JSON-Anfrage könnte folgendermaßen aussehen:

```
{
  "totalResultsCount": 14526,
  "geonames": [
    ...
    {
      "countryName": "Germany",
      "adminCode1": "16",
      "fclName": "city, village,...",
      ...
      "countryCode": "DE",
      "lng": 13.410530090332,
      "adminName2": "",
      "adminName3": "",
      ...
      "fcl": "P",
      "continentCode": "EU",
      "name": "Berlin",
      "geonameId": 2950159,
      "lat": 52.5243681651343,
      "adminName1": "Berlin",
      "population": 3426354
    },
    ...
  ]
}
...
```

Listing 5.74 JSON-Daten (Ausschnitt)

Hinweis

Falls Sie Abschnitt 4.12, »Ajax & JSON«, übersprungen haben sollten, schlagen Sie noch einmal nach, denn dort finden Sie weitere Informationen zum `XMLHttpRequest`-Objekt und auch über `JSON(P)`.

Doch nun noch einmal zurück auf `START`. Legen Sie innerhalb des Formulars aus den letzten Beispielen ein neues `Input`-Element an:

```
...
<p>
  <label for="city">Stadt</label>
  <input id="city" name="city" />
</p>
...
```

Und fügen Sie in das Script, das Sie in den letzten Beispielen angelegt haben, innerhalb des `$(document).ready()` Blocks, folgende Zeilen ein:

```
...
$("#city").autocomplete({
  source: function(request, response) {
    $.ajax({
      url: "http://ws.geonames.org/searchJSON",
      dataType: "jsonp",
      data: {
        featureClass: "P",
        style: "full",
        maxRows: 12,
        name_startsWith: request.term
      },
      success: function(data) {
        response($.map(data.geonames, function(item) {
          return {
            label: item.name +
              (item.adminName1 ? ", " + item.adminName1 : "") +
              ", " + item.countryName,
            value: item.name
          }
        })))
      }
    })
  },
  minLength: 2
});
...
```

Listing 5.75 JavaScript-Code – jQuery UI Autocomplete (Ausschnitt)

Diese Zeilen entstammen der Demo des Autocomplete Widgets, Sie fügen sie mehr oder weniger so ein, wie Sie sie in der Dokumentation vorfinden. Die Optionen der Methode `.autocomplete()` fallen vergleichsweise übersichtlich aus, Sie benötigen `source` und `minLength`.

Sie rufen die Methode `.autocomplete()` für das Input-Element mit dem ID `#city` auf, den Sie im Schritt vorher angelegt haben, und geben mit `url` die Datenquelle an, in diesem Beispiel ist es eine Ajax-Quelle. Mit der Option `minLength` geben Sie die Anzahl Zeichen an, die mindestens eingegeben werden muss, bevor die Anfrage an die Datenquelle gestartet wird.

Die Methode `$.ajax()` übergibt die Parameter `featureClass`, `style`, `maxRows` und `name_startsWith` als Datenpaket an den Server, wobei der Wert von `name_startsWith` der Suchbegriff des Input-Feldes `#city` ist.

Die Funktion `success` wird ausgeführt, wenn die JSON-Daten geladen wurden. Die Methode `$.map()` iteriert über alle Items des `data.geonames`-Objekts. Für jeden gefundenen Ort werden der `name` und gegebenenfalls auch der `adminName` sowie der `countryName` zurückgegeben.

Vergleichen Sie die anonyme Funktion innerhalb der Methode `$.map()` mit dem Auszug der JSON-Datei: Für dieses eine Item wird zurückgegeben: »Berlin«, »Berlin«, »Germany«. Den Rest erledigt die Methode `.autocomplete()`. Sie baut das Result-Set zu einem komplexen Auswahlmenü zusammen. Events innerhalb des Autocomplete Widgets sorgen dafür, dass die gewünschte Auswahl, hier ein Städtenamen, im Input-Feld `#city` landet. Selbstredend können Sie das Aussehen des Autocomplete-Auswahlmenüs über Themes steuern.

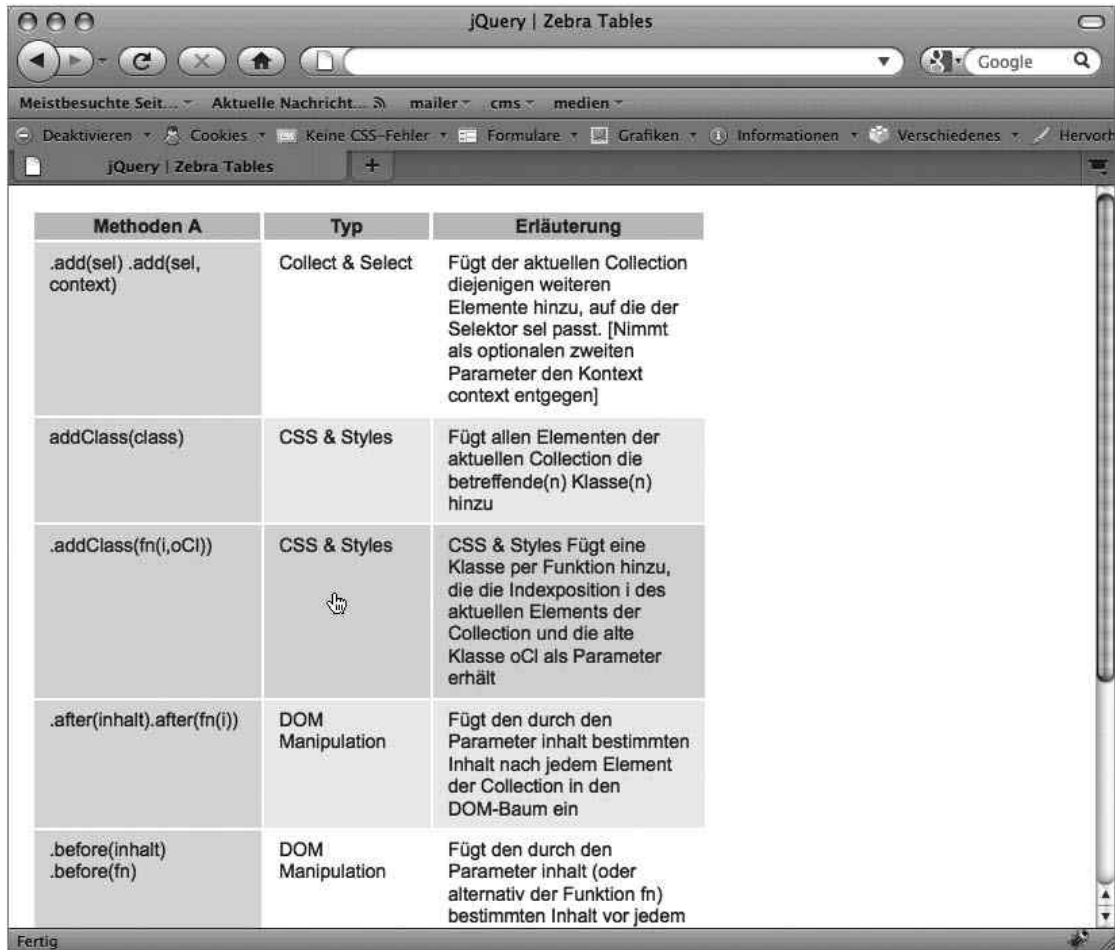
Dieses Beispiel zeigt, wie Sie mit einer äußerst geringen Anzahl von Anweisungen ein komplexes User Interface steuern können. Vergessen Sie nicht, in unkomprimiertem Zustand beträgt das Autocomplete Widget ca. 500 Zeilen, die Funktion `$.ajax()` tut ihr Übriges. Mit wenigen Anweisungen steuern Sie einen hochkomplexen Vorgang.

5.9 Flexiblere Tabellen mit jQuery

jQuery eignet sich auch dafür, Manipulationen in Tabellen vorzunehmen. Sie haben bereits gesehen, dass Sie über Ajax & Co sehr leicht Daten importieren und als Tabellen ausgeben lassen können. Mit jQuery können Sie aber noch mehr erreichen, Sie können Daten in Tabellen sortieren, Sie können Datensätze verschieben, ausblenden, Sie können Daten per Drag & Drop sortieren, Sie können Zeilen, Spalten in abwechselnden Farben gestalten, was mit CSS nur über einzelne Klassen für einzelne TR-Elemente realisierbar wäre.

5.9.1 Die Zebra-Tabelle

Mit jQuery können Sie mit einem »Muster« beispielsweise mit dem Selektor `$("tr:even")` alle geraden Zeilen einfärben. Dies soll auch das erste Tabellenbeispiel sein, Zusätzlich soll jeder Zeile ein `.hover()`-Event zugewiesen werden, mit dem eine Zeile zusätzlich eingefärbt werden soll.



Methoden A	Typ	Erläuterung
<code>.add(sel).add(sel, context)</code>	Collect & Select	Fügt der aktuellen Collection diejenigen weiteren Elemente hinzu, auf die der Selektor sel passt. [Nimmt als optionalen zweiten Parameter den Kontext context entgegen]
<code>addClass(class)</code>	CSS & Styles	Fügt allen Elementen der aktuellen Collection die betreffende(n) Klasse(n) hinzu
<code>.addClass(fn(i,oCi))</code>	CSS & Styles	CSS & Styles Fügt eine Klasse per Funktion hinzu, die die Indexposition i des aktuellen Elements der Collection und die alte Klasse oCi als Parameter erhält
<code>.after(inhalt).after(fn(i))</code>	DOM Manipulation	Fügt den durch den Parameter inhalt bestimmten Inhalt nach jedem Element der Collection in den DOM-Baum ein
<code>.before(inhalt).before(fn)</code>	DOM Manipulation	Fügt den durch den Parameter inhalt (oder alternativ der Funktion fn) bestimmten Inhalt vor jedem

Abbildung 5.36 Tabelle mit Zebra-Muster und Hover

Zuerst legen Sie wieder eine HTML-Seite an. Sie benötigen zusätzlich zum jQuery Framework die jQuery Effects Core. Der Einfachheit halber binden Sie das gesamte UI-Plugin ein. Falls Sie dieses Beispiel in einer Produktivumgebung laufen lassen, können Sie die jQuery UI-Datei später immer noch auf das Nötigste reduzieren. In Abschnitt 5.7, »jQuery UI«, finden Sie die Verweise auf den UI-Konfigurator:

```
<!DOCTYPE html>
<html>
<head>
<title>jQuery | Zebra Tables</title>
<link type="text/css" href="css/table-zebra.css" rel="stylesheet" />
```

```

<script type="text/javascript" src="js/jquery-1.4.2.js">
</script>
<script type="text/javascript" src="js/jquery-ui-1.8.2.custom.js">
</script>
<script type="text/javascript" src="js/table-zebra.js">
</script>
</head>
<body>
<table id="zebra">
  <thead>
    ...
  </thead>
  <tbody>
    <tr>
      <td>...</td> <td>...</td> <td>...</td>
    <tr>
      ...
    </tbody>
  </table>
</body>
</html>

```

Listing 5.76 HTML-Gerüst Zebra-Tabelle

Legen Sie nun eine JavaScript-Datei *table-zebra.js* für die jQuery-Anweisungen an. Die CSS-Angaben werden hier übersprungen, da sie lediglich das Aussehen verfeinern:

```

function zebra(obj,colors) {
  if(!obj.length) return;
  var tr = $("tr", obj);
  var td = $("td:first-child",tr);
  var even = $("tr:even",obj);
  even.css("background-color",colors.even);
  td.css("background-color",colors.column);

  tr.hover(function() {
    $(this).animate({
      backgroundColor: colors.over
    }).find("td:first-child").animate({
      backgroundColor: colors.over
    });
  },
  function() {
    var color;

```



```

        $(this).is(":nth-child(even)") ?
            color = colors.even :
            color = colors.odd ;
        $(this).animate({
            backgroundColor: color
        }).find("td:first-child").animate({
            backgroundColor: colors.column
        });
    });
}
$(document).ready(function(){
    var colors = {
        over: "#DDDDEC",
        odd: "#FFFFFF",
        even: "#EEEEEE",
        column: "#ECDDDD"
    }
    zebra($("#zebra"),colors);
});

```

Listing 5.77 JavaScript-Code – Zebra Tabelle

Zuerst deklarieren Sie im `$(document).ready()`-Block das Objekt `colors`, in dem Sie die benötigten Farben speichern. Dieses Objekt wird an die Funktion `zebra()` als Argument übergeben.

In der Funktion `zebra()` werden als Erstes benötigte Variablen deklariert, und mit den Zeilen

```

even.css("background-color",colors.even);
td.css("background-color",colors.column);

```

werden alle geraden Zeilen sowie die erste Spalte eingefärbt.

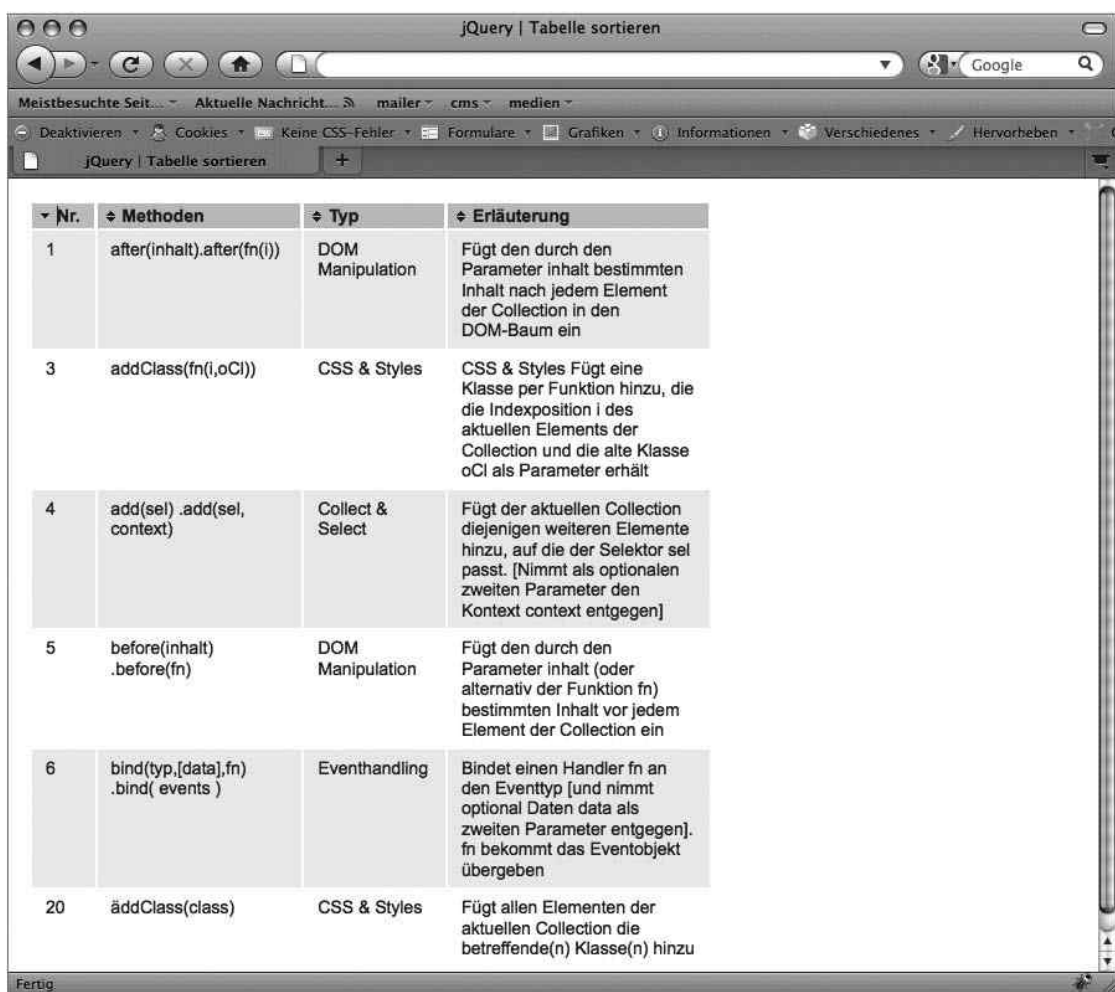
Im Anschluss wird der Event Handler `.hover()` mit seinen beiden anonymen Funktionen initialisiert. Der Wechsel der Farben wird über eine Coloranimation realisiert, dafür benötigen Sie die Effects Core, Sie haben ja bereits im `<head>` das jQuery UI-Plugin eingebunden.

Eigentlich ein einfaches Beispiel. Der einzige Pferdefuß verbirgt sich vielleicht in dem Selektor `.is(":nth-child(even) ")`. Warum wird hier nicht einfach der Selektor `:even` verwendet? Der Grund ist, dass `:even` und `:odd` das gesamte Result-Set betrachten. Im Falle von `$(this)` ist das gerade ein einziges Element. Der Ausdruck `$(this).is(":even")` wird für jedes TR-Element `true` zurückgegeben,

während `$(this).is(":nth-child(even)")` die Position des Elements in Bezug auf alle Nachbarelemente berechnet.

5.9.2 Die Tabelle sortieren

Ein weiterer Anwendungsfall ist das Sortieren von Tabelleninhalten. Sie können selbstverständlich die Daten serverseitig sortieren, und in eine HTML-Seite ausgeben lassen. Der Vorteil von clientseitiger Sortierung besteht darin, dass die Daten nicht neu geladen werden müssen. Sie müssen nur den DOM-Baum innerhalb einer Tabelle auseinandernehmen, genauer gesagt zeilenweise, anschließend sortieren und in einer anderen Reihenfolge ausgeben.



Nr.	Methoden	Typ	Erläuterung
1	<code>after(inhalt).after(fn(i))</code>	DOM Manipulation	Fügt den durch den Parameter <code>inhalt</code> bestimmten Inhalt nach jedem Element der Collection in den DOM-Baum ein
3	<code>addClass(fn(i,oCI))</code>	CSS & Styles	CSS & Styles Fügt eine Klasse per Funktion hinzu, die die Indexposition <code>i</code> des aktuellen Elements der Collection und die alte Klasse <code>oCI</code> als Parameter erhält
4	<code>add(sel) .add(sel, context)</code>	Collect & Select	Fügt der aktuellen Collection diejenigen weiteren Elemente hinzu, auf die der Selektor <code>sel</code> passt. [Nimmt als optionalen zweiten Parameter den Kontext <code>context</code> entgegen]
5	<code>before(inhalt) .before(fn)</code>	DOM Manipulation	Fügt den durch den Parameter <code>inhalt</code> (oder alternativ der Funktion <code>fn</code>) bestimmten Inhalt vor jedem Element der Collection ein
6	<code>bind(typ,[data],fn) .bind(events)</code>	Eventhandling	Bindet einen Handler <code>fn</code> an den Eventtyp [und nimmt optional Daten <code>data</code> als zweiten Parameter entgegen]. <code>fn</code> bekommt das Eventobjekt übergeben
20	<code>addClass(class)</code>	CSS & Styles	Fügt allen Elementen der aktuellen Collection die betreffende(n) Klasse(n) hinzu

Abbildung 5.37 Tabellendaten sortieren

Wenn Sie sich in den jQuery Plugin Repositorys umsehen, werden Sie feststellen, dass es zwar einige Plugins zu diesem Thema gibt. So richtig überzeugen sie jedoch nicht, obwohl die Autoren durchaus versiert vorgegangen sind.

Teilweise werden sie nicht weiterentwickelt, oder sie funktionieren nur für bestimmte Anwendungsfälle. Es gibt ein Plugin, das Bestandteil des jQuery UI Frameworks war, aber es ist seit der Version 1.5 nicht mehr enthalten. Es wird wohl in einer zukünftigen Version des UI Frameworks innerhalb eines Grid-Plugins wieder eine Sortierfunktion geben. Im Moment müssen Sie entweder eines der noch im weiteren Verlauf beschriebenen Grid-Plugins benutzen, das ehemalige UI-Plugin von Christian Bach verwenden, siehe <http://tablesorter.com/docs/>, oder ein eigenes Script entwerfen. Für kleinere Anwendungsfälle mag folgendes Script interessant und ausbaufähig sein. Setzen Sie dazu zunächst das HTML-Gerüst auf:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery | Tabelle sortieren</title>
  <link type="text/css"
        href="css/tablesorter.css" rel="stylesheet" />
  <script type="text/javascript" src="js/jquery-1.4.2.js">
  </script>
  <script type="text/javascript" src="js/tablesorter.js">
  </script>
</head>
<body>
  <table id="sortable">
    <thead>
      <tr>
        <th class="numeric">Nr.</th>
        <th>Methoden</th>
        <th>Typ</th>
        <th>Erläuterung</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td>1</td>
        <td>... </td>
        <td>...</td>
        <td>...</td>
      </tr>
      ...
    </tbody>
  </table>
</body>
</html>
```

Listing 5.78 HTML-Gerüst – Tabelle sortieren

Legen Sie eine Tabelle mit Inhalten an, hier sind sie nur angedeutet. Es sollten schon einige Datensätze sein, um zu sehen, ob Ihr Script auch nach den richtigen Kriterien sortiert. In der ersten Spalte werden Sie Ziffern eintragen, Sie werden später bei den Sortierfunktionen zwischen Ziffern und Strings unterscheiden. Binden Sie im Head wieder die nötigen JavaScript- und CSS-Dateien ein, und fertig ist das Grundgerüst. Um dem Benutzer ein Feedback auf seine Mausektionen zu geben, benötigen Sie einige CSS-Anweisungen:

```
.even {
  background-color:#EEEEEE;
}
.hover {
  background-color:#EEEEFF;
}
th.asc {
  background-image: url(../images/sort_asc.gif);
  background-color: #CCCCFF;
}
th.desc {
  background-image: url(../images/sort_desc.gif);
  background-color: #CCCCFF;
}
th {
  background-image: url(../images/sort_default.gif);
  cursor: pointer;
  background-repeat: no-repeat;
  background-position: center left;
  padding-left: 20px;
  text-align:left;
}
```

Listing 5.79 CSS Styles – Tabelle sortieren

Die Klasse `.even` benötigen Sie, um die Zebrastreifen auf die einzelnen Tabellenzeilen zu legen, die Klasse `.hover` benötigen Sie, um die aktuell mit der Maus überflogene Zeile zu markieren. Beim Klick auf einen Eintrag im Tabellenheader soll die Sortierung durchgeführt werden. Damit der Benutzer über die Aktion eine Rückmeldung erhält, auch in welche Richtung sortiert wird (aufsteigend oder absteigend), werden die Klassen `.asc` (aufsteigend) und `.desc` (absteigend) später via jQuery gesetzt. Per Default wird der Tableheader mit einem Hintergrundbild `sort_default.gif` versehen. Im folgenden Schritt wird die Datei `tablesorter.js` gefüllt:

```

var sortTable = function(obj) {
    if(!obj.length) return;
    var Helper = {
        replaceSpecials: function(str) {
            var specials = {"ö":"oe","ä":"ae","ü":"ue","ß":"ss"};
            for (var char in specials) {
                var regExp = new RegExp(char,"g");
                str = str.replace(regExp,specials[char]);
            }
            return str;
        }
    }
    obj.each(function() {
        $('tbody tr:even', obj).addClass("even");
        var th = $('th', obj);
        th.each(function(index) {
            var asc = false;
            $(this).click(function() {
                asc ? asc = false : asc = true;
                th.removeClass("desc asc");
                asc?$(this).addClass("asc"):$(this).addClass("desc");
                var tableRows = $('tbody > tr',obj).get();
                var isNumeric = $(this).hasClass("numeric");
                tableRows.sort(function(a, b) {
                    var sortA = Helper.replaceSpecials($(a)
                        .children('td')
                        .eq(index).text().toLowerCase();
                    var sortB = Helper.replaceSpecials($(b)
                        .children('td')
                        .eq(index).text().toLowerCase();

                    var num;
                    asc ? num = 1 : num = -1;
                    if (isNumeric) return asc ?
                        sortA - sortB :
                        sortB - sortA;
                    return (sortA == sortB) ? 0 :
                        (sortA > sortB) ? num : num * -1;
                });
                for (var tr in tableRows) {
                    $('tbody',obj).append(tableRows[tr]);
                }
                $('tbody tr', obj).removeClass("even");
                $('tbody tr:even', obj).addClass("even");
            });
        });
    });
}

```

```

    });
}

$(document).ready(function() {
    sortTable($('table#sortable'));
});

```

Listing 5.80 JavaScript-Code – Tabelle sortieren

Um die Sortierung vorzunehmen, bedienen Sie sich der JavaScript-Methode `.sort()`. Sie können mit dieser Methode Arrays sortieren. Wenn Sie lediglich absteigend alphabetisch sortieren wollen, benötigen Sie kein Argument. Wenn Sie aber eigene Sortierkriterien anwenden wollen, übergeben Sie der Methode eine Funktion als Argument. Dieser Callback-Funktion wiederum übergeben Sie zwei Vergleichswerte `a` und `b`. Die Methode `.sort()` iteriert über alle Elemente, indem sie bei jedem Durchlauf Element `a` und `b` miteinander vergleicht.

Sie müssen als Erstes einen Array aus einem jQuery-Objekt, den Tabellenzeilen, generieren. Das erreichen Sie mit der Methode `$('tbody > tr',obj).get()`. Diese jQuery-Methode gibt ein Array mit DOM-Elementen zurück, und Sie können auf den Rückgabewert JavaScript-Methoden anwenden. In diesem Fall werden Sie wie gesagt die `sort()`-Methode anwenden.

Als Erstes betrachten wir die alphabetische Sortierung. Sie werden diese in zwei Varianten benötigen, einmal aufsteigend, einmal absteigend sortiert. Sie können leider nicht die einzelnen Tabellenzellen im Array `tableRows` vergleichen, zuerst muss der zu sortierende Inhalt mit folgender Anweisung extrahiert werden:

```

($([Vergleiswerte a oder b]).children('td')
.eq(index).text().toLowerCase());

```

Diese Anweisung sucht innerhalb einer TR-Zeile das Kindelement TD mit dem Zähler `index`. Der `index` bestimmt die Spalte, die sortiert werden soll. Klickt man im Header das TH-Element mit dem `index 1`, wird das entsprechende TD-Element im TBODY mit dem `index 1` selektiert. Diesen extrahierten Vergleichswert übergeben Sie der Funktion `Helper.replaceSpecials()`. Diese Funktion ersetzt Umlaute. Wenn Sie die Funktion nicht einsetzen würden, würden beispielsweise bei einer alphabetischen Sortierung Worte mit dem Umlaut `ä` wie Ägypten am Ende der Tabelle landen, deshalb wird `a` mit `ae` etc. ersetzt.

Beachten Sie, es wird nicht der Text, der in den einzelnen Tabellenzellen steht ersetzt, sondern Sie generieren aus diesen Strings Sortierkriterien, nach denen die Zeilen sortiert werden.

Die Vergleichswerte werden in den beiden Variablen `sortA` und `sortB` abgelegt. In der folgenden Zeile wird das Sortierkriterium festgelegt, die Anweisung

```
asc ? num = 1 : num = -1;
```

legt fest, welchen Wert die Variable `num` erhalten soll, für aufsteigend den Wert 1, für absteigend den Wert -1.

Die Variable `asc` wird für jedes TH-Element einzeln bei jedem Klick von `true` auf `false` geändert:.

```
...
$(this).click(function() {
    asc ? asc = false : asc = true;
    ...
});
```

So wird bei jedem Mausklick für jede Spalte abwechselnd aufsteigend oder absteigend sortiert.

Die Variable `num` benötigen Sie in der Anweisung:

```
(sortA == sortB) ? 0 : (sortA > sortB) ? num : num * -1;
```

Diese Zeile ist die eigentliche Berechnung der Reihenfolge. Sie berücksichtigt folgende Fälle:

- ▶ Wenn `sortA` `sortB` entspricht, ändert sich nichts an der Reihenfolge, für diesen Vergleich wird 0 zurückgegeben.
- ▶ Wenn `sortA` ungleich `sortB` ist, gibt es zwei Möglichkeiten, erstens wenn `sortA` größer als `sortB` ist, soll `num` gesetzt werden.
- ▶ Zweitens, wenn `sortA` größer `sortB` nicht zutrifft, wird der negative Wert von `num` verwendet.

Wobei `num` wie bereits gesehen, entweder den Wert 1 oder -1 erhalten kann. Mit dem Wert 1 wird das betreffende Element um eine Stelle nach vorn verschoben, mit -1 eine Stelle nach hinten. Mit dem Wert 0 verändert sich nichts. Anschließend werden die nächsten Elemente im Array verglichen, gegebenenfalls verschoben etc.

Nun ist der Array `tableRows` umsortiert und kann innerhalb der `for-in`-Schleife mit `.append()` in die Tabelle zurückgeschrieben werden.

Wenn Sie nun versuchen, mit dieser Sortierung einen Array mit numerischen Werten zu sortieren, werden Sie folgende Reihe erhalten: [1,10,11,2,3,4,5,...]. Vom Standpunkt einer alphabetischen Sortierung mag das korrekt erscheinen, Sie werden aber eine Sortierung nach dem Muster [1,2,3,4,

5...10,11] bevorzugen. Zuerst müssen Sie ein Kriterium festlegen, dass für eine bestimmte Spalte eine alphanumerische Sortierung vorgenommen werden soll. Vergeben Sie im HTML-Quelltext dem TH-Element die Klasse `numeric`, in diesem Beispiel die Spalte mit dem Spaltennamen `Nr`:

```
...
<th class="numeric">Nr.</th>
...
```

In der JavaScript-Funktion werden Sie für das Element, das geklickt wurde, die Variable

```
...
var isNumeric = $(this).hasClass("numeric");
...
```

setzen. Besitzt die Variable den Wert `true`, wird die Anweisung

```
...
if (isNumeric) return asc ? sortA - sortB : sortB - sortA;
...
```

ausgeführt. Die Sortierung erfolgt nun wie gewünscht. Auch hier werden anschließend die Zeilen mit `.append()` in die Tabelle geschrieben und zum Abschluss die Zebrastreifen neu gesetzt. Fehlen noch die Anweisungen für die Benutzerfeedbacks beim Klick auf eine Tabellenüberschrift, also das Setzen und Entfernen der CSS-Klassen `desc` und `asc` der TH-Elemente, wobei zuerst alle Klassen entfernt werden, bevor eine neue gesetzt wird:

```
th.removeClass("desc asc");
asc ? $(this).addClass("asc") : $(this).addClass("desc");
```

sowie das initiale Setzen der Zebrastreifen mit der Zeile

```
$('tbody tr:even', obj).addClass("even");
```

Mit wenigen Zeilen können Sie dem Benutzer eine Sortierfunktion zur Verfügung stellen, das macht besonders dann Sinn, wenn Sie größere Datenmengen visualisieren müssen. Nur, bei größeren Datenmengen werden bekanntlich HTML-Seiten sehr lang, sodass man schnell die Übersicht verliert, da hilft die beste Sortierfunktion nichts. Deshalb gehört fast zwingend eine Paginierung einer solchen Tabellenfunktion dazu. Im Folgenden werden Sie das Script um eine solche Funktionalität erweitern.

5.9.3 Paginierung von Tabellen

Paginierung bedeutet, dass eine längere Tabelle in mehrere Abschnitte unterteilt wird, und der Benutzer zwischen den Teilen hin- und herblättern kann.

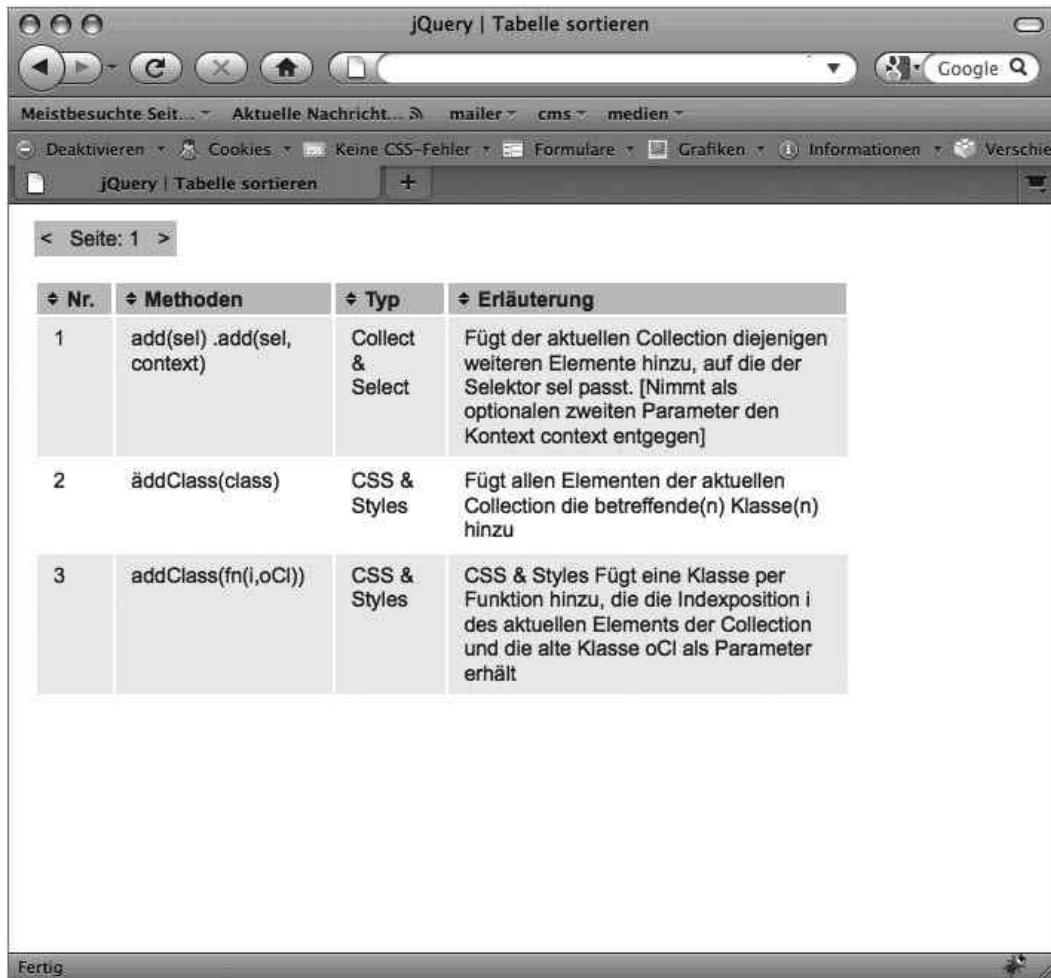


Abbildung 5.38 Tabelle paginieren

In diesem zugegebenermaßen kleinen Beispiel werden Sie die Zahl pro angezeigter Datensätze auf drei reduzieren; in anderen Anwendungsfällen werden das sicherlich zehn oder mehr sein, das werden Sie im Script festlegen oder das Script weiterentwickeln und dem Benutzer ein Pulldown-Menü bereitstellen, in dem er die Anzahl festlegen kann. Es wird Ihre Aufgabe sein, das ohne Anleitung zu implementieren.

Ebenfalls wird es Ihre Aufgabe sein, die Blätternavigation in das Script zu übernehmen. In diesem Beispiel werden Sie es hart in den HTML-Code einbetten, um das Script kompakt zu halten, aber wenn man die Methode der »progressiven Erweiterung« ernst nimmt, kann auch ein Benutzer ohne JavaScript die HTML-Seite benutzen, dann allerdings ohne Sortierfunktion, dann haben die Buttons zum

Vor- und Zurückblättern aber nichts im HTML-Gerüst zu suchen, sondern sollten per JavaScript in den DOM-Baum gesetzt werden. Mit der Methode `.html()` werden Sie das schnell selbst hinbekommen. Nun zurück zur HTML-Erweiterung:

```
...
<div id="tools">
  <a id="down" href="#">&lt; </a>
  <span id="msg"></span>
  <a id="up" href="#"> &gt;</a>
</div>
...
```

Listing 5.81 HTML-Gerüst – Tabelle paginieren (Ausschnitt)

Zur JavaScript-Datei fügen Sie folgenden Code hinzu:

```
var sortTable = function(obj) {
  if(!obj.length) return;
  var Helper = {
    replaceSpecials: function(str) {
      var specials = {"ö":"oe","ä":"ae","ü":"ue","ß":"ss"};
      for (var char in specials) {
        var regExp = new RegExp(char,"g");
        str = str.replace(regExp,specials[char]);
      }
      return str;
    },
    paginate: function() {
      tableRows = $('tbody tr',obj);
      msg.text("Seite: " + (current + 1));
      tableRows.hide()
        .slice(current * numMax,(current + 1) * numMax)
        .show();
    }
  }
  // pagination
  var current = 0;
  var numMax = 3;
  var tableRows = $('tbody tr',obj);
  var pages = Math.ceil(tableRows.length / numMax);
  var msg = $("#tools #msg");
  Helper.paginate();
  $("#up").click(function(e) {
    e.preventDefault();
    (current < pages -1) ?
```

```

        current += 1 : current = pages - 1;
        Helper.paginate();
    })
    $("#down").click(function(e) {
        e.preventDefault();
        (current > 0) ? current -= 1 : current = 0;
        Helper.paginate();
    })
    // sort
    obj.each(function() {
        $('tbody tr:even', obj).addClass("even");
        var th = $('th', obj);
        th.each(function(index) {
            var asc = false;
            $(this).click(function() {
                asc ? asc = false : asc = true;
                th.removeClass("desc asc");
                asc ? $(this).addClass("asc") : $(this).addClass("desc");
                var tableRows = $('tbody > tr', obj).get();
                var isNumeric = $(this).hasClass("numeric");
                tableRows.sort(function(a, b) {
                    var sortA = Helper.replaceSpecials($(a)
                        .children('td')
                        .eq(index).text().toLowerCase());
                    var sortB = Helper.replaceSpecials($(b)
                        .children('td')
                        .eq(index).text().toLowerCase());

                    var num;
                    asc ? num = 1 : num = -1;
                    if (isNumeric) return asc ?
                        sortA - sortB :
                        sortB - sortA;
                    return ( sortA == sortB) ? 0 :
                        ( sortA > sortB) ? num : num * -1;
                });
                for (var tr in tableRows) {
                    $('tbody', obj).append(tableRows[tr]);
                }
                $('tbody tr', obj).removeClass("even");
                $('tbody tr:even', obj).addClass("even");
                Helper.paginate();
            });
        });
    });
}

```

```
$(document).ready(function() {
    sortTable($('table#sortable'));
});
```

Listing 5.82 JavaScript Code – Tabelle paginieren

Die Funktion `paginate()` ist dafür zuständig, den gewählten Ausschnitt an Tabellenzeilen zu zeigen und den Rest zu verstecken:

```
tableRows.hide()
    .slice(current * numMax, (current + 1) * numMax)
    .show();
```

Zuerst werden alle Zeilen mit `.hide()` ausgeblendet. Mit der Methode `.slice()` wird die gewünschte Range selektiert und mit `.show()` angezeigt. Die Range wird berechnet mit den Argumenten »von bis«, wobei das »von«-Argument hier die aktuelle Seitenzahl mit der maximal für eine Seite angezeigten Zeilen bestimmt wird. Der »bis«-Wert errechnet sich, wie unschwer zu erkennen ist, aus der Seitenzahl +1, multipliziert mit der maximalen Anzahl der anzuzeigenden Zeilen. Stellen Sie sich vor, Sie würden die Range manuell eintragen, dann würden auf Seite 1, sofern Sie die maximale Anzahl auf 3 gesetzt haben, die Argumente folgendermaßen aussehen:

`.slice(0,3)`, für Seite 2 `.slice(3,6)`. Das ist die wichtigste Anweisung, mit der die Range gesteuert wird. Weiter unten werden über Click-Events die anzuzeigenden Seiten gesteuert.

Dazu brauchen Sie die Anzahl der Seiten:

```
var pages = Math.ceil(tableRows.length / numMax);
```

Mit der Anzahl der Tabellenzeilen, dividiert durch die Anzahl der anzuzeigenden Zeilen, ist es noch nicht ganz getan, Sie müssen das Ergebnis noch aufrunden, dies erreichen Sie mit `Math.ceil()`.

Wenn Sie aufwärts blättern wollen, erstellen Sie innerhalb des Click-Events folgende Anweisung:

```
(current < pages -1) ? current += 1 : current = pages -1;
```

Wenn die aktuelle Seitenzahl kleiner als die Gesamtzahl aller Seiten ist, setzen Sie `current` um einen Zähler nach oben, wenn die letzte Seite erreicht ist, bleibt `current` auf dem höchstmöglichen Zähler. Die `-1` benötigen Sie deswegen, weil Sie den Zähler bei 0 beginnen lassen müssen, die Variable `pages` zeigt dagegen die tatsächliche Anzahl der Seiten.

Für den »Nach unten«-Event verwenden Sie diese Anweisung:

```
...
(current > 0) ? current -= 1 : current = 0;
...
```

Wenn die aktuelle Seitenzahl größer als 0 ist, wird sie heruntersgesetzt, bei 0 angekommen, bleibt sie auf 0.

Vergessen Sie nicht, anschließend `Helper.paginate()` aufzurufen, damit die neu ausgewählte Range auch angezeigt wird. Das Gleiche gilt für den Fall, dass ein Benutzer die Sortierung ändert, anschließend wird `Helper.paginate()` aufgerufen, damit die Range erneut angezeigt wird.

Damit sind Sie für ein paar Anwendungsfälle gewappnet. Sie können für die Sortierung dieses Beispiel noch ausbauen, es gibt noch weitere Datentypen, wie Datum oder Währung, die gesondert zu berechnen sind, Sie können Daten per Ajax laden, dann bewegen Sie sich langsam in den Bereich von Grid-Plugins, die im nächsten Beispiel besprochen werden.

5.9.4 Grid-Plugins

Einige Tabellenmanipulationen haben Sie bereits kennengelernt. In so manchem Anwendungsfall werden Sortierung und Zebramuster genügen. Es existiert aber eine Reihe weiterer Tabellenfunktionen, die man zusammenfassend als Tabellen-Grids oder einfach Grids bezeichnet. Grids, Raster, Gitternetze sind eines der wichtigsten interaktiven Benutzerschnittstellen. Gerade dort, wo viele Daten visualisiert werden müssen, spielen sie eine zunehmend wichtige Rolle. Grids sind HTML-Tabellen, die mit zusätzliche Funktionalitäten ausgestattet sind, mit Sortierfunktion, Filterfunktionen, Funktionen zur Vergrößerung von Spalten bis hin zu Editierfunktionen, mit denen man Daten clientseitig bearbeitet. Ein besonderes Kriterium für Grids sind Ajax-Funktionen, mit den Daten geladen und wieder in einer Datenbank gespeichert werden können. Wenn man ein, zwei, drei Schritte weiterdenkt, können sich solche Widgets in absehbarer Zeit hin zu Online-Tabellenkalkulation entwickeln.

Nun muss man leider sagen, dass es derzeit kein Grid-Plugin auf Basis von jQuery gibt, das den Anforderungen und Ansprüchen des jQuery UI-Teams genügen würde. Und genau hier, im Bereich von jQuery UI, wäre ein solches Widget hervorragend untergebracht. Das Ziel soll es nach den Diskussionen innerhalb des jQuery UI-Teams sein, nicht nur eine Vielzahl an Funktionen zu erfassen und in ein solches Widget zu integrieren, sondern es soll auch mittels jQuery UI-Themes zu gestalten sein.

Die Entwicklung eines in das jQuery UI Framework integrierten Grid-Plugins genießt hohe Priorität bei der Entwicklung neuer UI-Plugins. Im Moment müssen Sie aber auf Plugins zurückgreifen, die sicherlich beeindruckend sind, aber nicht den Segen des jQuery UI-Teams erhalten. Im Folgenden werden Sie ein Plugin kennenlernen, das sicherlich einen hohen Funktionsumfang besitzt, mit dem Sie arbeiten können, bis das UI-Team eine integrierte Lösung zur Verfügung stellt.

Die Diskussion zum Thema GridTable können Sie hier verfolgen:

<http://wiki.jqueryui.com/GridTable>

Ein Plugin soll hier trotzdem kurz vorgestellt werden, da es einen faszinierenden Funktionsumfang besitzt. Es besitzt Suchfunktionen, vielfältige Editiermöglichkeiten, einzelne Zellen lassen sich bearbeiten, Zeilen lassen sich löschen, verschieben, oder hinzufügen, es lassen sich JSON-Daten genauso laden wie XML-Daten, es lassen sich Subgrids integrieren, also Datensätze, die sich zu Datensätzen unterordnen lassen. Sie können zum Editieren von Datumsformaten den jQuery UI-Datepicker integrieren. Und das jQuery Grid-Plugin lässt sich über jQuery UI Themes gestalten. Sie finden ein umfangreiches Demo unter dieser Adresse:

<http://trirand.com/blog/jqgrid/jqgrid.html>

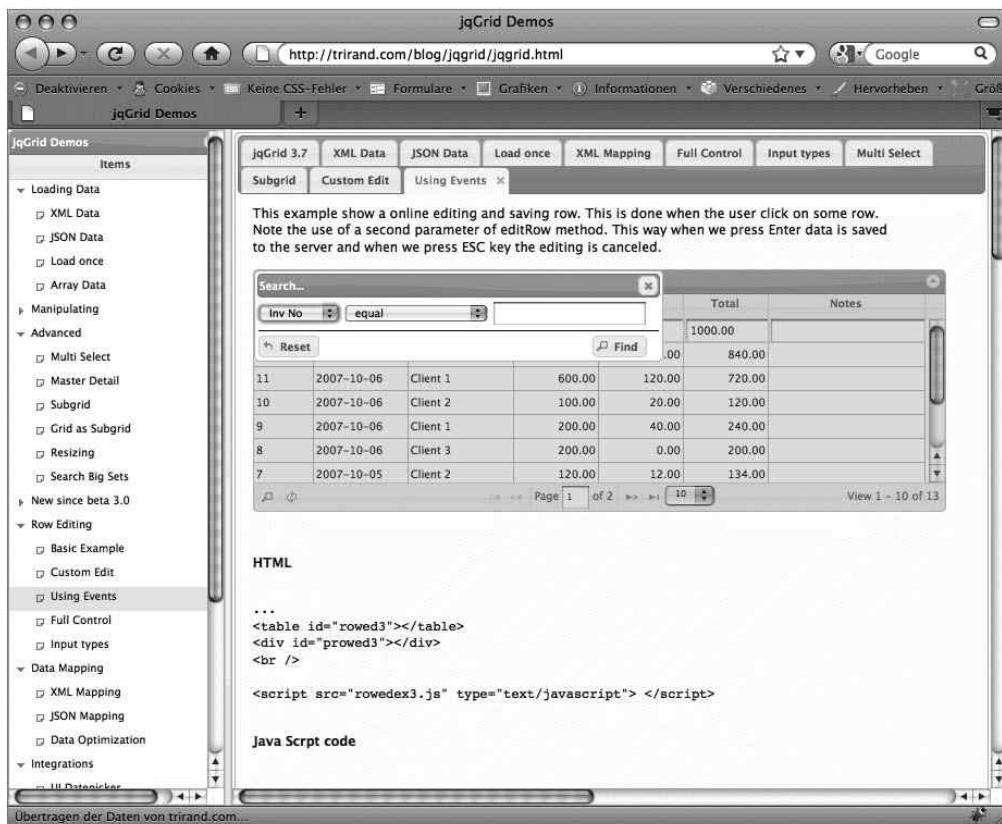


Abbildung 5.39 jQuery Grid-Plugin

Den vollen Umfang erfassen Sie auf der Downloadseite. Hier finden Sie einen Konfigurator, mit dem Sie Features mit in die »minified-Version des Plugins hineinrechnen lassen können – oder eben nicht. Das »eben nicht« hat seine Berechtigung, ist doch das gesamte Script mit allen Features in der optimierten Version noch über 200 KB groß. Sie sollten für den Produktiveinsatz genau überlegen, welche Funktionalitäten Sie benötigen, und ein optimiertes Paket zusammenstellen.

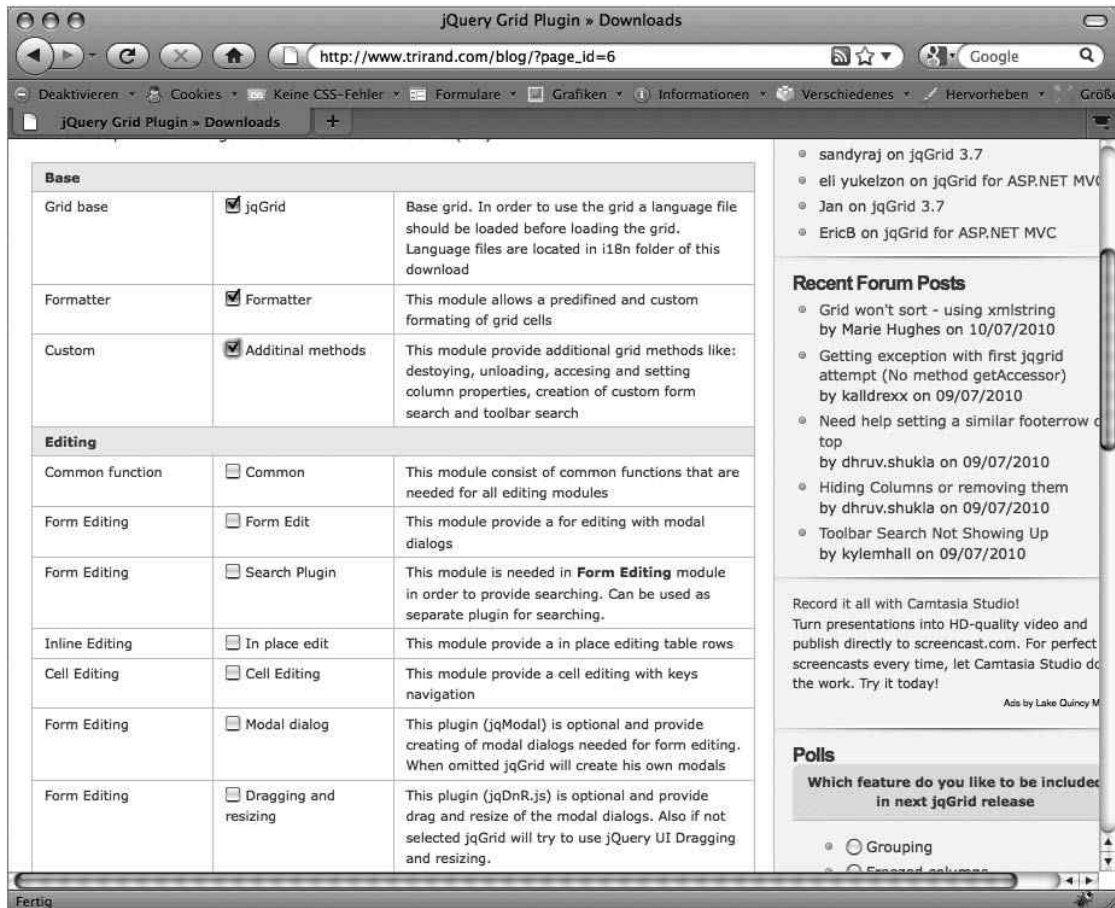


Abbildung 5.40 jQuery Grid-Plugin – Download-Konfigurator

Hier kann nicht auf den Gesamtumfang eingegangen werden, zumal es empfehlenswert ist, auch einen Server mit Datenbank anzulegen, um Daten dynamisch aus der Datenbank zu laden und wieder hineinzuschreiben. Auf der Demoseite finden Sie auch PHP-Beispielscripte, mit denen Sie experimentieren können.

Im Folgenden soll nur kurz angerissen werden, wie Sie dieses Plugin benutzen können. Es gibt zu viele Optionen, auf die hier nicht im Detail eingegangen werden kann. Das hier verwendete Script ist in leicht abgewandelter Form der Demoseite entnommen.

jQuery | jqGrid Demo

XML Example

Injr No	Date	Client	Amount	Tax	Total	Notes
1	2007-10-01	Client 1	100.00	20.00	120.00	note 1
2	2007-10-03	Client 1	200.00	40.00	240.00	note 2
3	2007-10-02	Client 2	300.00	60.00	360.00	note invoice 3 & and amp test
4	2007-10-04	Client 3	400.00	80.00	480.00	

Seite 1 von 1 | 20 | Zeige 1 - 13 von 13

Fertig

Abbildung 5.41 jQuery Grid-Plugin – Beispiel

Es gibt ein eigenes Wiki, in dem das Plugin ausführlich dokumentiert ist, Sie finden es unter:

<http://www.trirand.com/jqgridwiki/doku.php>

Dieses Script wird eine statische XML-Datei in eine HTML-Seite laden und als Tabelle visualisieren. Sie können die Daten anzeigen lassen und können die Anzahl der Zeilen beschränken sowie einen Pager integrieren:

```
<!DOCTYPE html>
<html>
  <head>
    <title>jQuery | jqGrid Demo</title>
    <link rel="stylesheet"
      type="text/css"
      href="css/ui-lightness/jquery-ui-1.8.2.custom.css"/>
    <link rel="stylesheet"
      type="text/css" href="css/ui.jqgrid.css" />
    <link rel="stylesheet"
      type="text/css" href="css/grid.css" />
    <script src="js/jquery-1.4.2.js" type="text/javascript">
    </script>
    <script src="js/jquery-ui-1.8.1.custom.min.js"
      type="text/javascript"></script>
    <script src="js/i18n/grid.locale-de.js"
      type="text/javascript"></script>
    <script src="js/jquery.jqGrid.min.js"
      type="text/javascript"></script>
    <script src="js/grid.js"
      type="text/javascript"></script>
  </head>
</body>
```



```

    <table id="list"></table>
    <div id="pager"></div>
</body>
</html>

```

Listing 5.83 HTML-Gerüst – Grid-Plugin

Sie können ein beliebiges Theme des jQuery UI CSS Frameworks einbinden, es wird automatisch zur Formatierung der Tabelle genutzt. Einbinden reicht, Sie müssen im Scriptteil keinerlei zusätzliche Optionen angeben. Hier wurde wieder das Theme `ui-lightness` verwendet.

Binden Sie unterhalb des jQuery Frameworks zuerst die Lokalisierungsdatei *grid.locale-de.js* für das Plugin ein und danach das Plugin selbst. Diese Reihenfolge ist unbedingt einzuhalten, andere Plugins gehen hier anders vor. Die JavaScript-Datei *grid.js* enthält den Code für die Deklaration des Plugins. Sie sieht folgendermaßen aus:

```

$(document).ready(function (){
    $("#list").jqGrid({
        url:'grid.xml',
        datatype: "xml",
        colNames:['Inv No','Date', 'Client', 'Amount',
                'Tax','Total','Notes'],
        colModel:[
            {name:'id',index:'id', width:75},
            {name:'invdate',index:'invdate', width:90},
            {name:'name',index:'name', width:100},
            {name:'amount',index:'amount', width:80,align:"right"},
            {name:'tax',index:'tax', width:80, align:"right"},
            {name:'total',index:'total', width:80,align:"right"},
            {name:'note',index:'note', width:150, sortable:false}
        ],
        rowNum:10,
        autowidth: true,
        rowList:[10,20,30],
        pager: jQuery('#pager'),
        loadonce: true,
        sortname: 'id',
        viewrecords: true,
        sortOrder: "desc",
        caption:"XML Example"
    }).navGrid('#pager',{edit:false,add:false,del:false});
});

```

Listing 5.84 JavaScript-Code – Grid-Plugin

Die ersten Zeilen erinnern an die jQuery-.ajax()-Methode, jqGrid greift auch auf diese Methode zurück. Hier geben Sie den Datentyp und die URL der Datenquelle an. Anschließend müssen Sie die Spaltennamen mit der Option `colNames` definieren sowie die Eigenschaften der Spalten mit der Option `colModel`. Mit `rowNum` bestimmen Sie die maximal auf einer Seite anzuzeigenden Zeilen, mit der Option `pager` legen Sie fest, in welches HTML-Element der Pager hineingesetzt werden soll.

Wichtig für dieses Beispiel ist die Option `loadonce`. Sie gibt an, dass Sie keine Daten serverseitig sortieren lassen wollen oder dass Daten dynamisch nachgeladen werden sollen. Sie laden eine statische XML-Seite und fertig. Versuchen Sie, die Option `loadonce` auszukommentieren, werden Sie feststellen, dass die clientseitige Sortierung nicht funktioniert. Sie benötigen diese Option, um folgerichtig damit ein Sortierung der geladenen DOM-Elemente zu erzielen.

5.9.5 Zusammenfassung

Wie können Sie sich nun solche Datenvisualisierung in der Praxis vorstellen? Die einfachsten Beispiele wären Warenkörbe von Shopsystemen. Anspruchsvoller wird es, solche Plugins für die Verwaltung von Daten zu verwenden, also in Backends von Content-Management-Systemen oder für die Verwaltung von Artikeln und Kundendaten in CRM- oder Shopsystemen einzusetzen.

Mit diesen Beispielen konnten die Möglichkeiten der Manipulation von Tabellen nur angedeutet werden. Mit Plugins wie *jqGrid* wird aber klar, welches Potenzial darin steckt, solche komplexen Benutzerschnittstellen zu programmieren. Es ist davon auszugehen, dass wir in den nächsten Jahren hier noch einen Entwicklungsschub erwarten können. Ein Beispiel dafür ist *jQuery.sheet*. Ein neu entwickeltes Plugin des Jahres 2010, die Entwickler haben sich die Aufgabe gemacht, eine Online-Tabellenkalkulation zu realisieren. Sie finden sie unter:

<http://www.visop-dev.com/jquerysheet.html>

5.10 Browser und Fenster

In diesem Abschnitt werden Sie einige Beispiele finden, wie Sie mithilfe von jQuery Cookies setzen, Seiten scrollen oder die VOR- und ZURÜCK-Buttons des Browsers benutzen können, um das Blättern über Pager oder Tabs über das history-Objekt zu registrieren.

5.10.1 Cookies

So umstritten Cookies sind, so praktisch und unverzichtbar sind sie für den Weballtag. Sie speichern eine Einstellung, sie speichern Waren für eine Übergabe an den Warenkorb, sie merken sich Informationen, die beim Wechsel von einer HTML-Seite auf die nächste verloren gehen würden.

Was sind Cookies? Es sind kleine Textdateien, die auf dem Rechner des Benutzers abgelegt werden, die bestimmte Informationen speichern können. Die Informationen können später wieder abgerufen werden. Cookies speichern Informationen in einem Zuweisungspaar nach dem Muster Bezeichner = Wert ab. Es gibt Sicherheitsbestimmungen – so darf ein Cookie nur von einer Domain gesetzt und gelesen werden. Jeder Benutzer hat die Wahl, über die Browsereinstellungen Cookies zu unterbinden, aber in der Praxis wird es keinen Einkauf in einem Onlineshop ohne Cookies geben, geschweige denn eine Session in einer Webapplikation.

Cookies können serverseitig über PHP oder Java an den Client über den HTTP-Header gesendet werden. Falls dies aus irgendeinem Grund nicht möglich ist oder die clientseitige Logik so komplex ist, bietet es sich an, Cookies auch über JavaScript zu setzen und auszulesen.

Das Cookie-Objekt ist ein Objekt, das im Sprachkern von JavaScript verankert ist. Sie können komfortabel darauf zurückgreifen. Frage: Warum benötigen Sie dann ein Plugin, das auf dieses JavaScript-Cookie-Objekt zurückgreift? Antwort: Sie erweitern damit das jQuery-Objekt, und können eigene jQuery-Plugins schreiben, die auf dieses jQuery-Cookie-Objekt aufbauen können.

Das Cookie-Plugin ist sehr kurz gehalten, es vereinfacht aber den Umgang mit Cookies ungemein. In einem ersten Beispiel werden Sie die grundlegenden Möglichkeiten des Plugins kennenlernen, in einem zweiten Beispiel verwenden Sie das Plugin als Erweiterung des jQuery UI Frameworks, indem Sie es für das Widget Tabs verwenden.

Zuerst das Minibeispiel, um das Cookie-Plugin kennenzulernen:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery | Cookies</title>
  <script type="text/javascript" src="js/jquery-1.4.2.js">
  </script>
  <script type="text/javascript" src="js/jquery.cookie.js">
  </script>
  <script type="text/javascript" src="js/cookie.js">
```

```

    </script>
</head>
<body>
  <p>
    <a id="set" href="#">Cookie setzen</a><br>
    <a id="get" href="#">Cookie lesen</a><br>
    <a id="delete" href="#">Cookie löschen</a><br>
  </p>
  <p id="msg"></p>
</body>
</html>

```

Listing 5.85 HTML-Gerüst – Cookie-Plugin

Die `<a>`-Elemente allein reichen natürlich noch nicht aus, um Cookies zu steuern, Sie müssen noch die JavaScript-Datei *cookies.js*, die im Header eingebunden wurde, füllen. Nicht zu verwechseln mit der Datei *jquery.cookies.js*, sie beherbergt das Plugin. Wie Sie vielleicht gemerkt haben, wurde noch gar nicht darauf eingegangen, wo das Plugin zu beziehen ist. Wenn Sie das jQuery UI Framework heruntergeladen haben, finden Sie eine aktuelle Version im Ordner *development-bundle/external*. Wenn Sie das Framework nicht heruntergeladen haben, nutzen Sie folgende URL:

<http://github.com/jquery/jquery-ui/tree/master/external/>

Bitte beachten Sie, dass Sie unter folgendem Link lediglich eine ältere Version des Plugins finden (Stand Mitte 2010):

<http://plugins.jquery.com/project/Cookie>

Nun zum JavaScript-Code:

```

$(document).ready(function() {
  var COOKIE_NAME = 'cookie';
  var options = { path: '/', expires: 10 };
  var clicked = 0;
  // cookie setzen
  $('a#set').click(function(e) {
    e.preventDefault();
    clicked++;
    $.cookie(COOKIE_NAME, 'clicked='+clicked, options);
  });
  // cookie lesen
  $('a#get').click(function(e) {
    e.preventDefault();
    clicked++;
  });
});

```

```

    $("#msg").html("Cookie: " + $.cookie(COOKIE_NAME));
  });
  // cookie löschen
  $('a#delete').click(function(e) {
    e.preventDefault();
    clicked++;
    $.cookie(COOKIE_NAME, null, options);
  });
});

```

Listing 5.86 JavaScript-Code – Cookie-Plugin

Bei der Analyse des Codes stellen Sie fest, es handelt sich um drei `click()`-Events, die mit dem Aufruf der Methoden `$.cookie()` arbeiten. Sie sehen, dass Sie zuerst einen Namen für den Cookie setzen müssen, hier wird der Name in der Variablen (Pseudo-Konstante) `COOKIE_NAME` gespeichert, und Sie sehen, dass sich im Objekt `options` die Optionen setzen lassen, die Sie auch im reinen JavaScript-Cookie-Objekt setzen können.

Allerdings wird bei der Option `expires` ein Wert in Tagen angegeben. Cookie-Werte werden automatisch im Hintergrund mit der JavaScript-Methode `encodeURIComponent()` kodiert, sodass Sie sich nicht um Sonderzeichen kümmern müssen, diese werden beim Schreiben des Cookies kodiert und beim Lesen wieder dekodiert. Lassen Sie sich im Firefox das Cookie über EXTRAS/SEITENINFORMATIONEN und dort im Reiter SICHERHEIT anzeigen:



Abbildung 5.42 Cookie-Anzeige im Firefox

Sie erkennen, dass der Wert (Inhalt) des Cookie-URL kodiert wurde. URL-Kodierung folgt dem Muster, dass ein Prozentzeichen, gefolgt von einem hexadezimalen Wert, einem bestimmten festgelegten Zeichen entspricht. %3A entspricht einem Doppelpunkt, %20 einem Leerzeichen.

Wichtig

Bitte beachten Sie, Sie können dieses Beispiel im Firefox lokal ausführen, für andere Browser benötigen Sie unter Umständen einen Webserver, damit das Cookie gesetzt werden kann.

Das *jquery.cookie*-Plugin ist ein kleiner, einfacher Helfer, der es erlaubt, innerhalb des jQuery Frameworks und der jQuery-Syntax auf einfache Weise mit kleinen Keksen umzugehen.

Im nächsten Schritt werden Sie verstehen, warum es sinnvoll sein kann, eine Funktionalität, die in der Kernsprache von JavaScript vorhanden und auch ähnlich einfach anzuwenden ist, trotzdem als jQuery-Plugin zur Verfügung zu stellen.

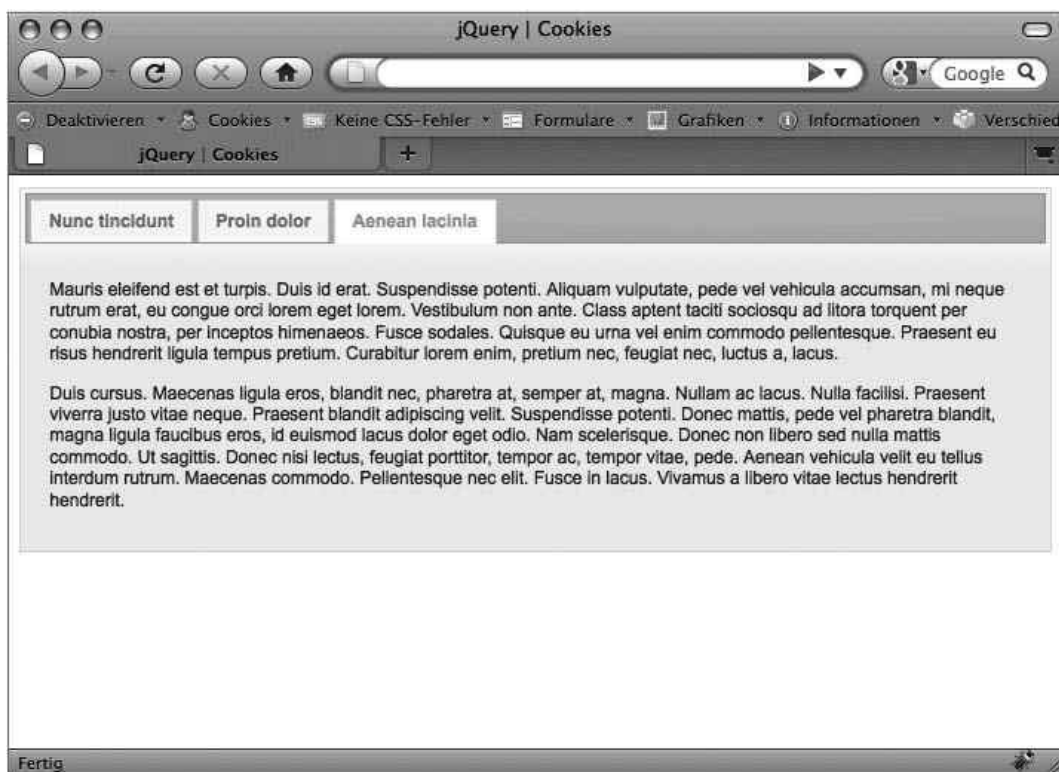


Abbildung 5.43 jQuery UI-Tabs mit Cookies

Dazu greifen Sie auf ein Widget des jQuery UI Frameworks zurück, nämlich das Widget »Tabs«. Tabs haben Sie am Anfang des Praxisteils in diesem Buch bereits kennengelernt, das vorige Beispiel wurde aber von Grund auf entwickelt, also

ohne Zuhilfenahme des UI-Plugins. Zur Erinnerung, unter *Tabs* werden Karteireiter verstanden, zwischen denen man hin- und herschalten kann, jede Karteikarte zeigt seinen eigenen Inhalt.

Das Cookie, das gesetzt werden soll, wird sich den zuletzt angeklickten Karteireiter merken, bei einem neuerlichen Besuch werden Sie direkt auf Ihre zuletzt getätigte Auswahl geführt, allerdings nur in einem festgelegten Zeitraum. Sie verwenden in diesem Beispiel die Option `expires`, also bestimmen, wann das Cookie vom Browser gelöscht werden soll, hier verwenden Sie den Wert `10`, also 10 Tage.

Der HTML-Teil beinhaltet wieder alle Referenzen auf externe Dateien, wie Stylesheets und JavaScript-Dateien sowie eine Liste mit Menüpunkten und den Textcontainern:

```
<!DOCTYPE html>
<html>
<head>
<title>jQuery | Cookies</title>
  <link type="text/css"
        href="css/ui-lightness/jquery-ui-1.8.2.custom.css"
        rel="stylesheet" />
  <link type="text/css"
        href="css/tabs-cookie.css" rel="stylesheet" />
  <script type="text/javascript" src="js/jquery-1.4.2.js">
  </script>
  <script type="text/javascript"
        src="js/jquery-ui-1.8.2.custom.js"></script>
  <script type="text/javascript"
        src="js/jquery.cookie.js"></script>
  <script type="text/javascript"
        src="js/tabs-cookie.js"></script>
</head>
<body>
  <div id="tabs">
    <ul>
      <li><a href="#tabs-1">Nunc tincidunt</a></li>
      <li><a href="#tabs-2">Proin dolor</a></li>
      <li><a href="#tabs-3">Aenean lacinia</a></li>
    </ul>
    <div id="tabs-1"><p>Inhalt...</p></div>
    <div id="tabs-2"><p>Inhalt...</p></div>
    <div id="tabs-2"><p>Inhalt...</p></div>
  </div>
</body>
</html>
```

Listing 5.87 HTML-Gerüst – Cookie-Plugin und Tabs

Verwenden Sie wieder das jQuery UI-Plugin. Wie Sie das bewerkstelligen, haben Sie in Abschnitt 5.7, »jQuery UI«, erfahren. Inkludieren Sie das jQuery Framework, darunter der Datei *jquery-ui-1.8.2.custom.js* sowie alle zugehörigen CSS-Dateien. Diese binden Sie wieder oberhalb der *js*-Dateien ein. Den jQuery-Code fügen Sie in die Datei *tabs-cookies.js* ein, er ist denkbar einfach:

```
$(document).ready(function() {
    $("#tabs").tabs({
        cookie: {
            expires: 10,
            path: "/"
        }
    });
});
```

Listing 5.88 JavaScript-Code – Cookie-Plugin und Tabs

Mit der Methode `.tabs()` initialisieren Sie das Tabs-Widget. Dieses besitzt bereits eine eingebaute Option `cookie`, Sie müssen nur noch die Cookie-Optionen festlegen, und fertig ist das Script. Das Tabs-Plugin greift implizit auf das Cookie-Plugin zurück. Sie müssen aufpassen, wenn Sie diese Option setzen, dass Sie nicht vergessen, das Plugin einzubinden, sonst erhalten Sie eine JavaScript-Fehlermeldung. Dieser Fehler wird vom Tabs-Widget nicht abgefangen.

Wird das Cookie nun erfolgreich gesetzt, wird der Benutzer, wenn er den Browser schließt und innerhalb von zehn Tagen die Seite erneut besucht, den letzten Karteireiter, den er angeklickt hatte, wieder bereits ausgewählt vorfinden.

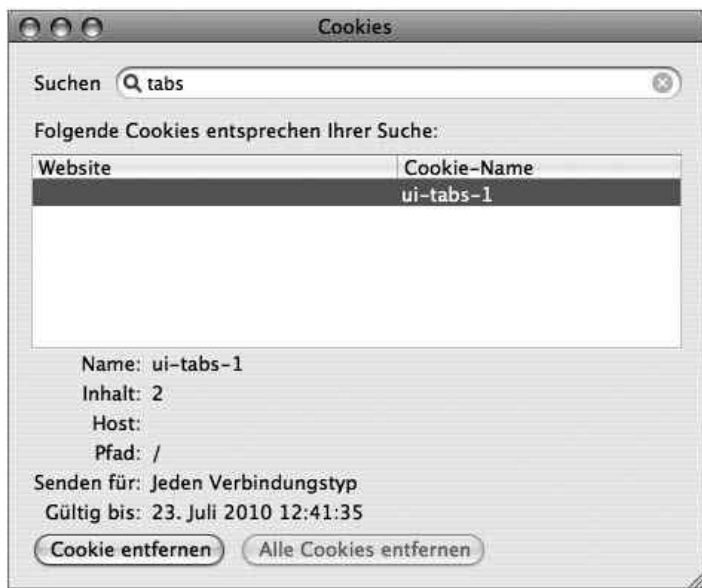


Abbildung 5.44 Cookie gesetzt für jQuery UI-Tab

In diesem Cookie ist der Wert 2 für den dritten Karteireiter mit dem Namen `ui-tabs-1` gespeichert.

5.10.2 History des Browsers

Einen großen Nachteil teilen sich die Technologien Ajax und Flash. Die Zurück- und Vorwärts-Buttons des Browsers bleiben wirkungslos, wenn man eine komplexe Webanwendung mit vielen Menüs und nachladbaren Inhalten in eine einzige HTML-Seite platziert. Eigentlich.

Für Flash gibt es ein Framework namens *SWFAddress*. Diese Bibliothek ermöglicht es Flash-Entwicklern einerseits, *deep linking* zu ermöglichen, also dem Benutzer (und dem Browser) zu suggerieren, dass er, wenn er Menüpunkte innerhalb einer Flash-Applikation klickt, sich physisch innerhalb einer Ordnerstruktur bewegt und die Adresse auch in der Adresszeile angezeigt wird.

Setzt man ein Bookmark in einer tiefen Ebene der Flash-Applikation und ruft später dieses Bookmark auf, wird der Benutzer exakt zu dem Menüpunkt zurückgeführt. Obwohl lediglich eine einzige HTML-Seite geladen wurde, in der eben diese Flash-Applikation eingebettet wurde, funktioniert das Setzen von Bookmarks. *SWFAddress* sorgt auch dafür, dass die Browserhistory, also der Verlauf, jeden Klick auf einen Menüpunkt registriert.

Das Problem, dass der Browserverlauf nachgeladene oder dynamisch gezeigte Inhalte nicht wahrnimmt, trifft auch Ajax-Websites. Schließlich wird (nehmen wir das Tab-Beispiel wieder auf) nur eine einzige HTML-Seite geladen, und innerhalb dieser Seite werden bei jedem Klick neue Inhalte gezeigt und wieder versteckt. Benutzt der Benutzer den ZURÜCK-Button des Browsers, wird dies genauso wenig zum gewünschten Erfolg führen wie in einer Flash-Applikation. Eigentlich.

Es gibt für Ajax und dynamische, auf jQuery basierende Websites einige Ansätze, die Browserhistory zu beeinflussen: Auch die Entwickler von *SWFAddress* haben sich daran gemacht, ein Plugin für jQuery zu schreiben, sie nennen es folgerichtig *jQuery Address*.

Zu finden ist es unter <http://www.asual.com/jquery/address/>.

Das Plugin ermöglicht es zudem, einen Pfad in der Adressleiste zu setzen und so für einzelne Tab-Reiter Bookmarks zu setzen. Außerdem wird das `<title>`-Element mitgeführt, mit jedem Klick auf einen Reiter wird das Label des Reiters im »Title« angezeigt. Im Verlauf (»Chronik«) des Browsers werden die Titel der einzelnen Reiter akkurat angezeigt – allerdings: Safari und Internet Explorer lassen

sich nicht hinters Licht führen. Sie zeigen im Verlauf nur eine einzige Seite an. (Seien Sie aber beruhigt, die VOR- und ZURÜCK-Buttons lassen sich in allen Browsern bedienen, und das Bookmarks funktioniert auch. Damit ist dieses Plugin durchaus einsetzbar.)

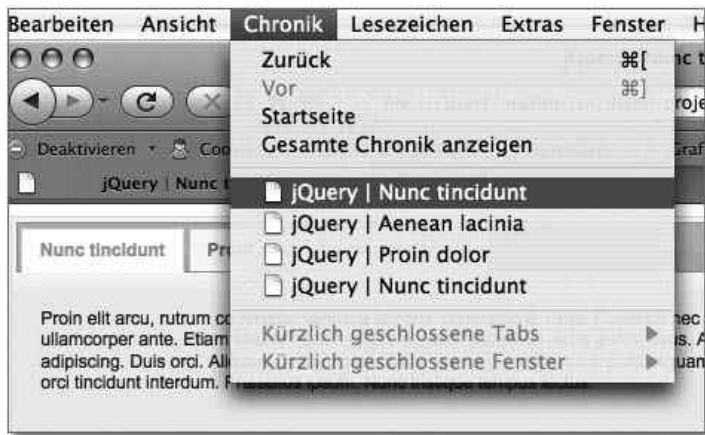


Abbildung 5.45 Browserhistory

Nun nehmen Sie sich, wie gesagt, wieder das Tabs-Beispiel vor. Sie verwenden exakt das gleiche Beispiel, das Sie bereits im Cookie-Beispiel eingesetzt haben. Sie werden wieder das jQuery UI-Plugin verwenden, nur, dass Sie zusätzlich das jQuery Address-Plugin hinzufügen. Im HTML-Teil müssen Sie nur wenige Änderungen vornehmen, zum einen die Scriptdateien einbinden:

```
...
<script type="text/javascript" src="js/jquery.address-1.2.1.js">
</script>
<script type="text/javascript" src="js/tabs-history.js">
</script>
...
```

In der Datei *tabs-history.js* werden Sie das Plugin initialisieren. Sie müssen noch ein Attribut für die Karteireiter setzen, dann können Sie sich an den JavaScript-Code machen:

```
...
<ul>
  <li><a href="#tabs-1" rel="address:/item_1.html">
    Nunc tincidunt
  </a>
</li>
  ...
</ul>
...
```

Das Attribut `rel` wird verwendet, um den Wert im URL an den Hash `#` zu hängen. In diesem Fall sähe der URL folgendemmaßen aus, nachdem Sie auf das `<a>`-Element in der `<ul>`-Liste geklickt hätten:

`http://[domain]/[Verzeichnis]/#/item_1.html`

Damit ist dieser URL eindeutig zurückzuführen auf einen bestimmten Karteireiter. Wenn Sie das Beispiel statt mit einer *`index.html`* auf Ihrem Webserver lokal in Ihrem Arbeitsverzeichnis testen, wird der Pfad ein wenig anders aussehen (es funktioniert aber trotzdem):

`file://[...]/tabs-history.html#/item_1.html`

Nun werden Sie die JavaScript-Datei `tabs-history.js` mit Anweisungen füllen:

```
$(document).ready(function() {
    var tabs;
    // Address handler
    $.address.init(function(event) {
        tabs = $('#tabs').tabs();
    }).change(function(event) {
        $.address.title($.address.title()
            .split(' | ')[0] + ' | ' + $('a')
            .filter('[rel=address:' + event.value + ']').text());
    }).externalChange(function(event) {
        tabs.tabs('select', $('a')
            .filter('[rel=address:' + event.value + ']').attr('href'));
    });
});
```

Listing 5.89 JavaScript-Code – Browserhistorie mit jQuery Address-Plugin

Mit `$.address.init()` initialisieren Sie die jQuery UI-`.tabs()`-Methode. Diese Methode kennen Sie bereits aus dem vorangegangenen Beispiel.

Mit `$.address.change()` wird der »Title« des Dokuments immer dann geändert, sobald eine Mausaktion stattfindet, also wenn ein Benutzer auf einen Tab klickt.

Die Methode `$.address.title()` wird aufgerufen und wiederum mit dem Argument `$.address.title()` übergeben, was bedeutet, der bisherige Titel wird mit `.split()` und `.text()` manipuliert und als neuer Titel gesetzt. Im Klartext könnte man es so übersetzen: Der String wird mit vom Anfang bis zum Zeichen »|« aufgeteilt, daraufhin wird das Zeichen »|« neu gesetzt, und mit der Methode `.filter()` wird das aktive `<a>`-Element bestimmt und dessen Inhalt (wenn Sie obiges Beispiel verwenden) »Nunc tincidunt« an den »Title« gehängt.

Die Methode `$.address.externalChange()` setzt dann den Status der Tabs neu, wenn über die Zurück- oder Vortasten des Browsers navigiert wird. Diese Methode ruft erneut das Widget `.tabs()` mit der Option `select` auf. Der Select Handler setzt die aktuelle Selektion. Dieses Beispiel ist sehr einfach gehalten, zu erwähnen ist, dass Sie externe Inhalte über Ajax laden können, dann wird der Nutzen dieses Plugins noch größer.

In diesem Beispiel sehen Sie eine Verzahnung von JavaScript-, jQuery-, jQuery UI- und jQuery Address-Methoden. Sie sehen, wie Sie aufeinander aufbauend Plugins integrieren können. Und der Nutzen des hier besprochenen Plugins ist nicht zu unterschätzen. Viele Benutzer sind es gewohnt, über die ZURÜCK-Buttons des Browsers zu navigieren, allzu oft kommen auch Rückfragen, warum eine Site nicht erwartungsgemäß funktioniert. Sie sollten stets davon ausgehen, dass der Benutzer nichts von Webtechnik versteht, auch nicht verstehen muss, und es ist nicht vermittelbar, warum in dem einen oder anderen Fall die Buttons nicht funktionieren. Und hier hilft dieses Plugin.

5.10.3 Flashfilme einbinden mit jQuery

Seit geraumer Zeit hat es sich durchgesetzt, Flash-Filme nicht direkt mit Object- bzw. Embed-Elementen einzubinden, sondern »unobtrusive« JavaScript zu verwenden, um einen vorhandenen `<div>`-Container zu nutzen und dort dynamisch einen Flash-Film einzubinden. Bekanntester Vertreter eines solchen JavaScripts ist *SWFObject*. Der Code ist zu finden unter:

<http://code.google.com/p/swfobject/>

So manch ein Entwickler hat sich die Aufgabe gestellt, genau diese Funktionalität für jQuery zu adaptieren. Das bringt den Vorteil, sich innerhalb eines einzigen Frameworks zu bewegen. Ganz zu schweigen davon, dass es Ladezeiten spart, wenn man sowieso an anderer Stelle jQuery einsetzt. Dann kann man die Bibliothek nutzen, um auch Flash-Filme einzubetten. Es gibt zahlreiche Ansätze in diesem Zusammenhang. Einer davon ist das Plugin von Stephan Belander. Sie können es unter folgender Adresse herunterladen:

<http://github.com/Qard/jquery-flash>

Ein kleines Beispiel: Sie finden in der Begleit-DVD zu diesem Buch einen Dummy-Flash-Film, den Sie in eine Seite einbetten können. Es handelt sich um eine kleine Slideshow. Die Bilder kennen Sie bereits aus den Galerie-Beispielen.



Abbildung 5.46 Flash-Filme einbinden

Erstellen Sie als Erstes wieder ein HTML-Gerüst, das denkbar einfach gehalten ist, und der jQuery-Code zur Initialisierung ist direkt im `<head>` notiert:

```
<!DOCTYPE html>
<html>
<head>
<title>jQuery | Flash</title>
<script type="text/javascript" src="js/jquery-1.4.2.js">
</script>
<script type="text/javascript" src="js/jquery.flash.js">
</script>
<script type="text/javascript">
$(document).ready(function(){
    $('.flash').flash({
        "src":"jquery.dummy.swf",
        "width":500,
        "height":332,
        "quality":"high",
        "wmode":"transparent",
        "express":"express_install.swf",
        "classid":"clsid:D27CDB6E-AE6D-11cf-96B8-444553540000",
        "pluginspace":"http://get.adobe.com/flashplayer",
        "version":"9.0.24",
    });
});
```

```
});
</script>
</head>
<body>
  <div class="flash">
    <h1>Alternativer Inhalt</h1>
  </div>
</body>
</html>
```

Listing 5.90 Flash – SWF-Dateien laden mit jQuery Flash-Plugin

Sie müssen im Body einen Container mit, sagen wir, der Klasse `flash` anlegen. Innerhalb des Containers können Sie bzw. sollten Sie einen alternativen Content einfügen, der dem Benutzer einen adäquaten Inhalt bietet. Dieses Konzept folgt wieder dem Prinzip der »progressiven Erweiterung«, so wie einige Beispiele in diesem Buch. Demjenigen Browser, der die notwendigen Technologien nicht integriert und bestimmte Inhalte nicht darstellen kann, wird mit einem Fallback trotzdem ein Inhalt geboten, er wird nicht einfach im Regen stehen gelassen. Dem Benutzer, der über diese Technologie verfügt, wird ein Mehrwert geboten. Von all den Abläufen sollte er nichts mitbekommen, er kennt die technischen Hintergründe vielleicht nicht und soll sie auch nicht kennen müssen.

Zurück zum Beispiel: Der alternative Content wird, falls JavaScript aktiviert ist, mit dem `<object>`-Element überschrieben, und der SWF-Film wird eingebettet.

Mit der Anweisung `$('.flash').flash()` wenden Sie die Methode `.flash()` auf das jQuery-Objekt `$('.flash')` an. Die CSS-Klasse haben Sie für den `<div>`-Container vergeben. Mit der Methode `.flash()` sollten Sie auf jeden Fall die Option `src` setzen, hier werden der Pfad und der Dateiname des SWF-Filmes angegeben. Alle Optionen sind optional, wie Breite und Höhe oder Expressinstallation. Selbst `src` ist optional, wenn Sie im Element, das Sie selektiert haben, ein `href`-Attribut mit einem Verweis auf einen SWF-Film gesetzt haben. Im Folgenden eine Tabelle mit allen Optionen.

Option	Defaultwert	Beschreibung
<code>id</code>	<code>\$(this).attr('id')</code>	ID
<code>class</code>	<code>\$(this).attr('class')</code>	Klasse
<code>width</code>	<code>\$(this).attr('width')</code>	Breite
<code>height</code>	<code>\$(this).attr('height')</code>	Höhe
<code>src</code>	<code>\$(this).attr('href')</code>	Quelle

Tabelle 5.2 jQuery Flash-Plugin – alle Optionen

Option	Defaultwert	Beschreibung
flashvars	null	JSON-strukturierte Daten
bgcolor	null	Hintergrundfarbe
quality	null	low, medium, high, autolow, autohigh, best
wmode	null	Window, Opaque, Transparent (nicht von allen Browsern unterstützt)
allowscriptaccess	null	always, never, samedomain
allowfullscreen	null	true, false – erlaubt den Vollbildmodus
classid	'clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'	Identifiziert die ActiveX-Steuerung für den Browser (nur Internet Explorer).
codebase	'http://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version='	Gibt die Adresse an, von der Codes nachgeladen werden können. (nur <object>-Elemente).
Pluginspace	'http://get.adobe.com/flashplayer'	Gibt den Ort für nachladbaren Code an, (ähnliche Funktionalität wie codebase, gilt nur für <embed>-Element)
Version	'9.0.24'	Minimum Version ist 9.0.24
express	null	Gibt den Ort für die Datei an, mit der Adobes Express Install ausgeführt werden kann, z. B. <i>express-install.swf</i> .

Tabelle 5.2 jQuery Flash-Plugin – alle Optionen (Forts.)

Es mag sein, dass der Klassiker unter den Scripts, *SWFObject*, immer noch leistungsfähiger ist als dieses Script, aber eine Adaption für jQuery ist ein interessanter Ansatz und in vielen Anwendungsfällen voll und ganz ausreichend.

5.10.4 Scrolling

Scrollen ist eine gängige Aufgabe von Browsern. Von so manchem Designer und Benutzer zu Unrecht verpönt, ermöglicht dieses Feature auf jedem Rechner, in den meisten Programmen eine größere Menge von Inhalten bei beschränkter Fenster- oder Bildschirmgröße benutzerfreundlich zugänglich zu machen.

Mit jQuery können Sie das Scrollen zum Erlebnis machen, indem Sie die Bewegung nach oben oder nach unten animieren und zur gewünschten Seitenposition sanft hingeleiten lassen. So macht Scrollen richtig Spaß.

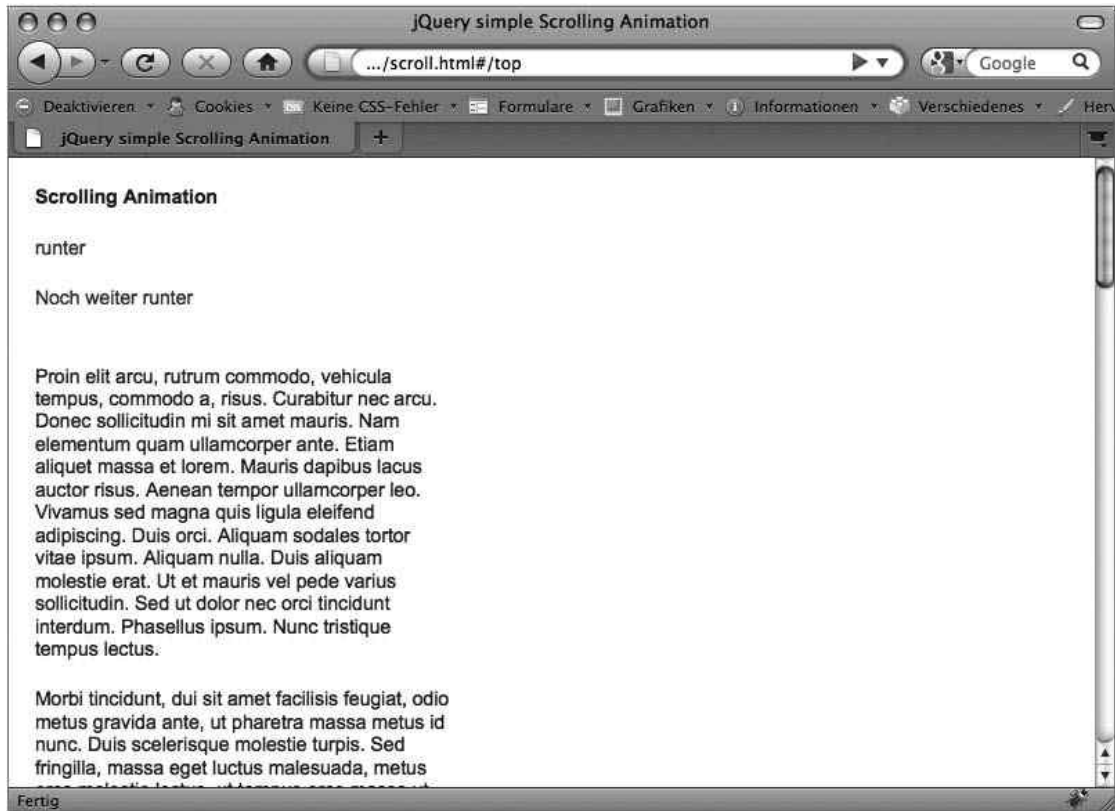


Abbildung 5.47 Animiertes Scrollen

Zuerst brauchen Sie eine HTML-Seite mit vielen Inhalten oder zumindest mit vielen Absätzen, damit auch etwas zu scrollen ist. Der Inhalt sollte mehr Höhe einnehmen als der sichtbare Bereich des Fensters.

Im Head benötigen Sie die obligatorischen `<script>`- und `<link>`-Elemente:

```
<!DOCTYPE html>
<html>
<head>
  <title>jQuery simple Scrolling</title>
  <link type="text/css" href="css/scroll.css" rel="stylesheet" />
  <script type="text/javascript" src="js/jquery-1.4.2.js">
  </script>
  <script type="text/javascript"
    src="js/jquery.address-1.2.1.js">
  </script>
  <script type="text/javascript" src="js/scroll.js">
  </script>
```



```

</head>
<body>
  <a name="top"></a>
  <ul id="top">
    <li><a href="#down">runter</a></li>
    <li><a href="#bottom">Noch weiter runter</a></li>
  </ul>
  <div>
    <p>...</p>
  </div>
  <a id="down" name="down" href="#top">nach oben</a>
  <div>
    <p>...</p>
  </div>
  ...
  <a id="bottom" name="bottom" href="#top">nach oben</a>
</body>
</html>

```

Listing 5.91 HTML-Gerüst – animiertes Scrollen

Sie sollten am Anfang der Seite noch ein paar lokale Anker setzen, eine Art von lokaler Navigation in der Mitte und unten <a>-Elemente. Die CSS-Angaben werden hier wieder eingespart, damit Sie sich umgehend mit dem JavaScript-Code auseinandersetzen können:

```

$(document).ready(function(){
  var localAnchors = $("a[href*=#]");
  if (!localAnchors.length) return;
  localAnchors.click(function(event) {
    event.preventDefault();
    var target = $($ (this).attr("href"));
    if (!target.length) return;
    var position = target.position().top;
    var obj = $('html,body');
    var selectedObj = obj.eq(1);
    obj.each(function(i) {
      var eachElement = $(this);
      var top = eachElement.scrollTop();
      eachElement.scrollTop(top + 1);
      var test = eachElement.scrollTop();
      eachElement.scrollTop(top - 1);
      if (test > 0) {
        selectedObj = eachElement;
        return false;
      }
    });
  });
});

```

```

    }
  });
  selectedObj.animate({scrollTop: position}, 1000);
});
});

```

Listing 5.92 JavaScript-Code – animiertes Scrollen

Zunächst benötigen Sie ein jQuery-Objekt mit allen lokalen Ankern. Dazu verwenden Sie den Selektor `"a[href*=#]"`, er filtert sämtliche `<a>`-Elemente, die im Attribut `href` das Hash-Zeichen enthalten. Da jeder lokale Anker dieses Zeichen mit sich führen muss, ist das der Filter, der sämtliche Elemente zuverlässig selektiert. An diese Elemente binden wir den `.click()`-Event. Mit `event.preventDefault()` stellen Sie sicher, dass die browsereigene Aktion unterbunden wird. Die Variable `target` speichert das Sprungziel, das Sie im `href`-Attribut festgelegt haben, als jQuery-Objekt. Mit der Anweisung `if (!target.length)` fangen Sie den Fall ab, dass Sie versehentlich eine Sprungmarke ansprechen, die gar nicht existiert.

In der nun folgenden Zeile legen Sie die Position fest, die der Browserinhalt einnehmen soll. In der Variablen `position` wird mit der Methode `.position()` und deren Eigenschaft `.top` der Abstand des Inhalts vom oberen Dokumentursprung festgelegt. Da das `href`-Attribut das Sprungziel in einem `id`-Attribut angibt und Sie sichergestellt haben, dass es das Sprungziel gibt, können Sie so leicht die Position des selektierten Elements ermitteln.

Auffällig ist hier der Selektor `$('html, body')`. Wieso wird hier nicht einfach allein das `<body>`-Element genutzt, um die Seite zu animieren? Weil die verschiedenen Browser unterschiedlich reagieren. Im Safari und Google Chrome müssen Sie das `<body>`-Element animieren, in allen anderen Browsern das HTML-Element. Hinzu kommt, dass Opera den Selektor `'html, body'` nicht korrekt interpretiert, sondern ausschließlich `'html'` verlangt.

Sie könnten jetzt einfach die Utility-Methode `$.browser` verwenden, um den richtigen Browser anzusprechen. Dies werden Sie aber unterlassen, da das jQuery-Team zwar diese Methode weiterhin unterstützt, jedoch davon abrät, sie zu benutzen. Es ist nicht sonderlich zuverlässig, sie zu benutzen, da einige Browsertypen einen anderen User Agent vorzugaukeln wissen, als sie in Wirklichkeit darstellen. Außerdem steht es in den Sternen, wie sich zukünftige Browserversionen verhalten werden.

Die Idee ist es nun, eine Abfrage zu entwerfen, um herauszubekommen, welcher Browser welches Verfahren unterstützt. Zuerst iterieren Sie mit `.each()` über das

jQuery-Objekt. Um genau zu sein, mit dieser Methode werden die folgenden Anweisungen genau zweimal durchlaufen: einmal für 'html' und einmal für 'body'.

Innerhalb der `.each()`-Schleife erhöhen Sie den Wert der Methode `.scrollTop()` um ein Pixel, speichern das Ergebnis und senken den Wert wieder um ein Pixel. Ist das Ergebnis größer als 0, so können Sie davon ausgehen, dass das Script mit diesem Element operieren kann. Sollte es 0 sein, erlaubt dieses Element hingegen keine Veränderung über `.scrollTop()`. Mit dieser Testabfrage erkennt das Script genau, welches Element zu scrollen ist. Dies wird in allen aktuellen und zukünftigen Browserversionen so sein und bleiben. Selbst wenn also Safari (Webkit) sein Verhalten in dieser Hinsicht ändern sollte, bleibt dieses Script funktionstüchtig. Hier wird der Vorteil dessen deutlich, eine Abfrage über die Eigenschaften, bzw. Eigenarten der Browser zu generieren und nicht aus der Browserversion: Die Browserversion ist nicht zwingend an ein bestimmtes Verhalten gekoppelt.

Zurück zum Script: Ist das brauchbare Element gefunden, wird die `.each()`-Schleife mit `return false` abgebrochen. In der Variablen `selectedObj` wird das Ergebnis gespeichert. In der letzten Anweisung benutzen Sie die Methode `.animate()`, um das sanfte Scrollen am selektierten Objekt auszuführen.

5.10.5 Scrollen mit Geschichte

Jetzt fragen Sie sich, was ist mit dem Verlauf, der History des Browsers? Wenn Sie lokale Sprungmarken mit der normalen Browseraktion benutzen, registriert dies das History-Objekt. Im vorangegangenen Beispiel haben Sie das jQuery Address-Plugin kennengelernt. Sie können es jetzt wieder einsetzen. Mit seiner Hilfe können Sie mit den ZURÜCK- und VOR-Buttons zwischen Seitenteilen hin- und herspringen und können lokale Anker als Lesezeichen abspeichern.

Sie sollten sich aber darüber im Klaren sein, dass Sie die Aktionen der Vor- und Zurücktasten nicht animieren können: Der Browser speichert die Fensterposition, die er für richtig hält (und die auch korrekt sind). Nur nicht für diesen Anwendungsfall: Wenn Sie versuchen, mit `.scrollTop()` sofort zum gewünschten Ziel zu springen, veranlasst das den Browser zu unschönen Zuckungen, da Sie die browserinterne Aktion nicht verhindern können.

Sie können dieses Verhalten auch mit JavaScript kaum abfangen. Möglich wäre, mittels `location.hash` den lokalen Anker zu ersetzen. Dadurch wird jedoch die Liste der Browserhistory gelöscht bzw. neu gesetzt. (Als »Hash« versteht man in diesem Zusammenhang innerhalb einer Webadresse den Teilstring, der dem #-Zeichen folgt. Den Hash zu ändern ist also nicht zielführend.)

Fazit: Auf das History-Objekt selbst können Sie in keinem Fall schreibend zugreifen. Dies wird von allen Browsern aus Sicherheitsgründen nicht ermöglicht. So bleibt nur, die Position der Seite anzuspringen und keine Animation anzuwenden (dass es noch einen anderen Weg gibt, werden Sie noch sehen).

Im nächsten Schritt werden Sie aber den sicheren und sauberen Pfad beschreiten, das Ziel ohne Animation anzusteuern. Beginnen Sie mit der Erweiterung des HTML-Gerüsts. Hier genügt es, nur Ausschnitte wiederzugeben. Im `<head>`-Container müssen Sie das Plugin einbinden:

```
<script type="text/javascript"
      src="js/jquery.address-1.2.1.js">
</script>
```

Sie finden es, wie bereits beschrieben unter

<http://www.asual.com/jquery/address>

oder auf der Begleit-DVD zu diesem Buch.

Im Body müssen sie nur die `rel`-Attribute nach folgendem Muster einfügen:

```
...
<a href="#down" rel="address:/down">runter</a>
...
```

Wenn Sie jetzt das Beispiel testen, werden Sie feststellen, dass es schon recht gut funktioniert. Nur wenn Sie die Seite mit einer Sprungmarke hinzufügen, sagen wir, mit `#/down` als Lesezeichen, den Browser schließen und die Seite mittels Lesezeichen erneut ansteuern, wird der gewünschte Absatz nicht korrekt angesteuert. Dazu müssen Sie den jQuery-Code verändern und um jQuery Address-Events erweitern:

```
$(document).ready(function(){
  var localAnchors = $("a[href*=#]");
  if (!localAnchors.length) return;
  localAnchors.click(function(event) {
    event.preventDefault();
    scrollPage(true);
  });
  $.address.externalChange(function(event) {
    scrollPage(false);
  });
  function scrollPage(animate) {
    var targetID = $.address.value();
    targetID = targetID.replace("/", "#");
    if (targetID == "#") return;
```

```

var target = $(targetID);
if (!target.length) return;
var position = $(targetID).position().top;
var obj = selectedObj = $('html,body');
obj.each(function(i) {
    var eachElement = $(this);
    var top = eachElement.scrollTop();
    eachElement.scrollTop(top + 1);
    var test = eachElement.scrollTop();
    eachElement.scrollTop(top - 1);
    if (test > 0) {
        selectedObj = eachElement;
        return false;
    }
});
if (animate) {
    selectedObj.animate({scrollTop: position}, 1000);
    return false;
}
selectedObj.scrollTop(position);
});

```

Listing 5.93 JavaScript Code – animiertes Scrollen und Browserhistory

Sie packen die Scrollanweisungen in eine Funktion `.scrollPage()`. Diese rufen Sie an zwei Orten auf, einmal innerhalb des `.click()`-Events und einmal in der jQuery Address-Methode `.externalChange()`. Sie wird aufgerufen, wenn sich außerhalb des Dokuments etwas ändert, d.h., wenn die Vor- und Zurücktasten geklickt werden oder die Seite über ein Lesezeichen aufgerufen wird.

Damit erfüllen Sie die Anforderung, an den Absatz innerhalb der Seite zu springen, den Sie über den Hash in dem URL definiert haben. Aber: Sie müssen der Funktion zuerst das Argument mit einem Booleschen Wert übergeben, ob eine Animation ausgeführt werden soll oder ob die Methode `.scrollTop()` direkt aufgerufen werden soll. Sie können selbstverständlich damit herumspielen und versuchen, bei einem `.externalChange()` eine sinnvolle Animation zu erzielen, indem Sie dem Funktionsaufruf ein `true` übergeben, aber auf diesem Weg werden Sie keinen Erfolg haben.

Es gibt noch einen Weg, wie Sie eine Animation mittels Vor- und Zurücktasten stabil und ohne Nebenwirkungen realisieren können. Diese Lösung führt über einen Umweg, der nicht in jedem Fall angewendet werden kann, er sei hier trotz-

dem kurz vorgestellt, Sie müssen konkret in Ihrem Projekt entscheiden, ob Sie diesen *Hack* verwenden wollen.

Sie umschließen den zu scrollenden Inhalt mit zwei Containern:

```
<div id="wrapper">
  <div id="container">
    <!--... gesamter Inhalt hier ...-->
  </div>
</div>
```

Sie müssen nun noch die Styles anpassen:

```
html, body {overflow:hidden; }
```

```
div#wrapper {
  width:100%;
  overflow:auto;
}
div#container {
  width:100%;
  height:500px;
}
```

Der Trick besteht darin, dass Sie jetzt den Div mit dem ID `wrapper` scrollen und animieren, der Container wird mit `overflow:auto` so gesetzt, dass er die Anzeige der Scrollbalken übernimmt. Dadurch wird mit den Vor- und Zurücktasten die Seite immer auf die Position 0 gesetzt, und jQuery kontrolliert nun vollständig die Scrollbewegung.

Sehen Sie es als Ihre Aufgabe, mittels der jQuery-Methode `$('body').wrapInner('<div id="wrapper" />')` die Wrapper- und Container-Elemente dynamisch hinzuzufügen, um das HTML-Markup von Ballast freizuhalten. In diesem Beispiel werden Sie es der Einfachheit halber im HTML-Dokument wiederfinden. Sie müssen noch eine kleine Funktion einfügen, die die Größe des Wrappers dynamisch an die Fenstergröße anpasst, und die Event Handler bereitstellen. Das gesamte Script sieht folgendermaßen aus:

```
$(document).ready(function(){
  resizeWindow();
  var localAnchors = $("a[href*=#]");
  if (!localAnchors.length) return;
  localAnchors.click(function(event) {
    event.preventDefault();
    scrollPage(true);
  });
});
```

```

$.address.externalChange(function(event) {
    scrollPage(true);
});
function scrollPage(animate) {
    var targetID = $.address.value();
    targetID = targetID.replace("/", "#");
    if (targetID == "#") return;
    var target = $(targetID);
    if (!target.length) return;
    var position = $(targetID).position().top;
    var selectedObj = $("#wrapper");
    if (animate) {
        selectedObj.animate({scrollTop: position}, 1000);
        return false;
    }
    selectedObj.scrollTop(position);
}
});
$(window).resize(function() {
    resizeWindow();
});
function resizeWindow() {
    var h = $(window).height();
    var w = $(window).width();
    var wrapper = $("#wrapper");
    if (!wrapper.length) return;
    wrapper.css({'height':h,'width': w });
}

```

Listing 5.94 JavaScript-Code – Scrollen und Browserhistory (Ausbaustufe)

Wie Sie sehen können, können Sie die Methode `.each()`, mit der Sie das HTML- und das `<body>`-Element untersucht hatten, herausnehmen, da Sie jetzt den `<div>`-Container manipulieren und nicht mehr die gesamte Seite. Hinzu gekommen ist die Funktion `resizeWindow()`, die einmal nach dem Laden des DOM-Baums und nach jedem `.resize()` des Browserfenster ausgeführt wird.

5.10.6 Das jQuery.ScrollTo-Plugin

Jetzt gibt es findige Webdeveloper, die sich Gedanken gemacht haben, wie man animierte Scrollfunktionen bequemer implementieren kann. Sie haben für diese Zwecke Plugins entwickelt. Das Plugin, das nun vorgestellt werden soll, heißt *jQuery.ScrollTo* und stellt genau diese Funktionalität zur Verfügung.

Sie müssen sich keine Gedanken zu Problemen bestimmter Browser machen. Mit einem simplen Methodenaufruf können Sie an der X- oder der Y-Achse entlang oder an beiden Achsen gemeinsam, also diagonal, einen Container oder eine ganze Seite animieren. Sie finden das Script entweder auf der Begleit-DVD zu diesem Buch oder im Internet unter:

<http://flesler.blogspot.com/2007/10/jqueryscrollto.html>

Stellen Sie sich einen `<div>`-Container vor, der gewisse Dimensionen einnimmt. In diesem Container befinden sich, ähnlich einer Tabelle, fünf Felder in der Waagerechten, und drei Zeilen untereinander. Die insgesamt 15 Felder nehmen wesentlich mehr Raum ein als der Container, in dem sie sich befinden. Sie werden diese verschachtelten Felder scrollen müssen, um alle Inhalte betrachten zu können.

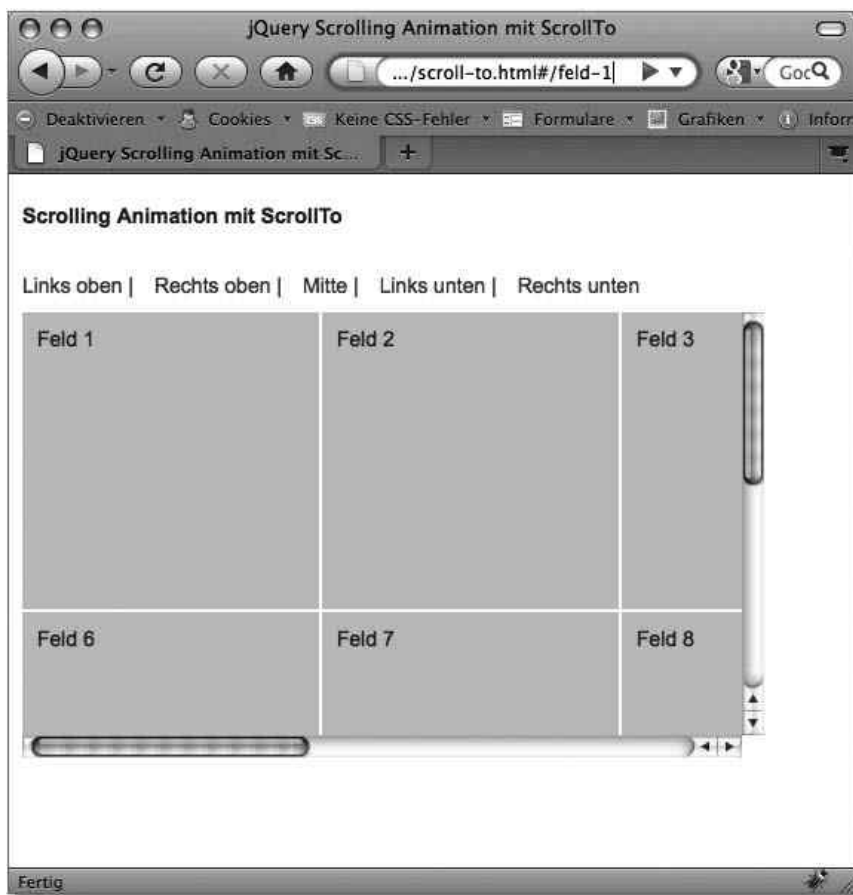


Abbildung 5.48 ScrollTo-Plugin

Um die Navigation zu erleichtern, platzieren Sie `<a>`-Elemente, denen Sie mittels jQuery die Anweisung zuweisen, zu welchem Punkt sich die Felder bewegen müssen. Die Bezeichner »Links oben«, »Rechts unten« etc. sind aussagekräftig genug.

Zusätzlich zum Plugin *jQuery.ScrollTo* können Sie noch das Plugin *jQuery Address* einbinden, sodass die Scrollziele in der Browserhistory registriert werden. Wie üblich, werden Sie als Erstes das HTML-Gerüst aufbauen:

```
<!DOCTYPE html>
<html>
<head>
<title>jQuery Scrolling Animation mit ScrollTo</title>
<link type="text/css" href="css/scroll-to.css" rel="stylesheet" />
<script type="text/javascript" src="js/jquery-1.4.2.js">
</script>
<script type="text/javascript" src="js/jquery.address-1.2.1.js">
</script>
<script type="text/javascript" src="js/jquery.scrollTo.js">
</script>
<script type="text/javascript" src="js/init-scroll-to.js">
</script>
</head>
<body>
  <h1>Scrolling Animation mit ScrollTo</h1>
  <a id="button-1" href="#" rel="address:/feld-1">
    Links oben
  </a>
  ...
  <div id="wrapper">
    <div id="container">
      <div id="feld-1">Feld 1</div>
      ...
      <div id="feld-15">Feld 15</div>
    </div>
  </body>
</html>
```

Listing 5.95 HTML-Gerüst – ScrollTo-Plugin

Die Buttons sind nur angedeutet. Fügen Sie unter dem `<a>`-Element mit dem ID `button-1` weitere vier Anker (nennen wir sie der Einfachheit halber »Buttons«) hinzu. Das Gleiche gilt für die `<div>`-Elemente für die »Felder«. Fügen Sie unter dem Element mit dem ID `feld-1` 14 weitere Felder hinzu. Jetzt werden Sie das Stylesheet bearbeiten. In der HTML-Seite ist es bereits unter dem Namen *scroll-to.css* eingebunden:

```
...
div#wrapper {
  margin:10px;
  width:500px;
```

```

    height:300px;
    overflow:auto;
}
div#container {
    width:1010px;
}
div#container div {
    width:180px;
    height:180px;
    padding:10px;
    float:left;
    border-right:#FFFFFF solid 2px;
    border-bottom:#FFFFFF solid 2px;
    background-color:#CCCCCC;
}
...

```

Listing 5.96 CSS Styles – ScrollTo-Plugin (Ausschnitt)

Wichtig ist, dass der Container mit dem ID `wrapper` kleiner ist als der Container mit dem ID `container` und dass `overflow` den Wert `auto` zugewiesen bekommt. Die Felder werden mit dem Selektor `div#container div` formatiert, mit `float:left` stellen Sie sicher, dass die Felder sich Stück für Stück von links nach rechts und dann Zeile für Zeile aneinander ausrichten.

Jetzt können Sie sich daran machen, den jQuery-Text aufzubauen:

```

$(document).ready(function(){
    var container = $("#wrapper");
    $.address.externalChange(function(event) {
        container.scrollTo("div#"+(event.value).replace("/",""),
            800);
    });
    $('#a#button-1').click(function(event){
        event.preventDefault();
        container.scrollTo( "div#feld-1", 800, { queue:true } );
    });
    $('#a#button-2').click(function(event){
        event.preventDefault();
        container.scrollTo( {top:'0px',left:'525px'}, 800 );
    });
    $('#a#button-3').click(function(event){
        event.preventDefault();
        container.scrollTo('50%', 800 );
    });
    $('#a#button-4').click(function(event){

```

```

        event.preventDefault();
        container.scrollTo( {top:'321px',left:'0px'},
                           800, { queue:true } );
    });
    $('a#button-5').click(function(event){
        event.preventDefault();
        container.scrollTo( {top:'321px',left:'525px'},
                           800, { queue:true } );
    });
});

```

Listing 5.97 JavaScript-Code – ScrollTo-Plugin

Im Grunde genommen ist das eine Aneinanderreihung von Event Handlern, die jeweils die Methode `.scrollTo()` aufrufen. Der erste Handler ist ein Handler des jQuery Address-Plugins, mit dem Sie sicherstellen, dass beim Betätigen mit den VOR- und ZURÜCK-Buttons das korrekte Sprungziel, besser »Animationsziel«, angesteuert wird (das kennen Sie aus dem letzten Beispiel). Zusätzlich wird das richtige Ziel angesteuert, falls Sie die Seite als Lesezeichen aufgenommen haben, und die Seite so ansteuern, oder den Link kopiert und in ein neues Browserfenster eingefügt haben: Sie erhalten volle Deep-Linking-Funktionalität.

Betrachten Sie nun die Optionen, die Sie verwenden können. Im Button `#button-1` haben Sie als Sprungziel einen Selektor angegeben: `div#feld-1`. Die Methode `.scrollTo()` ermittelt automatisch die Koordinaten und bewegt das Element auf die Punkte der oberen linken Ecke zu. Sie können auch, wie in den anderen Anknüpfungen angewendet, Werte für `top` und `left` als Objekt übergeben: `{ top:'0px',left:'525px' }`.

Es gibt noch weitere Möglichkeiten, die Sprungziele anzugeben:

- ▶ eine einfache Zahl
- ▶ einen String: `'+=30px'` oder `'100px'` oder `'30 %'`
- ▶ ein DOM-Element
- ▶ einen Selektor, wie gesehen: `div#feld-1`
- ▶ den String `'max'`, um an das Ende zu scrollen

Neben den Möglichkeiten, das Sprungziel anzugeben, können Sie noch weitere Optionen angeben. In den hier verwendeten Optionen sehen Sie noch die Dauer, die Sie, wie auch für die Methode `.animate()`, in Millisekunden angeben können. Sie können die Option `queue` auf `true` setzen, damit bewirken Sie, dass die Scrollbewegung über die X- und die Y-Achse nacheinander und nicht parallel abgearbeitet wird. Weitere Optionen im Überblick finden Sie in folgender Tabelle.

Option	Beschreibung
axis	Mögliche Werte: 'x', 'y', 'xy' oder 'yx'. 'xy' ist der Standardwert.
easing	Hier können Sie eine Formel für das Easing angeben, Sie können auch das <i>jQuery.easing</i> -Plugin einbinden und dessen Berechnungen als Werte setzen.
margin	true/false – wenn die Option true ist, werden Außenabstände abgezogen.
offset	Ein zusätzlich zu den Zielkoordinaten anzugebender Wert, der als Objekt notiert wird: { top: '5px', left: '5px' }
over	Wert, der mit den Werten der Zielkoordinaten multipliziert wird. Er verändert das Sprungziel um einen angegebenen Faktor.
onAfterFirst	Als Wert wird eine Funktion übergeben. Sie wird ausgeführt, falls Queue aktiviert wurde, und zwar nach der Ende der ersten Animation in der Queue-Warteschlange.
onAfter	Als Wert wird eine Funktion übergeben. Sie wird ausgeführt, sobald die gesamte Animation beendet wurde.

Tabelle 5.3 jQuery scrollTo-Plugin – Optionen

Das Plugin *jQuery.ScrollTo* ist eine wertvolle Hilfe im Umgang mit Animationen. Der Autor Ariel Flesler stellt neben vielen anderen noch zwei weitere Plugins zur Verfügung, die auf ScrollTo aufbauen. Damit sind Slideshows möglich, Paginierungen oder die von vielen Nutzern ungeliebten animierten Newsticker.

Sie haben in diesem Kapitel den Aufbau von Scrollfunktionen kennengelernt, einmal von Grund auf und selbst gemacht und einmal unter Zuhilfenahme eines Plugins. Sie haben in beiden Fällen das Zusammenspiel von mehreren Plugins erlebt und gesehen, wie einfach sie einsetzbar und wie leicht sie kombinierbar sein können. Sie haben gesehen, dass man manchmal sehr trickreich die Einschränkungen der Browser umgehen kann, müssen aber in Kauf nehmen, zusätzliches HTML-Markup zu produzieren, das Sie aber getrost über jQuery generieren können.

Die hier angerissenen Beispiele können beliebig weitergesponnen werden. Hier wurden nur ein Bruchteil der verfügbaren und wundervollen Plugins vorgestellt. Es gibt unzählige weitere taugliche Plugins für Scrollfunktionen, für das Auslesen des Browserhistory-Objekts. Es gäbe sicherlich auch einiges am Code des jQuery Address-Plugins zu verbessern ... Dass es ständig überarbeitete Plugins gibt, das ist das Schöne an jQuery. Es existiert eine große, überaus kreative Community, die stets neuen Input für das gesamte Projekt zur Verfügung stellt und so das Wissen und die Tools rund um das Framework bereichert. Machen Sie sich auf die Suche!

5.11 Going mobile mit jQTouch

Es muss nicht mehr darüber debattiert werden, ob im Bereich Webdesign der Sektor »Mobilgeräte« eine größere Rolle spielen wird oder nicht. Diese Zukunft ist längst absehbar – einige Analysten sprechen bereits davon, dass mobile Clients wie Smartphones, Netbooks oder Personal Assistants eher über kurz als über lang dem Desktop-Computer den Rang ablaufen werden, was den Zugang zu internet-gestützten Informationen und Webdiensten angeht. Zeit also, sich mit der Thematik auseinanderzusetzen.

Ohne ins Detail zu gehen, kann man davon ausgehen, dass auf allen neuen Geräten der eben genannten Klassen ein Webbrowser laufen wird, der im Kern aktuellen Desktop-Browsern entspricht. Oft wird dies Webkit sein (also Safari) oder Gecko (sprich Firefox). Ebenfalls wichtig sind Opera-Browser wie Opera mini. Sie alle können mit JavaScript umgehen. Trotz anderer zu berücksichtigender Randbedingungen (geringere Prozessorleistung, Akkubetrieb, meist kleineres Display und mehr) bedeutet dies, dass auch für Websites, die auf Mobilgeräte ausgerichtet sind, ein Framework wie jQuery einsetzbar ist.

So weit, so gut. Allerdings ist ein Mobilgerät kein Desktop-Browser und verfügt (meist) über keine Tastatur oder Maus als Eingabegeräte. Vielmehr setzen sich (und dies gilt auch für Netbooks oder Notebooks) zunehmend Touchdisplays als Eingabemedium durch. Dies hat Folgen:

- ▶ Touchevents sind keine Mouse-Events.
- ▶ Die Bedienungsmetaphern sind andere (Wischen, Blättern).
- ▶ Es treten neuartige Events wie Mehrfingergersten auf (für Zoom etc.).

Im August 2010 hat das jQuery Team angekündigt, bis zum Jahresende eine eigens für mobile Plattformen zugeschnittene Version von jQuery zu veröffentlichen. Dieses neuartige Framework wird nicht nur äußerst schlank sein, es soll auch spezielle UI Widgets erhalten, die über einen Themeroller konfiguriert werden können, es wird Event Handler für die neuartigen Gesten wie Wischen und Blättern geben. Es wird neben dem Browser Webkit in einer Reihe von weit verbreiteten mobilen Browsern unterstützen. Im Moment kann man nur erahnen, wohin die Reise geht, aber es gibt bereits ein Plugin, das viele der neuen Funktionen abbildet, mit Namen *jQTouch*. Es gibt gewissermaßen einen Ausblick auf das *Mobile Design* mit jQuery der nächsten Generation, mit einfachen Mitteln sollten so ansprechende *Online Apps* erstellt werden können.

Unter <http://jquerymobile.com> können Sie sich über den aktuellen Stand der Entwicklung auf dem Laufenden halten. Auch wenn jQuery mit einem eigenen mobilen Framework in Erscheinung treten wird, erhalten Sie schon jetzt mit dem

Plugin jQTouch von David Kaneda einen Einblick, wie man zukünftig genauso einfach Mobile Apps erstellen kann, wie man es heute mit Websites machen kann. Sie erhalten jQTouch unter folgender Adresse: www.jqtouch.com/



Abbildung 5.49 Startseite von www.jqtouch.com

5.11.1 Emulatoren und IDEs für Mobilgeräte

Will man für Mobilgeräte entwickeln, wird man zwar am Ende nicht darum herumkommen, auch auf dem »echten« Zielgerät zu testen. Sämtliche Geräte hierfür vorrätig zu haben, verbietet sich aus Kostengründen allerdings von selbst. Daher greift man vorerst – auch wir werden dies hier tun – zu Emulatoren, die Oberfläche und Bedienung (günstigstenfalls auch das Verhalten) eines Mobilgeräts mehr oder weniger gelungen auf dem Desktoprechner nachempfinden. Hierzu gleich mehr.

Ein dritter (der einfachste) Weg besteht in der Nutzung eines normalen Desktop-Browsers für den Test einer Mobilwebsite, indem man (wie es Opera von Hause aus und Firefox über Chris Pedericks Web Developer Toolbar bietet) dessen sogenannten *Smallscreen-Mode* nutzt. Allerdings stellt dies bestenfalls eine Annäherung an die Realität dar und keine Kontrolle dafür, wie sich ein echtes Mobilgerät verhalten würde.

Kommen wir daher zu den Emulatoren zurück, von denen praktisch jede Firma, die mit Mobilgeräten zu tun hat, einige im Angebot hat.

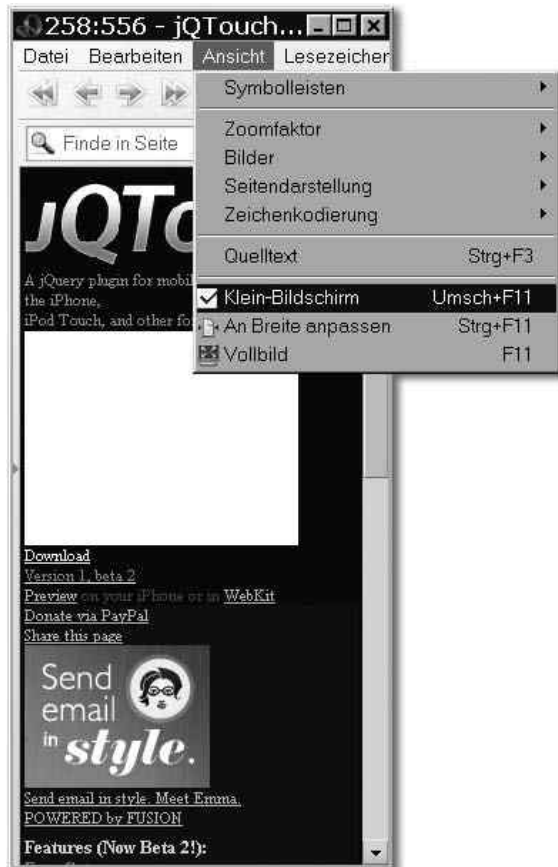


Abbildung 5.50 Opera im Smallscreen-Mode (www.jqtouch.com)

Sie finden Emulatoren und Entwicklungsumgebungen u. a. bei:

- ▶ **Research in Motion**
<http://na.blackberry.com/eng/developers/>
- ▶ **Palm**
<http://developer.palm.com/>
- ▶ **Android**
<http://developer.android.com/>
- ▶ **Openwave**
<http://developer.openwave.com>
- ▶ **Nokia**
<http://www.forum.nokia.com/>

5.11.2 Emulatoren und IDEs für iPhone

In diesem Abschnitt beschränken wir uns auf das iPhone von Apple. Man kann jedoch davon ausgehen, dass die hier betrachteten Applikationen auf anderen Plattformen ebenso gut lauffähig sind.

Apple selbst stellt für die Entwicklung von iPhone-Applikationen eine Umgebung zur Verfügung, das *iOS SDK*, die auch einen Emulator für das iPhone enthält. Sie erhalten es hier:

► **Apple**

<http://developer.apple.com/iphone/>

Leider hat dieses SDK einen empfindlichen Nachteil: Es ist nur für Mac-User einsetzbar, da es nicht in einer Version für Windows existiert. Das iOS SDK setzt *zwingend Mac OS X* als Plattform voraus.



Abbildung 5.51 iOS SDK auf Mac OS X

Zum Glück existieren inzwischen auch für die *Windows-Plattform* brauchbare Emulatoren, von denen hier zwei kurz vorgestellt werden. Es ist zum einen der Emulator vom MobiOne, der praktisch eine ganze Entwicklungsumgebung nebst Oberflächendesigner mitbringt, zum anderen der iPhone/iPad-Emulator iBBDemo, der zwar spartanisch, aber ressourcenschonender ist.

iPhone/iPad Emulator iBBDemo2

Mit Shaun Sullivans iPhone Emulator *iBBDemo2* ist es auch unter anderen Plattformen als Mac OS X möglich, für iPhone und iPad optimierte Websites zu entwickeln und zu testen. Das Programm verwendet als Basis Adobe AIR 2, das daher mitinstalliert wird (sofern nicht bereits auf dem Zielrechner vorhanden)

und den Emulator so unter Windows oder Linux zur Verfügung stellt. Das Ergebnis ist eine ähnliche Ansicht, wie sie einen auf dem iPhone oder iPod Touch erwartet. Auch funktional kommt iBBDemo2 recht gut an das Original heran.

Sie erhalten iBBDemo2 hier:

<http://www.puresimstudios.com/ibbdemo/>



Abbildung 5.52 iBBDemo2 unter Windows (iPhone-Modus)

Bevor Sie später lange suchen, werden hier die wichtigsten Tastenkürzel zur Steuerung von iBBDemo2 aufgelistet, da der Emulator selbst kein Menü bietet.

Shortcut	Funktion
Strg + 1	Oberfläche iPad
Strg + 2	Oberfläche iPhone
Strg + → / Strg + ←	Umschaltung Portrait/Landscape
Strg + + / Strg + -	Zoom in/out
Strg + D	Adressleiste ein-/ausblenden

Tabelle 5.4 Tastatur-Shortcut für iBBDemo2 (Auszug)

Als kleine Randbemerkung sei noch gesagt, dass iBBDemo auf Wunsch auch Flash-Inhalte ausführt – im Gegensatz zu den Originalgeräten, die simuliert werden. Von Haus aus ist das Feature jedoch deaktiviert.

MobiOne iPhone Emulator für Windows

MobiOne bezeichnet sich selbst als das »beste derzeitig am Markt befindliche« Emulatorsystem für iPhone und Palm Pre und könnte damit durchaus richtig liegen. Das System befindet sich noch in der Entwicklungsphase, verfügt jedoch bereits über zahlreiche Features, die es bei Fertigstellung quasi als IDE für Mobilgeräte prädestinieren. Für die Zukunft ist ein integrierter Editor für HTML/CSS, eine Portierung nach Linux und Mac OS X sowie die Emulation weiterer Endgeräte (u.a. Blackberry und Android) vorgesehen. Derzeit ist die Umgebung noch kostenlos erhältlich, was sich jedoch irgendwann ändern mag.

Sie erhalten MobiOne beim Hersteller *Genuitec*:

<http://www.genuitec.com/mobile/>



Abbildung 5.53 MobiOne unter Windows (Test Center im iPhone-Mode)

Die Umgebung installiert sich in Form zweier Untieranwendungen, dem »Test Center« (der eigentliche Emulator, der zwischen iPhone und Palm Pre umgeschaltet werden kann) und dem »Design Center«, in dem man Weboberflächen per Drag & Drop gestalten kann. Darüber hinaus werden ein Debugger, ein DOM-Inspektor und noch viel mehr geboten. Eine ausführlichere Vorstellung der umfangreichen Möglichkeiten von MobiOne verbietet sich an dieser Stelle leider aus Platzgründen.

5.11.3 Ins mobile Web mit jQuery und jQTouch

Beginnen wir relativ kommentarlos damit, dass wir jQTouch in eine HTML5-Seite einbinden. Dies geschieht, wie man es von Plugins her kennt, einfach durch Verlinken der benötigten JavaScript- und CSS-Ressourcen.

Letztere sind nötig, da es beim Einsatz von jQTouch einerseits um Funktionalität geht, andererseits um »Eye Candy«. Dies schlägt sich darin nieder, dass eine jQTouch-Seite stets mit einem Theme verlinkt sein sollte, das das Grundlayout festlegt. An die zu stylenden Seitenelemente wird hierfür eine CSS-Klasse gebunden, falls sich die Rolle und damit die Präsentation eines Elements nicht aus der Dokumentstruktur erschließt (die auch ein wenig reglementiert ist). Doch beginnen wir von vorn. Hier der Dokumentkopf nach der Einbindung (wir bleiben bei HTML5):

```
<!DOCTYPE HTML>
<html lang="de-DE">
<head>
<meta charset="UTF-8">
<title>jQuery | Listing 5.98</title>
  <link rel="stylesheet" type="text/css"
    href="../js/jqtouch/jqtouch.css"/>
  <link rel="stylesheet" type="text/css"
    href="../js/themes/jqt/theme.css"/>
<script type="text/javascript" src="/js/jquery-1.4.2.js"></script>
  <script src="../js/jqtouch/jqtouch.js" type="application/x-
javascript" charset="utf-8"></script>
<script type="text/javascript">
  $.jQTouch({
    statusBar: 'black'
  });
</script>
</head>
<body>

</body>
</html>
```

Listing 5.98 jQTouch-Beispiel 01

Vielleicht ein Seitenblick auf den Scriptcontainer, der hier so aussieht:

```
<script type="text/javascript">
    $.jQueryTouch({
        statusBar: 'black'
    });
</script>
```

Hier geschieht die Grundkonfiguration der jQueryTouch-Seite, indem der \$.jQueryTouch-Methode ein Konfigurationsobjekt übergeben wird. Allerdings wird an dieser Stelle lediglich das Aussehen der Statusleiste festgelegt. Natürlich ist viel mehr möglich. Sehen wir uns stattdessen im Body um.

jQueryTouch-Dokumente können mehrere sogenannte *Seiten* enthalten, die jeweils aus <div>-Containern mit ID bestehen. Wer sich an dieser Stelle an das alte Schema in WML erinnert, ein Multidokument aus »Decks« und »Cards« zusammenzusetzen, liegt nicht ganz falsch.

Im Body liegen also <div>-Container. In diesem Fall hätten wir vier »Seiten«. Im momentanen Zustand ist allerdings noch nicht viel mehr zu sehen als der schwarze Hintergrund (siehe Abbildung 5.54, links):

```
<body>
    <div id="s1"></div>
    <div id="s2"></div>
    <div id="s3"></div>
    <div id="s4"></div>
</body>
```

Der ID einer Seite (bleiben wir bei diesem Begriff) dient auch als Linkziel. Da jeweils nur eine Seite gezeigt werden wird, benötigen wir in allen Fällen eine Navigationsmöglichkeit, um auf eine andere Seite zu gelangen. Dies kann als vollständige Navigation zur Verfügung gestellt werden oder als Link im Bereich der sogenannten *Toolbar*.

Meist wird man einer Seite eine Toolbar zuweisen wollen. Sie besteht aus einem <div>-Container mit der Klasse `toolbar`, der eine Überschrift `h1` beinhaltet, die als Seitenüberschrift dient.

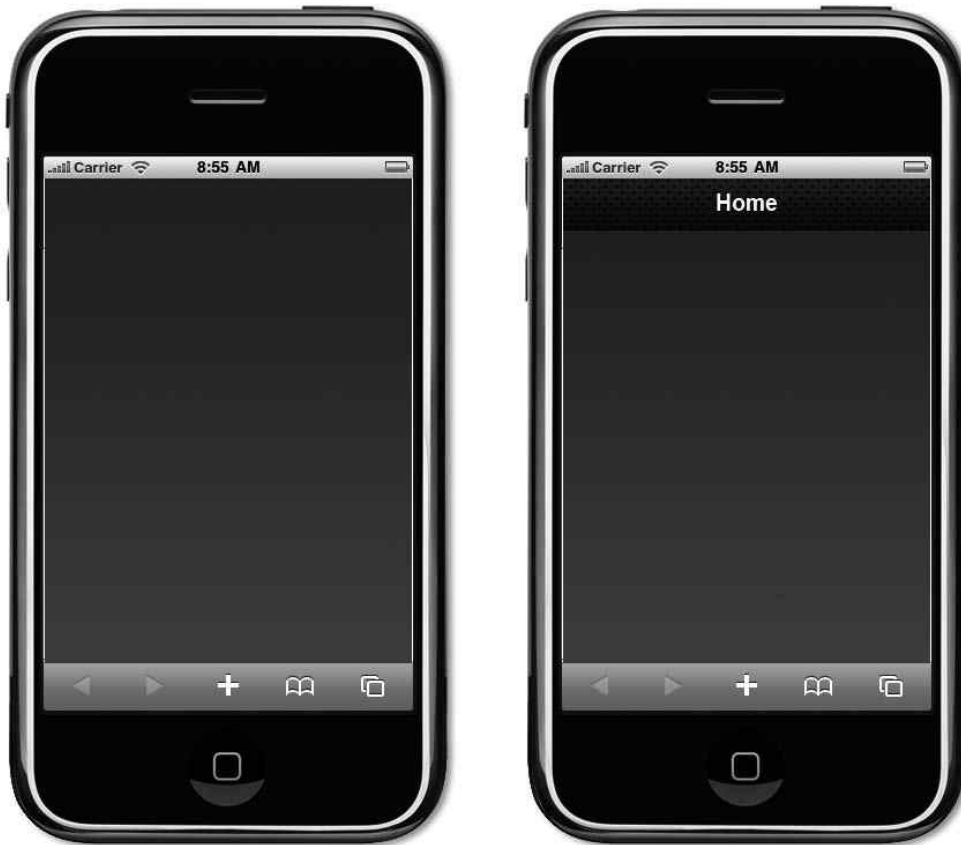


Abbildung 5.54 jQTouch-Seiten ohne und mit Toolbar

Erweitert man die ersten beiden Seiten jeweils um die Toolbar, bietet sich folgendes Bild:

```
<body>
  <div id="s1">
    <div class="toolbar">
      <h1>Home</h1>
    </div>
  </div>
  <div id="s2">
    <div class="toolbar">
      <h1>Seite 2</h1>
    </div>
  </div>
  <div id="s3">
  </div>
  <div id="s4">
  </div>
</body>
```

Listing 5.99 jQTouch-Beispiel 02

Das können wir bereits erfolgreich im Emulator betrachten (siehe Abbildung 5.54, rechts). Es ist aber lediglich die Seite HOME zu sehen (defaultmäßig wird stets die erste Seite des Dokuments angezeigt). Um zu den anderen Seiten zu gelangen, muss eine Navigation her.



Abbildung 5.55 Home- und Zurück-Buttons in der Toolbar

Eine Navigation wird auch in jQTouch durch eine `<ul>`-Liste definiert. Die Optik der Navigation wird durch eine CSS-Klasse gesteuert. Wir wählen die Klasse `edgetoedge`:

```
<ul class="edgetoedge">
  <li><a href="#s2">Seite 2</a></li>
  <li><a href="#s3">Seite 3</a></li>
  <li><a href="#s4">Seite 4</a></li>
</ul>
```

Jeweils angepasst (auf die aktuelle Seite muss ja nicht verlinkt werden), wird diese Navigation nun in Seite 1 (HOME) und Seite 2 eingefügt. Seite 3 und 4 lassen wir nach wie vor leer:

```
<div id="s1">
  <div class="toolbar">
```

```

        <h1>Home</h1>
    </div>
    <ul class="edgetoedge">
        <li><a href="#s2">Seite 2</a></li>
        <li><a href="#s3">Seite 3</a></li>
        <li><a href="#s4">Seite 4</a></li>
    </ul>
</div>
<div id="s2">
    <div class="toolbar">
        <h1>Seite 2</h1>
    </div>
    <ul class="edgetoedge">
        <li><a href="#s1">Home</a></li>
        <li><a href="#s3">Seite 3</a></li>
        <li><a href="#s4">Seite 4</a></li>
    </ul>
</div>

```

Listing 5.100 jQTouch-Beispiel 03

Sie können jetzt von der HOME-Seite zur Seite 2 wechseln. Wie Sie bemerken, geschieht dies mit einer Slide-Animation, bei der die neue Seite stets von rechts nach links in den Viewport fährt. Seite 3 und 4 stellen zurzeit noch Sackgassen dar. Damit das nicht so bleibt, sollen sie jetzt auch erweitert werden.

Seite 3 bekommt nun auch ihre Toolbar:

```

<div class="toolbar">
    <h1>Seite 3</h1>
    <a href="#s1" class="button">Home</a>
</div>

```

Diesmal enthält die Toolbar außer der Überschrift noch einen Link, der die Seite neu lädt (siehe Abbildung 5.55, links). Damit er die gewünschte Wirkung zeigt, nämlich die HOME-Seite aufzurufen, muss der Link auf den entsprechenden Div zeigen (sonst geht es nicht). Die Klasse `button` sorgt für Position und Aussehen. Auf der linken Seite ist noch Platz für einen weiteren Button. Fügen Sie einen zweiten Link hinzu und vergeben diesmal die Klassen `button` und `back` (siehe Abbildung 5.55, rechts):

```

<div class="toolbar">
    <h1>Seite 3</h1>
    <a href="#s1" class="button">Home</a>
    <a href="#" class="button back">Zurück</a>
</div>

```

Dem neuen Button genügt der Hash als href-Wert. Er benötigt auch kein festes Ziel, da seine Aufgabe darin besteht, stets zur vorhergehenden Seite zu springen. Nun soll auch Seite 4 endlich einen Link-Button bekommen und dazu noch etwas Inhalt (siehe Abbildung 5.56, links):

```
<div id="s4">
  <p style="margin:14px;padding:5px;
    color:black;background:white;">
    <b>Alice</b> schloß die Thür auf und fand, daß sie zu
    einem kleinen Gange führte, nicht viel größer als ein
    Mäuseloch. Sie kniete nieder und sah durch den Gang in
    den reizendsten Garten, den man sich denken kann.<br/>

    
  </p>
  <div>
    <a style="margin:10px;color:rgba(250,250,250,.7)"
      href="#" class="whiteButton goback">
      Und zur&uuml;ck!
    </a>
  </div>
</div>
```

Listing 5.101 jQTouch-Beispiel 04



Abbildung 5.56 Back-Button und Navigation »rounded« (Theme »jq«)

Nun kommen mehrere Dinge hinzu. Über dem ZURÜCK-Button soll ein WEITER-Link erscheinen, der auf eine Folgeseite #s5 zeigt. Auch diese soll einen ZURÜCK- und einen WEITER-Link besitzen, wobei der WEITER-Link wiederum auf eine neue Seite #s6 zeigt (siehe Abbildung 5.56, rechts). Diese letztlich verfügt nur noch über den ZURÜCK-Button.

Einen WEITER-Link realisieren wir wieder mit einer `<ul>`. Diesmal bekommt diese nicht die Klasse `edgetoedge`, sondern die Klasse `rounded`. Auch das LI bekommt eine Klasse, und zwar `arrow`. Hier exemplarisch der Link:

```
<ul class="rounded">
  <li class="arrow"><a href="#s5">weiter</a></li>
</ul>
```

Hier der gesamte zusätzliche Quelltext (und ja, alles findet noch immer in ein und derselben HTML-Seite statt):

```
<div id="s4">
  ...
  <ul class="rounded">
    <li class="arrow"><a href="#s5">weiter</a></li>
  </ul>
</div>
  <a style="margin:10px;color:rgba(250,250,250,.7)"
    href="#" class="whiteButton goback">
    Und zur&uuml;ck!</a>
</div>
</div>
<div id="s5">
  <p style="margin:14px;padding:5px;
    color:black;background:white;">
    Wie w&uuml;nchte sie, aus dem dunkeln Corridor zu gelangen,
    und unter den bunten Blumenbeeten und k&uuml;hlen
    Springbrunnen umher zu wandern; aber sie konnte kaum den
    Kopf durch den Eingang stecken. "Und wenn auch mein Kopf
    hindurch ginge," dachte die arme Alice, "was w&uuml;rde es
    n&uuml;tzen ohne die Schultern?"<br/>
    
  </p>
  <ul class="rounded">
    <li class="arrow"><a href="#s6">weiter</a></li>
  </ul>
</div>
  <a style="margin:10px;color:rgba(250,250,250,.7)"
    href="#" class="whiteButton goback">
    Und zur&uuml;ck!</a>
</div>
```

```

</div>
<div id="s6">
  <p style="margin:14px;padding:5px;
        color:black;background:white;">
    "Oh, ich möchte mich zusammenschieben können wie ein
    Teleskop! Das geht gewiß, wenn ich nur wüßte, wie man es
    anfängt." Denn es war kürzlich so viel Merkwürdiges mit
    ihr vorgegangen, daß Alice anfang zu glauben, es sei
    fast nichts unmöglich. Es schien ihr ganz unnütz, länger
    bei der kleinen Thür zu warten. Daher ging sie zum Tisch
    zurück, halb und halb hoffend, sie würde noch einen
    Schlüssel darauf finden, oder jedenfalls ein Buch mit
    Anweisungen, wie man sich als Teleskop zusammenschieben
    könne.</p>
  <div>
    <a style="margin:10px;color:rgba(250,250,250,.7)"
      href="#" class="whiteButton goback">
      Und zur&uuml;ck!</a>
  </div>
</div>

```

Listing 5.102 jQTouch-Beispiel 05

Zum Styling: Sicher haben Sie bemerkt, dass die Vergabe einer CSS-Klasse das Layout stark beeinflusst. Die `<ul>`-Liste wirkt völlig anders, wenn Sie ihr statt `edgetoedge` die Klasse `rounded` zuweisen. Versuchen Sie auch mal `metal` oder `plastic`.

Klasse	Beschreibung
<code>edgetoedge</code>	Liste nimmt ganze Viewport-Breite ein.
<code>rounded</code>	Runde Ecken, seitliches Margin im Viewport
<code>metal</code>	Ganze Viewport-Breite, Top- und Bottom-Padding, silber-metallischer Farbverlauf über Hintergrundgrafik
<code>plastic</code>	Ganze Viewport-Breite, ähnlich <code>edgetoedge</code>
<code>individual</code>	Halbe Viewport-Breite, runde Ecken

Tabelle 5.5 jQTouch CSS-Klassen für Navigationslisten (»`<ul>`«)

Auch die LI können für Navigationslisten mit Klassen versehen werden. Beachten Sie, dass eine Abhängigkeit von der Klasse des umgebenden `<ul>` besteht!

Klasse	Beschreibung
<code>arrow</code>	Blendet Pfeil auf dem Navigationsbalken ein; nicht in Zusammenhang mit <code>edgetoedge</code> anwendbar.

Tabelle 5.6 jQTouch CSS-Klassen für Navigationslisten (LI)

Klasse	Beschreibung
forward	Blendet sogenannten <i>Chevron-Pfeil</i> (runde Plakette) ein.
sep	Minimale Höhe, Farbverlauf (für <code>edgetoedge</code>)

Tabelle 5.6 jQTouch CSS-Klassen für Navigationslisten (LI) (Forts.)

Noch deutlicher ist das Styling bei den Toolbar-Buttons, bei denen die Klasse `back` Aussehen, Position und Funktionalität beeinflusst.

Klasse	Beschreibung
button	Button rechts in der Toolbar
back	Button links in der Toolbar (als Pfeil nach links)
cancel	Button links in der Toolbar

Tabelle 5.7 jQTouch CSS-Klassen für Toolbar-Buttons

Auch die Buttons im Inhaltsbereich werden durch einen Grundstyle festgelegt (in unserem Beispiel wurde die Schriftfarbe durch ein `style`-Attribut verändert, nicht aber das Gesamterscheinungsbild).

Klasse	Beschreibung
blueButton	Blaue Hintergrundgrafik (nicht im <code>jqt</code> -Theme!)
whiteButton	Weißer Hintergrundgrafik
grayButton	Graue Hintergrundgrafik

Tabelle 5.8 jQTouch CSS-Klassen für allgemeine Buttons

Grundsätzlich gilt: Auch einiges an Funktionalität wird über CSS-Klassen festgelegt. Experimentieren Sie ein wenig damit, indem Sie den `<a>`-Tags innerhalb der WEITER-Links ebenfalls eine Klasse zuweisen. Etwa so:

```
<ul class="rounded">
  <li class="arrow"><a href="#s5" class="pop">
    weiter</a></li>
</ul>
```

Oder so:

```
<ul class="rounded">
  <li class="arrow"><a href="#s6" class="flip">
    weiter</a></li>
</ul>
```

Mit diesen Klassen werden unmittelbar die Animationen für Seitenübergänge getriggert. Folgende Klassen sind per Default verfügbar:

Klasse	Animation
slideup	Neues Panel fährt von unten nach oben ein.
dissolve	Pixelüberblendung zum neuen Panel
fade	Fade-Überblendung zum neuen Panel
flip	Neues Panel dreht sich in den Viewport.
pop	Neues Panel wird aus der Viewport-Mitte aufgezo-gen.
swap	Neues Panel schiebt sich von rechts über das bestehende Panel.
cube	Mischung aus fade- und swap-Übergang ¹

Tabelle 5.9 jQueryTouch CSS-Klassen für Seitenübergänge

Nun soll auf dem Panel »Seite 3« ein Formular angezeigt werden. Das `<form>`-Element wird einfach nach der Toolbar in den `<div>`-Container `#s3` eingefügt. Um die Eingabeelemente auf Linie zu bekommen, ist die Standardvorgehensweise in jQueryTouch, sie mit einer `<ul>`-Liste zu umgeben. Diese erhält die beiden Klassen `edit` und `rounded`:

```
<form>
<ul class="edit rounded" >
  <li>
    <input type="text" name="user"
      id="user" placeholder="User" />
  </li>
  <li>
    <input type="password" name="pw"
      id="pw" placeholder="Passwort" />
  </li>
  <li>Newsletter:
    <span class="toggle">
      <input type="checkbox" name="nl" id="nl" />
    </span>
  </li>
  <li>
    <textarea name="kom" rows="4">
      Ihr Kommentar
    </textarea>
  </li>
</ul>
```

¹ Schauen Sie sich das einfach selbst an.

```

        <input style="padding:6px;font-size:smaller;color:#999"
            type="submit" value="Abschicken" />
    </li>
</ul>
</form>

```

Listing 5.103 jQTouch-Beispiel 06

Interessant sind hier folgende Details: In den Textinputs tritt an die Stelle des `value`-Attributs ein `placeholder`-Attribut. Die Aufgabe ist dieselbe, nämlich einen Defaulttext im Input zu platzieren. Dieser wird bei `placeholder` allerdings entfernt, sobald der User das Feld in den Fokus nimmt. Spannend ist die Lösung für die Checkbox, die mit einem umliegenden `<span>` mit der Klasse `toggle` arbeitet. Die Checkbox wird durch eine Grafik ersetzt. Hier ein kleiner Auszug aus *theme.css*:

```

.toggle input[type="checkbox"] {
    ...
    background: #fff url(img/on_off.png) 0 0 no-repeat;
    ...
}

```

Die optische Anmutung können Sie in Abbildung 5.57 sehen.



Abbildung 5.57 Ein Formular mit Toggle-Checkbox (Theme »jq«)

Die ganze Zeit über haben wir nur mit dem Theme `jqt` gearbeitet, das über den CSS-Link

```
<link rel="stylesheet" type="text/css"
      href="../../js/themes/jqt/theme.css"/>
```

festgelegt wurde.

Dieses Theme ist jedoch nur eines von zweien, die zur Verfügung stehen. Bevor Sie daran gehen, ein eigenes zu entwerfen (nicht unmöglich, aber arbeitsintensiv), versuchen Sie einmal diese Variante:

```
<link rel="stylesheet" type="text/css"
      href="../../js/themes/apple/theme.css"/>
```

Listing 5.104 jQTouch-Beispiel 07



Abbildung 5.58 jQTouch mit Theme »apple«

Interessant an jQTouch ist die Konfigurationsmöglichkeit. Zu Beginn war bereits angedeutet, dass hierfür der Initialisierungsfunktion ein Objekt übergeben werden muss. Betrachten wird das nun ein wenig näher.

5.11.4 Konfiguration von jQTouch

Sie erinnern sich noch an die Einbindung des Plugins, bei der zwei CSS-Dateien und eine JavaScript-Datei mit der Seite verknüpft wurden. Anschließend folgte noch ein Scriptblock, in dem sich jedoch nicht sonderlich viel abspielte:

```
<script type="text/javascript">
    $.jqTouch({
        statusBar: 'black'
    });
</script>
```

Dass dies genügt, wurde im Vorfeld zur Genüge bewiesen. Im Grunde wurde kaum Arbeit in die Seiten gesteckt, sieht man von der Vergabe der geeigneten CSS-Klassen einmal ab (die man natürlich kennen muss). Alle relevanten Vereinbarungen werden nämlich in einem Defaultobjekt getroffen (Sie erinnern sich, dass darauf im Abschnitt 5.6, »Plugin-Entwicklung«, hingewiesen wurde), der zuvor übergebene Wert überschreibt lediglich einen einzigen von 21 Konfigurationswerten.

In den folgenden Tabellen finden Sie eine Übersicht über die Konfigurationsmöglichkeiten, die sich Ihnen bei der Initialisierung von jQTouch bieten.

Eigenschaft	Erwarteter Wert	Erläuterung
addGlossToIcon	true/false	Setzt einen Glanz auf das Webclip-Icon. Default: true
icon	Der Wert null oder ein relativer oder absoluter Pfad zum Web-Icon	Das Aushängeschild der Web-App, das beim Speichern auf dem Homescreen als Icon verwendet wird. Eine PNG-Datei, deren Abmessungen 57x57 px betragen muss. Default: null
startupScreen	Der Wert null oder ein relativer oder absoluter Pfad zu einer IMG-Datei	Bezeichnet den Pfad zur Startgrafik einer Fullscreen-App. Deren Größe sollte 320x460 px betragen. (Hinweis: Ist die Statusbar auf "black-transparent", also teiltransparent, gesetzt, verwenden Sie besser 320x480 px). Default: null

Tabelle 5.10 Konfiguration von jQTouch (Icon, Startscreen, Statusbar)

Eigenschaft	Erwarteter Wert	Erläuterung
statusBar	Ein Wert aus default, black, black-translucent	Steuert das Erscheinungsbild der Statusleiste, wenn die App im Fullscreen-Modus läuft. Erlaubt gegenwärtig nur drei vorgegebene Werte. Default: 'default'

Tabelle 5.10 Konfiguration von jQuery Touch (Icon, Startscreen, Statusbar) (Forts.)

Eigenschaft	Erwarteter Wert	Erläuterung
cacheGetRequests	true/false	Legt fest, ob die Ergebnisse aus GET-Requests gecacht werden oder nicht, um anschließend auf diese Daten lokal zuzugreifen. Default: true
fixedViewport	true/false	Setzt die Zoom-Funktion außer Kraft. Default: true
formSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Nennt den Elementtyp, auf den die Anweisungen für das Formularlayout des aktuellen Themes angewendet werden sollen. Default: 'form'
fullScreen	true/false	Startet die App im Fullscreen-Modus, wenn auf true gesetzt und die App vom Homescreen aus aufgerufen wird (ohne Effekt in Mobile Safari). Default: true
fullScreenClass	String	Bestimmt die CSS-Klasse, die im Fullscreen-Mode dem <body>-Element zugewiesen wird, um so für diesen Modus spezielle Styleanweisungen zu aktivieren. Default: 'fullscreen'
preloadImages	Ein Array aus Pfaden, die auf IMG-Dateien zeigen, die im Vorfeld zu laden sind	Beispiel: ['img/link_aktiv.png', 'img/link_auswahl.png'] Default: false

Tabelle 5.11 Konfiguration von jQuery Touch (Basiskonfiguration)

Eigenschaft	Erwarteter Wert	Erläuterung
submitSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt die CSS-Klasse des Elements eines Formulars, das auf Klick dessen Submit-Vorgang einleitet und gegebenenfalls das eingeblendete Keyboard schließt. Default: <code>'.submit'</code>

Tabelle 5.11 Konfiguration von jQTouch (Basiskonfiguration) (Forts.)

Eigenschaft	Erwarteter Wert	Erläuterung
backSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Nimmt eine Liste von CSS-Selektoren als String entgegen, die Elemente kennzeichnen, die als »Zurück«-Trigger dienen sollen. Default: <code>'.back, .cancel, .goback'</code>
cubeSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt die CSS-Klasse des Trigger-Elements, das das Target-Panel per Cube-Animation einblendet. Default: <code>'.cube'</code>
dissolveSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt die CSS-Klasse des Trigger-Elements, das das Target-Panel per Dissolve-Animation einblendet. Default: <code>'.dissolve'</code>
fadeSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt die CSS-Klasse des Trigger-Elements, das das Target-Panel per Fade-Animation einblendet. Default: <code>'.fade'</code>
flipSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt die CSS-Klasse des Trigger-Elements, das das Target-Panel per Flip-Animation einblendet. Default: <code>'.flip'</code>
popSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt die CSS-Klasse des Trigger-Elements, das das Target-Panel per Pop-Animation einblendet. Default: <code>'.pop'</code>

Tabelle 5.12 Konfiguration von jQTouch (Seitenübergänge)

Eigenschaft	Erwarteter Wert	Erläuterung
slideInSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt den CSS-Selektor des Trigger-Elements, das das Target-Panel per SlideIn (horizontal) einblendet. Entspricht dem Verhalten normaler Linklisten. Default: 'body * ul li a'
slideupSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt die CSS-Klasse des Trigger-Elements, das das Target-Panel per SlideUp (von unten nach oben) einblendet. Default: '.slideup'
swapSelector	Beliebige, in der Liste durch Kommata getrennte CSS-Selektoren	Bestimmt die CSS-Klasse des Trigger-Elements, das das Target-Panel per Swap einblendet. Default: '.swap'
useAnimations	true/false	Unterbindet alle Animationen, wenn auf false gesetzt. Default: true

Tabelle 5.12 Konfiguration von jQTouch (Seitenübergänge) (Forts.)

5.11.5 Das jQTouch-Objekt referenzieren

Noch einmal zurück auf Start – wichtig ist, dass die jQTouch-Methode *überhaupt* aufgerufen wird. Sie könnten jQTouch auch so initialisieren:

```
<script type="text/javascript">
  $.jqTouch();
</script>
```

In diesem Fall würden ausschließlich Defaultwerte gelten. Noch ein anderer Aspekt. So wurde jQTouch quasi »freischwebend« aufgerufen. Es gibt keine Möglichkeit, an das so initialisierte Funktionsobjekt heranzukommen – falls man dies überhaupt wollen sollte.

Warum aber sollte man? Ganz einfach deshalb, weil das jQTouch-Objekt selbst einige Methoden und Eigenschaften besitzt, über die man es manipulieren kann. Und das Objekt zu manipulieren bedeutet, Zugriff auf die History und auf Animationen zu besitzen.

Aus diesem Grunde erzeugt man, falls man so etwas benötigt, eine Referenz:

```
<script type="text/javascript">
  // Rückgabewert wird in Referenz JQT gespeichert:
  var JQT = $.jQTouch();
</script>
```

Wie Sie das referenzierte Objekt nennen, ist eigentlich egal – Hauptsache, Sie fangen den Rückgabewert der Initialisierungsfunktion in einer Variablen auf. Dann können Sie Folgendes tun:

```
// zur vorletzten Seite zurückgehen
JQT.goBack(2);
// Zur Seite #s1 springen (falls diese in der History ist)
JQT.goBack('#s1');
// genau eine Seite zurückgehen
JQT.goBack();
```

Sie sehen, dass die Methode `goBack()` von `jQTouch` der `history.back()`-Methode von JavaScript entspricht. Nur dass `goBack()` mit der Seitenmetapher von `jQTouch` zu arbeiten in der Lage ist:

```
// Springt zur Seite #s6 mit dem Übergang 'cube':
JQT.goTo('#s6', 'cube');
```

Mithilfe von `goTo()` kann `jQTouch` veranlasst werden, zu einer beliebigen Seite zu springen und dafür eine Animation als Seitenübergang zu verwenden, die ebenfalls als Argument übergeben wird. Last, but not least ist es möglich, eigene Animationen zu schreiben und zu deklarieren. Die Deklaration geschieht über ein Objekt mit der Methode `addAnimation()`:

```
JQT.addAnimation({ name: 'slideDown',
                  selector: '.slideDown' });
```

Die Animationen werden jedoch nicht über jQuery definiert, sondern greifen ausschließlich auf die Webkit-CSS-Transitions zurück (in anderen Browsern werden sie daher nicht funktionieren). Dies hat den Vorteil, dass sie einfach zu definieren sind, nämlich innerhalb des Stylesheets:

Hierfür wird zuerst die Animation selbst definiert und benannt (dieses Beispiel ist relativ fiktiv):

```
@-webkit-keyframes slideDown {
  from {
    top: 0px;
  }
  to {
```

```

        bottom: 0px;
    }
}

```

Anschließend wird mit sogenannten *Keyframes* der Anfangs- und Endpunkt einer Animation bestimmt (beachten Sie den Bezug auf den Namen):

```

.slideDown {
  -webkit-animation-name: slideDown;
  -webkit-animation-duration: 0.5s;
  -webkit-animation-iteration-count: 10;
  -webkit-animation-direction: normal;
}

```

Anmerkung

Die -webkit-Eigenschaften gehören zu den »inoffiziellen« CSS-Eigenschaften, die bislang nur von Safari und iPhone unterstützt werden. Bei der aktuell avisierten Zielplattform ist dies kein Problem. Dennoch werden CSS-Animation und CSS-Transitions über kurz oder lang auch ein Thema auf anderen Browsern sein (so ist zu hoffen).

Hier finden Sie die offiziellen Working-Drafts zum Thema:

- ▶ www.w3.org/TR/css3-transitions/
- ▶ www.w3.org/TR/css3-animations/

5.11.6 Zusammenfassung

Hiermit sind wir am Ende des Praxisteils zu jQuery und speziell des Themas »mobiles Webdevelopment« angelangt, soweit Letzteres im Rahmen der Besprechung von jQueryTouch und dieses Buches sinnvoll zu erörtern war. Natürlich gibt auch dieses Thema weit mehr her – wir hoffen, dass wir Sie zumindest neugierig gemacht und angeregt haben, selbstständig und gut gerüstet auf Entdeckung zu gehen.

Sie finden im Anschluss noch ein Kapitel über »Unit Testing« in Zusammenhang mit JavaScript-Entwicklung. Dies mag Ihnen vielleicht ein wenig »randständig« vorkommen, aber seien Sie versichert, es handelt sich um eine nützliche Sache – so wie wir grundsätzlich hoffen, dass dieses Buch Ihnen nützlich ist und sein wird.

»QUnit is a powerful, easy-to-use, JavaScript test suite. It's used by the jQuery project to test its code and plugins but is capable of testing any generic JavaScript code.« – QUnit Dokumentation

6 Unit Tests

Wenn Sie größere JavaScript-Projekte bearbeiten, kommen Sie an einen Punkt, an dem Sie sich Testroutinen wünschen. Die beliebte Ausgabe mit `alert()`, um Variablenwerte auszulesen, hilft ein wenig, kostet aber Zeit und ist einfach mühselig und obendrein nicht besonders zuverlässig.

Für andere Programmiersprachen wurden zuerst sogenannte *Unit Tests* entwickelt, zuerst für Smalltalk, dann für Java. Eines der bekanntesten Testframeworks für Java ist *JUnit*. Die Idee dahinter war es, einem Konzept, das in Zusammenhang mit *extreme programming* entstanden ist, zuerst Testroutinen zu schreiben und im Anschluss daran den eigentlichen Code, um alle erdenklichen Testfälle im Vorfeld der eigentlichen Entwicklung zu berücksichtigen.

Wenn neue Erweiterungen für eine solche Software geschrieben werden, kann davon ausgegangen werden, dass der vorhergehende Code fehlerarm ist. Treten nach der Entwicklung der Erweiterungen dann Fehler auf, kann davon ausgegangen werden, dass diese Fehler in dem neu hinzugefügten Code zu suchen sind. Vielmehr, der Entwickler wird zuerst neue Testroutinen schreiben, bevor er an die eigentliche Erweiterung geht (»test driven development«).

Dieses Prinzip der »testgetriebenen« Softwareentwicklung hat man auch auf die Sprache JavaScript übertragen und *QUnit* entworfen. QUnit besteht also aus Testroutinen, mit denen man Codes, Codeabschnitte bzw. einzelne Klassen oder Module testen kann. Ohne dass das gesamte Programm getestet wird, können Sie Tests für einen ausgewählten Codeabschnitt, ein Modul oder eine Klasse formulieren, Bedingungen entwerfen und den Code daraufhin testen.

Wer steht hinter QUnit?

Entwickelt und betreut wird das Framework von John Resig und Jörn Zaefferer. Sie können sich die Dateien unter folgenden beiden Links herunterladen:

- ▶ <http://github.com/jquery/qunit/raw/master/qunit/qunit.js>
- ▶ <http://github.com/jquery/qunit/raw/master/qunit/qunit.css>

Sie werden in diesem Kapitel in das *QUnit Framework* hineinschnuppern und erste Gehversuche machen können. Sie werden jedoch (dies verbietet die Beschränkung durch den Umfang dieses Buches) nicht eine komplette Einführung in das Thema bekommen.

6.1 Ein Blick auf QUnit

QUnit ist jedoch zu wichtig, um es in diesem Buch zu übergehen. Sobald Sie Plugins für jQuery herunterladen, werden Sie feststellen, dass viele Entwickler, die ernst zu nehmende Plugins entwerfen, mit QUnit arbeiten. Wollen Sie komplexe Softwareerweiterungen für jQuery schreiben, werden Sie um dieses Thema nicht herumkommen.



Abbildung 6.1 QUnit – Browserausgabe

Zunächst soll hier ein erstes einfaches Exempel aufgestellt werden. Dazu wird eine Test-HTML-Seite geschrieben:

```

<!DOCTYPE html>
<html>
<head>
<title>jQuery | Unit Testing</title>
<link type="text/css" href="css/qunit.css"
      rel="stylesheet" />
<script type="text/javascript"
      src="js/jquery-1.4.2.js"></script>
<script type="text/javascript" src="js/qunit.js"></script>
<script type="text/javascript">
  test("Das ist mein erster Test", function () {
    ok(true, "Dieser Test war erfolgreich");
  });
  QUnit.log = function(result, message) {
    $("#log").append(result + " | " + message + "<br />");
  }
</script>
</head>
<body>
  <div id="container">
    <h1 id="qunit-header">QUnit Beispiele</h1>
    <ol id="qunit-tests"></ol>
  </div>
  <div id="log"></div>
</body>
</html>

```

Listing 6.1 HTML-Gerüst – QUnit

Das jQuery Framework müssen Sie nicht zwingend einbinden, QUnit funktioniert auch ohne jQuery. Für dieses und die folgenden Beispiele sollten Sie jQuery aber einbinden. Im `<body>` benötigen Sie auf jeden Fall das `<ol>`-Element (auf die Überschrift können Sie verzichten).

6.2 Assertions

Es existieren im QUnit Framework drei verschiedene Tests für *Assertions*, also »Bedingungen« oder »Annahmen«, die erfüllt werden wollen. Die Tests machen verschiedene »Annahmen«. Sind alle Teste durchlaufen, wird in Klammern die Auswertung angezeigt: Die erste Ziffer gibt an, wie viele Tests fehlgeschlagen sind, die zweite, wie viele erfolgreich waren. Die dritte zeigt die Gesamtzahl der durchlaufenen Tests an.

6.2.1 Test mit .ok()

In diesem Beispiel wird genau ein Test mit der Methode `.ok()` durchlaufen, die die einfachste *Assertion* darstellt und die einfach den Booleschen Wert `true` erwartet. Der eigentliche Test wird in der Zeile `ok(true, "...");` durchgeführt. Das Ergebnis wird folgendermaßen aussehen:

Das ist mein erster Test (0, 1, 1)

Der Test würde in den Fällen `"irgendein String"`, `true` oder `"3956"` den Test als »erfolgreich« abschließen. In den Fällen `false`, `""`, `undefined` oder `0` würde der Test als fehlgeschlagen beendet.

6.2.2 Test mit .equals()

Neben `.ok()` gibt es noch `.equals()` und `.same()`. Die beiden letztgenannten *Assertions* vergleichen zwei Werte miteinander, einmal den Ist-Wert (»actual«) und den erwarteten Wert (»expected«).

Ein Test für `.equals()` würde folgendermaßen aussehen:

```
test("Das ist mein zweiter Test", function () {
  equals(true,true, "Dieser Test war erfolgreich");
  equals(true,false, "Dieser Test ist fehlgeschlagen");
});
```

Die erste Assertion ist erfolgreich, die zweite ist fehlgeschlagen. Mit `.equals()` können Sie also zwei Werte miteinander vergleichen.

Im Folgenden wird das Beispiel etwas ausgebaut. Mit dem `jQuery`-Objekt, das alle `<div>`-Elemente des `HTML`-Dokuments speichert, wird mit `.each()` über jedes Element iteriert. Sie können voraussagen, dass es sich um zwei Tests handeln muss, die durchlaufen werden:

```
...
var obj = $("div");
test("Das ist ein Test mit equals", function () {
  expect(2);
  obj.each(function() {
    equals($(this).length, true, "Test war erfolgreich");
  });
});
...
```

Das Ergebnis wird folgendermaßen ausfallen:

Das ist ein Test mit equals (0, 2, 2)

6.2.3 Test mit `.same()`

Jedes Element, das mit `.each()` durchlaufen wird, hat natürlich eine Eigenschaft `length`, eben die Zahl 1. Sie werden einwenden, dass die Zahl 1 ja nicht dem erwarteten Wert `true` entspräche. Nun, die Methode `.equals()` geht nicht »strikt« mit dem Überprüfen der Datentypen um. Sie vergleicht Boolean, Numbers und Strings gleichermaßen. Das gleiche Ergebnis erhalten Sie, wenn Sie `(obj.length == true)` ausgeben lassen.

Was dahinter steckt: JavaScript geht grundsätzlich sehr großzügig mit Datentypen um, wenn Sie den Vergleichsoperator `==` verwenden. Setzen Sie stattdessen `===` ein, wird JavaScript zwei Werte *strikt* vergleichen. Nur dann wird auch nach Datentyp unterschieden.

Mit QUnit erreichen Sie dies mit `same()`. Verändern Sie den Versuchsaufbau:

```
var obj = $("div");
test("Das ist ein Test mit same", function () {
  expect(2);
  obj.each(function() {
    same($(this).length, true, "Test wird fehlschlagen");
  });
});
```

So erhalten Sie für beide Tests ein »Test fehlgeschlagen«. `same()` arbeitet also viel genauer im Vergleichen mit Werten, 1 entspricht hier eben nicht `true`.

6.2.4 Module

Sie können zusätzlich noch »Module« definieren. Module sind hilfreich, um eben genau den Zweck zu erfüllen, einzelne Units zu testen. Zusätzlich können Sie einer Variablen Testwerte übergeben, ohne die Variablen für nachfolgende Funktionen zu verändern. Einem Modul muss das Argument `name` übergeben werden. Des Weiteren kann ein »Lifecycle« definiert, es können also mit den Optionen `setup` oder `teardown` Funktionen übergeben werden. Hierbei erwartet `setup`, wie der Name bereits sagt, einen Wert, der am Anfang des Tests verarbeitet wird. Meist ist dies eine anonyme Funktion. Die Anweisung `teardown` wird ausgeführt, bevor die Tests beendet werden (für das folgende Beispiel wird der Klarheit halber auf `teardown` verzichtet). Im Folgenden das gesamte Script:

```
$(document).ready(function () {
  var obj = $("div");
  module("objectLength", {
    setup: function () {
      this.obj = obj.filter("#container");
    }
  });
```

```

    }
  });
  test("Test mit setup", function () {
    expect(1);
    this.obj.each(function() {
      equals($(this).length,
        true,
        "Element: "+$(this).get(0).tagName);
    });
  });
});
QUnit.log = function(result, message) {
  $("#log").append(result + " | " + message + "<br />");
}

```

Listing 6.2 JavaScript-Code – QUnit

Das Ergebnis wird im Div mit dem ID `#log` folgendermaßen ausgegeben:

```
true | Element: DIV, expected: true result: 1
```

Mit der Option `setup` wurde die Eigenschaft `this.obj` initialisiert. Die Erwartung ist, dass ein einziger Testlauf durchgeführt wird, `expect()` wird deshalb auf 1 gesetzt.

Noch ein kleiner Hinweis: Wenn Sie viele Tests aufgebaut haben, können Sie über die URL einen Parameter mit dem Modulnamen übergeben. Dann wird ausschließlich dieses Modul ausgeführt. In diesem Beispiel wäre das `file:///[/pfad] / qunit.html?objectLength`.

6.3 Alle QUnit-Methoden im Überblick

6.3.1 Setup

Methode	Erläuterung
<code>test(name, erwartet, test)</code>	Test hinzufügen
<code>asyncTest(name, erwartet, test)</code>	Asynchronen Test hinzufügen
<code>expect(Menge)</code>	Wie viele Assertions werden innerhalb eines Tests erwartet?
<code>module(name, lifecycle)</code>	Teilt Tests in Module auf.

Tabelle 6.1 QUnit-Übersicht (Setup)

Methode	Erläuterung
<code>init()</code>	Initialisiert den Test. Reinitialisiert den Test, wenn er bereits läuft. Wird in der Praxis nicht benötigt.

Tabelle 6.1 QUnit-Übersicht (Setup) (Forts.)

6.3.2 Assertions

Methode	Erläuterung
<code>ok(status, message)</code>	Wenn das erste Argument <code>true</code> ist, ist der Test verfolgreich verlaufen. Entspricht <code>assertTrue</code> des JUnit Testframeworks für Java.
<code>equals(ist, erwartet, message)</code>	Eine Vergleichsassertion, entspricht <code>assertEquals</code> von JUnit.
<code>same(ist, erwartet, message)</code>	Eine Vergleichsassertion, die primitive Datentypen, Arrays und Objekte vergleicht.

Tabelle 6.2 QUnit-Übersicht (Assertions)

6.3.3 Asynchrone Tests

Methode	Erläuterung
<code>start()</code>	Startet Testläufe, wenn sie vorher gestoppt wurden.
<code>stop(timeout)</code>	Stoppt Testläufe und wartet, bis sie mit <code>start()</code> aufgerufen werden.

Tabelle 6.3 QUnit-Übersicht (asynchrone Tests)

Weiterführende Informationen finden Sie unter:

<http://docs.jquery.com/QUnit>

Dort auch ein Link zu umfassenden Unit-Tests, beispielsweise jener zu jQuery UI Akkordion, der auch Benutzereingaben simuliert:

<http://jquery-ui.googlecode.com/svn/trunk/tests/unit/accordion/>

Anhang

A	HTML und CSS	495
B	JavaScript und DOM	535
C	Inhalt der DVD-ROM	569

»HTML has the potential interest of millions of people: anyone who has designed a web page may have useful views on new HTML features. It is the earliest spec of W3C, a battleground of the browser wars, and now the most widespread spec.« – Tim Berners-Lee in seinem Webblog

A HTML und CSS

Vielleicht haben Sie dieses Buch ja zur Hand genommen, ohne bereits jahrelange Erfahrung im Webdesign zu haben und möchten mit seiner Hilfe zügig zum Entwerfen interaktiver Weboberflächen gelangen. Gerne – genau hierzu wollen wir Ihnen mit diesem Buch verhelfen. Auf den folgenden Seiten geben wir Ihnen noch etwas zusätzliches Rüstzeug auf den Weg – lesen Sie sie daher sorgfältig.

Möglicherweise sind Sie aber auch vertraut mit HTML und CSS und arbeiten bereits jahrelang mit JavaScript. Das Thema »jQuery« stellt für Sie eher Aufbauwissen dar, mit dem Sie Ihr Portfolio erweitern möchten. Anhang A, »HTML und CSS«, sowie Anhang B, »JavaScript«, können Sie als Profi getrost überschlagen. Es sei denn, Sie haben Spaß an einer lockeren kleinen Wiederholung ...

A.1 Trennungen – Struktur, Präsentation, Verhalten

Teile und herrsche – diese Devise gilt auch im Webdesign. Anders gesagt, beim Erstellen einer Webseite genügt uns eine einzige Formulierungsmöglichkeit nicht, um alle anfallenden Aufgaben abdecken zu können. Vielmehr greifen wir für unterschiedliche Teilbereiche auf entsprechend spezialisierte Sprachen zurück, die uns das jeweils geeignete Werkzeug an die Hand geben.

Lassen Sie uns diese Teilbereiche kurz umreißen. Es sind dies (meist auch in dieser Reihenfolge genannt) im Einzelnen:

Beschreibung der Struktur

Die Beschreibung der *Struktur* wird durch eine Sprache übernommen, die hierarchisch geordnete Informationen möglichst semantisch (ihrer Bedeutung entsprechend) darzustellen in der Lage ist.

Im Webdesign geschieht dies durch *HTML* oder *XHTML*. Im Volksmund werden HTML und XHTML oft als »Programmiersprachen« bezeichnet, was allerdings nicht zutrifft. Bei beiden handelt es sich vielmehr um Auszeichnungs- oder *Markup-Sprachen*.

Beschreibung der Präsentation

Die in HTML beschriebene Struktur muss optisch präsentiert werden, wobei die Darstellung gemäß des User Agents (Browser, Mobilgerät, Drucker) *unterschiedlich* erfolgen muss (eben darum darf die Definition der Darstellung nicht Teil der Struktur sein). Für dieses Gebiet ist CSS die Sprache der Wahl.

Auch CSS ist keine Programmiersprache. Hier wird lediglich beschreibend festgelegt, welche Elemente auf welche Weise dargestellt werden und in welchem Kontext welche der definierten Darstellungsanweisungen einzusetzen ist. Man bezeichnet CSS daher auch als *Präsentationssprache*. Da kein Prozess, sondern ein Ergebnis beschrieben wird, nennt man dies auch *deklarative Sprache*.

Beschreibung des Verhaltens

Interessant wird es, sobald eine Interaktion mit dem dargestellten Dokument möglich wird. Hier kommt *JavaScript* ins Spiel, mit dessen Hilfe Useraktionen erkannt und auf sie reagiert werden kann.

Bei JavaScript handelt es sich tatsächlich um eine ausgewachsene *Programmiersprache*. Mit seiner Hilfe kann man die eigentlich statischen Struktur- und Präsentationsaspekte dynamisch machen, also (auch nachträglich) sowohl das Aussehen als auch die Inhalte eines Dokuments verändern. Ein JavaScript Framework wie jQuery hilft Ihnen bei den dabei anfallenden, komplexen Programmieraufgaben.

Das Schichtenmodell

Da diese drei Aspekte aufeinander aufbauen (mit der Struktur als Basis), kann man sie auch als Schichtenmodell betrachten (siehe Abbildung A.1). Hier wird deutlich, dass die HTML-Struktur die Basis bildet, auf der die darüberliegenden Schichten aufsetzen.

Tatsächlich liest der Browser auch zunächst diese Struktur ein, die er anschließend gemäß den CSS-Anweisungen »dekoriert« und schließlich mit dem in JavaScript beschriebenen Verhalten ausstattet. Setzen wir uns kurz mit den Grundlagen von HTML auseinander.

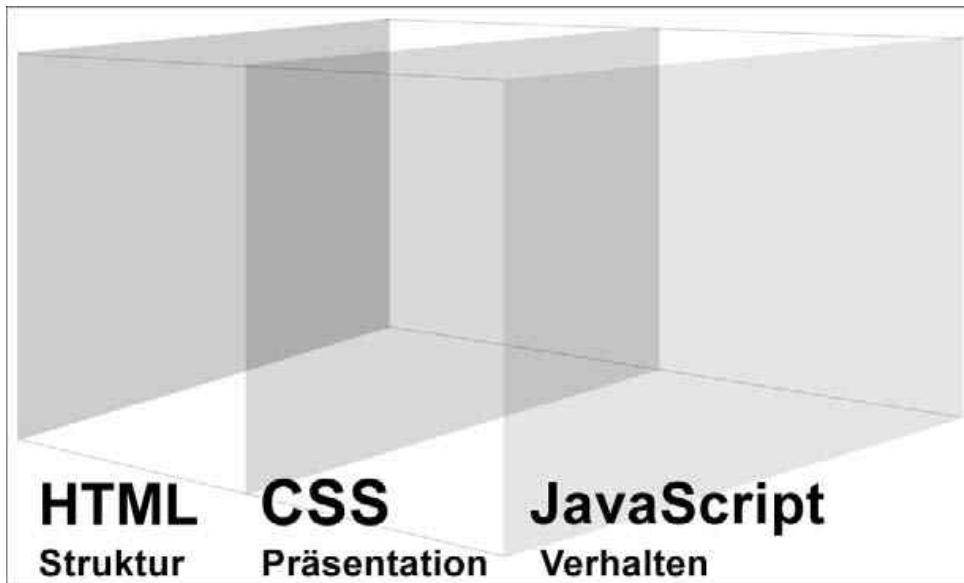


Abbildung A.1 Die drei Ebenen – Struktur, Präsentation und Verhalten

A.2 (X)HTML – Beschreibung der Struktur

Was auch immer Sie in Bezug auf Webdesign oder -programmierung unternehmen – HTML (heutzutage meist als XHTML) wird stets dabei sein.

A.2.1 Was ist eine Markup-Sprache?

Während eine Programmiersprache (wie JavaScript), einfach ausgedrückt, Abläufe und Vorgänge beschreibt, dient eine Auszeichnungssprache dazu, die *Struktur* von in Textform vorliegender Information in einer semantischen Form zu formulieren.

Hierfür wird ein Informationsblock in Marken eingeschlossen, die dessen Aufgabe oder Bedeutung benennen und gleichzeitig seinen Anfang und sein Ende bestimmen. Diese Marken werden als *Tags* bezeichnet; man unterscheidet einen Start- und einen End-Tag. Die gesamte Struktur aus Inhalt sowie Start- und Endmarke wird als *Element* bezeichnet:

```
<name>Eine Information</name>
```

A.2.2 Grammatik und Dokumenttyp

Die *Bezeichner* der Elemente einer Markup-Sprache, sprich die Tag-Namen, ergeben sich aus der zu beschreibenden Informationsstruktur. Ihre möglichen Ausprägungen bestimmen damit den Sprachumfang der Markup-Sprache, der im Allgemeinen in einer *Grammatik* festgelegt wird.

Hiervon existieren im Falle von HTML und XHTML (historisch bedingt) mehrere gleichzeitig gebräuchliche Varianten (u.a. *HTML 4 transitional*, *strict* und *frameset*, *XHTML 1.0 transitional*, *strict* und *frameset*).

Eine an die aktuellen Anforderungen angepasste Grammatik ist in Gestalt von *HTML 5.0* in Vorbereitung. Der Sprachumfang von HTML5 ist noch um einiges größer als der von HTML 4, um mehr Möglichkeiten der semantischen Abbildung einer Informationsstruktur zu bieten.

A.2.3 HTML vs. XHTML

HTML und XHTML haben *dieselbe Aufgabe* und gleichen sich daher in vielen Belangen. Hierzu gehören die Anzahl und die Bezeichnung der Elemente und Attribute und deren Verwendung. Da beide Sprachen *unterschiedlicher Herkunft* sind (HTML basiert auf SGML, XHTML hingegen auf XML), gibt es prinzipbedingt zwischen beiden Sprachen einige Unterschiede. Diese sind in der folgenden Tabelle kurz zusammengefasst.

	HTML	XHTML
Wurzelement	Optional	Obligatorisch
Namensraum	Namensraumkonzept existiert nicht.	XHTML-Namensraum erforderlich
Doctype-Deklaration	Optional	Erforderlich
Elementverkürzungen	Gestattet	Nicht gestattet
Attributwerte	Anführungszeichen um Zahlenwerte optional	Anführungszeichen obligatorisch
Marker-Attribute	Gestattet	Nicht gestattet
Leere Elemente	Werden nicht geschlossen	Müssen geschlossen werden
Bezeichner	Schreibweise nicht reglementiert	Kleinschreibung vorgeschrieben

Tabelle A.1 Zusammenfassung der Unterschiede zwischen HTML und XHTML

Welche HTML-Variante in Zusammenhang mit jQuery?

Ohne vorgreifen zu wollen, soll hier kurz auf die Frage eingegangen werden, ob die eine oder die andere HTML-Variante für die Arbeit mit jQuery geeigneter ist. Wir haben uns im Rahmen dieses Buches für XHTML entschieden. Für jQuery ist es jedoch unerheblich, welchen HTML-Dialekt Sie bevorzugen. Auch mit HTML5 arbeitet jQuery problemlos zusammen.

Die Regeln dieser Grammatiken liegen als *Doctype Definition* (DTD) vor, praktisch der Duden des Webdesigners (den Sie allerdings nur selten konsultieren müssen). Welche Grammatik für ein Dokument speziell gelten soll, kann an dessen Anfang bekannt gegeben werden – dies geschieht im Rahmen einer *Doctype-Deklaration*.

Gemäß dieser Grammatik kann das Dokument anschließend validiert (auf Konformität geprüft) werden. Eine solche Doctype-Deklaration braucht für HTML-Dokumente nicht unbedingt zu erfolgen; für XHTML-Dokumente ist sie allerdings vorgeschrieben. Hier ein Beispiel für *XHTML 1.0 transitional*:

```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Wenn Ihnen das übermäßig komplex vorkommt, lassen Sie sich dadurch trösten, dass die meisten aktuellen Editoren Ihnen die Arbeit, diesen Prolog zu schreiben, abnehmen werden. Ansonsten verwenden Sie HTML5, wo die Doctype-Deklaration stark vereinfacht einfach `<!DOCTYPE html>` lautet.

Spezifikationen zu HTML und XHTML

Die Spezifikationen zu HTML, XHTML und auch zu deren designiertem Nachfolger HTML5 finden Sie beim W3C:

- ▶ **HTML 4.01:** www.w3.org/TR/html401/
- ▶ **XHTML 1.0:** www.w3.org/TR/xhtml1/
- ▶ **HTML 5.0:** www.w3.org/TR/html5/

A.3 Aufbau von HTML- und XHTML-Dokumenten

Der Grundaufbau von HTML- und XHTML-Dokumenten ist, abgesehen von sprachspezifischen Details, identisch. Im *Wurzelement* `<html>` befinden sich die beiden funktionalen Dokumentbereiche `<head>` als *Dokumentkopf* und `<body>` als *Dokumenttrumpf*. Ersterer stellt einen Container für den Dokumenttitel `<title>` und optional weitere dokumentbezogene Informationen dar, Letzterer enthält alle sichtbaren Bestandteile des Dokuments.

Für die Grundstruktur des Dokuments werden also genau *vier Elemente* benötigt. Man bezeichnet sie als *Strukturelemente*. Wie die beiden folgenden Beispiele verdeutlichen, sind die Unterschiede zwischen HTML 4, XHTML und HTML5 auf dieser Ebene nur gering:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

```

<html lang="de">
<head>
  <meta http-equiv="Content-Type"
    content="text/html; charset=UTF-8">
  <title>Beispiel HTML 4.01 strict</title>
</head>
<body>

</body>
</html>

```

Listing A.1 Beispiel HTML 4.01 strict

Beachten Sie, dass die Doctype-Deklaration in HTML 4 nicht Pflicht ist. Der HTML-Tag verfügt über ein `lang`-Attribut. In XHTML ist Doctype hingegen Pflicht, der HTML-Tag besitzt einen Namensraum und verfügt hier über ein `xml:lang`-Attribut:

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="de">
<head>
  <meta http-equiv="Content-Type"
    content="text/html; charset=UTF-8" />
  <title>Beispiel XHTML 1.0 strict</title>
</head>
<body>

</body>
</html>

```

Listing A.2 Beispiel XHTML 1.0 strict

Das neue HTML5 ist demgegenüber stark vereinfacht. Der Doctype ist der denkbar simpelste, eine Namensraumsdeklaration ist nicht erforderlich (sie ist implizit), und wir haben wieder ein `lang`-Attribut. Prägen Sie sich diesen Dokumentaufbau gut ein, denn hierin liegt die Zukunft:

```

<!DOCTYPE HTML>
<html lang="de-DE">
<head>
  <meta charset="UTF-8">
  <title>Beispiels HTML 5</title>
</head>
<body>

```

```
</body>
</html>
```

Listing A.3 Beispiel HTML5

A.3.1 Aufgaben des Dokumentkopfs

Im Dokumentkopf `<head>` existieren neben dem obligatorischen `<title>`-Element noch weitere optionale Elemente. Ihre Anzahl und Reihenfolge sind nicht reglementiert. Die gebräuchlichsten hiervon sind:

- ▶ Metainformationen (als `<meta>`-Tags)
- ▶ Styleinformationen (über den `<link>`-Tag oder im `<style>`-Container)
- ▶ Scriptinformationen (im `<script>`-Container)

Der Scriptcontainer `<script>` in HTML und XHTML

Im Rahmen dieses Buches ist der `<script>`-Tag von besonderem Interesse, weshalb wir ihn etwas näher beleuchten.

Das `<script>`-Element wird zum direkten Einbetten von *Scriptanweisungen* verwendet. Es darf sowohl im `<head>` als auch im `<body>` des Dokuments in beliebiger Anzahl auftreten, wobei die Blöcke in der Reihenfolge abgearbeitet werden, in der sie im Dokument erscheinen (Scripts im Head also zuerst). Obligatorisch ist die Angabe des MIME-Typs mittels des `type`-Attributs. In der Regel lautet er `"text/javascript"` für JavaScript. Der Programmcode kann sich direkt im Element befinden, aber auch von außen »hereingeholt« werden.

Soll eine außerhalb des HTML-Dokuments liegende JavaScript-Datei ausgeführt werden, wird sie über das `src`-Attribut adressiert. Die externen Scriptdaten verhalten sich, als ob sie sich direkt im Container befinden würden. Das `<script>`-Element bleibt in diesem Falle leer, muss aber dennoch mit einer Endmarke geschlossen werden. Das Pflichtattribut `type` nennt auch hier den MIME-Typ der verwendeten Scriptsprache:

```
<script type="text/javascript" src="extern.js"></script>
```

Diese Schreibweise ist in HTML und XHTML identisch. Problematischer ist das direkte Einfügen von Scripten, da das Inhaltsmodell von `<script>` laut Grammatik in HTML (CDATA-Inhalt) und XHTML (PCDATA-Inhalt) unterschiedlich definiert ist.¹ Um das Parserverhalten zu vereinheitlichen, schließt man, um sicherzugehen, in XHTML-Dokumenten den Containerinhalt in einen CDATA-Bereich ein.

¹ Vereinfacht gesagt, werden in XHTML die Inhalte überprüft, ob sie Steuerzeichen enthalten, in HTML nicht. Dies kann beim Validieren der Dokumente einen Unterschied machen.

Einbindung in HTML	Einbindung in XHTML
<pre><script type="text/javascript"> <!-- alert("Hallo Welt!"); //--> </script></pre>	<pre><script type="text/javascript"> /*<![CDATA[*] alert("Hallo Welt!"); /*]]>*/ </script></pre>

Tabelle A.2 Unterschiede in der Einbindung von Scriptdaten bei HTML und XHTML

Was ist besser – lokale oder externe Scripte?

Diese Unterschiede im Inhaltsmodell und entsprechende Kompatibilitätsprobleme legen es nahe, in XHTML die Einbindung externer Scriptdateien vorzuziehen. Wirklich relevant ist der Unterschied nur, falls das Dokument validiert wird. Im Rahmen dieses Buches werden wir, um die Beispieldateien kompakt zu halten, in vielen Fällen lokal eingebettete Scripte einsetzen.

A.3.2 Der Dokumentrumpf – strukturierte Information

Die Mehrzahl der Elemente, die in HTML und XHTML definiert sind, wird im Dokumentrumpf, also dem im Browserfenster sichtbaren Teil des Dokuments, eingesetzt. An dieser Stelle erübrigt sich eine detaillierte Übersicht – höchstens ein Blick auf das allgemeine Prinzip kann im Rahmen dieses Buches erfolgen.²

Eine Webseite untergliedert man auch in bestimmte funktionale Bereiche, die wir gleich noch betrachten werden. Befassen wir uns zunächst mit der darzustellenden Information selbst, also dem eigentlichen Inhalt unseres Dokuments.

Semantische Auszeichnung

Was eine Information in einem Zusammenhang bedeutet, ist für einen menschlichen Betrachter, der zur Interpretation fähig ist, einfach ersichtlich. Soll ein Dokument von einem Computerprogramm verarbeitet werden, ist es hingegen erforderlich, diese Bedeutung durch eine entsprechende Kennzeichnung zu verdeutlichen. So auch in HTML.

Ein Textabsatz wird beispielsweise in `<p>`-Tags eingeschlossen, was dem Browser verdeutlicht, dass es sich um einen `paragraph` handelt. Die *Rolle* des Informationsabschnitts ist somit klar:

```
<p>Dies ist ein Textabsatz.</p>
```

² Wir empfehlen als Vertiefung das Referenzwerk »XHTML, HTML und CSS« des Autors Frank Bongers, das ebenfalls bei Galileo Press erschienen ist.

Sinnvoll ist es, den Tag entsprechend der Rolle zu wählen, sofern ein passendes Element zur Verfügung steht – wie in folgenden Beispielen:

```
<address>Donald-Duck-Allee 1, 12345 Entenhausen</address>
<blockquote>Ein Blockzitat</blockquote>
```

Damit die Rolle einer Information stets eindeutig bestimmt ist, sind Grundregeln bei der Verschachtelung von Elementen zu beachten. Die Bereiche von zwei Elementen dürfen sich nicht teilweise überlagern (im fett markierten Bereich des folgenden Beispiels ist die Rolle des Inhalts deshalb undefiniert):

```
<gross> ... <klein> ... </gross> ... </klein>
```

Allerdings ist es gestattet, dass ein Element in einem anderen vollständig enthalten ist: `<gross> ... <klein> ... </klein> ... </gross>`

Wollen wir an einen Abschnitt eines Dokuments und damit an eine bestimmte Information gelangen (hierbei wird uns jQuery später helfen), dann ist zumindest klar, dass der Bereich der gesuchten Information eindeutig begrenzt ist.

Block- und Inlineelemente

Je nachdem, ob ein blockartiger, zusammenhängender Abschnitt oder eine Teilinformation in einem größeren Zusammenhang zu kennzeichnen ist, hat man in HTML verschiedene Elementtypen zur Verfügung. Für die »grobe« Struktur sind die *Blockelemente* zuständig, mit denen man Überschriften, Textabsätze, aber auch komplex aufgebaute Strukturen wie Tabellen und Formulare gestaltet. In ihrer Grundpräsentation bilden diese logischen Blöcke ebenfalls in sich einen abgeschlossenen Dokumentabschnitt – wir werden darauf gleich wieder bei der Kurzabhandlung über CSS treffen.

Die Auszeichnungsmöglichkeiten wären zu gering, hätte man nicht im Inneren der Blöcke weitere Elemente zur Verfügung, die zur Darstellung einer Feinstruktur in der Lage sind. Da ihre Aufgabe darin besteht, innerhalb von Zeilenstrukturen einzelne Worte oder Passagen zu markieren, ohne dabei aber den logischen Block, in dem dies stattfindet, zu unterbrechen, nennt man sie *Inlineelemente*.

Man unterscheidet hierbei *logische* und *physische* Auszeichnungen, die Rolle oder Bedeutung ihrer Inhalte kennzeichnen, ohne damit notwendigerweise eine Präsentation festzulegen. Des Weiteren existieren auch Inlineelemente mit rein präsentativer Funktion, von deren Verwendung man jedoch nach Möglichkeit zugunsten von CSS absehen sollte.

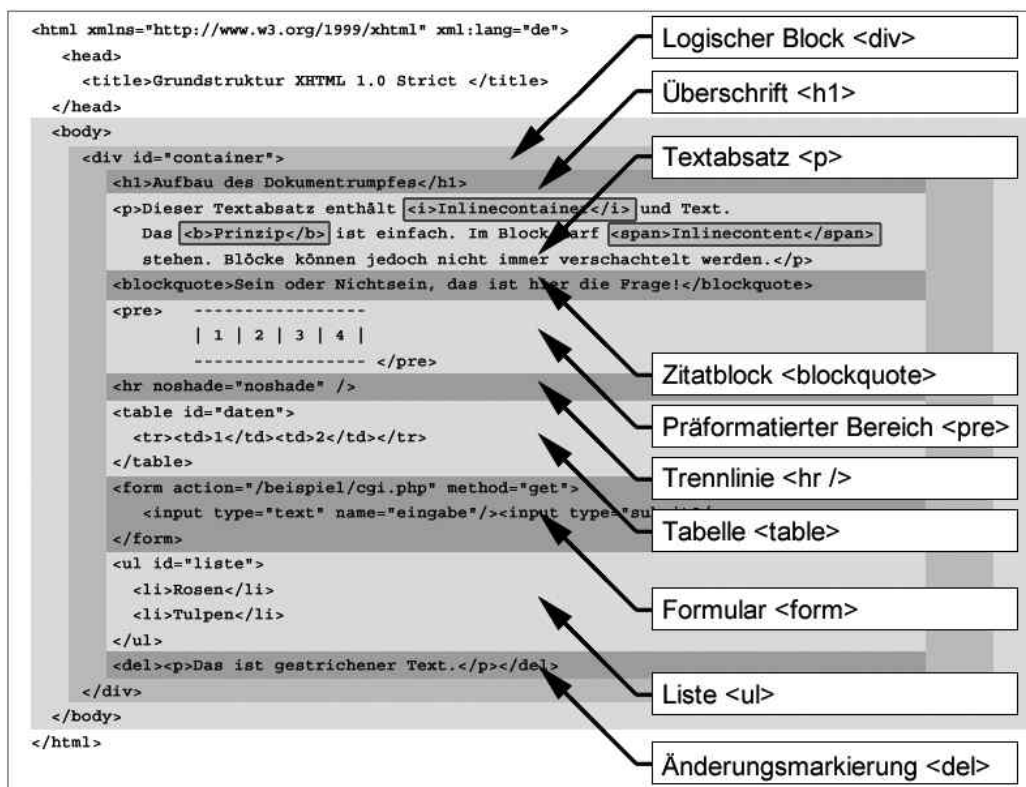


Abbildung A.2 Überblick über den Vorrat an semantischen Blockelementen

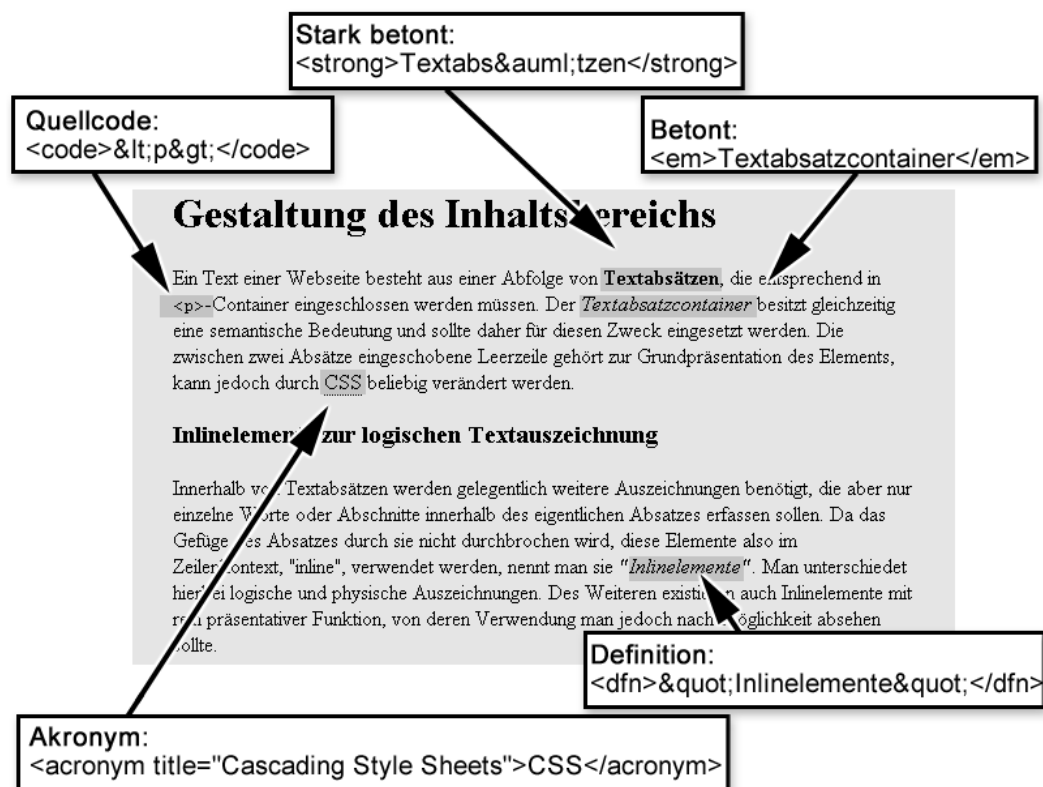


Abbildung A.3 Verschiedene logische Textauszeichnungen

In der folgenden Tabelle sind die wichtigsten Inlinenelemente aufgelistet.

Element	Rolle
<code><cite> ... </cite></code>	Bezeichnet die Quelle (Herkunft) eines Zitats.
<code><code> ... </code></code>	Bezeichnet Quellcode; gewöhnlich mit nicht proportionaler Schrift dargestellt.
<code><dfn> ... </dfn></code>	Bezeichnet eine Definition; gewöhnlich kursiv dargestellt.
<code> ... </code>	Einfache Hervorhebung; gewöhnlich kursiv dargestellt
<code><kbd> ... </kbd></code>	Bezeichnet eine Tastatureingabe.
<code><q> ... </q></code>	Bezeichnet ein Inlinезitat durch Anführungszeichen.
<code><samp> ... </samp></code>	sample; Auszug z. B. eines Programmquellcodes
<code> ... </code>	Bezeichnet starke Hervorhebung; gewöhnlich fett dargestellt.
<code><var> ... </var></code>	Bezeichnet eine Variable.
<code> ... </code>	Neutraler Container; wichtig in Zusammenhang mit Style-sheets
Attribute (alle)	class, id, style, title (Universalattribute)

Tabelle A.3 Inlinenelemente zur logischen und physischen Textauszeichnung

Die hier vorgestellten Inlinenelemente haben gemeinsam, dass sie nicht auf eine spezielle Präsentation ihrer Inhalte abzielen (obwohl diese teilweise durchaus damit verbunden ist), sondern eine Aussage über ihre Bedeutung machen. Eine Sonderstellung hat der Anker-Container, der eine Linkbeziehung zu einer Resource darstellt:

Element	Rolle
<code><a> ... </code>	anchor; bezeichnet Quelle oder Ziel eines Hyperlinks.
Attribute (Auswahl)	accesskey, href, name, tabindex, target, title

Tabelle A.4 Der `<a>`-Container zum Erzeugen von Hyperlinks

Da der Anker-Tag (neben Formularen) praktisch eine natürliche Schnittstelle zwischen Benutzer und der Webseite darstellt, steht er im Mittelpunkt der Interaktivität und damit auch des Scriptings mit jQuery.

Formulare, Listen und Tabellen

Die einfachsten Blockelemente wie Überschriften und Textabsätze können einfach nur Text enthalten. Andere Blöcke dienen dazu, Informationen hierarchisch darzustellen und dabei Zusammenhänge wie Zugehörigkeit und Reihenfolgen zu ver-

deutlichen. Es sind dies alle Arten von Listen (HTML kennt drei Typen davon) sowie die Tabellen, für die HTML eine ganze Reihe von Elementen zur Verfügung stellt.

Eine Sonderstellung besitzen die Formulare, die neben dem Anker-Tag die andere »native« Interaktionsmöglichkeit mit einem HTML-Dokument ermöglichen: Sie dienen der Eingabe von Daten durch den User, die zur Auswertung an eine andere Anwendung weitergeschickt werden. Keine Frage, dass Formulare heutzutage in vielen Belangen (E-Commerce, Communitys) im Zentrum stehen und der Nutzer entsprechenden Komfort erwartet.

Element	Rolle
<code><form></code> ... <code></form></code>	Bildet einen Container für Formularelemente; bestimmt über Attribute das Formularverhalten.
Attribute (Auswahl)	action, enctype, method
<code><input /></code>	Erzeugt verschiedene Arten von Inputelementen, deren Form über das type-Attribut bestimmt wird.
Attribute (Auswahl)	accesskey, checked, disabled, maxlength, name, size, tabindex, type, value
<code><label> ...</code> <code></label></code>	Erzeugt zu einem Formularelement über eine IDREF-ID-Beziehung eine klickbare Beschriftung.
Attribute	accesskey, for

Tabelle A.5 Grundelemente des HTML-Formulars

Suchen:

```

<form action="/cgi/form.pl" method="get">
  Suchen: <input type="text" name="suche" accesskey="s"/>
  <input type="submit" value="Suchen"/>
</form>

```

Formular

Input (Textfeld)

Input (Submitbutton)

```

<form action="/cgi/form.pl" method="get">
  <label for="suche">Suchen:</label>
  <input type="text" id="suche" name="suche" accesskey="s"/>
  <input type="submit" value="Suchen"/>
</form>

```

Label

Abbildung A.4 Einfaches Formular, unten mit hinzugefügtem `<label>`

Da wir im Praxisteil des Buchs mit Listen, Tabellen und Formularen genügend Kontakt hatten, erübrigt sich hier eine genaue Erörterung. So viel sei gesagt, dass die Möglichkeiten von HTML sich in einer rein *statischen Darstellung* erschöpfen. Erwarten wir beispielsweise eine Sortierbarkeit von Listen oder Tabellen, das Ein- oder Ausblenden von Informationen oder vielleicht auch nur etwas Dynamik im Umgang mit Formularen (heutzutage bezeichnet man dies als »Eye-Candy«), so ist nicht viel geboten.

Natürlich wird uns jQuery wieder zu Hilfe kommen – sortierbare Listen und Tabellen, dynamische Formulare? Kein Problem. Was früher tagelange Programmierung erforderte, erledigen Sie mit wenigen Zeilen Code. Versprochen!

Das Gesamtbild

Der gesamte Sprachumfang von *XHTML 1.0 transitional* (der dem von *HTML 4.01 transitional* entspricht) umfasst 89 Elemente.

<a> <abbr> <acronym> <address> <applet> <area />
 <base /> <basefont /> <bdo> <big> <blockquote> <body> <button>
<caption> <center> <cite> <code> <col> <colgroup>
<dd> <del> <dfn> <dir> <div> <dl> <dt>
 <fieldset> <form>
<h1> <h2> <h3> <h4> <h5> <h6> <head> <hr /> <html>
<i> <iframe> <input /> <ins> <isindex />
<kbd> <label> <legend> <link />
<map> <menu> <meta />
<noframes> <noscript>
<object> <optgroup> <option>
<p> <param> <pre> <q>
<s> <samp> <script> <select> <small> <strike> <style> <sub> <sup>
<table> <tbody> <td> <textarea> <tfoot> <th> <thead> <title> <tr> <tt>
<u> <var>

Tabelle A.6 Übersicht – Elemente von HTML 4.01 und XHTML 1.0 transitional

Eigene semantische Auszeichnung mit IDs und Klassen

Nun kann es sein, dass der HTML-Wortschatz, so groß er auch sein mag (immerhin sind es fast hundert verschiedene Elemente), für unsere Zwecke nicht aus-

reicht. Soll z. B. eine Bestellnummer als solche gekennzeichnet werden, wäre ein Tag `<bestellnummer>` nützlich. Woher nehmen?

An dieser Stelle kommen die neutralen Elemente `<div>` und `<span>` ins Spiel, die für sich betrachtet zunächst »nichts tun«. Hierbei ist `<div>` der neutrale Blockcontainer und `<span>` das neutrale Inlineelement. Zu Hilfe kommt uns der Umstand, dass (beinahe) jedes HTML-Element einen Identifizierer oder eine Klasse zugewiesen bekommen kann – in Form der `id`- und `class`-Attribute.

Wählen wir den Bezeichner nach semantischen Gesichtspunkten, können wir auf diesem Weg mit `<div>` und `<span>` ein »fehlendes« Element emulieren:

Best.Nr.: `<span class="bestellnummer"> 2-1-55-99-X</span>`

Ein paar einfache Grundregeln sind zu beachten:

- ▶ Wählen Sie den Bezeichner nach semantischen Gesichtspunkten, nicht nach gewünschter Präsentation (`class="wichtig"` statt `class="rote_schrift"`)
- ▶ Darf die Information nur einmal im Dokument auftreten, wählen Sie das `id`-Attribut zur Kennzeichnung (funktionale Dokumentabschnitte).
- ▶ Tritt die Information mehrfach auf (»wiederholbar«), wählen Sie das `class`-Attribut (wie beispielsweise für eine Bestellnummer denkbar).
- ▶ Bildet die Information einen eigenständigen Block, wählen Sie das `<div>`-Element als Basis.
- ▶ Tritt die Information im Fließtext (also »inline«) auf, bauen Sie sie auf das `<span>`-Tag auf.

Es muss klargestellt werden, dass es sich hier um eine »persönliche« Semantik handelt und nicht um eine allgemeingültige. Es wäre nach wie vor erforderlich, einer verarbeitenden Anwendung die Bedeutung von `class="bestellnummer"` und ähnlichen Konstrukten zu vermitteln.

Deutlich sollte geworden sein, dass Informationen nicht nur mit Elementnamen, sondern alternativ oder zusätzlich auch mittels IDs und Klassen bezeichnet werden können. Anhand dieser IDs und Klassen können sie gegebenenfalls später wieder im Dokument aufgefunden werden. (Für Sie inzwischen sicher nicht unerwartet, wird uns jQuery auch hierbei nützliche Dienste leisten.)

A.3.3 Semantischer Grundaufbau eines Dokuments

Die eben vorgestellte Technik lässt sich auf die Grundstruktur des Dokumentinhalts übertragen: Die Mehrzahl der Grundlayouts von Webseiten lässt sich auf die gleichen *vier funktionalen Grundbereiche* reduzieren, die gelegentlich noch feiner unterteilt werden (beispielsweise für mehrspaltige Inhalte):

- ▶ Kopfbereich
- ▶ Hauptnavigation
- ▶ Inhaltsbereich
- ▶ Seitenfuß

Daraus ließe sich schließen, dass ein Layout mit vier Grundbereichen nicht mehr als vier, mit semantisch gewählten ID-Bezeichnern versehene `<div>`-Container benötigen wird (IDs ziehen wir Klassen vor, da es sich um nicht wiederholbare Dokumentteile handelt). Ein erster Ansatz könnte wie folgt sein (Dokumentkopf weggelassen):

```
<body>
  <div id="kopfbereich"><h3>Kopfbereich</h3></div>
  <div id="navigation"><h3>Navigation</h3></div>
  <div id="inhalt"><h3>Inhaltsbereich</h3></div>
  <div id="seitenfuss"><h3>Seitenfuß</h3></div>
</body>
```

Aus verschiedenen Gründen umgibt man die vier funktionalen Bereichs-Divs meist mit einem weiteren Container (meist als *Wrapper* bezeichnet), der für die Gesamtbreite des Layouts und die Positionierung im Viewport verantwortlich ist. Zusätzlich kann auch der `<body>`-Tag selbst einen sprechenden ID (oder eine Klasse) bekommen, umso leichter ist es, zwischen verschiedenen Layoutvarianten zu unterscheiden.³ Wir gelangen zu folgender Grundkonfiguration:

```
<body class="layoutname">
  <div id="wrapper">
    <div id="kopfbereich"><h3>Kopfbereich</h3></div>
    <div id="navigation"><h3>Navigation</h3></div>
    <div id="inhalt"><h3>Inhaltsbereich</h3></div>
    <div id="seitenfuss"><h3>Seitenfuß</h3></div>
  </div>
</body>
```

Weitere semantische Elemente in HTML5

Im Prinzip reichen neutrale Blockcontainer wie `<div>` oder Inlineelemente wie `<span>` zusammen mit einem ID oder einer Klasse aus, um die semantische Aufgabe eines Bereichs innerhalb eines Dokuments zu kennzeichnen. Selbst beschreibend wird er dadurch jedoch nicht – es bleibt dem Webdesigner überlassen, den ID-Namen zu wählen und die Rolle zuzuweisen.

³ Dies ist gängige Praxis beim »Theming« von Webseiten, wo durch Wechsel der Body-Klasse die Präsentation (bei gleichbleibender Semantik) vollständig geändert werden kann.

Bezeichnet man beispielsweise eine Randspalte mit `id="rand"`, `id="spalte_rechts"` oder mit `id="aside"`? In keinem Fall wird die Rolle für den Browser zwingend festgelegt – sie ergibt sich (und das nur für den Betrachter!) erst aus der zugewiesenen Präsentation.

Dies ist der Grund, warum in HTML5 weitere, semantisch beschreibende Blockelemente eingeführt werden, die Dokumentbereiche auf Strukturebene kennzeichnen sollen. Diese müssen zwar ebenfalls anschließend noch mit einer Präsentation belegt werden, ihre Rolle hingegen ist von vornherein klar.

HTML 5 Element	Erläuterung
<code><aside></code>	Bezeichnet nebenständige Inhalte, die dem Hauptthema erläuternd beigelegt werden.
<code><header></code>	Bezeichnet den Kopfbereich des sichtbaren Teils eines Dokuments (also nicht mit <code><head></code> zu verwechseln).
<code><footer></code>	Bezeichnet den Fußbereich eines Dokuments (oder eines Dokumentabschnitts).
<code><nav></code>	Bezeichnet eine Navigation.
<code><article></code>	Bezeichnet einen Informationsblock innerhalb des Dokuments wie einen Artikel, Blogeintrag etc.
<code><section></code>	Bezeichnet einen (in sich weiter unterteilbaren) Abschnitt eines Dokuments oder Artikels.

Tabelle A.7 Neue semantische Blockelemente in HTML5 (Auswahl)

Unterschiede zwischen HTML 4 und HTML5
Prinzipiell ist der gesamte Wortschatz von HTML 4 in HTML5 enthalten. Die neue HTML-Version stellt darüber hinaus viele weitere Elemente und Attribute zur Verfügung. Die »feinstoffliche« Unterschiede zwischen den Sprachen gehen aber über einen reinen Versionssprung hinaus und würden ein eigenes Buch zum Thema rechtfertigen. Einen guten Überblick über die Neuerungen in HTML5 finden Sie beim W3C in folgendem Dokument: HTML5 Differences from HTML4: www.w3.org/TR/html5-diff/

Wir bewegen uns, nachdem wir unsere Informationsstruktur (unterstützt durch geeignete ID- und Klassenbezeichner) zufriedenstellend durch HTML-Elemente ausgedrückt haben, noch vollständig auf der semantischen Ebene. Noch ist keine Aussage über die optische Darstellung getroffen worden. Warum uns Struktur und Semantik auch hierbei helfen, erfahren Sie im Rahmen der jetzt folgenden Grunderläuterungen zu CSS.

A.4 CSS – Beschreibung der Präsentation

Aufgrund der andersartigen Syntax stellen Stylesheets für angehende HTML-Autoren eine kleine Hürde dar, die aber nicht schwer zu nehmen ist. Wie bereits erläutert wurde, übernehmen Stylesheets die Aufgabe, die Darstellung eines Dokuments definieren.

A.4.1 Einbindung von CSS in ein HTML-Dokument

Bevor wir auf die eigentliche Syntax der CSS-Anweisungen selbst eingehen, werden wir zunächst die Einbindung der Anweisungen in das HTML-Dokument ansprechen. Als gängigste Methode wird eine CSS-Datei über einen Link aus dem Dokumentkopf mit einer HTML-Seite verknüpft.

Verlinkte, externe CSS-Datei

Ein externes Stylesheet ist, ebenso wie HTML, ein reines Textformat. Es besteht aus einer Abfolge von Styleanweisungen. Um diese auf eine Webseite anzuwenden, muss eine Verknüpfung zwischen beiden Dokumenten definiert werden. Dies geschieht mit dem `<link>`-Element (nachfolgend wird die XHTML-Syntax verwendet):

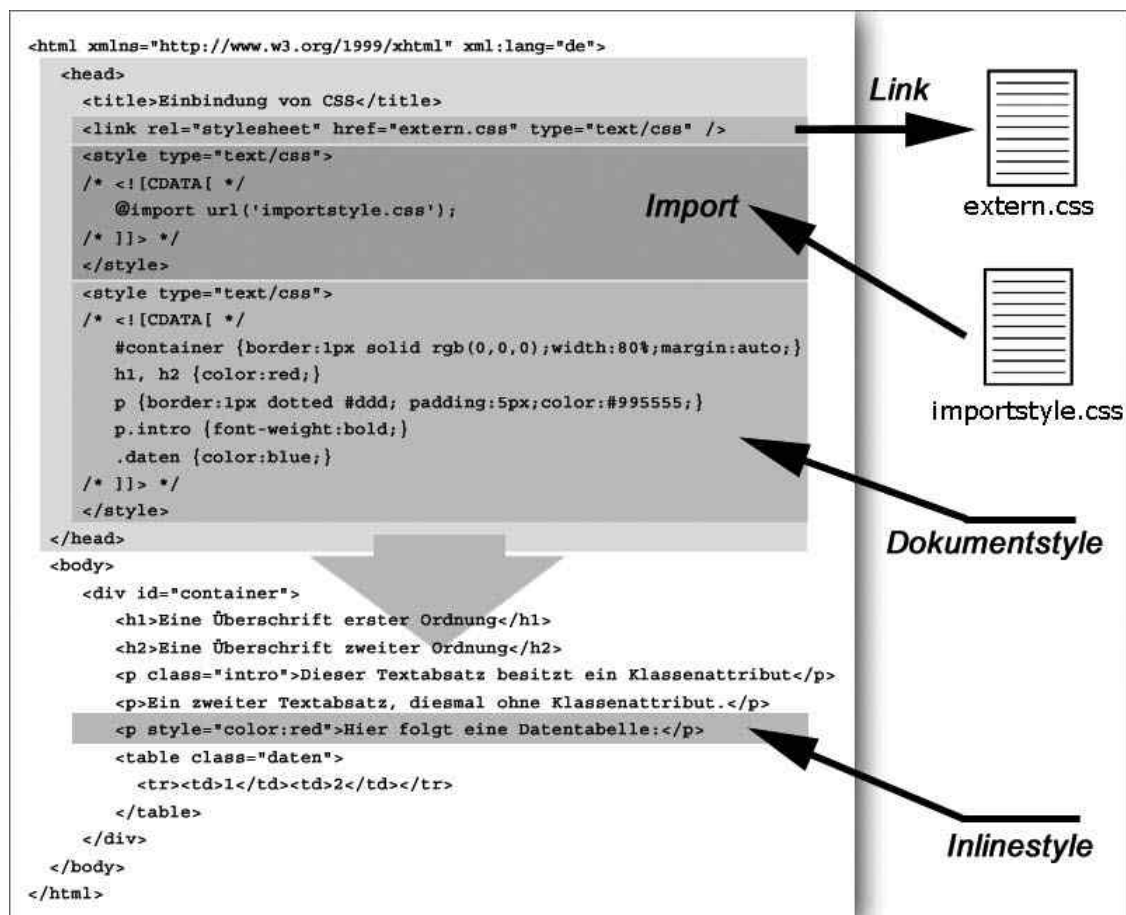
```
<link rel="stylesheet" href="extern.css" type="text/css"/>
```

Da das `<link>`-Element verschiedenen Zwecken dient, muss definiert werden, welche »Relation« zwischen Dokument und externer Ressource besteht. Dies geschieht mit dem `rel`-Attribut, das hier den Wert `"stylesheet"` erhält. Der Verweis selbst (eine Hyperreferenz) wird durch das `href`-Attribut ausgedrückt, das als Wert den Dateinamen und Pfad zur Ressource übergeben bekommt. Schließlich wird mit dem `type`-Attribut noch der MIME-Typ der Ressource genannt, der für CSS-Dateien stets `"text/css"` lautet. Alle Anweisungen der externen Datei werden nun auf das Dokument angewendet.

Insgesamt existieren jedoch vier Wege, wie eine Styleanweisung in ein Dokument und letztlich zu einem zu präsentierenden Element gelangen kann (siehe Abbildung A.5). Werden sie alle gleichzeitig angewendet, wirkt die Summe ihrer CSS-Anweisungen auf das Dokument (symbolisiert durch den Pfeil).

Die Methoden sind der Reihe nach:

- ▶ externe, verlinkte CSS-Datei (wie eben vorgestellt)
- ▶ externe, importierte CSS-Datei
- ▶ Dokumentstyle im `<style>`-Container
- ▶ Inlinestyle direkt im `style`-Attribut eines Elements



```
]]>
</style>
```

In diesem Fall handelt es sich jedoch nicht um eine Verknüpfung, sondern, wie der Name bereits sagt, um den *Import* von Anweisungen: Dies bedeutet, dass die extern liegenden Anweisungen gelesen und so behandelt werden, als ob sie *physisch* anstelle der Importanweisungen stehen würden. Dies ist wichtig, denn falls mehrere Importanweisungen nacheinander stehen, gilt die *zuletzt* erfolgte im Konfliktfall als maßgeblich.

Dokumentstyle

Enthält der Stylecontainer unmittelbar CSS-Anweisungen, spricht man von einem *Dokumentstyle*, da sich diese Anweisungen ausschließlich auf das aktuelle Dokument beziehen. Dem Reglement entsprechend, muss der Containerinhalt in XHTML als CDATA-Bereich ausgezeichnet werden. In der Praxis verursacht die Einbindung mit dem CDATA-Bereich Schwierigkeiten – die enthaltenen Anweisungen werden nur teilweise oder gar nicht ausgewertet (abhängig vom Browser), weil die CDATA-Marken nicht der CSS-Syntax entsprechen. Es empfiehlt sich daher, die Marken durch CSS-Kommentare zu verstecken:

```
<style type="text/css">
/* <![CDATA[ */

    h1 { color:red; }

/* ]]> */
</style>
```

Listing A.4 Stylecontainer in XHTML im praktischen Einsatz

In HTML ist die Einbindung eines Dokumentstyles einfacher, weil man hier keine CDATA-Marken benötigt – in HTML ist das Containerinnere von vornherein als CDATA definiert. (Es ist aus historischen Gründen noch oft üblich, die CSS-Anweisungen mit einem HTML-Kommentar zu umgeben. Erforderlich ist dies im Grunde nicht mehr.) Folgendes genügt in HTML:

```
<style type="text/css">
    h1 { color:red; }
</style>
```

Was ist denn nun eigentlich das »Stylesheet«?

Mit dem Begriff *Stylesheet* benennt man so salopp wie unkorrekt oft eine externe CSS-Datei. Technisch exakt ist *das Stylesheet* jedoch die Summe aller für ein Dokument gültigen CSS-Anweisungen, die sich aus dem Zusammenspiel von externen, importierten und lokalen Regeln ergeben.

A.4.2 Aufbau einer CSS-Anweisung

CSS besteht aus nichts weiter als aus einer Abfolge von Formatierungsregeln (schematisch in Abbildung A.6 zu sehen), sogenannten *Rule Sets*. Jede Regel zerfällt in zwei Teile:

- ▶ Den **Selektor**
Dieser weist die Präsentationsregel über ein Vergleichsmuster einem Elementcontainer eindeutig zu.
- ▶ Den **Deklarationsblock** (Declaration Block)
Dieser enthält, in geschweifte Klammern eingeschlossen, die eigentlichen Formatierungsanweisungen (Eigenschaftsdeklarationen).

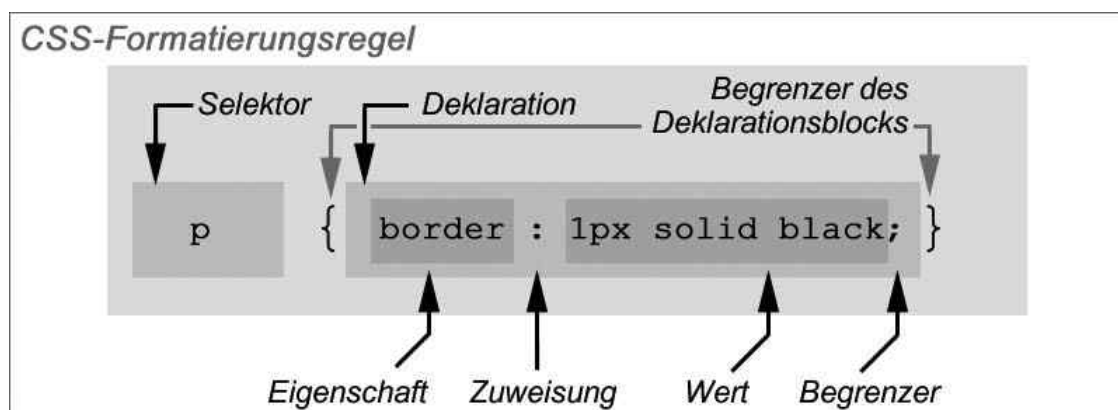


Abbildung A.6 Schema einer CSS-Formatierungsregel

Die Deklaration einer CSS-Eigenschaft

Jede Eigenschaftsdeklaration ist zusammengesetzt aus Paaren von *Eigenschaftsbezeichner* (name) und *Eigenschaftswert* (value) – getrennt durch einen Doppelpunkt, der als Zuweisungsoperator dient. Leerzeichen um den Doppelpunkt sind erlaubt. Das Ende der Wertzuweisung wird durch ein abschließendes Semikolon gekennzeichnet. (Ein zugewiesener Wert kann daher selbst auch Leerzeichen enthalten oder die Form einer kommagetrennten Liste annehmen.)

Ein Deklarationsblock kann eine oder mehrere Wertzuweisungen (Einzeldeklarationen) enthalten, die zusammen mit dem Selektor die Formatierungsregel bilden. Der Deklarationsblock wird durch geschweifte Klammern begrenzt.

Leerzeichen und Zeilenumbrüche zwischen den Regeln sind irrelevant. Sie können Styleanweisungen daher beliebig durch Einrückungen und Zeilenumbrüche formatieren und so ihre Lesbarkeit erhöhen. Hier eine Regel, die drei Anweisungen enthält:

```
p { border:1px dotted #ddd; padding:5px; color:#995555; }
```

Dieselbe Regel kann – etwas übersichtlicher – auch über mehrere Zeilen verteilt werden. Es ist Geschmackssache, ob Sie die schließende Klammer in eine neue Zeile setzen oder nicht:

```
p { border:1px dotted #ddd;
    padding:5px;
    color:#995555;
}
```

Achtung – das schließende Semikolon nicht vergessen!

Ein vergessenes *Semikolon* nach einer Wertzuweisung führt dazu, dass diese und die auf sie folgenden Zuweisungen nicht beachtet werden (Syntaxverstoß). Die vorangegangenen Zuweisungen werden immerhin erkannt. Nur bei der letzten Zuweisung darf auf das beendende Semikolon verzichtet werden.

A.4.3 CSS-Selektoren – die wichtigsten Grundformen

Da CSS-Selektoren, wie Sie später noch sehen werden, auch bei der Arbeit mit jQuery eine große Bedeutung besitzen, lohnt sich eine genauere Beschäftigung mit ihnen. Eine tabellarische Gesamtübersicht folgt später. Betrachten wir erst die Grundformen und werfen dann einen Blick auf deren praktische Anwendung.

Der Typselektor

Im einfachsten Fall besteht ein Selektor aus einem Elementnamen. Der Typselektor zielt auf Elemente des genannten Tag-Namens (»generic identifier«). Er besteht daher im einfachsten Fall aus einem Tag-Bezeichner.

Die folgende Stylesheet-Regel betrifft alle `<h1>`-Container des Dokuments:

```
h1 { color:red; }
```

Die Anweisung besagt, dass die Schriftfarbe im Inneren des Containers rot sein soll. Eine solche Regel gilt global im gesamten Dokument für alle Elemente des bezeichneten Typs.

Gruppierung von Selektoren

Soll der gleiche Regelsatz auf mehrere Elemente wirken, kann man ihn entweder mehrmals, mit verschiedenen Selektoren wiederholen (schlecht) oder eine *Selektorgruppe* bilden.

Hierfür werden die Einzelselektoren zu einer kommasetrennten Liste zusammengefügt, und dieser wird die Regel zugeordnet. Im einfachsten Fall ist es eine Liste aus Typselektoren:

```
h1, h2 { color:#ff0000; }
```

Es ist nicht erforderlich, die Liste in eine Zeile zu schreiben – da eine Selektoranweisung sehr viel länger ausfallen kann, tragen Zeilenumbrüche zur Übersichtlichkeit bei (nach dem letzten Selektor darf kein Komma mehr stehen):

```
h1,
h2 {
    color:#ff0000;
}
```

Klassenselektor ohne Typbindung

Manchmal sollen Anweisungen nicht pauschal allen Elementen einer Sorte zugewiesen werden, sondern nur einigen ausgewählten. Weil sie, als Untergruppe, quasi als »eigene Klasse« bezeichnet werden können, erhalten sie ein Attribut `class`:

```
<p class="intro">Den Ausgangspunkt für die Planungen
jedes Webauftritts bilden ... </p>
```

Um auf ein Element mit diesem Klassenattribut zuzugreifen, verwendet man dementsprechend einen Klassenselektor. Dieser besteht aus dem Namen der Klasse (achten Sie auf den *Punkt* vor dem Bezeichner).

```
.intro { font-weight:bold;
}
```

Eine solche Klasse wirkt auf jedes Element mit entsprechendem `class`-Attributwert, ungeachtet des Elementtyps. Eine Klasse ohne Elementbindung bezeichnet man als *freie Klasse*. Eine solche Klasse wirkt auf jedes Element mit entsprechendem Klassenattribut, ungeachtet des Elementtyps. Die Klasse `intro` könnte erfolgreich auch einem `<li>`-Container zugewiesen werden:

```
<ul>
  <li class="intro">Inhaltliche Ziele</li>
  <li>Funktionale Ziele</li>
</ul>
```

Klassenselektor mit Typbindung

Die Wirkung eines Klassenselektors kann auf eine Elementgattung beschränkt werden, indem ihm ein Typselektor vorangestellt wird. Der Punkt dient nun als Verbinder zwischen den beiden Teilen des Selektors:

```
p.frage { font-style:italic;
}
```

Diese Anweisung gilt nur für `<p>`-Container mit dem Klassenattributwert "frage", wobei die Anweisung `font-style:italic` die Schrift kursiv macht. Man spricht von einer *typgebundenen Klasse*. Eine typgebundene Klasse wirkt nur auf die Elemente ihres Typs, auch wenn die Klasse (versehentlich) einem anderen Element zugewiesen wird. Bei einem `<li>`-Element ist die Klasse `frage` daher unwirksam:

```
<ul>
  <li class="intro">Inhaltliche Ziele</li>
  <li class="frage">Funktionale Ziele</li>
</ul>
```

Der ID-Selektor

Sie haben das `id`-Attribut als Kennzeichner für funktionale Bestandteile des Dokuments kennengelernt. Hierfür ist es prädestiniert, da ein Identifier nur einmal pro Dokument mit demselben Wert erscheinen darf:

```
<body>
  <div id="inhalt">
    <h1>Websitegestaltung</h1>
    <p class="intro">Den Ausgangspunkt...
    ...
  </div>
</body>
```

Der ID-Selektor wirkt immer auf *genau ein Element*, das ein `id`-Attribut mit dem übergebenen Bezeichner besitzt. Hierfür wird der ID-Wert mit vorangestelltem Hash-Zeichen eingesetzt:

```
#inhalt { width:70%;
          border:1px dotted #4994CD;
          padding:1em;
}
```

Sie könnten hier, unter Hinzunahme des Elementbezeichners, auch genauer `div#inhalt` schreiben, was die Anweisungen unmissverständlicher macht. Dies erhöht die Übersichtlichkeit Ihres CSS-Dokuments.

Kontextselektoren

Ein Typselektor wirkt auf alle Elemente seines Tag-Namens, unabhängig von deren Position im Dokument. Dies ist oft unerwünscht – nehmen wir nur den Fall, dass wir Links in der Navigation anders gestalten wollen als solche im Fließtext (das wird meist so sein). Eine allgemein adressierte Regel wie

```
a { text-decoration:none; }
```

erweist sich als kontraproduktiv (grundsätzlich raten wir von solch pauschal gestalteten Selektoren ab). Vielmehr möchten wir die Position in der Dokumentstruktur (den »Kontext« des Auftretens) in den Selektor mit einfließen lassen:

```
div#navigation a { text-decoration: none; }
div#inhalt a { text-decoration: underline; }
```

Ein solcher Selektor liest sich am einfachsten von rechts nach links. Der zusammengesetzte Selektor `div#navigation a` bedeutet demnach »ein `<a>`-Tag in einem `<div>` mit dem ID `navigation`«. Dies kann beliebig komplex werden

```
div#inhalt p a.externerLink { ... }
div#inhalt li a.externerLink { ... }
```

was uns beispielsweise ermöglicht, einen extern gerichteten Link im Inhaltsbereich unterschiedlich darzustellen, je nachdem, ob er in einem Textabsatz oder in einer Liste auftritt.

A.4.4 CSS-Selektoren in der Praxis

Es existieren weitere spezialisierte Selektortypen, die wir abschließend vorstellen werden. Auch mit den bereits bekannten lässt sich eine Methodik entwickeln, die ausgehend von der Struktur des Dokuments bei der Planung von CSS-Dateien hilft. Gehen wir exemplarisch von folgender zu stylender Struktur im Dokument aus (ein typischer Navigationsbereich):

```
<div id="hauptnavi">
<ul>
  <!-- Hauptmenüebene: -->
  <li class="aktuell"><a href="index.html">Home</a></li>
  <li><a href="katalog.html">Katalog</a>
    <!-- Untermenüebene: -->
    <ul>
      <li><a href="produkt1.html">Produkt 1</a></li>
      ...
    </ul>
  </li>
  ...
</ul>
```

```

    <li><a href="Impressum.html">Impressum</a></li>
</ul>
</div>

```

Folgende Anforderungen werden gestellt:

- ▶ Hauptebene und Unterebene sollen verschieden gestaltet werden.
- ▶ Der Eintrag der aktuell gezeigten Seite soll hervorgehoben werden (hierfür wird er durch die Klasse `aktuell` im betreffenden `<li>`-Element gekennzeichnet).
- ▶ Für die Links soll ein Hover-Zustand definiert werden, um dem Nutzer ein optisches Feedback zu bieten (immer gut).

Betrachten wir nur die zu verwendenden Selektoren. Zu uneindeutig ist:

```
div#hauptnavi a { ... }
```

da dies auf alle Links sowohl in der Haupt- als auch in der Unterebene wirkt. Die Regel trifft aber auch eventuelle Links im Bereich `#hauptnavi`, die nicht Teil der Navigationsliste sind. Besser, weil spezifischer (d.h. »passgenauer«), ist:

```
div#hauptnavi ul li a { ... }
```

Leider trifft dies auf Links in der Unterebene ebenfalls zu. Für die erste Ebene folgt der Selektor jedoch exakt der Dokumenthierarchie, bildet diese also optimal ab. Gehen wir für die zweite Ebene analog vor, bilden also die *gesamte umliegende Hierarchie* ab:

```
div#hauptnavi ul li ul li a { ... }
```

Die vorher durch den Selektor für die Hauptebene unfreiwillig auch für die Unterebene vorgenommenen Anweisungen werden durch diese zweite, passgenauere Regel überschrieben.

Folgten Sie dieser Vorgehensweise konsequent, kämen Sie nach systematischer Auswertung der Anforderungen und der Dokumentstruktur auf folgende sinnvolle Selektoren:

```

/* Allgemeine Eigenschaften Navigationsbereich: */
div#hauptnavi { ... }

/* Styles Hauptebene: */
div#hauptnavi ul { ... }
div#hauptnavi ul li { ... }
div#hauptnavi ul li a { ... }
div#hauptnavi ul li a:hover { ... }
div#hauptnavi ul li.aktuell a { ... }

```



```
div#hauptnavi ul li.aktuell a:hover { ... }

/* Styles Unterebene: */
div#hauptnavi ul li ul { ... }
div#hauptnavi ul li ul li { ... }
div#hauptnavi ul li ul li a { ... }
div#hauptnavi ul li ul li a:hover { ... }
div#hauptnavi ul li ul li.aktuell a { ... }
div#hauptnavi ul li ul li.aktuell a:hover { ... }
```

Natürlich werden Sie nicht für alle diese Selektoren Regelsätze benötigen. Viele könnten Sie, bei gleicher Wirkung, auch einfacher schreiben (also unter Auslassung von Zwischenhierarchien). Zugegeben, eine Dokumentstruktur auch für die restlichen logischen Dokumentbereiche und deren mögliche Inhalte auf diese Weise vollständig in CSS abzubilden ist arbeitsintensiv. Drei Vorteile haben Sie dadurch jedoch:

- ▶ Die Gültigkeit der CSS-Regeln ist stets eindeutig.
- ▶ Die Selektoren sind selbstbeschreibend, da sie der Dokumentstruktur folgen.
- ▶ Eine Gliederung des CSS-Dokuments erfolgt anhand der HTML-Dokumentbereiche quasi »von selbst«.

A.4.5 CSS-Selektoren in der Übersicht

Eine detaillierte Erläuterung zu allen hier im Anschluss vorgestellten Selektoren ist nicht möglich. Da sie jedoch alle auch von jQuery eingesetzt werden, sollen sie zumindest kurz angerissen werden. Beginnen wir mit den Selektoren aus CSS 2.1, die in allen aktuellen Browsern unterstützt werden.

Selektor	Erläuterung
*	Universeller Selektor, trifft auf alle Elemente unabhängig vom Elementnamen zu.
E	Typselektor, gilt für Elemente mit Tag-Name »E« (hier und in Folge ist »E« ein Platzhalterbezeichner).
.klassenname	Freie Klasse, alle Elemente mit <code>class="klassenname"</code>
E.klassenname	Typgebundene Klasse; gilt für Elemente mit Namen E und <code>class="klassenname"</code> .
#identifizier E#identifizier	Freier und typgebundener ID-Selektor; im Prinzip wirkungsgleich

Tabelle A.8 Selektoren aus CSS 2.1

Selektor	Erläuterung
E F	Descendant-Selektor (»F enthalten in E«), wobei die Tiefe der Verschachtelung unerheblich ist
E > F	Child-Selektor (»F ist Kindknoten von E«), d.h. unmittelbar ohne Zwischenhierarchie in E enthalten
E + F	Adjacent Sibling-Selektor (»F folgt unmittelbar auf E«), d.h. auf gleicher Hierarchieebene (»direct adjacent«)
E[att]	Attributselektor (»E besitzt Attribut att«)
E[all=wert]	Attributselektor (»E besitzt Attribut att« mit Wert »wert«)
E[att~=val]	Attributselektor (»E besitzt Attribut att als Listentyp«, das den Teilwert »wert« enthält)
E[att =val]	Attributselektor (»E besitzt Attribut att«, dessen Wert mit »wert-« beginnt); speziell für Sprachbezeichner
E:hover	Pseudoselektor; gilt für E beim Überfahren des Containerbereichs mit der Maus

Tabelle A.8 Selektoren aus CSS 2.1 (Forts.)

Diese Selektoren und ihre Kombinationen können unbedenklich im Rahmen von Stylesheet-Definitionen eingesetzt werden.

A.4.6 Neue Selektoren in CSS 3

In CSS 3 kommen sehr viele Selektoren hinzu, von denen wir hier nur eine Auswahl auflisten können. Die meisten von ihnen sind Pseudoselektoren, die das Auftreten eines Elements innerhalb eines Containers oder die Containereigenschaften selbst beschreiben. Hier finden Sie die offizielle Spezifikation:

CSS 3-Selektoren: <http://www.w3.org/TR/css3-selectors/>

CSS 3-Selektoren noch nicht für praktischen Einsatz geeignet

Leider sind diese Selektoren auch in aktuellen Browsern noch nicht durchgängig unterstützt, weshalb sie beim Entwurf von Stylesheets derzeit noch mit Vorsicht zu verwenden sind.

Keine Frage jedoch, dass es gegebenenfalls praktisch wäre, mit einem Selektor wie `p:empty` beispielsweise auf leere Textabsätze zuzugreifen. Zum Glück interpretiert jQuery diese Selektoren sehr wohl, was uns eine Selektion passender Elemente dann zwar nicht über CSS, aber immerhin mittels JavaScript ermöglicht. Praktisch. Schauen wir uns diese Selektoren daraufhin doch einfach einmal an.

Selektor	Erläuterung
<code>E ~ F</code>	Sibling-Selektor (»F folgt auf E«), auf gleicher Hierarchieebene; jedoch nicht unbedingt unmittelbar (»indirect adjacent«)
<code>E[att^=val]</code>	Attributselektor (»E besitzt Attribut att«, dessen Wert mit dem Teilwert »wert« beginnt)
<code>E[att\$=val]</code>	Attributselektor (»E besitzt Attribut att«, dessen Wert mit dem Teilwert »wert« endet)
<code>E[att*=val]</code>	Attributselektor (»E besitzt Attribut att«, dessen Wert den Substring »wert« enthält)
<code>E:first-child</code>	Ein Element E, wenn es erstes Kindelement seines Elternelements ist
<code>E:last-child</code>	Ein Element E, wenn es letztes Kindelement seines Elternelements ist
<code>E:only-child</code>	Ein Element E, wenn es einziges Kindelement seines Elternelements ist
<code>E:nth-child(n)</code>	Ein Element E, wenn es in seinem Elternelement n–1 Vorgängerknoten besitzt
<code>E:nth-last-child(n)</code>	Ein Element E, wenn es in seinem Elternelement n–1 Nachfolgerknoten besitzt
<code>E:first-of-type</code>	Ein Element E, wenn es erstes Kindelement vom Typ E seines Elternelements ist
<code>E:last-of-type</code>	Ein Element E, wenn es letztes Kindelement vom Typ E seines Elternelements ist
<code>E:only-of-type</code>	Ein Element E als einziges Kindelement vom Typ E seines Elternelements
<code>E:empty</code>	Ein Element E, wenn es keinen Inhalt besitzt
<code>E:not(s)</code>	Negativer Selektor; Ein Element E, auf das der Selektor <i>s</i> <i>nicht</i> zutrifft
<code>E:enabled</code>	Form-Input-Selektor, spricht alle eingabebereiten (»enabled«) Inputs an.
<code>E:disabled</code>	Form-Input-Selektor, spricht alle deaktivierten (»disabled«) Inputs an.
<code>E:checked</code>	Form-Input-Selektor, spricht alle ausgewählten (»checked«) Radiobuttons oder Checkboxes an.

Tabelle A.9 Selektoren aus CSS 3 (Auswahl)

Die Bedeutung der Selektoren in jQuery

Prägen Sie sich die CSS-Selektoren gut ein – Sie werden Sie in jQuery häufig benötigen: Dort dienen sie als zentrales Argument, mit dessen Hilfe Sie diejenigen Teile Ihres Dokuments bezeichnen werden, die Sie über ein jQuery-Script bearbeiten möchten – sei es, um dort zusätzliche Interaktivität zu ermöglichen oder die Präsentation dynamisch zu verändern.

A.4.7 CSS-Eigenschaften

Sobald mittels eines Selektors ein Element oder eine Elementgruppe ausgewählt ist, wird diesem bzw. dieser eine Präsentation zugewiesen. Hierzu dienen die *CSS-Eigenschaften*, die in den Regelsätzen zusammengefasst werden.

Definiert werden können u. a.:

- ▶ **Text- und Fonteigenschaften**

direction, letter-spacing, line-height, text-align, text-decoration, text-indent, text-transform, unicode-bidi, vertical-align, white-space, word-spacing, font-family, font-size, font-style, font-variant, font-weight

- ▶ **Eigenschaften für Farben und Hintergründe**

background, background-attachment, background-color, background-image, background-position, background-repeat, color

- ▶ **Boxproperties** (Abmessungen, Padding, Margin, Rahmen)

padding, margin, border, border-color, border-style, border-width, border-radius (CSS 3), height, width, max-height, max-width, min-height, min-width, outline

- ▶ **PageLayout, Float- und Positioning-Properties**

bottom, clear, clip, float, left, overflow, position (static, relative, absolute, fixed), right, top, visibility, z-index

- ▶ **Display-Properties** (Sichtbarkeit, Displayverhalten)

display, visibility, opacity (CSS 3)

- ▶ **Eigenschaften für Listen und Tabellen**

list-style, list-style-type, list-style-image, list-style-position, border-collapse, border-spacing, caption-side, empty-cells, table-layout

Viele weitere Eigenschaften kommen in CSS 3 neu hinzu. Bereits in Gebrauch und daher in der Auflistung enthalten ist die `opacity`-Eigenschaft, die auch von jQuery an vielen Stellen eingesetzt wird (allerdings auf browserübergreifende Weise). Webdesigner freuen sich über die `border-radius`-Eigenschaft.



Abbildung A.7 Das Demonstrationstool »CSS Visual« auf der Begleit-DVD

In diesem Rahmen können wir natürlich nicht näher auf alle CSS-Eigenschaften eingehen – auf der Begleit-DVD zu diesem Buch finden Sie jedoch das Tool »CSS Visual«, das zum einen eine geeignete Übersicht bietet, zum anderen das Ausprobieren verschiedener Werte ermöglicht.

Da wir jedoch im Rahmen der folgenden Abschnitte (speziell bei Navigationen) ein Grundwissen über Dinge wie Positionierung und das Floaten von Containern voraussetzen, kann ein kleiner Überblick speziell über diese beiden Aspekte nicht schaden. Beginnen wir dafür aber beim »Normalfall«, nämlich den, wie man sagt, »im Flow« dargestellten Inhalten.

A.4.8 Dokumentflow

Jedes HTML-Element besitzt (sofern es zu den »sichtbaren« Elementen gehört) eine Grundeigenschaft, die sein Verhalten entweder als Block- oder als Inlineelement festlegt. (Diese Grundeigenschaft kann über die CSS-Eigenschaft `display` nachträglich verändert werden, was in vielen Fällen praktisch ist. Lassen wir dies aber zunächst beiseite.)

Die Darstellung des Dokuments erfolgt in Quelltextreihenfolge, wobei zunächst die Elemente mit Blockeigenschaften im Viewport »gestapelt« werden und dort

dessen gesamte Innenbreite einnehmen. Enthält ein Blockelement andere Blockelemente, wird es von diesen entweder unterbrochen (gilt für `<p>`-Container), oder innen liegende Böcke werden im Inneren des sie umgebenden Blocks ihrerseits gestapelt (z. B. Div in Div).



Abbildung A.8 Stapelung und Verschachtelung der Blöcke im »Flow«

Im Inneren der Blockelemente befinden sich entweder Texte oder Inlineelemente (hier je vier `<span>`-Tags). Beide verhalten sich insofern gleich, als dass sie das Innere der Blöcke (bis zu deren gesamter Innenbreite) Zeile für Zeile auffüllen – denken wir uns um jede dieser Zeilen einen virtuellen Container und nennen diesen »Zeilencontainer« (»line-box«) – in Abbildung A.8 durch den gepunkteten Umriss symbolisiert.

Setzen wir dieses Spiel fort, bis der Quelltext durchgearbeitet ist, so hat jeder Inhalt im Viewport seine Position gefunden. Diese Position entspricht derjenigen im »Flow«. Sie wird als »statische« Position bezeichnet. (Bis jetzt betrachten wir keine CSS-Besonderheiten, sondern nur das Defaultverhalten der HTML-Elemente.)

A.4.9 Positionierung

Interessant, besonders im Hinblick auf die sich ergebenden Layoutmöglichkeiten, wird es, sobald man von dieser »natürlichen« Positionierung abweicht. Hierzu dient die CSS-Eigenschaft `position`. Von Haus aus gilt ein Elementcontainer als »nicht positioniert«. Durch Zuweisung der `position`-Eigenschaft wechselt der Status des Containers auf POSITIONIERT, was, abhängig vom zugewiesenen Wert, gravierende Auswirkungen auf das Layout hat.

position:static

Die statische Positionierung entspricht der des Elements im normalen Dokumentfluss. Etwaige Angaben zur Positionierung (`left`, `top`, `bottom`, `right`) werden ignoriert. Die Zuweisung `position:static` hat keine Auswirkungen auf die Flow-Positionierung und ist deshalb eher rein formal definiert. Wir können sie also vernachlässigen.

position:relative

Interessanter ist der folgende Wert:

► `position: relative`

Die Positionierung bezieht sich auf die Position des Elements im normalen Dokumentfluss. Etwaige Angaben zur Positionierung (`left`, `top`, `bottom`, `right`) beziehen sich auf diese Ursprungsposition.

In der folgenden Abbildung A.9 hat der Span `#d1s2` den Status **RELATIV POSITIONIERT** erhalten, ohne dass eine Koordinatenangabe erfolgt ist:

```
span#d1s2 {
    position:relative;
    border-color:red;
    border-style:dotted;
}
```

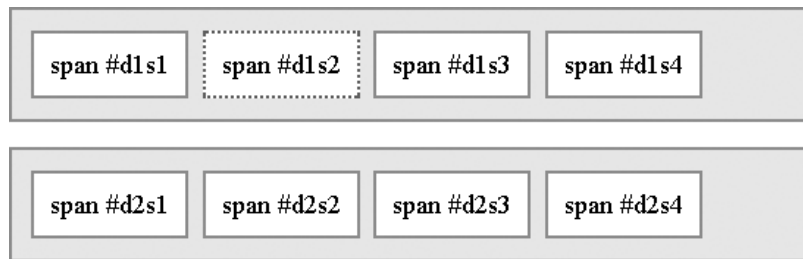


Abbildung A.9 Relative Positionierung ohne Repositionierung

Ein relativ positioniertes Element kann auch (wie in Abbildung A.10 gezeigt) von seiner Flow-Position verschoben werden. Die umliegenden Flow-Elemente verhalten sich, als ob der Container noch an Ort und Stelle wäre:

```
span#d1s2 {
    position:relative;
    top:18px;
    left:18px;
    border-color:red;
    border-style:dotted;
}
```

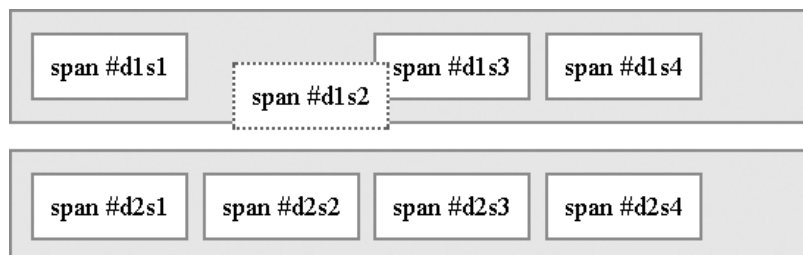


Abbildung A.10 Relative Positionierung mit Repositionierung

Wichtig ist die Tatsache, dass allein die Zuweisung der `position`-Eigenschaft die Statusveränderung zu `POSITIONIERT` bedeutet (ja, das haben Sie bereits gelesen, es ist aber elementar): Auch falls der so »positionierte« Container nicht aus seiner Position verrückt wird, dient sein Ursprung (linke obere Ecke) nunmehr als *Bezugspunkt* für in seinem Inneren befindliche Container. Doch dazu gleich mehr.

position:absolute

Die eben erläuterte Positionierung bringt den Dokumentfluss nicht durcheinander. Dies ist bei »absoluter« Positionierung nicht der Fall – ein so positionierter Container wird dem normalen Fluss entnommen und »schwebt« praktisch über der Grundebene der Seite.

► `position: absolute`

Die Positionierung mit `left`, `top`, `right` und `bottom` bezieht sich auf einen absoluten Koordinatenursprung, der entweder durch die obere linke Ecke des Browserfensters (d.h. des `<body>`-Elements) gebildet wird oder durch die obere linke Ecke des übergeordneten Elementcontainers, sofern dieser ebenfalls absolut oder fixiert (»fixed«) positioniert ist.

Folgecontainer ignorieren ein absolut positioniertes Element. In Abbildung A.11 ist `<span> d1s2` absolut positioniert; die folgenden `<span> #d1s3` und `#d1s4` rücken im Flow nach; der erste schiebt sich dabei hinter den positionierten Container. Es wurden hier, wohlgemerkt, keine Positionsangaben gemacht: Kommen dem positionierten Element nicht (wie hier) Padding- und Margin-Angaben in die Quere, bewegt es sich (noch) nicht von der Stelle:

```
span#d1s2 {
    position:absolute;
    border-color:green;
    border-style:dashed;
}
```

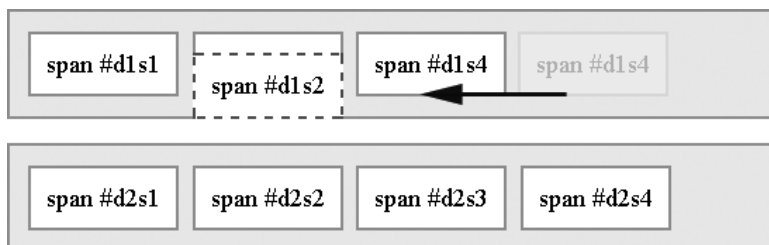


Abbildung A.11 Absolute Positionierung ohne Repositionierung

Gibt man, wie in Abbildung A.12, dem absolut positionierten Container Koordinaten (mit `top`, `left`, `right` oder `bottom`), beziehen sich diese auf den `Body-Tag`⁴ – die absolute Beziehung erfolgt also relativ (!) zu einem Bezugsobjekt:

```
span#d1s2 {
  position:absolute;
  top:18px;
  left:18px;
  border-color:green;
  border-style:dashed;
}
```

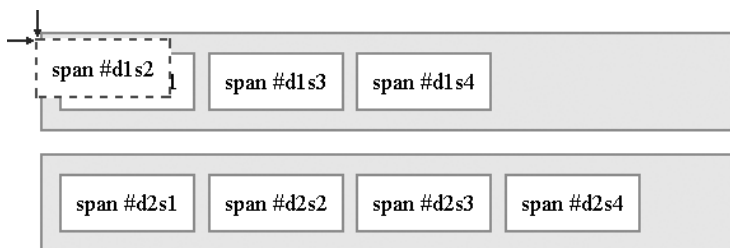


Abbildung A.12 Absolute Positionierung relativ zum Body-Element

Interessant wird es, sobald sich dieses Bezugsobjekt ändert. Dies geschieht, sobald in der Hierarchie zwischen dem positionierten Objekt und dem Body ein weiteres positioniertes Element erscheint – unabhängig davon, ob es absolut oder relativ positioniert ist (und ja, es kommt nur auf den *Status* an).

Als einzige Veränderung zu Abbildung A.12 wird dem umgebenden Div `#d1` der Status `position:relative` zugewiesen. Voilà, die Koordinaten des positionierten `<span> #d1s2` beziehen sich nicht mehr auf den `<body>`-Tag, sondern auf den `<div>`-Container (siehe Abbildung A.13):

```
div#d1 {
  position:relative;
  border-color:red;
  border-style:dotted;
}
```

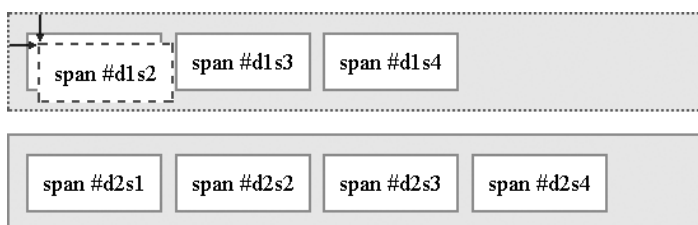


Abbildung A.13 Absolute Positionierung relativ zu positioniertem Div

⁴ Technisch ausgedrückt, ist der Body der »Containing Block« des positionierten Elements.

Praktische Anwendung bei FlyOut Navigationen

Bei der in Abschnitt 5.1.1, »Die FlyOut-Navigation«, vorgestellten FlyOut Navigation wird dieses Prinzip eingesetzt, um ein Subnavigationspanel korrekt zum Hauptmenüpunkt zu platzieren. Hierfür erhält das `<li>`-Element der Hauptnavigation den Status `position:relative`, die in ihm liegende `<ul>`-Liste wird durch *absolute* Positionierung *relativ* (sic!) zu diesem festgelegt.

position:fixed

Der Vollständigkeit halber: Eine weitere Positionierungsmöglichkeit ist die Angabe einer festen (*fixed*) Position gegenüber dem Viewport. Dies kann eingesetzt werden, um Teile des Layouts, ähnlich einem Frameset, im Browserfenster festzusetzen. Eine Verschachtelung des fixierten Elements in einen anderen positionierten Container spielt dabei keine Rolle (unabhängig von der Art dessen Positionierung).

► position: fixed

Der Elementcontainer bleibt beim Scrollen relativ zum Viewport fixiert. Die Positionierung mit `left`, `top`, `right` und `bottom` bezieht sich auf die linke obere Ecke des Browserfensters (nicht auf den Body-Container!).

A.4.10 Floats

Bei ineinandergeschachtelten Boxen ist es möglich, diese durch Zuweisen der Eigenschaft `float:left` bzw. `float:right` dazu zu veranlassen, sich entweder an den linken oder an den rechten Innenrand der umliegenden Box anzudocken. So weit, so gut. Obwohl gängige Praxis, ist das Prinzip dabei recht komplex.

Betrachten wir zunächst zwei Blockcontainer im Flow. Diese liegen übereinander im Viewport – der obere hat einen transparenten Hintergrund. (Die Rolle des umgebenden Containers spielt hier das `<body>`-Element.)

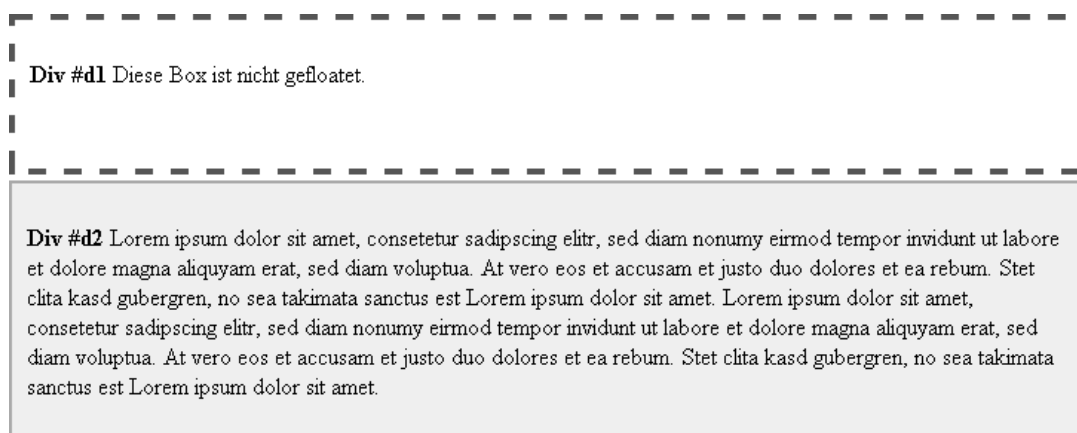


Abbildung A.14 Zwei Blockcontainer im Flow

Sobald man den ersten Container `Div #d1` mit der Eigenschaft `float:left` belegt, ändert sich das Bild: Da der gefloatete Container aus den Flow entfernt wird, schiebt sich der zweite an dessen Position vor. Dies betrifft allerdings (wie man sieht) nur dessen *Box* (beachten Sie dabei die Überlappung der Rahmen), nicht aber die im zweiten Block liegenden *Zeilencontainer*, die sich rechts um den gefloateten Block anordnen:

```
#d1 {
  float:left;
  height:60px;
  padding:10px;
  border:4px dashed #555;
  margin-bottom:4px;
}
```

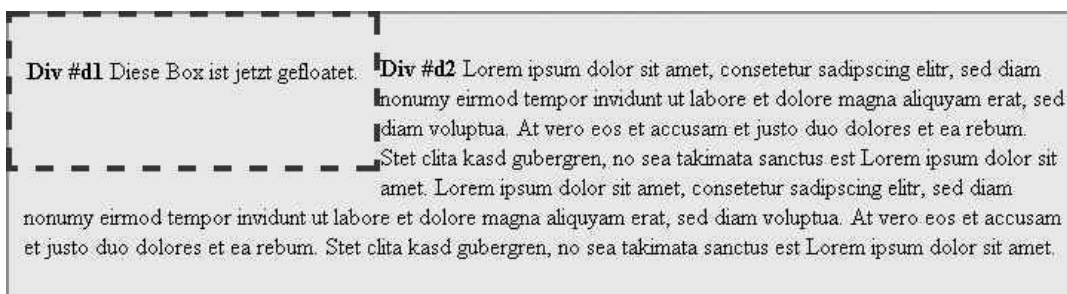


Abbildung A.15 Ein gefloateter und ein im Flow befindlicher Container

Bemerkenswert ist auch, dass der gefloatete Block seine Defaultbreite von 100 % verliert und nur noch die minimal durch die Inhalte erforderliche Breite einnimmt! Da der Block bereits vorher hart am linken Rand des Bodys lag, rührt er sich scheinbar nicht.

Sind gefloatete in nicht gefloateten Containern enthalten, ragen sie unter Umständen aus diesen hinaus – da sie nicht im Flow sind, tragen sie nicht zur Höhe der Inhalte des umgebenden Containers bei (dieser schrumpft auf 0, wenn er nichts als gefloatete Inhalte besitzt).

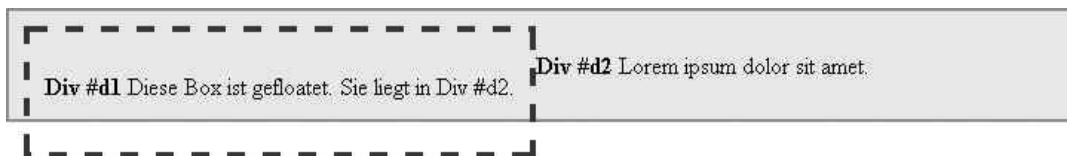


Abbildung A.16 Ein gefloateter, in einem im Flow befindlicher Container

Ein dem umgebenden Block folgender Container schiebt sich ebenfalls unter die Floatbox. Das ist unter Umständen natürlich nicht erwünscht (beispielsweise bei Seitenlayouts).

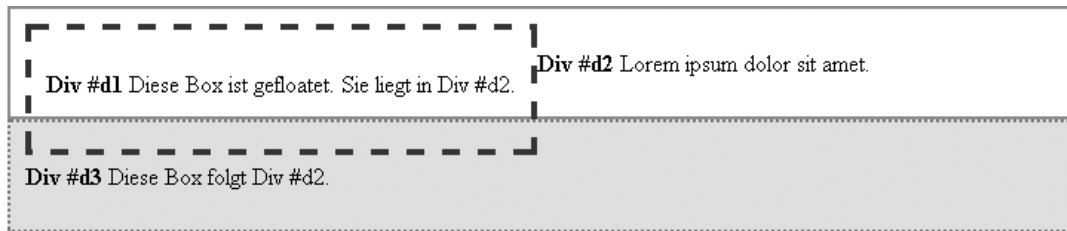


Abbildung A.17 Die gefloatete Box ragt auch über den folgenden Container hinaus.

Hier ist eine Lösung einfach. Um den Folgecontainer aus dem Bereich des Floats hinauszuschieben, genügt es, ihm die Eigenschaft `clear:both` zu geben (bzw. wie hier `clear:left`):

```
#d3 {
    clear:left;
    padding:10px;
    background-color:#e9e9e9;
    border:2px dotted #999;
}
```

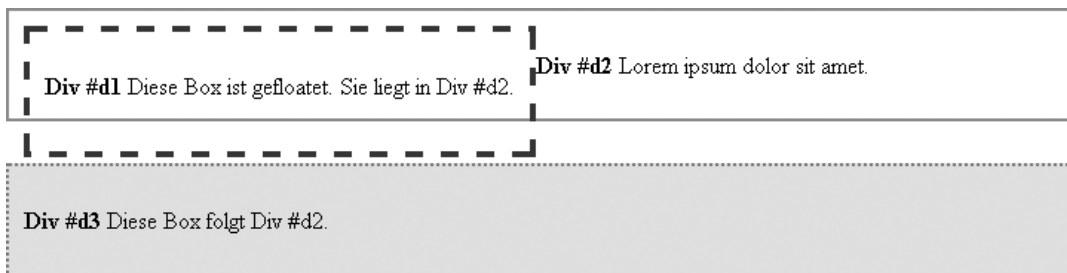


Abbildung A.18 Der Folgecontainer besitzt die Eigenschaft »clear:left«.

Eine Lösung, das Herausragen der Floatbox aus seinem übergeordneten Block zu beseitigen, besteht darin, das *übergeordnete* Objekt seinerseits zu floaten – damit brächten Sie allerdings den Flow nachfolgender Elemente in Unordnung (siehe Abbildung A.19).

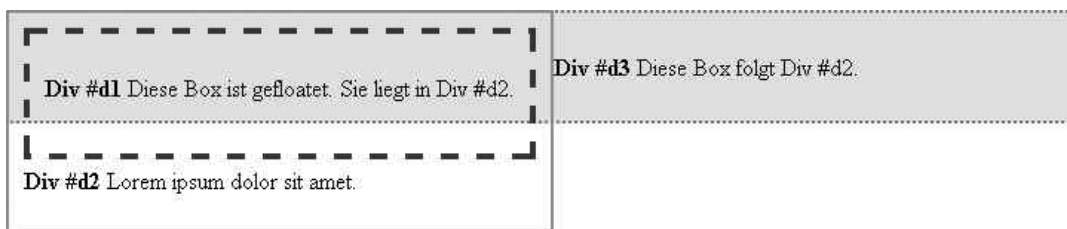


Abbildung A.19 Die Floatbox überdeckt den folgenden Container.

Der nachfolgende Container Div #d3 schiebt sich (wie zu erwarten) hinter das gefloatete Gesamtgebilde – sofern er nicht eine `clear`-Eigenschaft besitzt.

Besser wäre eine autarke Lösung, bei der man den äußeren Container nicht floaten muss und sich über die Eigenschaften eventueller Folgecontainer trotzdem keine Gedanken zu machen braucht. Hier hilft ein Griff in die Trickkiste (leider bleibt hier der Internet Explorer bis Version 7 außen vor).

Wir nehmen das Pseudoelement `:after` zu Hilfe, um nach dem Inhalt des äußeren Containers einen künstlichen Block mit `clear`-Eigenschaft zu erzeugen:

```
#d2:after {
    content:"x";
    display:block;
    height:0;
    clear:both;
    visibility:visible;    /* in der Praxis: hidden ! */
}
```

Das Ergebnis überzeugt (siehe Abbildung A.20). Natürlich würde man im Ernstfall die Sichtbarkeitseigenschaft des »Zwischenblocks« auf `hidden` setzen. Anstelle eines `x` genügt auch ein Leerzeichen (meist wird ein Punkt verwendet).

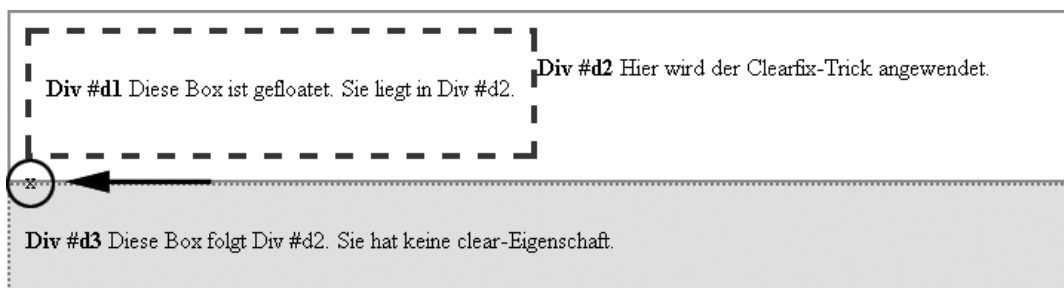


Abbildung A.20 Internes Clearing mit dem Clearfix-Trick

Alles ist zu Positionierung und Floats damit bei Weitem noch nicht gesagt – um dies Thema erschöpft behandelt zu sehen, greifen Sie besser zu einer Einführung zu CSS. Hierfür gibt es zahlreiche spannende und interessante Bücher – eine Liste mit Literaturvorschlägen finden Sie im Anhang.

A.4.11 Statische Präsentation dynamisieren

Sobald wir mit HTML eine Struktur erstellt und an diese Struktur mithilfe von CSS-Selektoren eine Präsentation gebunden haben, stoßen wir wieder an eine Grenze. Prinzipiell sind CSS-Präsentationen nämlich wiederum statisch, also unveränderlich.

Gut, mithilfe der `Hover`-Eigenschaft kann man für manche Elemente zwischen zwei Präsentationen »umschalten« und damit bereits ein Mindestmaß an Interaktion ermöglichen:

```

a      { text-decoration: none;
        font-weight:normal;
        color: black;
      }
a:hover { text-decoration:underline;
        font-weight:bold;
        color:red;
      }

```

Man kann hier selbstverständlich noch wesentlich imposantere Effekte erzielen – es ist aber stets der harte Wechsel zwischen zwei Zuständen, vergleichbar mit einem Schalter: An, Aus. Und manchen Browser muss man selbst hierbei noch auf die Beine helfen (jQuery hilft hierbei – wir kommen noch darauf zurück).

Was fehlt,⁵ ist eine Interpolation zwischen zwei Zuständen, die eine Animation ermöglicht. Dies kann eine Positionsänderung sein, ein gradueller Farbwechsel, ein sanftes Ein- oder Ausblenden anstelle eines schlagartigen Versteckens oder ein Aufziehen und Schließen per Größenveränderung. Oder eine Kombination aus alldem ...

Für praktisch alle CSS-Eigenschaften wäre eine entsprechende Dynamisierung wünschenswert. Hierfür haben Sie mit jQuery ein Werkzeug an der Hand, das Ihnen die Tür zur Animation von CSS weit aufstößt. Schauen wir uns aber jetzt an, wie JavaScript grundsätzlich funktioniert und wie es die HTML-Struktur betrachtet. Das nächste Thema ist also: JavaScript und das DOM.

5 Die Spezifikationen »CSS Transitions« und »CSS Animations« versprechen für die Zukunft Abhilfe: www.w3.org/TR/css3-transitions, www.w3.org/TR/css3-animations

»Over the course of the past decade, the perception of JavaScript has evolved from a simple toy language into a respected programming language used by corporations and developers across the globe to make incredible applications.« – John Resig

B JavaScript und DOM

Nachdem wir uns im vorigen Abschnitt mit den »statischen« Aspekten der Beschreibung einer Website beschäftigt haben, also mit HTML und CSS, wenden wir uns nun dem dynamischen Aspekt zu. Dieser wird durch JavaScript abgedeckt. In Form des Document Object Models (DOM) lernen wir darüber hinaus die Schnittstelle kennen, die eine Zusammenarbeit zwischen HTML, CSS und JavaScript erst möglich macht. Spätestens ab hier wird es spannend.

B.1 JavaScript – Beschreibung des Verhaltens

Man könnte damit anfangen, dass JavaScript einfach »die Programmiersprache für das Web« ist. Zu sehr ins Detail gehen wollen wir nicht, ebenso wenig, wie wir Sie mit historischen Dingen behelligen wollen. Wenn Sie sich für die Geschichte und Syntax von JavaScript interessieren, dann finden Sie im Internet genügend Quellen – auch gute Bücher zum Thema gibt es reichlich.

ECMA-Script – der offizielle Sprachstandard

Zu JavaScript existiert in Form von ECMA-Script (ECMA-262) ein offizieller, nicht an einen kommerziellen Hersteller gebundener Standard. Aktueller Stand ist ECMA-Script 5 (www.ecmascript-lang.org). Die Weiterentwicklung von JavaScript liegt ansonsten in der Hand der Mozilla Foundation.

JavaScript wird stets direkt als Quelltext verarbeitet. Die Ausführungsgeschwindigkeit ist zwar gegenüber kompilierten Sprachen geringer, der Vorteil, nämlich die leichtere Editierbarkeit, sollte dies aber aufwiegen. Obendrein ist JavaScript in den Grundzügen wegen der einfachen Syntax leicht zu erlernen.

Die Aufgabe des Browsers in diesem Zusammenhang besteht in der Interpretation und Ausführung der Scripte. Hierfür verwendet er eine sogenannte *Script-*

Engine. Diese unterscheidet sich von Browser zu Browser geringfügig. Aktuelle Engines wie *TraceMonkey* (Firefox), *V8* (Google Chrome) oder *Chakra* (IE9) kompilieren das Script, was die Verarbeitungsgeschwindigkeit erhöht.

Vom Typ her ist JavaScript eine »prototypisierte«, objektorientierte Sprache, ein Begriff, den wir gleich noch erläutern werden. Hierin unterscheidet sie sich von Java (Namensvetter, aber ansonsten nicht verwandt), das eine »klassenbasierte«, objektorientierte Sprache ist. JavaScript ist dennoch zu allem (und teilweise mehr) in der Lage, was im Rahmen von Objektorientierung möglich ist – auch wenn einige Leute anderes behaupten.¹

B.1.1 Grundlagen

Entgegen anderen Sprachen brauchen Sie sich in JavaScript beim Speichern von Werten und Deklarieren von Variablen keine Gedanken über deren Typ zu machen. Datentypen existieren jedoch sehr wohl.

Datentypen – die Arten von Daten

JavaScript unterscheidet drei einfache (*primitive*) Datentypen und einen (oder je nach Lesart mehrere) komplexen Datentyp (*complex*).

Die einfachen Datentypen sind

- ▶ String (Zeichenkette)
- ▶ Number (Zahlen aller Art, egal, ob Ganzzahl oder Bruchzahl)
- ▶ Boolean (Wahrheitswert, `true` oder `false`)

Als komplexer Datentyp begegnet uns das »Objekt«, das wir der Einfachheit als »Wert mit innerer Struktur« definieren. Die einfachen Typen hingegen besitzen keine innere Struktur sondern werden als »nicht untergliederbar« und »unveränderlich« verstanden. Wir betrachten gleich einige Aspekte der komplexen Typen, die uns in Form von Arrays, »normalen« Objekten und Funktionen begegnen. Da Werte in Programmen stets irgendwo »abgelegt«, d.h. gespeichert werden müssen, benötigen wir hierfür Variablen, die gewissermaßen »benannte« Speicherplätze darstellen.

Variablen – Speicher für Werte

Dass eine Variable in JavaScript nicht fest an einen Typ gebunden bleibt, vereinfacht zwar das Schreiben von Programmen, die implizite (kontextabhängige)

¹ Richtig ist, dass einige Paradigmen der Objektorientierung in JavaScript nicht erfüllt scheinen. Diese lassen sich jedoch alle, teilweise auf Umwegen, dennoch implementieren.

Typänderung bei Operationen ist allerdings auch die Quelle lustiger Fehler (abhängig vom Humor des Programmierers). Aber zunächst die einfachen Dinge:

Eine Variable wird mit dem Schlüsselwort `var` eingeführt:

```
var beispiel;
```

Hierbei kann gleichzeitig ein Wert zugewiesen werden. In diesem Fall ist es eine Zeichenkette (`string`), gekennzeichnet durch die Anführungszeichen (Stringbegrenzer). Die Variable braucht jedoch nicht als »Stringvariable« deklariert zu werden:

```
var text = "Hallo Welt!";
```

Mehrere Variablen können kommagetrennt gemeinsam deklariert werden:

```
var a, b = 7, c;
```

Die zweite Variable hat gleichzeitig einen Wert erhalten. Das Semikolon am Ende schließt die Anweisung ab. Fehlt es, wird das Zeilenende als Anweisungsende verstanden. Achtung – dies gilt nur dann, wenn (erstens) die so in der Luft hängende Anweisung als abgeschlossen betrachtet werden kann und (zweitens) die folgende Zeile nicht als deren Weiterführung betrachtet werden muss.

Sie können also durchaus *Zeilenumbrüche innerhalb einer Anweisung* vornehmen, wie hier demonstriert (das machen wir uns später häufiger zunutze):

```
var a,
    b = 7,
    c;
```

Dies ist wirkungsgleich zum darüberstehenden Beispiel. Achten Sie auf die Kommas, die sich leicht mit dem Semikolon verwechseln lassen.

Die Wertzuweisung

Eine Variable kann verschiedene Werte aufnehmen und *wechselt ihren Typ* (*string*, *number*, *boolean* etc.) je nach Art des aktuell gespeicherten Wertes. Bei einer Zuweisung steht auf der linken Seite (*left hand side*, LHS) des Gleichheitszeichens stets eine Variable.

Auf der rechten Seite (RHS) befindet sich der Wert oder ein Ausdruck, der den Wert erzeugt – die rechte Seite wird gegebenenfalls interpretiert, bevor die Zuweisung stattfindet (wir arbeiten also von *rechts nach links*). Innerhalb des auszuwertenden Ausdrucks wird wiederum von *links nach rechts* gearbeitet.

Hier wird auf der rechten Seite eine Addition vorgenommen und das Ergebnis zugewiesen:

```
var summe = 5 + 12; // ergibt 17
```

Treffen Werte unterschiedlichen Typs in einem Ausdruck zusammen, entscheidet der »stärkere« Kontext:

```
var summe = "08" + 15; // ergibt "0815" (als String)
```

Da die Ziffernfolge »08« als String gekennzeichnet ist, erzwingt sie die Umwandlung der Zahl 15 in einen String *vor* der Verknüpfung (implizite Typumwandlung). Anstelle einer Addition findet eine Verkettung statt.

Arrays – Speicher für mehrere Werte

In die bisher betrachteten Speicher für Werte passt jeweils nur eine Größe. Oft ist es praktisch, unter einem Bezeichner nicht nur einen, sondern eine ganze Reihe von Werten verfügbar zu haben.

Der komplexe Typ »Objekt« besitzt einige Spezialausprägungen, von denen uns an dieser Stelle das »Array« interessiert. Ein Array ist eine Art »Multivariable«, in der unter einem Namen beliebig viele Werte gespeichert werden können.

Ein Array wird wie folgt erzeugt und wird in eine normale Variable gespeichert:

```
var woche = new Array(); // erzeugt ein leeres Array
```

Die »Multivariable« speichert ihre Werte in »Fächern«, die über numerische Indizes erreicht werden. Die Nummerierung beginnt bei 0 (erstes Fach) und wird in eckigen Klammern an den Array-Namen angehängt:

```
woche[0] = "Montag";
```

Analog erfolgt das Auslesen:

```
alert(woche[0]); // -> Alert: "Montag"
```

Weitere Fächer können gefüllt werden, indem einfach der entsprechende Index angesprochen wird. Hierbei können (meist nicht sinnvoll) auch Fächer übersprungen werden. Oft benötigt wird das Feature, einen Wert an das Array anzuhängen. Hierfür wird die `push()`-Methode des Arrays eingesetzt:

```
woche.push("Dienstag");
```

Sie brauchen dabei nicht zu wissen, das wievielte Fach hierbei befüllt wird. Die Zahl der Array-Fächer lässt sich jedoch mit der `length`-Eigenschaft auslesen:

```
alert(woche.length); // -> Alert: 2
```

Auch das Entfernen von Array-Fächern ist möglich. Mit `pop()` wird das *letzte* Fach, mit `shift()` das *erste* Fach entfernt:

```
woche.pop(); // Entfernen des letzten Faches
woche.shift(); // Entfernen des ersten Faches
```

Das Array hat jetzt die Länge 0, da es keine Fächer mehr besitzt – es ist aber nach wie vor ein Array. Statt wie mit `push()` am *Ende* können Sie auch am *Anfang* des Arrays einen Wert hinzufügen. Dies geschieht mit `unshift()`:

```
woche.unshift("Montag"); // vor dem ersten Fach einfügen
```

Alle folgenden Fächer (zurzeit sind keine da) würden hierbei um einen Indexwert nach hinten geschoben.

Weitere Array-Methoden

Bekannte weitere Array-Methoden sind `reverse()`, `sort()`, `splice()`, `concat()`, `join()`, `slice()`, `toString()`. Es existiert zudem eine Reihe neuer Array-Methoden (aus JavaScript 1.6 und 1.8), die zum Teil noch nicht in allen Browsern unterstützt werden: `indexOf()`, `lastIndexOf()`, `every()`, `filter()`, `forEach()`, `map()`, `reduce()`, `reduceRight()` und `some()`.

Ausführliche Erläuterungen finden Sie bei: <https://developer.mozilla.org/>

Array-Literale

Ein leeres Array kann aber auch als *Literal* geschrieben werden, was Ihnen noch des Öfteren begegnen wird. Das Wesentliche dabei ist, dass ein Array nicht »erzeugt« wird (beispielsweise über `new Array()`), sondern einfach »hingeschrieben« werden kann:

```
var woche = []; // leeres Array wie mit new Array()
```

Hier ist es möglich, Inhalte gleich mit zu übergeben. Diese werden durch Komma getrennt:

```
var woche = ["Montag", "Dienstag", "Mittwoch", ...];
```

Ebenso ist es möglich (und üblich), ein Literal über mehrere Zeilen zu schreiben:

```
var woche = [
    "Montag",    // Komma nach dem Wert
    "Dienstag",  // Komma nach dem Wert
    "Mittwoch",  // Komma nach dem Wert
    // ...,
    "Sonntag"    // kein Komma nach letztem Wert
];
```

Wichtig – nach dem letzten Wert steht *kein Komma*! Dies ist einer der wenigen ernsthaften Fehler, die man bei Array-Literalen machen kann. Auf das Array-Li-

teral können selbstverständlich ebenfalls alle Array-Methoden angewendet werden – immerhin handelt es sich um ein normales Array.

B.1.2 Kontrollstrukturen – Bedingungen und Schleifen

Nun besteht ein Programm aus mehr, als einer Abfolge von Anweisungen. Eine grundsätzliche Notwendigkeit existiert in der Steuerung dieser Abfolge. Dies geschieht durch Kontrollstrukturen, nämlich Bedingungen und Schleifen. Sie sollten mit Konstrukten wie `if-else`, `while`, `do-while` und `for` sowie der `for-in`-Schleife (zumindest halbwegs) vertraut sein, da Sie dieses Buch in die Hand genommen haben. Sie werden hier auszugsweise kurz gestreift, und zwar `if-else` und `for`, weil diese im Buch am häufigsten erscheinen. Auf ein weiteres Konstrukt soll noch besonders eingegangen werden, da es gern verwendet wird, aber dennoch wenig geläufig ist – der *ternäre Operator*.

Bedingung `if-else`

Die einfachste Kontrollstruktur ist die *Bedingung*, die einen Anweisungsblock durch `if` an einen Booleschen Ausdruck (die Bedingung) knüpft, der in der Bedingungsklammer steht. Ist die Bedingung erfüllt (`true`), wird der Block ausgeführt, anderenfalls (`false`) übersprungen:

```
if(x > 5) {
    alert("x ist größer als 5!");
}
```

Ist die Bedingung nicht erfüllt, kann ein mit `else` gekennzeichneteter alternativer Block ausgeführt werden:

```
if(x > 5) {
    alert("x größer 5!");
}
else {
    alert("x nicht größer 5");
}
```

Der ternäre Operator

Diese Konstellation (tu dies oder tu das) ist in der Programmierung so häufig, dass es hierfür eine Kurzschreibweise gibt. Sie werden ihr recht oft begegnen, da sie sich durch Kompaktheit auszeichnet. Identisch mit der `if-else`-Anweisung ist Folgendes:

```
x > 5 ? alert("x größer 5!") : alert("x nicht größer 5");
```

Dieses Konstrukt wird als *ternärer Operator* bezeichnet, da hier *drei Ausdrücke* (Operanden) beteiligt sind:

- ▶ die zu prüfende Bedingung (vor dem Fragezeichen)
- ▶ die Anweisung bei Erfüllung (vor dem Doppelpunkt)
- ▶ die Anweisung bei Nichterfüllung (nach dem Doppelpunkt)

Auch hier ist bei Bedarf eine Aufteilung über mehrere Zeilen gestattet:

```
x > 5 ?
  alert("x größer 5!") :
  alert("x nicht größer 5");
```

Achten Sie auch hier auf die Verwechslungsgefahr zwischen Doppelpunkt und Semikolon.

Bedeutung für die Arbeit mit jQuery

Der *ternäre Operator* wird auch in jQuery häufig eingesetzt, weil er kompakt zu schreiben ist. Für Anfänger ergibt sich daraus eine erschwerte Lesbarkeit der Scripte – schärfen Sie also Ihr Auge. Sie können aber den ternären Operator stets durch ein (umständlicheres) *if-else*-Statement ersetzen, wenn Ihnen das lieber ist.

Die for-Schleife

Sollen Anweisungsblöcke wiederholt werden, muss eine Schleife als Kontrollstruktur her. Im Grunde ist dies nichts großartig anderes als eine Bedingung, nur dass nicht entschieden wird, ob ein Block ausgeführt wird, sondern, ob er wiederholt werden soll. Die Bedingung hierzu wird daher als *Schleifenbedingung* bezeichnet.

In JavaScript existieren außerdem noch *while*- und *do-while*-Schleifen. Wir beschränken uns hier aber auf die *for*-Schleife, weil sie in Zusammenhang mit Arrays extensiv eingesetzt wird. Außerdem sind die anderen Konstrukte vergleichsweise einfach. Die Besonderheit der *for*-Schleife ist der eingebaute Zähler. Das Schema ist Folgendes:

```
for(zählerdeklaration; bedingung; zähleriteration) {
    // Anweisungen
}
```

Als Zählervariable wird traditionell *i* (wie *iterator*) verwendet. Sie können allerdings auch einen beliebigen anderen Bezeichner verwenden. Da es sich um eine »echte« Variable handelt, wird sie am saubersten mit *var* deklariert. Hier eine Schleife, die viermal läuft:

```
for(var i=1; i<5; i++) {
    alert("Dies ist der " + i + ". Durchlauf.");
}
```

Sie sehen, dass auf das `i` im Inneren der Schleife zugegriffen werden kann. Dies ist extrem wichtig. Auch ist es möglich, das `i` auf einen anderen Startwert, beispielsweise 0, zu setzen. In Einerschritten hochgezählt, wird mit dem `++`-Inkrement-Operator (andere Zählrichtungen oder Schrittweiten sind machbar):

```
for(var i=0; i<woche.length; i++) {
    alert("Der "+i+". Tag der Woche ist "+woche[i]+"!");
}
```

Hier wird die Schleife eingesetzt, um ein Array auszulesen. Unökonomisch ist, dass bei jedem Schleifendurchlauf erneut über `woche.length` die Array-Länge ausgewertet werden muss. Sie können dies auch im Rahmen des Deklarationsstatements tun und in eine Variable ablegen (achten Sie auf das Komma zwischen den Variablendeklarationen):

```
for(var i=0, len=woche.length; i<len; i++) {
    alert("Der "+i+". Tag der Woche ist "+woche[i]+"!");
}
```

Die Schleifenbedingung wird so performanter (einen merklichen Unterschied werden Sie normalerweise allerdings kaum feststellen). Neben dem Auslesen eines Arrays ist der umgekehrte Fall ebenfalls denkbar – Sie füllen dann ein leeres Array mit den Zahlen von 1 bis 10:

```
var zahlenarray = [];
for(var i=1; i<11; i++) {
    zahlenarray.push(i);
}
alert(zahlenarray); // -> 1,2,3,4,5,6,7,8,9,10
```

Bedeutung für die Arbeit mit jQuery

Sie werden in jQuery viel mit *Arrays* und arrayähnlichen Gebilden, den sogenannten *Collections* zu tun haben. Arrays und `for`-Schleifen stehen in einer beinahe symbiotischen Beziehung und tauchen deshalb in vielerlei Zusammenhängen gemeinsam auf.

B.1.3 Funktionen

Eine Funktion ist auf den ersten Blick nichts weiter als ein benannter Anweisungsblock, der bei Bedarf auf Zuruf ausgeführt wird. Um diese Anweisungen von außen zu steuern, ist die Übergabe von Werten an die Funktion möglich, und damit die Funktion mit dem Rest der Welt Kontakt aufnehmen kann, existiert ein

sogenannter *Rückgabewert*, der durch das Schlüsselwort `return` gekennzeichnet ist. Nimmt eine Funktion anders als über den Rückgabewert Einfluss auf ihre Umgebung, beispielsweise indem sie eine globale Variable verändert, bezeichnet man dies als *Seitenwirkung* (»side effect«).

Eine *Funktion* in diesem Sinne könnte wie folgt deklariert werden – diese hier nimmt zwei Werte entgegen, addiert diese innerhalb des Anweisungsblocks und gibt das Ergebnis zurück. Wir haben sie deshalb *addieren* getauft:

```
function addieren(a, b) {
    var summe = a + b;
    return summe;
}
```

Aufgerufen wird die Funktion über ihren Namen, wobei in den dann unbedingt folgenden runden Klammern die Parameter übergeben werden:

```
var summe = addieren(5, 7);
alert(summe); // -> Alert: 12
```

Der Kern der Funktionsdeklaration ist das Schlüsselwort `function`, dem im vorigen Beispiel der Funktionsname folgt. Ziehen wir kurz in Betracht, dass eine Funktion mehr ist als ein benannter Anweisungsblock – nämlich ein »Objekt«. Genauer gesagt handelt es sich um ein *Funktionsobjekt* (das aber tatsächlich nur eine Spezialausprägung der »gewöhnlichen« Objekte darstellt, die wir später behandeln).

Ein Funktionsobjekt brauchen Sie im Prinzip nicht zu benennen. Sie können es stattdessen in eine Variable ablegen, um es später »zur Hand« zu haben:

```
var addieren = function (a, b) {
    var summe = a + b;
    return summe;
}
```

Der Aufruf der Funktion funktioniert haargenau so wie eben. In der Variable `addieren` ist das Funktionsobjekt. Sie hängen für den Aufruf an den Variablennamen einfach die Funktionsklammern an und fertig. Der Rückgabewert landet wieder in der Variable `summe`:

```
// das kennen wir schon:
var summe = addieren(9, 4);
alert(summe); // -> Alert: 13
```

Nebenbei, warum existieren hier zwei Variablen namens `Summe`? Könnte man nicht eine weglassen und kürzer wie folgt schreiben:


```
var addieren = function (a, b) {
    var summe = a + b;
}
addieren(9, 4);
alert(summe); // -> Alert: "undefined" (!)
```

Hier gibt es ein Problem: Seltsamerweise gilt `summe` für JavaScript als `undefined`, obwohl – und das ist sehr wohl richtig! – während der Verarbeitung der Funktion das Ergebnis unserer Rechenoperation tatsächlich in eine Variable dieses Namens abgelegt wurde.

Dies ist der Knackpunkt – es war *während* der Laufzeit der Funktion, wohingegen wir die Variable *nach* der Laufzeit der Funktion auszulesen versuchen. Leider existiert die Variable zu diesem Zeitpunkt nicht mehr – wir haben ein Problem mit ihrem *Gültigkeitsbereich*, auch *Scope* genannt, bekommen.

B.1.4 Der Scope von Variablen

Grundsätzlich besitzt jede Variable einen *Scope*, also einen Gültigkeitsbereich. Bewegen wir uns aus dem Gültigkeitsbereich hinaus, ist die Variable für uns unerreichbar. Kein Problem gibt es mit Variablen, die »ebenerdig«, also außerhalb aller Strukturen angelegt wurden. Diese gelten also »global« und sind von überall erreichbar. Man bezeichnet ihren Scope als *globalen Scope*.

Globale Variablen

Im *globalen Scope* deklarierte Variablen sind stets als Eigenschaften des `window`-Objekts erreichbar:

```
var a;                // global deklariert
alert(a==window.a);  // ergibt true
```

Dies ist unabhängig davon, ob die Variable explizit mit `var` deklariert oder implizit durch Verwendung eingeführt wurde:

```
b = "Ich bin implizit deklariert!";
alert(b==window.b);    // ergibt true
```

Des Weiteren ebenfalls global sind innerhalb von Funktionen implizit deklarierte Variablen (die dort also ohne `var` eingeführt werden):

```
function beispiel() {
    // implizit deklariert, daher global
    c = "Ich bin global!";
}
beispiel();    // Deklaration von c durch Aufruf
alert(c==window.c);    // ergibt true
```

Lokale Variablen

Anders ist es, wenn eine Variable *innerhalb* einer Funktion ordnungsgemäß mit `var` deklariert wurde – sie gehört dann praktisch zum Funktionsobjekt, das seinen eigenen Scope bildet. Außerhalb der Funktion sind solche Variablen daher nicht erreichbar:

```
function beispiel() {
    // lokaler Scope der Funktion
    var x = "Ich bin lokal!";
    // lokale Variable kann ausgegeben werden:
    alert(x);
}
// zurück im globalen Scope:
alert(typeof x);    // x nicht existent; ergibt undefined
```

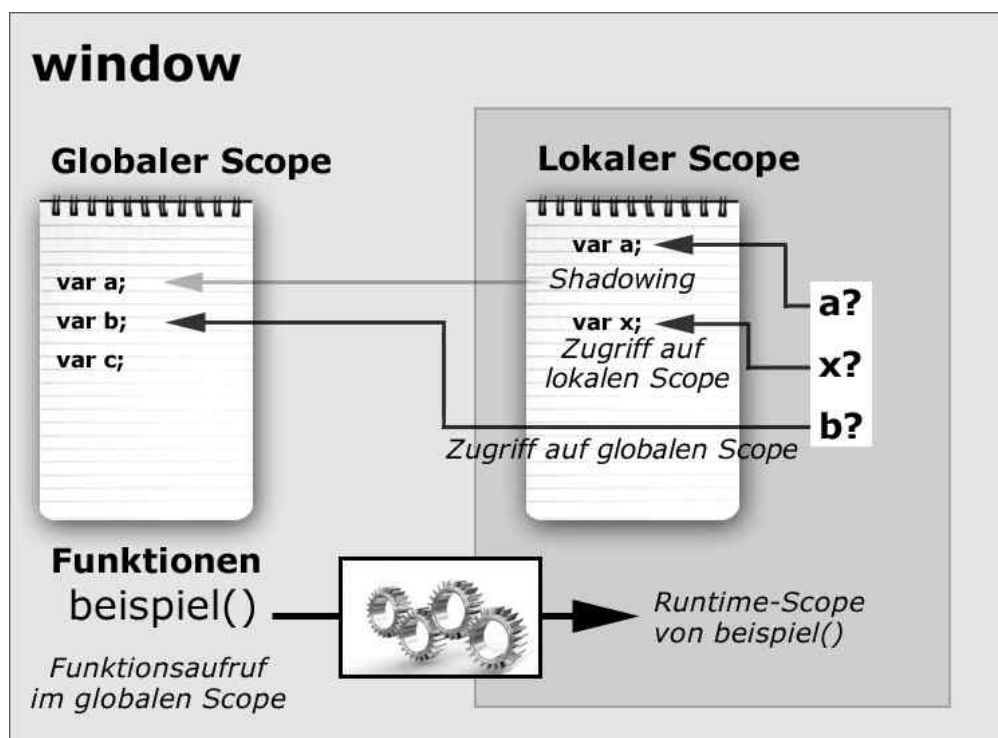


Abbildung B.1 Globaler Scope und lokaler Scope einer Funktion

Existieren im globalen und lokalen Scope gleichnamige Variablen, »verdeckt« innerhalb des lokalen Scopes die lokale Variable die gleichnamige globale Variable. Diesen Effekt bezeichnet man als *shadowing*:

```
var a = "Ich bin global deklariert!";
function beispiel() {
    // lokaler Scope der Funktion
    var a = "Ich bin lokal deklariert!";
    alert(a);    // -> "Ich bin lokal deklariert!"
```

```

}

// zurück im globalen Scope:
alert(a);           // -> Alert: "Ich bin global deklariert!"

```

Es ist allerdings durchaus möglich, außerhalb einer Funktion (d.h. scheinbar *nach* deren Laufzeit) auf eine ihrer lokalen Variablen zuzugreifen – allerdings kann dies nur über eine *innere Funktion* geschehen, mittels einer sogenannten *Closure*.

B.1.5 Closures

Wichtig ist also, dass Sie *zwei ineinander verschachtelte* Funktionen vor sich haben müssen: Über eine *Closure* kann eine innere Funktion `innen()` auf Variablen oder übergebene Parameterwerte aus dem Scope ihrer *übergeordneten* Funktion `aussen()` zugreifen – selbst wenn diese mittlerweile beendet ist. Normalerweise werden zwar alle lokalen Variablen einer Funktion nach deren Beenden gelöscht. Im Falle einer Closure werden die Variablen der äußeren Funktion nach deren Ausführung *nicht* gelöscht, sondern verbleiben im Speicher:

```

function aussen() {
    // Definiere eine lokale Variable
    var lokVar = "Ich bin ein lokaler Wert";
    // Lege eine Funktion als lokale Variable an
    var innen = function () {
        // Variable des umgebenden Scopes ist hier verfügbar:
        alert("Variable der äußeren Funktion: " + lokVar);
    };
    // Führe die eben definierte Funktion aus
    innen();
}
// Funktion gibt lokalen Wert über innere Funktion aus:
ausсен();

```

Die innere Funktion greift über eine Closure auf die Variable des umgebenden Scopes zu.

Wie definiert man eine Closure?

Eine Closure kann als »Objekt« betrachtet werden, das eine Funktion sowie deren Ausführungskontext (also deren »Variablentabelle«) beinhaltet. Wird eine Referenz auf ein Funktionsobjekt weitergegeben, behält die Funktion damit auch Zugriff auf die Variablen aus dem Kontext, in dem sie definiert wurde.

Nach Ablauf der äußeren Funktion behält die Closure den Zugriff auf deren Variable – vorausgesetzt, sie wird gespeichert und kann dadurch später ausgeführt werden.

Speichern einer Closure

Eine Möglichkeit des Speicherns besteht darin, die Closure über Event Handler an ein Element zu binden:

```
function aussen() {
    var lokVar = " Ich bin ein lokaler Wert";
    // Closure-Funktion in lokaler Variable ablegen:
    var meineClosure = function () {
        alert("Lokale Variable von aussen(): " + lokVar);
    };
    // Closure-Funktion Event-Handler zuweisen:
    document.getElementById("meinId").onclick = meineClosure;
}
// Funktion ausführen um Click-Event zu binden:
ausсен();
```

Die äußere Funktion muss einmal gelaufen sein, um den Click-Event zu binden. Das Element `#meinId` löst nun beim Anklicken die innere Funktion aus, die ihrerseits auf die Variable `lokVar` zugreift, die eigentlich nicht mehr existieren sollte. Die Referenz über die Eventbindung erzwingt jedoch die »Aufbewahrung« des Scopes in einer Closure. Die Variable bleibt erreichbar – allerdings ausschließlich auf diesem Weg (es ist nicht möglich, sie direkt anzusprechen!).

Speichern der Closure über eine anonyme Wrapper-Funktion

Closures können auch aus einer Schleife heraus gebildet werden. Hier soll der Wert von `i` zum Zeitpunkt der Bindung in einer Closure aufbewahrt werden. Diese Schleife bildet daher vier innere Funktionen, die extern über das `window`-Objekt in einer Variablen mit konstruiertem Namen abgelegt werden:

```
for(i=0; i < 4; i++) {
    (function() {
        var a = i;
        window['scope'+i] = function() {
            alert(a);
        }
    })();
}
```

Anschließend können die Funktionen aufgerufen werden, um die Closures auszuprobieren (einen praktischen Nutzen hat dieses Beispiel kaum, stellt aber ein gutes Anschauungsmaterial dar):

```
scope0(); // -> Alert: 0
scope1(); // -> Alert: 1
scope2(); // -> Alert: 2
scope3(); // -> Alert: 3
```

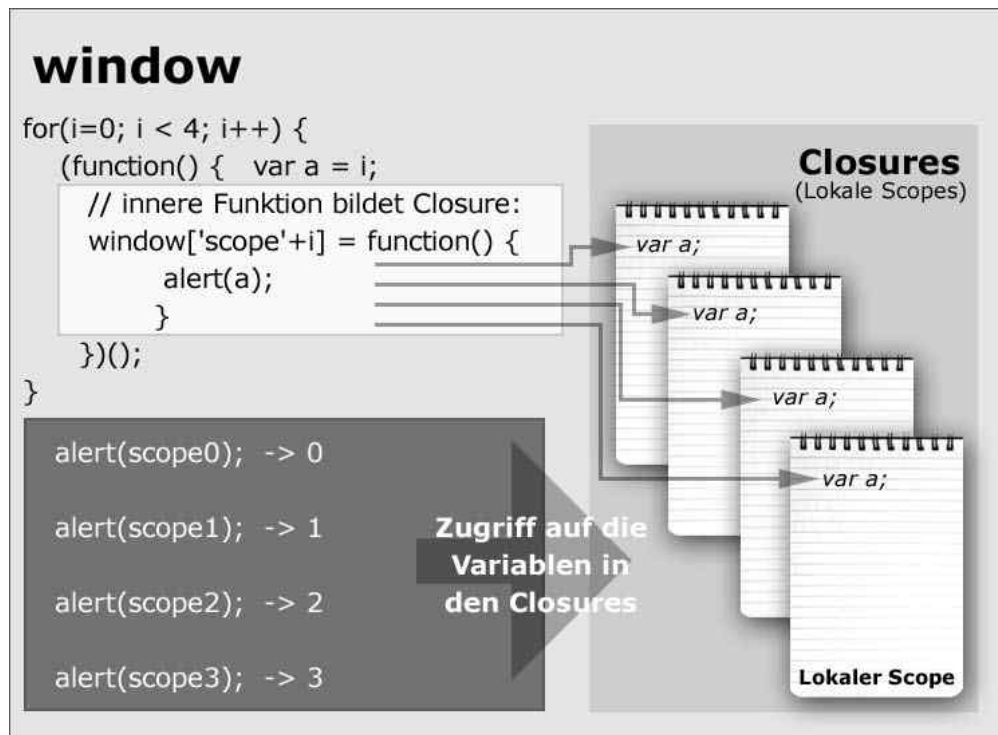


Abbildung B.2 Closures mit Wrapper-Funktion

Wie funktioniert das? Die Wirkung rührt von der anonymen Wrapper-Funktion her, die die innere Funktion umgibt und so die Closure erzeugt. Dieses Konstrukt mutet vielleicht zu merkwürdig an, um es ohne Erklärung hinzunehmen.

Kein Wrapper – keine Closure

Machen Sie ein Experiment. Anstatt an eine Variable binden Sie die innere Funktion an ein HTML-Element und lassen die Wrapper-Funktion versuchsweise weg:

```
window.onload = function() {
  var obj = document.getElementsByTagName('p');
  for(i=0; i < 4; i++) {
    var a = i;
    obj[i].onclick = function() {
      alert("Das ist Absatz " + (a + 1));
    }
  }
}
```

```

    };
  }
}

```

Im Beispieldokument befinden sich vier `<p>`-Container, die auf diesem Weg alle klickbar gemacht werden. Allerdings werden Sie bemerken, dass jeder von ihnen behauptet, »Absatz 4« zu sein. Das liegt daran, dass die gebundene Funktion stets nur auf den *letzten* Wert von `i` zugreift – nicht auf den Wert von `i` zum Zeitpunkt der Bindung. Das aber ist, was wir hier eigentlich wollen! Der Grund ist einfach – es wurde keine Closure gebildet.

Benannte Wrapper-Funktion

Modifizieren wir den Quelltext wie folgt:

```

window.onload = function() {
  var obj = document.getElementsByTagName('p');
  for(i=0; i < 4; i++) {
    function wrapper() {
      var a = i;
      // innere Funktion bildet jetzt eine Closure:
      obj[i].onclick = function() {
        alert("Das ist Absatz " + (a + 1));
      };
    }
    wrapper(); // Aufruf in jedem Schleifendurchlauf
  }
}

```

Für den zweiten Versuch wird nun die gebundene Funktion in eine Wrapper-Funktion gehüllt, also zu deren »innerer Funktion« gemacht. Dies friert den Stand von `i`, der zu jenem Zeitpunkt in die lokale Variable `a` umgespeichert wird, quasi in einer eigenen »Variablentabelle« ein (der Scope der Laufzeit der inneren Funktion).

Damit dies geschieht, muss eine *Funktionsinstanz* der äußeren Funktion `wrapper()` gestartet werden. Dies tun wir, indem wir sie direkt in der folgenden Zeile (innerhalb der Schleife) vor Ort aufrufen.

Anonymer statt benannter Wrapper

Wichtig ist zu verstehen, dass `wrapper()` hier praktisch eine »Wegwerffunktion« ist – wir brauchen sie nur, um die Closure zu erzeugen. Ihr einen Namen zu geben und sie extra aufzurufen ist eigentlich bereits »Luxus«. Dasselbe geht nämlich mit

einer namenlosen (anonymen) Funktion ebenfalls. Nur die Syntax ist etwas kurios. Tasten wir uns heran:

```
// eine benannte Funktion namens wrapper:
function wrapper() { ... }
// eine einfache, anonyme Funktion (guck mal, kein Name!):
function () { ... }
```

Jetzt, da Sie diese Funktion haben, wie rufen Sie sie auf? Ohne Namen? Sie müssen sie ja aufrufen, um eine *Funktionsinstanz* für die Closure zu bilden! Normalerweise rufen Sie eine Funktion über ihren Namen und die daran angehängten runden Klammern auf. Hier gibt es keinen Namen, daher geht dies nicht. Auch folgender Versuch scheitert:

```
// gut gemeint, aber leider ein Syntaxfehler:
function() { ... }()
```

Sie müssen die anonyme Funktion zunächst in einen *Auswertungskontext* bringen, bevor Sie sie derart auslösen können. Das ist einfach, sofern man darauf kommt – es genügt, den Funktionsausdruck in runde Klammern zu setzen:

```
// Funktionsausdruck hat Kontext, wird ausgeführt:
(function() { ... })()
```

Ersetzen Sie in unserem Szenario die benannte nun durch eine anonyme Wrapper-Funktion, gelangen Sie zu folgendem Quelltext:

```
window.onload = function() {
    var obj = document.getElementsByTagName('p');
    for(i=0; i < 4; i++) {
        (function() {
            var a = i;
            obj[i].onclick = function() {
                alert("Das ist Absatz " + (a + 1));
            };
        })();
    }
}
```

Da nun der Wert von *i* zum Zeitpunkt der Bindung in eine Closure »eingesperrt« wurde, kann er später (beim Klick auf den betreffenden Absatz) ausgelesen werden.

Die Bedeutung von Closures für jQuery

Bei *Closures* handelt es sich um ein wichtiges Prinzip, das in jQuery in vielerlei Hinsicht Bedeutung hat (im Grunde ist jQuery selbst eine riesige Closure). Es lohnt sich, das Konzept zumindest ansatzweise zu verstehen.

B.1.6 Objekte

JavaScript arbeitet mit Objekten und baut selbst im Kern auf Objekten auf, was eine wunderbare Homogenität des Gesamtbilds ergibt. Und bei Bedarf entpuppen sich auch Dinge als Objekte, die eigentlich keine zu sein scheinen. Das erleichtert den Umgang mit ihnen enorm.

Betrachten wir diesen Dreh- und Angelpunkt von allem – und damit auch vom Einsatz unseres Frameworks: Was versteht JavaScript unter einem Objekt? Wo kommen Objekte her und was macht man mit ihnen?

Objektorientierung gilt als schwieriges Thema, da es ein gewisses Maß an Abstraktion benötigt. Fangen wir mit einer saloppen Definition des Begriffs »Objekt« an. Sagen wir einfach, dass ein Objekt eine Art von »Behälter« ist. Dieser Behälter besitzt einen Namen (meistens zumindest) und enthält Daten (die wir als »Eigenschaften« bezeichnen). Außerdem enthält er Werkzeuge, die mit diesen Daten oder denen seiner Umgebung interagieren können. Diese zum Objekt gehörende Funktionalität (um nicht von vornherein das Wort »Funktionen« in den Mund zu nehmen) nennen wir »Methoden«.

Kurze Definition von Objekt

Ein Objekt ist ein so benannter Behälter für Eigenschaften und Methoden.

Wozu ist ein Objekt gut?

Stellen Sie sich den Inhalt eines Rucksacks über den Boden verstreut vor, so ist klar, dass der Rucksack (mit Inhalt darin) einfacher zu handhaben ist als die Summe der Einzelgegenstände, die er enthält. Diese müssten Sie separat zur Hand nehmen, Dinge gerieten durcheinander oder könnten vergessen werden.

Gäbe es eine vorgeschriebene »Inventarliste« (was hat im Rucksack zu sein), wäre man sicher, stets alles Benötigte dabeizuhaben. Anhand der Liste ließen sich identische Rucksäcke packen (d.h. gleichartige Objekte erstellen), falls man mehrere benötigte. JavaScript verwendet hierfür sogenannte *Konstruktoren*. Es gibt aber auch andere Wege, wie Sie zu neuen Objekten kommen können.

Ein Objekt kann in JavaScript entweder über den allgemeinen Konstruktor `Object()` oder, wirkungsgleich, als Literal `{}` erzeugt werden. Letzteres hat sich im Rahmen der Frameworks wegen der kompakteren Schreibweise als Standard durchgesetzt. Als dritte Möglichkeit kann auch ein eigener, spezialisierter Konstruktor geschrieben werden. Sehen wir uns dies alles einmal an.

Objekt per Object-Konstruktor

Mit einem einzeiligen Statement erzeugen Sie ein neues, leeres Objekt und legen es in eine Variable ab. Die rechte Seite wirkt auf den ersten Blick wie ein Funktionsaufruf (was im Grunde gar nicht so verkehrt ist). Auffällig ist das vorangestellte Schlüsselwort **new**. De facto ergibt ein Funktionsaufruf plus **new**-Schlüsselwort einen »Konstruktoraufruf«, wobei der Begriff **Object** (achten Sie auf die Schreibweise) vordefiniert ist:

```
// ein neues leeres Objekt erstellen:
var meinContainer = new Object();
```

Die Variable `meinContainer` ist jetzt keine »herkömmliche« Variable, sondern eine Variable vom »Type Object«. Dadurch ist es Ihnen gestattet, ihr Eigenschaften zu geben. Hierfür wird der Punktoperator eingesetzt. Die Namen der Eigenschaften können Sie frei wählen, wie bei Variablen. Es handelt sich quasi auch um Variablen, allerdings um solche, die diesem Objekt »gehören«:

```
// jetzt die Eigenschaften erzeugen:
meinContainer.meineEigenschaft = "wert";
```

Speichern wir in eine Objektvariable eine Funktion (eine anonyme Funktion tut hier gute Dienste), nennen wir dies eine »Methode« des Objekts.

```
meinContainer.meineMethode = function () {
    alert("Container-Eigenschaft: " + this.meineEigenschaft);
};
```

Diese Funktion ist nämlich anschließend nur im »Kontext« des Objekts, also über die Objektnotation mit Punkt, aufrufbar:

```
meinContainer.meineMethode();
```

Das Hinzufügen weiterer Methoden (oder Eigenschaften) erfolgt durch Deklaration. Hier wird dem Objekt eine zweite Methode `meinContainer.zweiteMethode` hinzugefügt, die die erste Methode aufruft. Der Bezeichner `this` bezeichnet in der Methode ebenfalls das im Kontext befindliche Objekt, also `meinContainer`.

```
meinContainer.zweiteMethode = function () {
    this.meineMethode();
};
// Aufruf:
meinContainer.zweiteMethode();
```

Nachteil dieser Methode der Objektherstellung ist, dass Sie anschließend genau eine Objektinstanz besitzen. Wollen Sie ein zweites, gleichartiges Objekt erstellen, muss die ganze Prozedur wiederholt werden.

Objekt über Literal

Wirkungsgleich zum Objektbau mit Konstruktor `Object()` ist das Schreiben eines *Objektliterals*. Mit anderen Worten, Sie schreiben das Objekt einfach hin, »wie es ist«. Um so ein neues, leeres Objekt zu erzeugen, genügt dies:

```
var meinContainer = {}
```

Nun könnten, wie eben, Eigenschaften und Methoden nachträglich hinzugefügt werden. Bei Literalen müssen Sie dies jedoch nicht tun: Sie können dies gleich »vor Ort« im Inneren der Literalklammern vornehmen. Achtung – zwischen den einzelnen Eigenschaftsdefinitionen steht ein Komma:

```
var meinContainer = {
  // Objektmembers werden im Inneren erzeugt:
  meineEigenschaft: "wert",
  meineMethode: function () {
    alert("Container-Eigenschaft: " + this.meineEigenschaft);
  }
}; // Ende des Literal-Statements
```

Der Aufruf der Objektmethode geht wie gehabt:

```
// Aufruf:
meinContainer.meineMethode();
```

Nachteil auch dieser Methode der Objektherstellung ist, dass Sie anschließend nur genau eine Objektinstanz besitzen. Wenn Sie nicht mehr als das benötigen, reicht dies (oft ist es so). Falls aber nicht, brauchen Sie eine maßgeschneiderte *Konstruktorfunktion*. Schauen wir uns erst, um nicht vorgreifen zu müssen, bei den Funktionen um.

B.1.7 Konstruktorfunktionen für Objekte

Nehmen wir an, dass wir Objekte wie das vorhin vorgestellte »konkrete« Objekt `meinContainer` erzeugen wollen (konkret bedeutet, dass die abstrakten Eigenschaften konkret für dieses eine Objekt festgelegt sind – man nennt dies eine *Instanz* dieses Objekttyps). Hier der äquivalente Literalausdruck:

```
var meinContainer = {
  // Objektmembers werden im Inneren erzeugt:
  meineEigenschaft: "wert",
  meineMethode: function () {
    alert("Container-Eigenschaft: " + this.meineEigenschaft);
  }
}; // Ende des Literal-Statements
```

Sie brauchen nun eine Möglichkeit, analoge Objektinstanzen zu erzeugen. Von der bereits gezeigten sollen Sie sich nicht in der Struktur unterscheiden, sollen also über gleiche (und gleich benannte) Eigenschaften und Methoden verfügen. Nur die konkreten *Werte* der Eigenschaften und natürlich der *Name* der Objektinstanz sollen anders sein.

Der Gedanke liegt nahe, eine Funktion mit der Erstellung eines Objekts zu betrauen und dieser Funktion die benötigten konkreten Werte als Parameter zu übergeben. Im Inneren der Funktion wird der übergebene Wert der Eigenschaft zugeordnet. Nehmen wir (als Hilfsvorstellung) an, dass wieder als erster Schritt ein neues, leeres Objekt erstellt wird (obwohl dies in der Funktion nicht explizit angeordnet wird). Dieses Objekt ist dann automatisch Kontextobjekt der Operation. Sie können sich also mit `this` darauf beziehen, was Sie verwenden, um Wert und Eigenschaft (die ja dem Objekt gehören sollen, nicht der Funktion) zusammenzubringen. Dies ergibt etwa Folgendes (die Methode lassen wir vorerst weg):

```
function MachContainer(wert) {
  // übergebener Wert wert landet in meineEigenschaft:
  this.meineEigenschaft = wert;
}
```

Bezeichner einer Konstruktorfunktion

Sie haben bemerkt, dass wir den Funktionsnamen des Konstruktors mit einem *Großbuchstaben* begonnen haben. Dies ist zwar nicht unbedingt nötig, aber es handelt sich um eine Konvention, die sich zur Bezeichnung von Klassen (und eben Konstruktoren) eingebürgert hat.

Kommen wir zum Hattrick der Angelegenheit – dem Aufruf des Konstruktors. Dies darf in der Tat *nicht* wie folgt geschehen (à la »normale« Funktion):

```
MachContainer("neuer Wert");
```

Warum? Weil `this` im Inneren der Funktion in diesem Fall leider auf das globale Objekt zeigt. Was dann passiert, ist, dass Sie damit im globalen Kontext (d.h. im `window`-Objekt) eine Variable namens `meineEigenschaft` erzeugen. Schlecht. Ein neues Objekt käme ohnehin nicht dabei heraus, weil die Funktion keinen definierten Rückgabewert hat:

```
// die Variable bleibt leer:
var leiderKeinObjekt = MachContainer("neuer Wert");
```

Aus diesem Grund muss der Aufruf eines Konstruktors mit vorangestelltem Schlüsselwort `new` erfolgen. Was bewirkt `new`? Ganz einfach – dies *ist* nun die An-

ordnung, zunächst ein neues, leeres Objekt zu erzeugen und es als Kontextobjekt zu verwenden. Bingo:

```
var neuerContainer = new MachContainer("neuer Wert");
```

Fügen Sie nun eine Methode hinzu. Hierfür gibt es zwei Möglichkeiten. Erstens könnten Sie den Konstruktor erweitern. Kein Problem, solange die Funktion, die der Methode hinterlegt ist, knapp genug ist. Ansonsten »sprengen« Sie die Übersichtlichkeit des Konstruktors möglicherweise:

```
function MachContainer(wert) {
  this.meineEigenschaft = wert;
  this.meineMethode = function () {
    alert("Container-Eigenschaft: "+this.meineEigenschaft);
  }
}
var neuerContainer = new MachContainer("neuer Wert");
neuerContainer.meineMethode(); // -> Alert: "neuer Wert"
```

Okay, das geht also. Beachten Sie, dass das `this` im Inneren der Methode tatsächlich nicht auf die Methode (als Funktionsobjekt), sondern auf das erstellte Objekt Bezug nimmt. So soll es auch sein.

Prototyping und die Prototypkette

Wenn die Methoden umfangreicher sind oder nachträglich hinzugefügt werden müssen, bietet sich ein anderer Weg (den viele Programmierer ohnehin bevorzugen – schon aus Prinzip sollten wir uns das daher einmal ansehen). Dieser Weg wird als *Prototyping* bezeichnet.

Erinnern Sie sich, dass am Anfang gesagt wurde, JavaScript sei eine »prototypisierte«, objektorientierte Sprache? Es handelt sich hier um ein grundlegendes Konzept der Sprache² – die Erweiterbarkeit von Objekten und Objekttypen.

Gehen wir nochmals vom einfachen Konstruktor aus, den wir durch *Prototyping* erweitern werden. Der Konstruktor sah so aus:

```
function MachContainer(wert) {
  this.meineEigenschaft = wert;
}
```

Lassen wir langwierige Erklärungen beiseite, erweitern wir ihn einfach:

2 Der Name des Frameworks »Prototype« (www.prototypejs.org) leitet sich ebenfalls hiervon ab.

```
MachContainer.prototype.meineMethode = function () {
    alert("Container-Eigenschaft: " + this.meineEigenschaft);
}
```

Der Aufruf des Konstruktors und der anschließende Ruf der Methode klappen erneut:

```
var neuerContainer = new MachContainer("neuer Wert");
neuerContainer.meineMethode(); // -> Alert: "neuer Wert"
```

Warum ist dies nun besser? Und wie funktioniert es überhaupt? Besser ist es, weil Sie so nachträglich – und zwar jederzeit! – die Konstrukturfunktion um Methoden erweitern können. Ob mit diesem Konstruktor bereits Objektinstanzen erzeugt wurden, ist dabei egal – die Erweiterung bezieht sich automatisch auch auf bereits existierende Objekte dieses Typs.

Schwieriger ist die Erklärung, *wie* es funktioniert. Nehmen wir an, jedes Objekt »erinnert« sich daran, wer (oder was) es konstruiert hat. Nehmen wir weiter an, dass das Objekt sich »an der Quelle« informiert, ob sich an den Konstruktionsvorschriften etwas geändert hat. Beides stimmt so weit auch – jedes Objekt besitzt eine Eigenschaft namens `constructor`, in dem eine Referenz auf »seine« Konstrukturfunktion gespeichert ist.

Wenn Sie den Namen des Konstruktors erfahren wollen, schreiben Sie:

```
alert(neuerContainer.constructor.name);
```

Vom Konstruktor zur Instanz existiert ebenfalls eine Verbindung, und zwar über die sogenannte *Prototype-Chain*. Dies führt dazu, dass jede Instanz von Änderungen am Konstruktor erfährt und diesem nachträglich hinzugefügte Methoden unmittelbar selbst nutzen kann. Zuvor wurde eine Eigenschaft namens `prototype` eingesetzt, um an den Konstruktor eine Methode zu binden.

Woher kommt diese Eigenschaft `prototype` eigentlich? Aus dem Ärmel geschüttelt, weil es grade opportun war? Nein – diese Eigenschaft ist bereits im Stammobjekttyp `Object` enthalten! Weil sich von diesem alles ableitet, existiert `prototype` auch in jedem anderen Objekt, also auch in `MachContainer`.

Nun besitzt jedes Objekt, analog zur `constructor`-Eigenschaft, auch eine Referenz auf die `prototype`-Eigenschaft seines Konstruktors. Da sie eigentlich nicht dem Objekt gehört, sondern dem Konstruktor, erreichen Sie sie vom Objekt aus nur indirekt:

```
// die prototype-Eigenschaft ist ein Objekt
alert(typeof neuerContainer.constructor.prototype);
```

Schreiben wir das Prototyping einmal um, damit der Objektcharakter der Eigenschaft etwas deutlicher wird, und erzeugen gleich eine zweite Methode:

```
MachContainer.prototype = {
  meineMethode: function() {
    alert("Container-Eigenschaft: "+this.meineEigenschaft);
  },
  andereMethode: function() {
    alert("Ich bin die andere Methode!");
  }
}
```

Das Manöver geschieht also in der `prototype`-Eigenschaft des Konstruktors, den dieser von `Object` geerbt hat. Was passiert dabei nun in der »betroffenen« Objektinstanz? Die Antwort ist einfach: gar nichts.

Das ist auch nicht nötig, da zwischen der Objektinstanz und der `prototype`-Eigenschaft des Konstruktors sozusagen eine »stehende Verbindung« vorliegt. Wird eine Objektinstanz um die Ausführung einer Methode »gebeten«, nimmt sie über diese `prototype`-Verbindung Kontakt mit dem Konstruktor auf und ruft die benötigte Methode von dort ab. Man sagt, die Instanz »erbt« die Methoden über die Prototypkette.

Die Bedeutung des Prototypings für jQuery

Unmittelbar bekommen wir es im Rahmen von jQuery mit *Prototyping* zu tun, sobald wir die Funktionalität des Frameworks um eigene Methoden erweitern. jQuery bezeichnet dies als ein *Plugin*. Zur Anwendung ist es zum Glück nicht unbedingt erforderlich, das Prinzip in allen Konsequenzen zu verstehen.

B.1.8 Funktionen als Objekte

Vorhin wurde bereits gesagt, dass es sich bei Funktionen um Objekte handelt. Ebenso wie andere Objekte, auch das haben wir gesehen, können wir eine Funktion in eine Variable abspeichern, mit anderen Worten, wie einen normalen Wert behandeln.

Callbacks – Funktionsobjekte als übergebene Werte

Wir können Funktionen demnach auch »herumreichen«, ebenso, wie wir dies mit anderen Werten auch machen. Ein Beispiel hierfür ist die globale Helferfunktion `setTimeout()`, die ein Funktionsobjekt als Parameter entgegennimmt:

```
var halloSagen = function() {
  alert("Es reicht. Ich habe lange genug gewartet.");
}
```

```

}
setTimeout(halloSagen, 5000);

```

Sie können das Funktionsobjekt auch unmittelbar übergeben, anstatt es umständlich in eine Variable abzulegen (dies macht nur dann Sinn, wenn Sie die Funktion anderenorts ein weiteres Mal brauchen):

```

setTimeout(function() {
    alert("Es reicht. Ich habe lange genug gewartet.");
}, 5000);

```

Das gleiche Prinzip gilt, wenn Sie eine Funktion an ein Ereignis binden:

```

var meldung = function() {
    alert("Die Seite ist jetzt fertig geladen.");
}
window.onload = meldung;

```

Analog dazu kann das Funktionsobjekt direkt übergeben werden:

```

window.onload = function() {
    alert("Die Seite ist jetzt fertig geladen.");
}

```

Eine Funktion, die an ein Ereignis gebunden ist (egal, auf welche Art) bezeichnen wir als eine *Callback-Funktion*.

Bedeutung von Callback-Funktionen in jQuery

Wir werden sehen, dass *Callbacks* für jQuery sehr wichtig sind – sie werden beispielsweise regelmäßig in Zusammenhang mit Ajax eingesetzt (»Sind die Daten eingetroffen? Dann tu etwas!«), aber auch bei weniger »abgehobenen« Gelegenheiten, wie dem Anklicken von Elementen (»Wurde etwas angeklickt? Dann tu dies!«) oder der Steuerung von Animationen (»Hat das Objekt seine Endposition erreicht? Dann reagiere entsprechend!«).

Methoden von Funktionsobjekten – call() und apply()

Wenn wir uns damit abgefunden haben, dass eine Funktion ein Objekt darstellt, dann mag es weniger verblüffend sein, dass es auch Methoden von Funktionsobjekten gibt – wir betrachten zwei davon, nämlich `call()` und `apply()`.

Bevor wir damit herausrücken, wozu diese benötigt werden, betrachten wir noch einmal den *Kontext*, in dem eine Funktion gilt. Für eine global deklarierte Funktion (alles, was nicht einem Objekt gehört, also eine Methode ist, und alles, was nicht eine »innere Funktion« darstellt, also einem anderen Funktionsobjekt gehört) ist der Kontext stets das `window`-Objekt. Der Kontext bestimmt die Bedeu-

tung des `this`-Objekts. Meist ist `this` also `window`. Will man das immer so haben? Sicherlich nicht.

Erstellen wir drei Dinge – eine globale Variable, ein Objekt und eine Funktion im globalen Kontext:

```
var name = "das window-Objekt";
var person = { name: "Herr Müller" };
function hallo() {
  alert("Hi, ich bin " + this.name);
}
```

Setzen wir die Funktion einmal in Betrieb:

```
hallo(); // Alert: "Hi, ich bin das window-Objekt"
```

Das ist nicht verwunderlich, weil ja der Kontext während der Ausführung der Funktion (es ist ja eine im globalen Scope) auf dem `window`-Objekt liegt. Wenn wir das aber nicht wollen? Dann nehmen wir `call()` zu Hilfe. Und das geht so:

```
hallo.call(person); // Alert: "Hi, ich bin Herr Müller"
```

Die globale Funktion `hallo()` verhält sich mithilfe von `call()` so, als ob sie eine Methode des Objekts `person` wäre, die ihr als Parameter übergeben worden ist. Die »Funktionsmethode« `call()` verschiebt also den Kontext hin zu dem ihr übergebenen Objekt. Analog arbeitet `apply()`:

```
hallo.apply(person); // Alert: "Hi, ich bin Herr Müller"
```

Der Unterschied zwischen `call()` und `apply()` kommt erst zum Tragen, wenn weitere Parameter übergeben werden. Während `call()` zusätzlich eine *Liste aus Einzelargumenten* akzeptiert, möchte `apply()` stattdessen ein *Array*, das diese weiteren Parameter enthält. (Wir werden das an dieser Stelle nicht weiter vertiefen, da dies im Rahmen des Buches nicht erforderlich ist. Wir empfehlen bei Interesse einen Blick in die Spezifikationen bei developer.mozilla.org).

Bedeutung von `call()` und `apply()` für jQuery

Wir werden beide Methoden selten direkt anwenden müssen, wenn wir mit jQuery arbeiten. Es mag aber interessant sein, zu wissen, dass jQuery selbst sie stets dann einsetzt, wenn der Verarbeitungskontext auf ein bestimmtes Objekt zu setzen ist.

B.1.9 »Unobtrusive« JavaScript

Ziel der zeitgenössischen JavaScript-Entwicklung ist es, sogenanntes *unobtrusive JavaScript* zu schreiben. Hierunter versteht man, dass der Entwickler es einerseits vermeidet, den HTML-Code mit JavaScript-Elementen (wie beispielsweise Event-

Handler-Attributen, Funktionsaufrufen oder Inline-Scripts) zu vermischen, andererseits dafür Sorge trägt, dass innerhalb von komplexen Anwendungen (die beispielsweise Bibliotheken einbinden) keine Konflikte zwischen Variablen- und Objektbezeichnern auftreten.

Definition »unobtrusive«:

dezent, unaufdringlich, unauffällig, zurückhaltend (<http://dict.leo.org>)

Keine Vermischung von HTML und JavaScript:

Diesem ersten Aspekt wird durch *DOM-Manipulation* des (fertig geladenen) HTML-Dokuments Rechnung getragen, indem nachträglich Event Listener an die gewünschten Elemente gebunden werden. Hierfür setzen Sie zum Auffinden der Zielelemente die im Rahmen des DOM angebotenen Funktionen (`getElementById`, `getElementsByTagName`) ein und binden über weitere DOM-Funktionen den gewünschten Event. Voraussetzung für den Erfolg ist die Überprüfbarkeit des Ladezustands des Dokuments (hierzu existiert eine Reihe von Ansätzen; dazu erfahren Sie später noch mehr).

jQuery und »unobtrusive« JavaScript

Mit jQuery können Sie dieses erste Konzept sehr leicht unterstützen. Wie Sie sehen werden, können Sie innerhalb einer JavaScript-Datei mithilfe jQuery ein bestimmtes HTML-Element ansprechen und ein Event daran binden.

Konfliktvermeidung zwischen benannten Objekten und Variablen

Konflikte zwischen gleich benannten Variablen ergeben sich, sobald sich deren *Gültigkeitsbereiche* (*Scope*) überschneiden. Dies geschieht vor allem bei global gültigen Variablen leicht, da sich deren Scope über die Laufzeit des gesamten Scripts erstreckt. Sobald Skripte aus mehreren Quellen gleichzeitig eingesetzt werden, wird es an dieser Front unübersichtlich.

Konfliktpotenzial bei globalen Variablen

Nicht immer ist es möglich, die Bezeichner aller eingesetzten Variablen im Vorfeld zu kennen (dies betrifft oft von extern eingebundene Bibliotheken), um Konflikten vorzubeugen. Die Lösung besteht darin, globale Variablendeklarationen nach Möglichkeit grundsätzlich zu vermeiden.

Namensräume zur Konfliktvermeidung

Ein Ansatz (im Prinzip eine Folgerung aus der vorher gestellten Prämisse) besteht in der Verwendung sogenannter *Namensräume*, indem Variablen, Objekte (Funk-

tionen also in Form von Methoden) einem benannten Objekt unterstellt werden, also nur »über« das Objekt erreichbar sind. Der Objektbezeichner übernimmt somit die Aufgabe eines Namensraums (eigentlich ein »benannter Scope«). Dieser Ansatz ist in *jQuery* ebenfalls wiederzufinden.

Ein anderer Ansatz, den globalen Scope von Variablen zu vermeiden, ist deren Deklaration innerhalb von (auch anonymen) Funktionen. Einige Frameworks (hier ist wieder *jQuery* zu nennen) bestehen deshalb im Grunde aus einer einzigen Funktionsdeklaration. In dieser Funktionsdeklaration ist es möglich, Variablen, Funktionen (hier spricht man von »inneren« Funktionen) und Objekte zu deklarieren und dabei eine fast beliebige Schachtelungstiefe zu erzielen. Hierbei ist es stets möglich, von einer weiter innen befindlichen Funktion auf die Variablen dieser direkt umgebenden Funktionen zuzugreifen, sodass die äußerste Funktion »scopetechnisch« die Rolle der globalen Ebene übernehmen kann (ohne natürlich wirklich global zu sein).

Erstellen wir probeweise einen Namensraum, zunächst als Objekt. Dieses wird in eine globale Variable gespeichert:

```
var meinNamensraum = {
  wert1 : 1,
  wert2 : 2,
  methode1 : function() {
    alert("Ich komme aus dem Namensraum!");
  }
};
```

Dies kapselt Eigenschaften und Methoden in das Objekt, die in Form namensraumgebundener Variablen und Funktionen eingesetzt werden können:

```
alert(meinNamensraum.wert1);
meinNamensraum.methode1();
```

Diese Methode ist simpel, aber nicht sehr elegant. Interessant und obendrein ausbaufähig ist die Variante, dies alles mithilfe eines automatisch aufgerufenen Funktionsobjekts vorzunehmen, das ein innen erzeugtes Objekt nach außen über `return` zurückgibt:

```
var meinNamensraum = function() {
  // hier beginnt das Namensraumobjekt:
  return {
    wert1 : 1,
    wert2 : 2,
    methode1 : function() {
      alert("Ich komme aus dem Namensraum!");
    }
  }
};
```

```
};
})();
```

Interessant ist dies, weil so die Möglichkeit besteht, zusätzlich in der Funktion weitere Variablen und Funktionen zu deklarieren. Diese sind von außen nicht zugänglich, also »privat«. Ihre Werte können aber über die inneren Funktionen erreicht werden – richtig, hier liegt eine Closure vor ...

```
var meinNamensraum = function() {

    var privaterWert = "geheim...";

    // hier beginnt das Namensraumobjekt
    return {
        wert1 : 1,
        wert2 : 2,
        methode1 : function() {
            alert("Ich komme aus dem Namensraum!");
        },
        getPrivaterWert : function() {
            return privaterWert;
        }
    };
}();
```

Auf diesem Weg ist es möglich, weitere Werte oder Methoden im Funktionsobjekt zu deponieren, die zwar nicht selbst Teil des Namensraums sind, über diesen aber (mittels Closures) zugänglich gemacht werden können – wie hier einfach demonstriert über *Getter-* und *Setter-Methoden*. Der Wert der lokalen Variable `privaterWert` des Funktionsobjekts ist von außen nicht erreichbar, kann aber über die Namensraummethode `getPrivaterWert()` ausgelesen werden. Analog ist eine Methode denkbar, die den Wert neu setzen könnte:

```
// dies und ähnliches geht nicht:
alert(meinNamensraum.privaterWert); // -> undefined
// geht:
meinNamensraum.getPrivaterWert(); // gibt Wert aus
```

Die Bedeutung von Namensräumen für jQuery

jQuery arbeitet direkt mit einem Namensraum, der den Namen `jQuery` besitzt. Alle jQuery-Eigenschaften und Methoden sind daher diesem Namensraum unterstellt. Alias für jQuery ist der Bezeichner `$`. (Zur Kollisionsvermeidung prüft jQuery bei der Initialisierung, ob `$` bereits durch eine andere Anwendung belegt ist und reagiert entsprechend.)

B.2 Die Synthese – das Document Object Model

Ein wichtiges Konzept, mit dem wir im Laufe dieses Buchs gearbeitet haben, ist das *Document Object Model* (DOM). Das DOM stellt die Schnittstelle zwischen der Dokumentstruktur und dem Rest dar und dient Ihnen einerseits zur Verknüpfung von HTML und CSS, andererseits ist es auch das Interface, das Ihnen erlaubt, per Script auf Struktur und Präsentation des Dokuments einzuwirken.

B.2.1 Das Erstellen des DOM-Baums

Das DOM stellt eine Abstraktion des Dokuments dar, die der Browser beim Einlesen des HTML-Quelltexts im Arbeitsspeicher erzeugt. Eigentlich existiert es also rein virtuell. Stellen Sie es sich als Baumstruktur vor, und zwar in Form eines nach unten hängenden Baums, der an einem Punkt aufgehängt ist und sich ab dort verzweigt. Bezeichnen wir den Punkt, an dem der Baum ansetzt, als *Dokumentknoten* (*documentNode*). Dieser Dokumentknoten besitzt keine Entsprechung im Dokument, sondern stellt das »Dokument an sich« dar.³

Einen solchen Dokumentknoten erzeugt der Browser beim Einlesen (*Parsing*) des Dokuments. Er hängt dann für jedes Element, das er in Quelltextreihenfolge antrifft, einen weiteren Knoten unten an diesen Dokumentknoten an. Das erste Element, das er antrifft, ist stets das Root-Element HTML (allgemein als *document-Element* bezeichnet). Ab hier spaltet sich der Baum in zwei Zweige, die den `<head>` und den `<body>` repräsentieren (siehe Abbildung B.3).

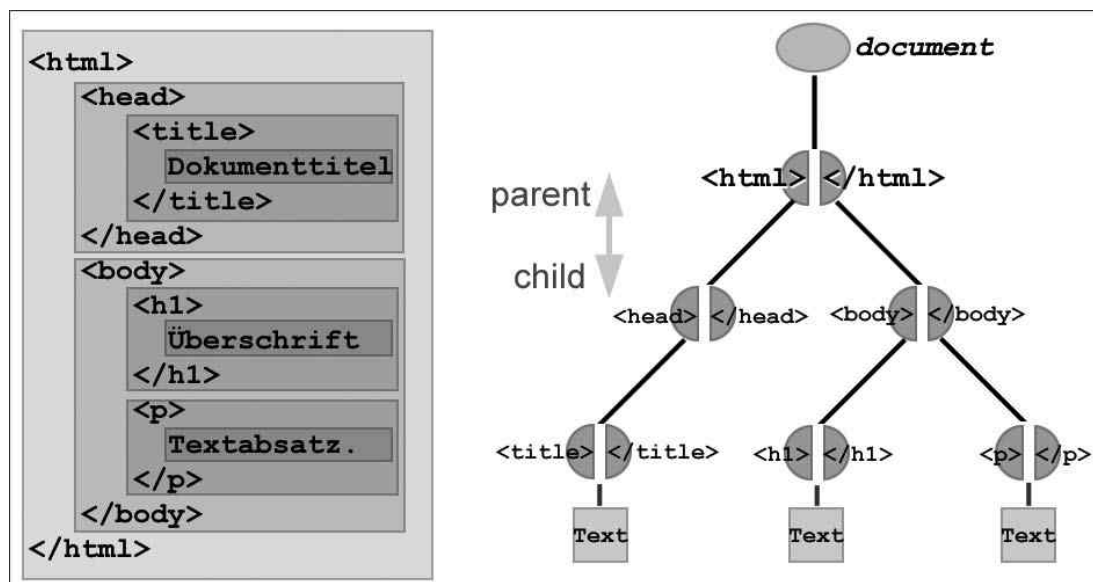


Abbildung B.3 Der HTML-Quelltext und seine DOM-Repräsentanz

³ Aus diesem Grund sind auch »leere« Dokumentknoten möglich, auch wenn dies im ersten Moment sinnlos erscheint.

Der Baum besteht also aus Verbindungen und Knoten. Im Falle eines *Elementknotens* fallen in diesem Start- und Endmarke des Quelltexts zusammen. Textknoten bilden stets das Ende eines Zweiges (»leaf nodes«). Enthält ein Element einen Inhalt (gehen wir also hierarchisch in dessen Inneres), fächert sich der Baum weiter nach unten auf. Hierdurch entsteht für jeden Bestandteil dieses Inhalts ein weiterer Ast mit daran hängenden Knoten.

Hierbei gilt der oben liegende Knoten als »Elternknoten« (»parent«), die von ihm unmittelbar abstammenden Knoten als »Kindknoten« (»children«). Sowohl Elemente als auch Textknoten stehen stets in einer Eltern-Kind-Beziehung, wobei jeder Knoten genau einen Elternknoten besitzt (niemals mehrere). Ein Elementknoten besitzt darüber hinaus weitere Eigenschaften, zu denen (salopp gesprochen) auch seine Attribute gehören. Der Browser baut auf diese Weise sukzessive ein komplettes Abbild der hierarchischen Struktur des Dokuments auf.

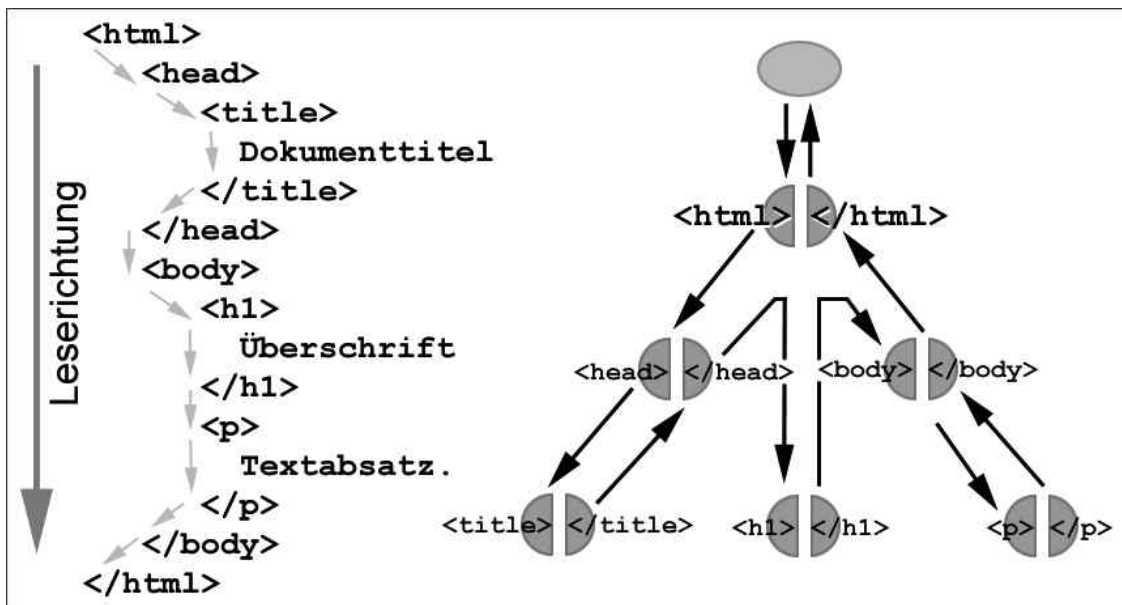


Abbildung B.4 Traversieren des Baums während des Parsing-Vorgangs

Das DOM ist also eine Abstraktion folgender Information:

- ▶ Wie ist der hierarchische Zusammenhang in der Dokumentstruktur?
- ▶ Welches Element ist an welcher Position der Hierarchie?
- ▶ Welche Eigenschaften hat es (z. B. Attribute, Inhalte, Nachbarelemente)?

Hier soll kurz ein Vorteil des DOM-Konzepts gegenüber »seriellem« Quelltext erwähnt werden, der darauf beruht, dass ein Knoten gleichzeitig Start- und Endmarke eines HTML-Tags repräsentiert: Bei einer Manipulation des Baums (wir werden gleich sehen wie das geht), also der Entnahme oder dem Hinzufügen einzelner Knoten oder ganzer Zweige, wird stets mit vollständigen Strukturen gear-

beitet. Ein Dokument behält so stets seine *Wohlgeformtheit*, also die Art von regelmäßiger Struktur, die in der XML-Datenverarbeitung gefordert ist.

B.2.2 Das »Schmücken« des DOM-Baums

Dieses Modell, in dem alle Elemente des Dokuments repräsentiert sind, ist noch eine Abbildung der »nackten« Struktur – durchaus wörtlich zu nehmen, denn bis jetzt hat der Browser die CSS-Informationen noch nicht eingebracht. Die nun folgende Phase wird als »decorating the tree« bezeichnet: Der Browser liest alle CSS-Informationen ein und löst dabei auftretende Konflikte und Unstimmigkeiten auf. Anschließend liegt das sogenannte *Stylesheet* vor: die vollständigen Präsentationsvorschriften, die auf das Dokument angewendet werden sollen.

Nun werden die Selektoren aller CSS-Regeln registriert und der Dokumentbaum anhand dieser Vergleichsmuster durchsucht. Jeder Elementknoten, auf den das Muster zutrifft, bekommt den Regelsatz zugewiesen. Auf diesem Weg werden alle Regeln durchgearbeitet. Regeln, für die keine Übereinstimmung mit dem Dokument gefunden wird, liegen brach (siehe Abbildung B.5).

Bekommt ein Element hierbei mehrfach Regeln zugewiesen, addieren sich diese. Ein dabei auftretender Konflikt wird nach Rang der Regel beigelegt. Sobald der Vorgang beendet ist, ist der Baum »dekoriert«, und der Browser geht daran, ihn im Viewport darzustellen.

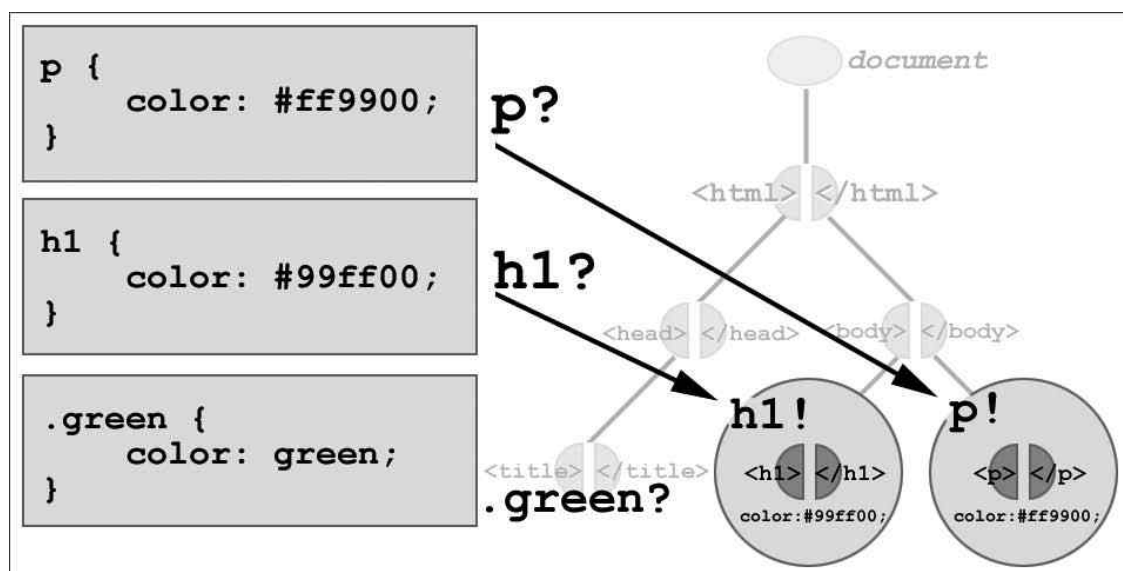


Abbildung B.5 Der DOM-Baum wird per Vergleichsmuster durchsucht.

Im dekorierten Baum sind folgende Informationen zusätzlich in den einzelnen Elementknoten gespeichert:

- Welche Präsentationsvorschriften existieren für dieses Element?

Die Stylevorschriften können wir uns durchaus als »Eigenschaften« des Elements vorstellen, die beim Elementknoten gelagert werden. Wir werden gleich sehen, dass dies uns hilft, die CSS-Regeln eines Elements auszulesen und auch bei Bedarf zu ändern. In diesem Augenblick kommt wieder JavaScript ins Spiel.

B.2.3 Manipulation von DOM und CSS per JavaScript

Der dekorierte Baum stellt, wie wir jetzt wissen, ein Abbild des Dokuments im Arbeitsspeicher des Browsers dar, das der Darstellung des Dokuments im Viewport entspricht. Da es sich um eine rein virtuelle Sache handelt, ist dieses Abbild per Programmierung beliebig manipulierbar. Genau dies ist überhaupt die Aufgabe des DOM – eine Schnittstelle (API) zu bieten, die es ermöglicht, mittels einer Programmiersprache auf das Dokument einzuwirken.

In Zusammenhang mit JavaScript bietet das DOM eine Reihe von Schnittstellenfunktionen, die eine Brücke schlagen zwischen der Scripting-Umgebung und dem Dokumentbaum. Einige dieser Methoden sind dem Dokumentknoten (den wir hierfür praktischerweise einfach als JavaScript-Objekt betrachten) unterstellt. Andere Methoden stehen auch direkt den Elementknoten zur Verfügung.

Diese DOM-Methoden sind nicht ausgesprochen zahlreich und zum Teil umständlich anzuwenden – ihre Anzahl hängt zudem von der Implementierung des DOM ab (»DOM-Level«), die der ausführende Browser jeweils unterstützt.

Dies sind ihre Aufgaben:

- Selektieren von Knoten
- Traversieren (Bewegung) innerhalb des Baums
- Wert (Inhalt) eines Knotens auslesen oder schreiben
- Attribute eines Elementknotens lesen oder schreiben
- Erzeugen von Element- und Attributknoten
- Einhängen von Knoten in den Baum
- Löschen von Knoten im Baum

DOM-Methode	Erläuterung
<code>addEventListener()</code>	Bindet Event Handler an DOM-Element.
<code>removeEventListener()</code>	Löst Event Handler an DOM-Element.
<code>createAttribute()</code>	Erzeugt einen Attributknoten.

Tabelle B.1 Die wichtigsten DOM-Methoden zur DOM-Manipulation

DOM-Methode	Erläuterung
<code>createTextNode()</code>	Erzeugt einen Textknoten.
<code>createElement()</code>	Erzeugt einen Elementknoten.
<code>getAttribute()</code>	Liest einen Attributwert.
<code>setAttribute()</code>	Schreibt einen Attributwert.
<code>removeAttribute()</code>	Entfernt einen Attributknoten.
<code>appendChild()</code>	Hängt Elementknoten am Ende des Inhalts des aktuellen Knotens ein.
<code>insertBefore()</code>	Hängt Elementknoten vor dem aktuellen Knoten ein.
<code>removeChild()</code>	Entfernt Kindknoten des aktuellen Knotens.
<code>replaceChild()</code>	Ersetzt Kindknoten des aktuellen Knotens.
<code>getElementById()</code>	Referenziert einen Elementknoten per ID.
<code>getElementsByName()</code>	Erstellt NodeList aus Elementknoten nach CSS-Klasse.
<code>getElementsByTagName()</code>	Erstellt NodeList aus Elementknoten nach Tag-Bezeichner.

Tabelle B.1 Die wichtigsten DOM-Methoden zur DOM-Manipulation (Forts.)

Auf die meisten dieser DOM-Methoden (es gibt jedoch eine Reihe mehr) wird in Kapitel 4, »jQuery – die Übersicht«, knapp eingegangen, sofern erforderlich.

An dieser Stelle sollten die Ausführungen zu JavaScript und zu DOM genügen. Falls nicht – eine umfassende *Onlinereferenz zu DOM* (in englischer Sprache) finden Sie bei der Mozilla Foundation unter dieser Adresse:

<https://developer.mozilla.org/en/DOM>

C Inhalt der DVD-ROM

Auf der beiliegenden DVD finden Sie eine Reihe nützlicher Tools sowie die Beispieldateien des Buches. Die Bilder, die für die Galeriebeispiele verwendet wurden, unterliegen dem Urheberrecht, und dürfen nur mit Genehmigung des Autors online wie offline für kommerzielle Zwecke verwendet werden.

C.1 Verzeichnis Listings

Auf der beiliegenden DVD finden Sie die Beispielcodes, geordnet nach Kapiteln und ihren Listingunterschriften. Die Beispiele finden Sie auf der DVD zusätzlich auch nach Themen (Navigationen, Formulare etc.) geordnet.

C.2 Verzeichnis jQuery

Die zur Zeit der Buchproduktion aktuelle Version des jQuery-Frameworks finden Sie ebenso im Verzeichnis *jQuery*, wie eine komplette Ausgabe des jQuery-UI-Frameworks. Es wurde mit dem UI-Lightness Theme heruntergeladen. Sämtliche Plugins, die im Buch besprochen werden, liegen zusätzlich gepackt im Verzeichnis *jQuery/Plugins*.

Es gibt noch eine unendlich große Zahl an jQuery Plugins, die es zwar wert sind, besprochen zu werden, aber nicht den Weg in das Buch geschafft haben. Eine kleine Auswahl zusätzlicher Plugins finden Sie deshalb im Verzeichnis *jQuery/Plugins/mehr*.

C.3 Verzeichnis Software

Im Verzeichnis *Software* finden Sie eine Version des *Google Closure Compilers*, mit dem Sie eigene Plugins minifizieren können. Damit Sie gleich komfortabel alle Beispiele ausprobieren können, sind einige Editoren und IDEs entweder als Vollversionen oder Demoversionen beigelegt. Sie finden Aptana Studio für Mac OS X und Windows, notepad++ für Windows sowie Adobe Dreamweaver als Demoversion für Mac OS X und Windows, ebenfalls im Verzeichnis *Software*.

Index

- \$()-Funktion 59
 - Argument Callback* 65
 - Argument CSS-Selektor* 66
 - Argument CSS-Selektor mit Kontext* 66
 - Argument DOM-Collection* 62
 - Argument DOM-Knoten* 62
 - Argument HTML-String* 63
 - Argument HTML-String, owner-Document* 65
 - Argument jQuery-Objekt* 62
 - \$.ajax 226
 - \$.ajax, async 232
 - \$.ajax, beforeSend 229
 - \$.ajax, cache 232
 - \$.ajax, complete 230
 - \$.ajax, contentType 228
 - \$.ajax, context 228
 - \$.ajax, data 228
 - \$.ajax, dataFilter 230
 - \$.ajax, dataType 228
 - \$.ajax, error 229
 - \$.ajax, global 232
 - \$.ajax, ifModified 233
 - \$.ajax, jsonp 231
 - \$.ajax, jsonpCallback 231
 - \$.ajax, password 233
 - \$.ajax, processData 228
 - \$.ajax, scriptCharset 233
 - \$.ajax, success 230
 - \$.ajax, timeout 233
 - \$.ajax, type 227
 - \$.ajax, url 227
 - \$.ajax, username 233
 - \$.ajax, xhr 233
 - \$.ajaxSetup 226
 - \$.boxModel 255
 - \$.browser 255
 - \$.contains() 262
 - \$.data() 143
 - \$.each() 257
 - \$.extend() 267
 - \$.extend(obj) 270
 - \$.fn.extend() 270
 - \$.get() 234
 - \$.getJSON() 237
 - \$.getScript() 238
 - \$.globalEval() 265
 - \$.grep() 259
 - \$.inArray() 261
 - \$.isArray() 273
 - \$.isEmptyObject() 273
 - \$.isFunction() 274
 - \$.isPlainObject() 274
 - \$.map() 260
 - \$.merge() 258
 - \$.noConflict() 252
 - \$.noConflict(), extreme 254
 - \$.param() 239
 - \$.post() 236
 - \$.proxy() 266
 - \$.pushStack() 263
 - \$.removeData() 143
 - \$.support 256
 - \$.trim() 262
 - \$.unique() 259
- ## A
-
- add() 102
 - addClass() 175
 - after() 162
 - Ajax 218
 - \$.ajax() 226, 371
 - \$.ajax(), *Konfiguration* 226
 - \$.ajaxSetup() 226
 - \$.get() 234
 - \$.getJSON() 237
 - \$.getScript() 238
 - \$.param() 239
 - \$.post() 236
 - .load() 241
 - Crossdomain-Request* 370
 - Cross-Domain-Schranke* 224
 - Datentypen* 223
 - Datentypen, HTML* 223
 - Datentypen, JSONP* 224
 - Datentypen, JSON* 224
 - Datentypen, Scriptdaten* 224
 - Datentypen, Textdaten* 223
 - Datentypen, XML* 224

Ajax (Forts.)

- klassischer Ansatz* 219
- Low-level Utilities* 225
- mit jQuery* 225
- Same Domain Policy* 241
- XHR-Objekt* 219
- XHR-Objekt, .open()* 219
- XHR-Objekt, .send()* 221
- XHR-Objekt, onreadystatechange* 221
- XHR-Objekt, readyState* 220
- XHR-Objekt,.responseText* 222
- XHR-Objekt, Spezifikation* 223
- XHR-Objekt, status* 222
- XHR-Objekt, Statusmeldung* 222
- XHT-Objekt, responseXML* 224

ajaxComplete() 246

ajaxError() 246

Ajax-Request

- .ajaxComplete()* 246
- .ajaxError()* 246
- .ajaxSend()* 245
- .ajaxStart()* 245
- .ajaxStop()* 246
- .ajaxSuccess()* 246
- allgemein* 226
- Defaultkonfiguration* 226
- Get* 234
- Get JSON* 237
- Get Script* 238
- Globale Handler* 245
- Konfiguration* 227
- Konfiguration, async-Eigenschaft* 232
- Konfiguration, beforeSend-Callback* 229
- Konfiguration, cache-Eigenschaft* 232
- Konfiguration, Callbacks* 229
- Konfiguration, complete-Callback* 230
- Konfiguration, contentType-Eigenschaft* 228
- Konfiguration, context-Eigenschaft* 228
- Konfiguration, data-Eigenschaft* 228
- Konfiguration, dataFilter-Callback* 230
- Konfiguration, dataType-Eigenschaft* 228
- Konfiguration, error-Callback* 229
- Konfiguration, global-Eigenschaft* 232
- Konfiguration, ifModified-Eigenschaft* 233
- Konfiguration, jsonpCallback-Eigenschaft* 231
- Konfiguration, jsonp-Eigenschaft* 231

Ajax-Request (Forts.)

- Konfiguration, password-Eigenschaft* 233
- Konfiguration, processData-Eigenschaft* 228
- Konfiguration, scriptCharset-Eigenschaft* 233
- Konfiguration, success-Callback* 230
- Konfiguration, timeout-Eigenschaft* 233
- Konfiguration, type-Eigenschaft* 227
- Konfiguration, url-Eigenschaft* 227
- Konfiguration, username-Eigenschaft* 233
- Konfiguration, xhr-Eigenschaft* 233
- Load* 241
- Post* 236

ajaxSend() 245

ajaxStart() 245

ajaxStop() 246

ajaxSuccess() 246

Akkordeon-Navigation 294

andSelf() 103

animate() 202

Animationen 189

- .animate()* 202

- .animate(), Argumente* 202

- .animate(), Dauer* 204

- .animate(), Easing* 205

- .animate(), Easingfunktionen* 206

- .animate(), Optionsobjekt* 209

- .animate(), Property-Map* 203

- .delay()* 199

- .dequeue()* 213

- .fadeIn()* 198

- .fadeOut()* 198

- .fadeTo()* 197

- .hide()* 190

- .queue()* 212

- .queue(), Callback* 213

- .queue(), Name der Queue* 214

- .show()* 190

- .slideDown()* 193

- .slideToggle()* 194

- .slideUp()* 193

- .stop()* 200

- .toggle()* 191

Abbrechen von 200

CSS-Eigenschaften animieren 202

Dauer 191

Animationen (Forts.)
Easingfunktionen 205
Easingfunktionen, Default 206
Fades 196
Farbwerte animieren 204
Globaler Stop 217
jQuery Easing Plugin 206
Queue 211
Slides 192
Utilities 199, 216
Verzögerung 199
Zeigen und Verstecken 190
 append() 164
 appendTo() 165
 Arrayverarbeitung 256
 \$.each() 257
 \$.grep() 259
 \$.inArray() 261
 \$.map() 260
 \$.merge() 258
 \$.unique() 259
 Test auf Array 273
 attr() 136
 Attribute 135
 entfernen 137
 lesen 136
 schreiben 136

B

before() 164
 bind() 122
 blur() 152
 Browserhistory 455

C

change() 152
 children() 93
 click() 131
 clone() 111, 163
 closest() 99
 Closure 376
 Collection, Eigenschaften 60
 Collection, primäre 59
 Conditional Comments 286, 323, 340
 Conditional Comments, Definition 287
 contents() 94
 context 90

Cookies 434
 css() 179
 CSS-Anweisung
 Aufbau 514
 Deklarationsblock 514
 Eigenschaftsdeklaration 514
 Selektor 514
 CSS-Datei, Import 512
 CSS-Datei, Verlinkung 511
 CSS-Eigenschaften
 .addClass() 175
 .css() 179
 .hasClass() 173
 .height() 182
 .innerHeight() 182
 .innerWidth() 182
 .outerHeight() 182
 .outerWidth() 182
 .removeClass() 177
 .toggleClass() 177
 .width() 182
 Abmessungen 181
 animieren 202
 computed height 181
 computed width 181
 Position 182
 Scrollposition 186
 CSS-Selektoren
 Gruppierung 516
 ID-Selektor 517
 Klassenselektor, frei 516
 Klassenselektor, gebunden 517
 Typ-Selektor 515

D

data() 141
 dblclick() 131
 delay() 199
 delegate() 125
 dequeue() 213
 detach() 167
 DHTML 17
 die() 125
 Doctype-Definition 499
 Doctype-Deklaration 499
 Document Object Model 563
 Child 564
 decorating the tree 565

Document Object Model (Forts.)
 Dokumentknoten 563
 Elementknoten 564
 Parent 564
 Dokumentstyle 513
 DOM-Eigenschaften
 checked 148
 disabled 147
 innerHTML 64, 135
 innerText 135
 nodeValue 135
 textContent 135
 DOM-Knotenverarbeitung
 \$.contains() 262
 \$.pushStack() 263
 DOM-Manipulation 159
 .after() 162
 .append() 164
 .appendTo() 165
 .before() 164
 .clone() 163
 .detach() 167
 .insertAfter() 162
 .insertBefore() 164
 .prepend() 165
 .prependTo() 165
 .remove() 166
 .replaceAll() 168
 .replaceWith() 169
 .unwrap() 172
 .wrap() 170
 .wrapAll() 171
 .wrapInner() 171
 Einfügen von Knoten 161
 Entfernen von Knoten 166
 Ersetzen von Knoten 168
 Wrapping 169
 DOM-Manipulation, klassisch 159
 DOM-Methoden
 .addEventListener() 113
 .appendChild() 159
 .createElement() 63, 159
 .createTextNode() 159
 .getAttribute() 135
 .getElementById() 61, 62, 113
 .getElementsByClassName() 63
 .getElementsByName() 61, 62
 .insertBefore() 160
 .removeChild() 160

DOM-Methoden (Forts.)
 .setAttribute() 135, 159
 DOM-Referenz, online 567
 DOM-Scripting 17

E

each() 87
 Easing Equations 396
 Editoren
 Adobe Dreamweaver 30
 Aptana 31
 Aptana, Code Assist 33
 BBedit 30
 Eclipse IDE 31
 Microsoft Expression Studio 31
 Microsoft Visual Studio 2010 31
 Notepad++ 30
 TextWrangler 30
 Elementinhalt
 lesen 137
 löschen 140
 schreiben 137
empty() 140
end() 109
eq() 105
error() 134
 Event
 Propagation 119
 Eventbindung
 .bind() 122
 .change() 152
 .click() 131
 .dblclick() 131
 .delegate() 125
 .die() 125
 .error() 134
 .focus() 152
 .focusin() 152
 .focusout() 152
 .hover() 127
 .keydown() 132
 .keypress() 132
 .keyup() 132
 .live() 124
 .load() 133
 .mousedown() 131
 .mouseenter() 131
 .mouseleave() 131

Eventbindung (Forts.)

- .mousemove()* 131
- .mouseout()* 131
- .mouseover()* 131
- .mouseup()* 131
- .resize()* 133
- .scroll()* 134
- .select()* 152
- .submit()* 152
- .toggle()* 128
- .unbind()* 123
- .undelegate()* 126
- .unload()* 134

Eventhandler 113

Eventhandling 112

EventListener 113

Eventobjekt 113, 153, 235

- .isDefaultPrevented()* 120
- .isImmediatePropagationStopped()* 119
- .isPropagationStopped()* 119
- .preventDefault()* 120, 235, 449, 458
- .stopPropagation()* 119
- currentTarget* 115
- keyCode* 117
- Koordinaten* 116
- pageX* 116
- pageY* 116
- relatedTarget* 116
- timeStamp* 116
- type* 117
- vereinheitlichtes E.* 115

Events

- Bubbling-Phase* 118
- Capture-Phase* 118
- Erzeugen von Events* 129
- Fensterevents* 132
- Formulare* 151
- Target-Phase* 118
- Triggern* 129
- Weitertragen des Events* 118

Extreme programming 485

F

- fadeIn()* 198
- fadeOut()* 198
- fadeTo()* 197
- filter()* 105
- find()* 97

- first()* 107

- focus()* 152

- focusin()* 152

- focusout()* 152

Formulare

- Autocomplete* 410
- Daten mit Ajax versenden* 404
- Datepicker* 407
- Validierung* 398

Formularverarbeitung 143

- .serialize()* 153
- .serializeArray()* 154
- .val()* 155
- Events* 151
- Extraktion von Werten* 155
- Selektoren* 144
- Serialisierung* 152
- Zustandsfilter* 146

Funktionsaufrufe

- \$globalEval()* 265
- \$noop* 265
- \$proxy()* 266

G

- Geodaten* 410

- Geonames* 410

- Webservice, Konfiguration* 410

- get()* 88

- Grids* 428

H

- has()* 108

- hasClass()* 173

- height()* 182

- hide()* 190

- hover()* 127

- html()* 137

I

- index()* 89

- innerHeight()* 182

- innerWidth()* 182

- insertAfter()* 162

- insertBefore()* 164

- iPhone Emulator*

- iBBDemo 2* 463

iPhone Emulator (Forts.)

MobiOne 465

is() 91

J

JavaScript

Callback 557

Closure 546

Cookie Objekt 434

Funktionen 542

Globale Variable 544

Konstruktor 552

Konstruktorfunktionen 553

Kontrollstrukturen 540

Lokale Variable 545

Namensräume 560

Objekte 551

Objektliteral 553

Prototypkette 555

Scope 544

ternärer Operator 540

JavaScript-Methoden

.apply() 558

.call() 558

.sort() 421

jQTouch 460, 466

.addAnimation() 482

.goBack() 482

.goTo() 482

Defaultkonfiguration 478

Formulare 476

Formulare, placeholder 476

Initialisierung 481

Konfiguration 467

Konfigurationsobjekt 478

Linkbuttons 472

Linkbuttons, Styling 474

Navigation 469

Navigation, Styling 473

Navigationselemente, Styling 473

Referenzierung 482

Seite 467

Seitenübergänge 470

Seitenübergänge, Animationen 475

Theme 466

Themes, apple 477

Themes, jq 477

Togglebuttons 476

jQTouch (Forts.)

Toolbar 467

Toolbarbuttons 470

Toolbarbuttons, Styling 474

jQuery UI 23, 382

.autocomplete(), Optionen 412

.datepicker() 409

.dialog() 393

Accordion-Widget 394

Accordion-Widget, Konfiguration 396

Autocomplete-Widget 410

CSS-Klassen animieren 390

Datepicker-Widget 407

Datepicker-Widget, Lokalisierung 408

Dialog-Widget 391

Effects core 414

Farbanimation 388

Iconsprites 386

Komponenten 384

Theme 384

Theme bearbeiten 386

Themeroller 384

jQuery-Eigenschaften

.context, Definition 90

.length, Definition 90

.selector, Definition 90

jQuery-Methoden 59

.add(), Definition 102

.addClass() 66, 292, 296, 423

.addClass(), Definition 175

.after() 163, 165

.after(), Definition 162

.ajaxError() 369

.ajaxSuccess() 369

.andSelf(), Definition 103

.animate() 450, 452, 458

.animate(), Beispiel 207

.animate(), Callback 206

.animate(), Dauer 204

.animate(), Definition 202

.animate(), Easing 205

.animate(), erweitern 389

.append() 292, 294, 368, 423

.append(), Definition 164

.appendTo() 111

.appendTo(), Definition 165

.attr() 88, 148, 292, 379

.attr(), Definition 136

.before(), Definition 164

jQuery-Methoden (Forts.)

- .bind()* 130
- .bind()*, Definition 122
- .blur()*, Definition 152
- .change()*, Definition 152
- .children()* 379
- .children()*, Definition 93
- .click()* 112, 292, 294, 296, 367, 449
- .click()*, Definition 131
- .clone()*, Definition 111, 163
- .clone(true)* 112
- .closest()*, Definition 99
- .contents()*, Definition 94
- .css()* 191, 284, 285
- .css()*, Definition 179
- .css()*, Eigenschaft über Funktion 180
- .css()*, Eigenschaftsnamen 180
- .css()*, Lesen einer Eigenschaft 179
- .css()*, Setzen einer Eigenschaft 179
- .css()*, Setzen mehrerer Eigenschaften 180
- .data()* 141, 166
- .data()*, Definition 141
- .dblclick()*, Definition 131
- .delay()*, Definition 199
- .delegate()* 130
- .delegate()*, Definition 125
- .dequeue()*, Definition 213
- .detach()*, Definition 167
- .die()*, Definition 125
- .each()* 89, 150, 292, 294, 368, 374, 379, 449, 454
- .each()*, Definition 87
- .empty()*, Definition 140
- .end()*, Definition 109
- .eq()* 292, 294
- .eq()*, Definition 105
- .error()*, Definition 134
- .fadeIn()* 292, 368
- .fadeIn()*, Definition 198
- .fadeOut()*, Definition 198
- .fadeTo()*, Definition 197
- .filter()*, Definition 105
- .find()* 284, 285, 292, 378
- .find()*, Definition 97
- .first()* 287, 292, 293
- .first()*, Definition 107
- .focus()*, Definition 152
- .focusin()*, Definition 152
- .focusout()*, Definition 152

jQuery-Methoden (Forts.)

- .get()* 91
- .get()*, Definition 88
- .has()*, Definition 108
- .hasClass()* 423
- .hasClass()*, Definition 173
- .height()*, Definition 182
- .hide()* 292, 296, 297, 427
- .hide()*, Definition 190
- .hide(fast)* 191
- .hide(slow)* 191
- .hover()* 284, 285, 416
- .hover()*, Definition 127
- .html()* 234, 235
- .html()*, Definition 137
- .index()* 292, 294
- .index()*, Definition 89
- .innerHeight()*, Definition 182
- .innerWidth()*, Definition 182
- .insertAfter()*, Definition 162
- .insertBefore()* 292, 293
- .insertBefore()*, Definition 164
- .is()* 148, 416
- .is()*, Definition 91
- .keydown()*, Definition 132
- .keypress()*, Definition 132
- .keyup()*, Definition 132
- .last()*, Definition 107
- .live()* 130
- .live()*, Definition 124
- .load()* (Ajax) 356
- .load()*, Callback 244
- .load()*, Daten-Argument 243
- .load()*, Definition (Ajax) 241
- .load()*, Definition (Event) 133
- .load()*, Drilldown-Selektor 242
- .mouseenter()*, Definition 131
- .mouseleave()*, Definition 131
- .mousemove()*, Definition 131
- .mouseout()*, Definition 131
- .mouseover()*, Definition 131
- .mouseup()*, Definition 131
- .next()* 296, 297
- .next()*, Definition 95
- .nextAll()*, Definition 100
- .nextUntil()*, Definition 101
- .not()* 296
- .not()*, Definition 106
- .offset()*, Definition 184

jQuery-Methoden (Forts.)

- .offsetParent()*, Definition 186
- .outerHeight()*, Definition 182
- .outerWidth()*, Definition 182
- .parent()*, Definition 95
- .parents()*, Definition 98
- .parentsUntil()*, Definition 98
- .position()* 449
- .position()*, Definition 186
- .prepend()*, Definition 165
- .prependTo()*, Definition 165
- .prev()* 298
- .prev()*, Definition 96
- .queue()*, Beispiel 214
- .queue()*, Definition 212
- .ready()* 284
- .remove()* 140
- .remove()*, Definition 166
- .removeAttr()* 111, 148
- .removeAttr()*, Definition 137
- .removeClass()* 292, 296
- .removeClass()*, Definition 177
- .removeData()*, Definition 142
- .replace()*, Definition 168
- .replaceWith()*, Definition 169
- .resize()* 454
- .resize()*, Definition 133
- .scroll()* 189
- .scroll()*, Definition 134
- .scrollLeft()*, Definition 188
- .scrollTop()* 449, 450, 452
- .scrollTop()*, Definition 189
- .select()*, Definition 152
- .serialize()* 235, 236
- .serialize()*, Definition 153
- .serializeArray()* 155
- .serializeArray()*, Definition 154
- .show()* 292, 293, 427
- .show()*, Definition 190
- .siblings()*, Definition 101
- .size()*, Definition 89
- .slice()* 427
- .slice()*, Definition 107
- .slideDown()* 284, 285, 296
- .slideDown()*, Definition 193
- .slideToggle()*, Definition 194
- .slideUp()* 284, 285, 296, 297
- .slideUp()*, Definition 193
- .stop()* 285

jQuery-Methoden (Forts.)

- .stop()*, Definition 200
- .submit()*, Definition 152
- .text()* 91, 292, 369
- .text()*, Definition 138
- .text()*, Rückgabewert 139
- .text()*, Stringmethoden 139
- .toArray()*, Definition 90
- .toggle()* 167
- .toggle()*, Definition 128
- .toggleClass()*, Definition 177
- .toggleClass()*, erweitern 391
- .trigger()*, Definition 129
- .triggerHandler()*, Definition 129
- .unbind()*, Definition 123
- .undelegate()*, Definition 126
- .unload()*, Definition 134
- .unwrap()* 379
- .unwrap()*, Definition 172
- .val()* 151
- .val()*, Definition 156
- .width()*, Definition 182
- .wrap()*, Definition 170
- .wrapAll()*, Definition 171
- .wrapInner()* 379
- .wrapInner()*, Definition 171
- Accessoren 86
- Animationen 189
- Animationen, Fades 196
- Animationen, Slides 192
- Animationen, Utilities 199
- Animationen, Zeigen und Verstecken 190
- Collection, Eigenschaften 86
- Eventbindung 121
- Traversierung 91
- Verkettbarkeit 59

jQuery-Methoden, destruktive 60

jQuery-Methoden, terminierende 60

jQuery-Methoden, transparente 60

jQuery-Objekt 59

Datenspeicherung in 141

Erweiterung mit *\$.extend()* 270

jQuery-Properties 60

jQuery-Utilities

\$.ajax() 413*\$.ajax()*, Definition 226*\$.ajax()*, success 413*\$.ajaxSetup()*, Definition 226*\$.boxModel*, Definition 255

jQuery-Utilities (Forts.)
\$.browser 449
\$.browser, Definition 255
\$.contains(), Definition 262
\$.data(), Definition 143
\$.dequeue(), Definition 216
\$.each() 155
\$.each(), Definition 257
\$.extend() 375
\$.extend(), Definition 267
\$.extend(), jQuery-Objekt 270
\$.extend(), Objektmerging 267
\$.fn.extend(), Definition 270
\$.fx.off, Definition 217
\$.get(), Definition 234
\$.getJSON(), Definition 237
\$.getScript(), Definition 238
\$.globalEval(), Definition 265
\$.grep(), Definition 259
\$.inArray(), Definition 261
\$.isArray(), Definition 273
\$.isEmptyObject(), Definition 273
\$.isFunction(), Definition 274
\$.isPlainObject(), Definition 274
\$.map(), Definition 260
\$.merge() 265
\$.merge(), Definition 258
\$.noConflict(), Definition 249
\$.noConflict(), Einsatz 252
\$.noConflict(), extreme 254
\$.noConflict(), mit Alias 254
\$.param(), Definition 239
\$.post(), Definition 236
\$.proxy(), Definition 266
\$.pushStack(), Definition 263
\$.queue(), Definition 216
\$.removeData(), Definition 143
\$.support, Definition 256
\$.trim() 152
\$.trim(), Definition 262
\$.unique(), Definition 259
Arrayverarbeitung 256
DOM-Knotenverarbeitung 262
Erweiterung von jQuery 267
Funktionsaufrufe 265
Objektverarbeitung 267
Stringbearbeitung 261
Test auf Datentyp 272
JSON 369

JSON-Daten
laden 367
JSONP 370, 371

K

keydown() 132
keypress() 132
keyup() 132

L

last() 107
length 90
Lifecycle 489
live() 124
load() (Ajax) 241
load() (Event) 133

M

MAMP 30
Mobile Design 460
mousedown() 131
mouseenter() 131
mouseleave() 131
mousemove() 131
mouseout() 131
mouseover() 131
mouseup() 131
Mozilla Foundation 567

N

next() 95
nextAll() 100
nextUntil() 101
not() 106

O

Objektverarbeitung
\$.extend() 267
\$.extend(), deep 269
Merging von Objekten 267
Test auf leeres Objekt 273
Test auf plain Object 274
offset() 184
offsetParent() 186

outerHeight() 182
outerWidth() 182

P

Paginierung 424
parent() 95
parents() 98
parentsUntil() 98
Personen
 Alsup, Mike 405
 Belander, Stephan 443
 Dhakar, Lokesh 337
 Edwards, Dean 22
 Flesler, Ariel 459
 Penner, Robert 396
 Resig, John 11, 22, 485
 Skarnelis, Janis 348
 Sullivan, Shaun 463
 Zaefferer, Jörn 398, 485
phpMyAdmin 30
PlugIn-Erstellung
 Allgemeine Regeln 382
 Default-Objekt 382
 Namensraum 382
 Namensschema 382
 Optionen 382
 Rückgabewert 382
 Semikolon-Regel 382
Plugin-Erstellung 270
PlugIn-Methoden
 .addAnimation() 482
 .flash() 445
 .goBack() 482
 .goTo() 482
 .scrollTo() 458
PlugIns 372
 Address 440
 jqGrid 429
 jqGrid, Konfigurator 430
 jQuery Address 456, 459
 jQuery.cookies 435
 jquery.easing 397
 jQuery.ScrollTo 454, 459
 jQuery.sheet 433
 jquery-flash 443
 Tabs 439
position() 186
prepend() 165

prependTo() 165
prev() 96

Q

Querystring 153, 154, 224, 228, 231,
 236, 239, 240
queue() 212
QUnit 485
 .equals() 488
 .ok() 488
 .same() 489
 Assertions 487
 Lifecycle 489
 Module 489
 setup 489
 teardown 489

R

remove() 166
removeAttr() 137
removeClass() 177
removeData() 142
replaceAll() 168
replaceWith() 169
resize() 133

S

scroll() 134
scrollLeft() 188
scrollTop() 189
select() 152
selector 90
Selektorstring
 Basisselektoren 68
 Child E > F, Definition 70
 Descendant 70
 Filter :animated, Definition 75
 Filter :button 144
 Filter :checkbox 144
 Filter :contains, Definition 76
 Filter :empty, Definition 76
 Filter :eq(), Definition 74
 Filter :even, Definition 73
 Filter :file 144
 Filter :first, Definition 73
 Filter :first-child 83

Selektorstring (Forts.)

Filter :gt(), Definition 74
Filter :has(), Definition 77
Filter :header, Definition 74
Filter :hidden 296, 297
Filter :hidden, Definition 78
Filter :image 144
Filter :input 144
Filter :last, Definition 73
Filter :last-child 83
Filter :lt(), Definition 74
Filter :not() 77
Filter :not(), Definition 73
Filter :nth-child(), Definition 84
Filter :nth-child(even), Definition 85
Filter :nth-child(even) zu :even 85
Filter :nth-child(odd), Definition 85
Filter :odd, Definition 73
Filter :only-child, Definition 83
Filter :parent, Definition 77
Filter :password 144
Filter :radio 144
Filter :reset 144
Filter :submit 144
Filter :text 144
Filter :visible 296, 297
Filter :visible, Definition 79
Filter Attribut, Definition 80
Filter Attributwert (Anfang), Definition 83
Filter Attributwert (Ende), Definition 82
Filter Attributwert (Teilstring), Definition 81
Filter Attributwert (Teilwert), Definition 82
Filter Attributwert (Ungleichheit), Definition 82
Filter Attributwert, Definition 80
Filter für Formularelemente 143
Filter lang-Attribut, Definition 81
Filter Multi-Attribut 81
Filter nach Attribut 79
Filter nach Inhalt 75
Filter nach Position als Kindknoten 83
Filter nach Sichtbarkeit 77
Filterausdrücke 72
Following E ~ F, Definition 71
Following Sibling E + F, Definition 71
Gruppenselektoren 69

Selektorstring (Forts.)

ID-Selektor 68
Klassenselektor 68
Kontextselektor E F, Definition 70
Mehrfachklassen 68
Sonderzeichen, escapen von 67
Typselektor 68
Universalselektor 68
Zustandsfilter :checked 148
Zustandsfilter :disabled 147
Zustandsfilter :enabled 147
Zustandsfilter :selected 150
Zustandsfilter für Formularelemente 146
serialize() 153
serializeArray() 154
show() 190
siblings() 101
size() 89
slice() 107
slideDown() 193
slideToggle() 194
slideUp() 193
Spaltennavigation 298
stop() 200
Stringbearbeitung
 \$.trim() 262
Stylesheet, Definition 514
submit() 152

T

Tabellen

Paginierung 424
Sortierfunktion 423
Sortierung 417, 419
Zebra-Tabellen 414
Tab-Navigation 290
Test driven development 485
text() 91, 138
Textinhalt
 lesen 138
 löschen 140
 schreiben 140
Themeroller 384
 Konfigurator 384
 Theme erstellen 386
toArray() 90
toggle() 128, 191
toggleClass() 177

trigger() 129
triggerHandler() 129

U

unbind() 123
undelegate() 126
Unit Tests 485
 JUnit 485
 Lifecycle 489
 JUnit 485
unload() 134
Unobtrusive JavaScript 22
Unobtrusive JavaScript, Definition 559
unwrap() 172

V

val() 156
Viewport 186, 187, 218, 509, 565, 566

W

width() 182
WML 467
wrap() 170
wrapAll() 171
wrapInner() 171

X

XAMPP 30

Z

Zebra-Tabellen 414
Zustandsfilter 146