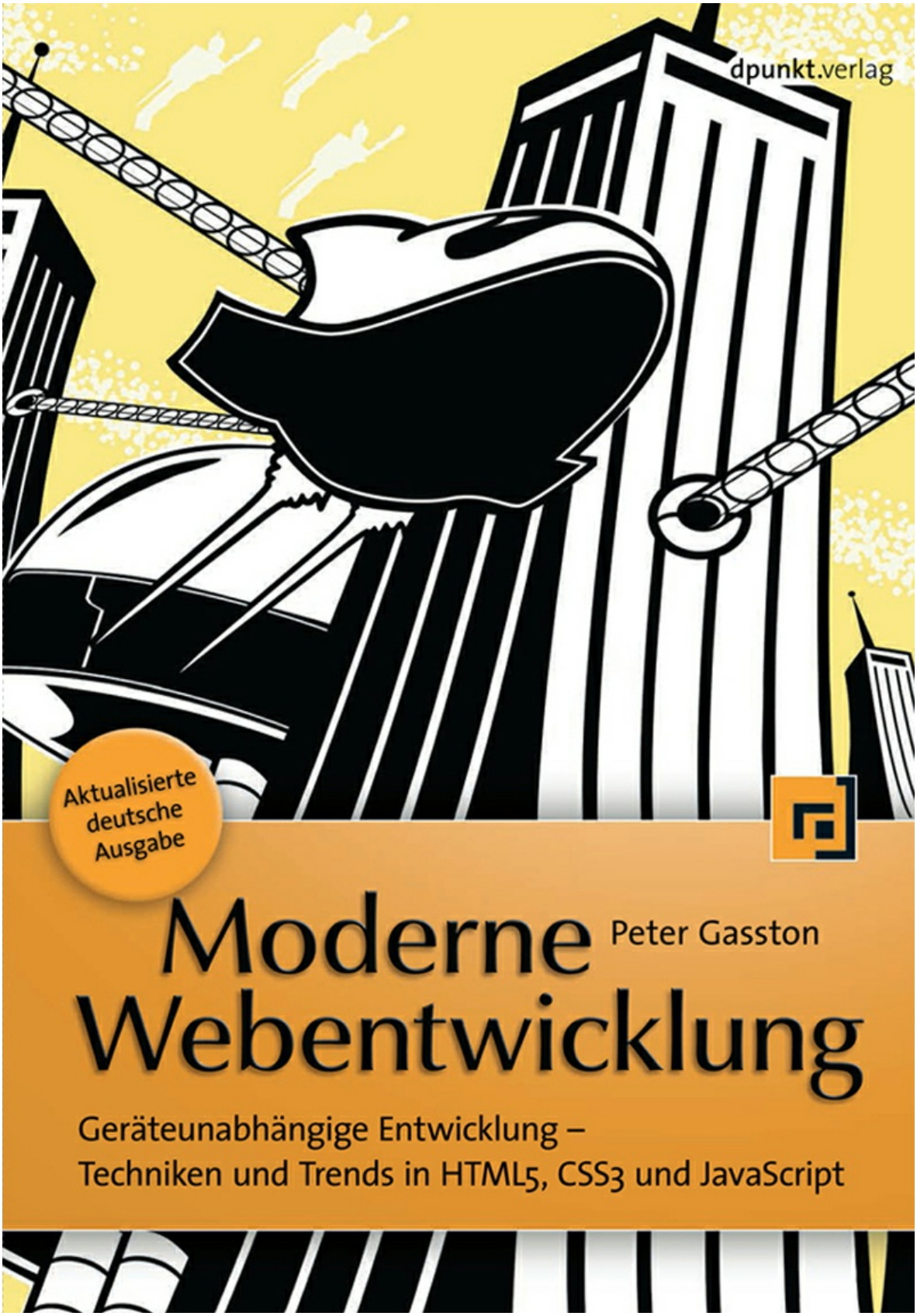
dpunkt.verlag

Aktualisierte deutsche Ausgabe

Moderne Webentwicklung

Peter Gasston

Geräteunabhängige Entwicklung – Techniken und Trends in HTML5, CSS3 und JavaScript


dpunkt.verlag
Aktualisierte deutsche Ausgabe
Moderne Webentwicklung
Peter Gasston
Geräteunabhängige Entwicklung –
Techniken und Trends in HTML5, CSS3 und JavaScript

Stimmen zur englischen Originalausgabe »The Modern Web«

»Dies ist ein nützliches und ein wichtiges Buch. Wenn Sie Peter Gasstons Ratschlägen folgen, Ihre Sites in allen Browsern und auf einer Fülle von Anzeigegeräten testen, werden Sie Ihren Chef beeindrucken und die Nutzer begeistern. Sie werden an der Verbesserung des Web mitwirken, sodass es eine offene Umgebung bleibt.«

Bruce Lawson, Autor von »Introducing HTML5«

»Peter Gasston hat nun für die moderne Webplattform dasselbe getan wie bereits für CSS: Er hat ein erstklassiges Kompendium mit Informationen für Webentwickler praktisch jeder Qualifikationsstufe geschrieben.«

Stephen Hay, Autor von »Responsive Design Workflow«

»Peter Gasstons Ausführungen sind ein großartiger Spagat zwischen der Produktion von lupenreinem Code für echte Anwendungen und aktuellen Informationen über die neuesten Entwicklungen der Webtechnologie. Er hat eine außergewöhnliche Begabung, schwierige Fachthemen auf eine einleuchtende und unterhaltsame Weise zu vermitteln.«

Chris Mills, Developer Relations Manager bei Opera Software und Autor von »Practical CSS3«

»Beim Lesen dieses Buchs lernte ich eine Menge Neues. Fantastische, nützliche Dinge, die ich bereits in meinen aktuellen Projekten angewandt habe. Techniken, von denen noch nichts gehört hatte, obwohl ich so viel lese und recherchiere. Wenn Sie moderne Websites entwickeln, dann brauchen Sie dieses Buch meiner Ansicht nach einfach.

Jane Morrison, FLOSS UK Secretariat

»*Moderne Webentwicklung* gehört in das Bücherregal jedes Webentwicklers!«

Raj Sharma, British Computer Society

»*Moderne Webentwicklung* ist nicht nur ein hervorragendes Nachschlagewerk; es ist auch sehr gut lesbar. Es besitzt einen guten Index, die Beispiele sind kurz und knackig und der Stil ist unterhaltsam.«

Lynn Greiner, SmartBear

»*Moderne Webentwicklung* ist eine angenehme Lektüre. Peter schreibt in guter und einfacher Sprache, sodass der Leser die Informationen genießen kann, statt über langweiligem technischem Zeug einzuschlafen. Peters geradlinige Weise, moderne Programmiertechniken zu lehren, ist angenehm und ansprechend.«

Gisele Muller, Abduzeedo

»Wenn Sie bereit, Ihre bisherige Herangehensweise an die Entwicklung von Webseiten aufzugeben und sich modernere Techniken anzueignen, ist dieses Buch eine große Bereicherung für Ihre Bibliothek. Die Zusatzressourcen, die in jedem Kapitel und im letzten Anhang aufgeführt sind, sind von unschätzbarem Wert. «

Jennifer Kyrnin, About.com Web Design/HTML

»Für mich als Front-End-Entwickler ist das Verständnis für den Kontext, in dem wir arbeiten, wichtig, für User-Trends, technische Entwicklungen und die Werkzeuge, die zur Verfügung stehen oder bald nutzbar sein werden ... Wenn Sie dieses Jahr nur ein einziges Buch über die neuesten Entwicklungen in HTML5, CSS3 und JavaScript lesen möchten, dann sollten Sie dieses wählen.«

Chris Webb, Front-End-Entwickler bei Funda

»Dies ist ein gut geschriebenes Buch. Auf klare und verständliche Weise führt Gasston Sie in zahlreichen Kapiteln detailliert durch die neuesten Technologien und Denkräume des modernen Web-Designs.«

Virginia DeBolt, *Web-Trainerin*

»Der Autor hat einen wirklich guten Schreibstil, der das Buch eine unterhaltsame Lektüre macht. Er deckt eine Menge Techniken ab und geht ausreichend in die Tiefe, um gründlich zu erläutern, wie man sie benutzt. «

Tad Anderson, *SYS-CON MEDIA*

»Wenn Sie von geschickten CSS-Techniken, semantischem Markup und verschiedenen Möglichkeiten, APIs zu verwenden und mit den Nutzern zu interagieren, fasziniert sind, ist *Moderne Webentwicklung* das richtige Buch für Sie.«

David Indish, *Red Star Web Development*

»Pflichtlektüre für jeden Web-Entwickler.«

Craig Buckler, *Optimal Works*

Peter Gasston entwickelt seit über 12 Jahren Webanwendungen für Agenturen und Unternehmen. Er ist Mitbegründer von CSS3.info, der führenden Online-Ressource für CSS3, und schrieb »The Book of CSS3« (No Starch Press) sowie Beiträge für Smashing Magazine, A List Apart und .net magazine. Er hält Vorträge auf Entwicklerkonferenzen und ist Inhaber des Webentwickler-Blogs Broken Links (broken-links.com). Peter Gasston lebt in London.

Peter Gasston

Moderne Webentwicklung

Geräteunabhängige Entwicklung – Techniken und Trends in HTML5, CSS3 und JavaScript

Aktualisierte deutsche Ausgabe

Lektorat: René Schönfeldt
Übersetzung: Isolde Kommer, Großerlach, und Christoph Kommer, Dresden
Fachgutachter für die deutsche Ausgabe: Florence Maurice (CSS, HTML),
Marcus Ross (JavaScript, Mobile Entwicklung)
Copy-Editing: Annette Schwarz, Ditzingen
Satz: Isolde Kommer, Großerlach, und Tilly Mersin, Großerlach
Herstellung: Frank Heidt
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

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:
Buch 978-3-86490-116-4
PDF 978-3-86491-464-5
ePub 978-3-86491-465-2

1. Auflage 2014
Copyright © 2014 dpunkt.verlag GmbH
Wieblinger Weg 17
69123 Heidelberg

Copyright © 2013 by Peter Gasston. Title of English-language original:

The Modern Web,
ISBN 978-1-59327-487-0, published by No Starch Press.
German-language edition copyright © 2013/2014 by dpunkt.verlag. All rights reserved.

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.
Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.
Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.
5 4 3 2 1 0

Für Dave, Jim, Morena, Nick, Rupert, Steve und alle anderen Mitwirkenden der London Web Standards Group, die für eine aktive Londoner Szene arbeiten und es mir ermöglichten, meinen ersten öffentlichen Vortrag zu halten.

Inhaltsverzeichnis

Einleitung
Die Gerätelandschaft
Desktop/Laptop
Mobilgeräte
Tablet
Fernseher
Die anderen
Die »Zwischendinger«
Die Multiscreen-Welt
Kontext: Was wir nicht wissen
Einige Kontext-Klischees
Nur die Geschwindigkeit zählt
Das werden Sie lernen
Literaturempfehlungen

1 Die Webplattform

1.1 Ein kurzer Hinweis zu den Begriffen
1.2 Wer Sie sind und was Sie wissen müssen
1.3 Begriffsklärung
1.4 Das echte HTML5
1.4.1 Das HTML5-Gerüst
1.4.2 Neue Standardverfahren
1.5 CSS3 und mehr
1.5.1 Herstellerspezifische Präfixe

1.5.2 CSS-Frameworks und Präprozessoren
1.6 Browserunterstützung
1.7 Testen, testen und nochmals testen
1.8 Zusammenfassung
1.9 Literaturempfehlungen

2 Struktur und Semantik

2.1 Grundsätzliche Anmerkungen
2.2 Neue HTML5-Elemente
2.2.1 Was bringen die neuen Elemente überhaupt?
2.2.2 Der Nachteil der Sektionselemente in HTML5
2.3 WAI-ARIA
2.4 Die Bedeutung semantischen Markups
2.5 Mikroformate
2.6 RDFa
2.7 Mikrodaten
2.7.1 Die API für Mikrodaten
2.7.2 Mikrodaten, Mikroformate und RDFa
2.7.3 Schema.org
2.7.4 Rich Snippets
2.8 Datenattribute
2.8.1 Die API für Datenattribute
2.8.2 jQuery und Datenattribute
2.8.3 Datenattribute in natura
2.9 Webkomponenten: Die Zukunft der Auszeichnungssprachen?
2.10 Zusammenfassung
2.11 Literaturempfehlungen

3 Mediengerechtes CSS

3.1 Media Queries
- 3.1.1 Media-Eigenschaften für die Abmessungen
- 3.1.2 Media Queries kombinieren und negieren
- 3.1.3 Ein kleiner Exkurs: Alles über Pixel
- 3.1.4 Media Queries für die Bildschirmauflösung
- 3.1.5 Geräteanpassung
- 3.1.6 Media-Eigenschaften für die Benutzerinteraktion
- 3.1.7 Weitere Media-Eigenschaften

3.2 Media Queries mit JavaScript

3.3 Adaptives oder responsives Webdesign?
- 3.3.1 Die Eigenschaft »box-sizing«
- 3.3.2 Längenmaße dynamisch berechnen

3.4 Längenangaben relativ zum Viewport
- 3.4.1 Einheiten relativ zum HTML-Wurzelelement (root)
- 3.4.2 »Mobile First« und Breakpoints für den Inhalt

3.5 Responsives Design und Objektersetzung
- 3.5.1 Das Bildproblem
- 3.5.2 Responsive HTML5-Bilder als Lösung

3.6 Zusammenfassung

3.7 Literaturempfehlungen

4 Neue CSS-Konzepte

4.1 Mehrspalten-Layout
- 4.1.1 Spaltenabstände und -linien
- 4.1.2 Spaltenspanne und Umbrüche

4.2 Flexbox

4.2.1 Das Flexbox-Modell deklarieren
4.2.2 Die Reihenfolge der Inhalte ändern
4.2.3 Anordnung innerhalb des Containers
4.2.4 Mehr Flexibilität
4.2.5 Wrap und Flow
4.3 Rasterlayouts
4.3.1 Das Raster deklarieren und definieren
4.3.2 Rasterlinien wiederholen
4.3.3 Elemente im Raster erzeugen
4.3.4 Ausrichtung und Stapelfolge
4.3.5 Die Grid-Layout-Syntax
4.3.6 Über die Terminologie des Rasterlayout-Moduls
4.3.7 Rasterbereiche
4.4 Die ferne Zukunft
4.5 Zusammenfassung
4.6 Literaturempfehlungen

5 Modernes JavaScript

5.1 Neu in JavaScript
5.1.1 Die Attribute async und defer
5.1.2 Die addEventListener-Methode
5.1.3 Das DOMContentLoaded-Ereignis
5.1.4 Eingabe-Events
5.1.5 CSS-Selektoren in JavaScript
5.1.6 Die Methode getElementsByClassName
5.1.7 Interaktion mit Klassen
5.2 JavaScript-Bibliotheken
5.2.1 jQuery

5.2.2 YepNope
5.2.3 Modernizr
5.2.4 Mustache
5.3 Polyfills und Shims
5.4 Testen und Debuggen
5.5 Zusammenfassung
5.6 Literaturempfehlungen

6 Geräte-APIs

6.1 Geolokalisierung
6.2 Orientierung und Bewegung im Raum
6.3 Vollbild
6.4 Vibration
6.5 Batteriestand
6.6 Verbindungsinformation
6.7 Kamera und Mikrofon
6.8 Web Storage
6.9 Drag & Drop
6.10 Mit Dateien arbeiten
6.11 Mozillas Firefox OS und WebAPIs
6.12 PhoneGap und native Wrapper
6.13 Zusammenfassung
6.14 Literaturempfehlungen

7 Bilder und Grafiken

7.1 Vektor- und Pixelgrafiken im Vergleich
7.2 Scalable Vector Graphics
7.2.1 Anatomie eines SVG-Bilds

7.2.2 Verknüpfte SVG-Dateien
7.2.3 Eingebettetes SVG
7.2.4 SVG-Filter
7.2.5 SVG und CSS kommen zusammen
7.2.6 Ein Nachteil von SVG
7.3 Das canvas-Element
7.3.1 Bildmanipulation
7.3.2 WebGL
7.4 Wann sollten Sie SVG oder Canvas wählen?
7.5 Zusammenfassung
7.6 Literaturempfehlungen

8 Neue Formulare

8.1 Neue Eingabetypen
8.2 Neue Attribute
8.2.1 autofocus
8.2.2 placeholder
8.2.3 autocomplete
8.2.4 spellcheck
8.2.5 multiple
8.2.6 form
8.3 Vorschlagslisten
8.4 Bedienelemente und Widgets
8.4.1 Zahlen
8.4.2 Datum und Zeit
8.4.3 Farben
8.5 Informationen anzeigen
8.5.1 progress

8.5.2 meter
8.5.3 output
8.6 Formularüberprüfung auf Anwenderseite
8.7 Die Constraint-Validation-API
8.8 Formulare und CSS
8.9 Zusammenfassung
8.10 Literaturempfehlungen

9 Multimedia

9.1 Die Medienelemente
9.1.1 Zusätzliche Attribute für das video-Element
9.1.2 Mehrere Quelldateien
9.1.3 Fallbacks
9.1.4 Untertitel und Beschriftungen
9.1.5 Codierung
9.2 Media Fragments
9.3 Die Media-API
9.3.1 Ladestatus
9.3.2 Zusätzliche Eigenschaften für Audio und Video
9.4 Medienereignisse
9.5 Erweiterte Medieninteraktion
9.5.1 Web-Audio-API
9.5.2 WebRTC
9.6 Zusammenfassung
9.7 Literaturempfehlungen

10 Web-Apps

10.1 Web-Apps

10.1.1 Hosted und Packaged Apps
10.1.2 Manifest-Dateien
10.1.3 W3C-Widgets
10.2 Hybrid-Apps
10.2.1 PhoneGap
10.2.2 Titanium
10.3 TV-Apps
10.4 Webinos
10.5 Application Cache
10.5.1 Inhalte der AppCache-Datei
10.5.2 Die Caching-Abfolge
10.5.3 Die AppCache-API
10.6 Zusammenfassung
10.7 Literaturempfehlungen

11 Die Zukunft

11.1 Webkomponenten
11.1.1 Templates
11.1.2 Decorator
11.1.3 Custom Elements
11.1.4 Das Shadow DOM
11.1.5 Die Puzzleteile zusammenfügen
11.2 Die Zukunft von CSS
11.2.1 Regions
11.2.2 Exclusions
11.2.3 Layouts: Noch weiter in die Zukunft geblickt
11.2.4 Feature Queries
11.2.5 Cascading Variables

11.3 Zusammenfassung
11.4 Literaturempfehlungen

Anhänge

A Browserunterstützung

B Literaturempfehlungen

Index

Dank

David Storey, dem Fachlektor dieses Buchs, schulde ich großen Dank. Seine fundierten Fachkenntnisse waren unschätzbar für mich. Obwohl er mich immer wieder korrigierte und beriet, gehen eventuelle Fehler in diesem Buch ganz auf mein Konto.

Keith Fancher, Serena Yang, Bill Pollock und der Rest des Teams von No Starch Press boten mir während der Entstehung dieses Buchs großartige Unterstützung und Beratung. Ihre Leistung kann nicht hoch genug eingeschätzt werden.

Dimitri Glazkov beantwortete mir zuvorkommend einige Fragen zu den Webkomponenten. Bruce Lawson arbeitete speziell meine Ausführungen zu den neuen HTML5-Elementen in Kapitel 2 durch. Während ich dieses Buch schrieb, griff ich ständig auf seine Arbeit und die der übrigen HTML5-Doktoren zurück.

Zwar kenne ich David Walsh nicht persönlich, danke ihm jedoch für seine herausragende Website, die ich oft zu Rate gezogen habe.

Stephen Hay und Chris Mills halfen mir nicht nur bei der Umsetzung meiner Ideen – sie sind auch unglaublich nette Menschen.

Ein großes Dankeschön geht an meine Freunde und Gelegenheitsmitarbeiter Giles McCartney, Richard Locke und Tom Shirley. Außerdem danke ich allen anderen Kollegen bei Preloaded, Poke, Top10, Believe.in und rehabstudio.

Und am allermeisten muss ich meiner Frau Ana danken – für ihre Geduld und Unterstützung während der Zeit, die ich an diesem Buch gearbeitet habe.

Einleitung

Heute erleben wir im Web eine beispiellose Innovation. Noch vor nicht allzu langer Zeit prägte ein einziges Unternehmen die Weblandschaft: Microsoft. Im Jahr 2003 wurde der Internet Explorer auf etwa 95% der Computer weltweit eingesetzt. Für den Entwickler hatte diese Dominanz den Vorteil eines stabilen Marktes. Es gab aber auch einen entscheidenden Nachteil: Microsoft beschränkte die Weiterentwicklung des IE irgendwann auf das Nötigste, und die Innovation im Web stagnierte – die Folge mangelnden Wettbewerbs und einer geschlossenen Umgebung.

Viele Browser

Inzwischen sieht es völlig anders aus. Es gibt am Browser-Markt vier bis fünf Hauptakteure und etwa genauso viele große Betriebssysteme. Ständig treten neue Mitspieler auf. Adobe verlagert seinen Schwerpunkt von Flash und Apps in Richtung Open Web, und neben Branchengrößen wie Google, Apple und Microsoft mischen auch Technologiekonzerne wie Samsung und Nintendo bei der zukünftigen Gestaltung des Web mit. Auch sie beeinflussen natürlich die Webentwicklung.

Viele Geräte

Die Zugangsmöglichkeiten haben sich ebenfalls drastisch verändert – denken Sie nur an die Geräte. Wir brauchen heute nicht mehr unbedingt einen Desktop- oder Laptop-Computer, um im Internet zu surfen. Inzwischen existieren so viele Gerätekategorien – etwa Smartphones, Tablets und Spielekonsolen –, dass mir die Finger zum Zählen ausgehen.

Moderne Webfrontends

In diesem Buch geht es um die Entwicklung von Webfrontends in diesem neuen Zeitalter des universellen

Internetzugangs. Es geht um Methoden zur Entwicklung hochklassiger Websites, Apps und allem anderen, was auf offenen Webtechnologien aufbaut. Dabei behalten wir immer die vielfältige Gerätewelt im Blick. Das Buch handelt nicht von der Programmierung mobiler Websites oder Smart-TV-Apps. Sie lernen hier die aktuellsten sowie die für die nahe Zukunft absehbaren Entwicklungen der Webtechnologie kennen. Mit diesem Wissen können Sie bessere Sites entwickeln und die beste User Experience für alle denkbaren Zugangsmöglichkeiten liefern.

Am Ende dieser Einleitung gehe ich noch näher auf die Inhalte des Buchs ein – zunächst möchte ich mich mit der inzwischen so verblüffenden Vielfalt internetfähiger Geräte beschäftigen.

Die Gerätelandschaft

2008 war ein herausragendes Jahr – und zwar nicht nur wegen des Kinostarts von »Indiana Jones und das Königreich des Kristallschädels«. Vielmehr überschritt in diesem Jahr die Anzahl internetfähiger Geräte erstmals die Anzahl ihrer Benutzer. Diese Super-Konnektivität zwischen Geräten ist gemeinhin als Internet der Dinge (IoT = engl.: Internet of Things) bekannt. Das vorliegende Buch zielt auf eine bereits beachtlich große und schnell wachsende Untermenge dieser Dinge ab, namentlich auf all jene Dinge mit integrierten Webbrowsern.

Webfähige Geräte

Es gibt eine enorme Bandbreite webfähiger Geräte, und täglich werden es mehr. Ich selbst besitze derzeit folgende Geräte mit einem Webbrowser: einen Desktop-Computer, einen Laptop, ein Tablet, ein Smartphone, einen E-Book-Reader und eine Spielekonsole. Ich denke, für etwas wohlhabendere Gesellschaftsmitglieder ist das kein ungewöhnliches Szenario; bei vielen kämen zu dieser Liste noch ein Smart-TV und eine tragbare Spielekonsole hinzu.

Das ungeheure Spektrum internetfähiger Geräte birgt natürlich auch

unglaubliche Möglichkeiten für Webentwickler. Dieses Buch soll Ihnen helfen, dabei das Maximum herauszuholen. Sehen wir uns aber zunächst die wichtigsten Gerätekategorien an und definieren wir einige Schlüsselbegriffe. So verstehen Sie besser, womit Sie es zu tun haben.

Desktop/Laptop

Desktop-Geräte

Das Web, wie wir es bis vor Kurzem kannten, zielte vornehmlich auf die größeren Bildschirme und leistungsfähigeren Prozessoren der Desktop-und Laptop-Computer (kurz: Desktop-PCs) ab. Üblicherweise sitzt der Nutzer vor diesen Geräten und greift dabei auf eine schnelle Netzwerkoder WLAN-Verbindung zu. Dies ist immer noch das bevorzugte Szenario am Büroarbeitsplatz. Daher sind viele Business-to-Business-Sites auf diese Konfiguration ausgelegt.

Der Einsatz von Desktop-Computern nimmt jedoch immer weiter ab; insbesondere Heimanwender tendieren heute verstärkt zu Mobiloder Tablet-Geräten (in Kürze werde ich darauf eingehen). Im letzten Quartal 2012 gingen die PC-Verkäufe gegenüber dem Vorjahreszeitraum um fast 5% zurück. Untersuchungen zeigen jedoch, dass die Anwender für textlastige Aufgaben, für Multitasking mit mehreren Browser-Tabs oder etwa für Bildbearbeitungs- und andere Programme, mit denen Inhalte exakt manipuliert werden müssen, nach wie vor den Desktop-PC bevorzugen. Auch wenn die Sicherheit im Vordergrund steht – etwa beim Internet-Banking –, kommen eher Desktop-PCs zum Einsatz.

Doch selbst innerhalb dieser respektablen und weitgehend stabilen Gruppe gibt es erhebliche Abweichungen. Die Bildschirmauflösung 1024×768 stirbt langsam aus, ist aber momentan noch recht verbreitet; die neuesten Versionen des Apple MacBook Pro bringen es indessen auf 2880×1800 Pixel Bildschirmfläche. Für die zuerst genannte Auflösung optimierte Sites werden auf einer Fläche von 2880×1800 Pixel ziemlich klein und verloren wirken.

Desktop-Browser

Bei den Desktop-Browsern teilen fünf Hauptbewerber den Markt unter sich auf: Chrome, Safari und Opera, die alle auf der WebKit-Engine aufsetzen (Opera hatte mit Presto eine eigene Engine, die aber in den Ruhestand geschickt wurde) sowie Firefox und Internet Explorer (Version 9 bietet die meisten und 10 sämtliche modernen Funktionen). Glücklicherweise werden die Funktionen bei allen Browsern in der Regel in standardkonformer Weise implementiert. Zudem verfügen alle Browser über eine regelmäßige Aktualisierungsroutine, oder die Entwicklung weist zumindest dorthin; die Anwender erhalten also recht schnell Zugriff auf neue Funktionen. Natürlich laufen bei einem nicht unerheblichen Prozentsatz der Anwender immer noch ältere Browser mit geringerem Funktionsumfang. Auch dies sollten Sie bei der Entwicklung stets im Hinterkopf behalten. (Viele Innovationen für Webstandards kommen zuerst auf Desktop-Browsern an, weil diese von den Entwicklern Tag für Tag eingesetzt werden. Viele Informationen in diesem Buch beziehen sich also auf Desktop-Browser. Was Sie dabei lernen, können Sie aber natürlich überall einsetzen.)

Mobilgeräte

Mobile Geräte

Dazu zähle ich sowohl ältere, preiswerte Mobiltelefone mit einfachen Webbrowsern, Geräte mit mittlerem Funktionsumfang und auf Geschwindigkeit statt auf Leistung optimierten Browsern. Dazu gehören aber auch leistungsstarke, teure Smartphones, deren Browser ihrem Desktop-Äquivalent ebenbürtig sind, und schließlich einige Apps, die häufig einen eingebetteten Browser zur Anzeige von Inhalten enthalten.

In vielen Ländern greift die Mehrzahl der Nutzer über Mobilgeräte auf das Internet zu. In Indien erfolgen etwa 55 Prozent der Internetzugriffe auf diese Weise, und in Ägypten nutzen nahezu 70

Prozent der Anwender ihr Handy als einzigen Zugangsweg zum Web. Schätzungen zufolge wird die weltweite Anzahl der mobilen Internetnutzer irgendwann zwischen 2014 und 2015 die Anzahl der Computernutzer überschreiten.

Ein für die Entwicklerszene heute wichtiges Mobilgerät ist das Feature Phone, dessen Funktionen über das einfache Telefonieren und Verfassen von Textnachrichten hinausgehen, das aber häufig keinen vollständigen Webzugang bietet. Das zunehmende Angebot an preiswerten und gebrauchten Smartphones krempelt diesen Markt allerdings bereits drastisch um. Innerhalb der nächsten fünf Jahre werden wohl die Smartphones die mobile Gerätelandschaft dominieren. Schätzungen zufolge erreichten Smartphones 2011 bereits einen weltweiten Marktanteil von 27 Prozent, und Ende 2012 war bereits ungefähr 1 Milliarde Smartphones im Einsatz – und diese Zahl dürfte sich bis 2015 verdoppeln.

Fokus des Buchs

Die schiere Vielfalt an Mobilgeräten und mobilen Browsern auf dem Weltmarkt zwingt mich dazu, den in diesem Buch abgehandelten Bereich einzuschränken. Aus Gründen der Zukunftssicherheit ziele ich daher auf Mobiltelefone mit voll ausgerüsteten Webbrowsern ab, deren Funktionen der entsprechenden Desktop-Version in nichts nachstehen.

Mobile Browser

Die WebKit-Engine dominiert den Markt der Smartphone-Browser – wir finden sie in vielen unterschiedlichen Browsern auf dem iPhone, Android, Blackberry und weiteren Geräten. Jedes Betriebssystem nutzt eine etwas andere Version von WebKit, aber die Interoperabilität ist in der Regel gut.

Der Internet Explorer entspricht im Betriebssystem Windows Phone von Version 7.5 aufwärts weitgehend der Desktop-Version; ältere Modelle verwenden IE9 und aktuellere Modelle IE10. Für Android und Firefox OS steht auch Firefox zur Auswahl. Firefox OS ist erschienen, und es gibt erste Geräte, bei denen es vorinstalliert ist. Auch das wird

wahrscheinlich den Markt langfristig nachhaltig beeinflussen.

Opera schneidet sich mit Opera Mini einen großes Stück des Kuchens vom weltweiten Mobilbrowser-Markt ab. Bei diesem Browser komprimiert ein Proxy-Server die angeforderten Seiten und leitet die Daten als eine Art Bild weiter. Interaktive Funktionen werden dabei stark eingeschränkt. Opera Mini ist leicht und schnell, daher ist der Browser in Ländern mit eingeschränkten Internet-Zugangsmöglichkeiten sehr beliebt. Da allerdings interaktive Anwendungen nicht vollständig unterstützt werden, widme ich ihm nicht viel Aufmerksamkeit. Seit Mai 2013 basieren die Opera-Browser auf einer Webkit-Rendering-Engine.

Mobile vs.Desktop

Die heutigen Smartphones werden zwar immer leistungsfähiger, haben aber meist trotzdem weniger freien Arbeits- und Datenspeicher und geringere Verbindungsgeschwindigkeiten als unsere Laptop- oder Desktop-Computer. Oberflächlich betrachtet erscheinen sie dadurch vielleicht als »armer Verwandter« unter den webfähigen Geräten, doch Jonathan Stark drückt es in »The 10 Principles of Mobile Interface Design« gut aus:

> *Smartphones sind in vielerlei Hinsicht sogar leistungsfähiger als Desktop-Computer. Sie sind stark personalisiert, immer eingeschaltet und meist mit dem Internet verbunden, wir haben sie stets dabei und sie lassen sich direkt ansprechen. Außerdem sind sie vollgepackt mit leistungsstarken Sensoren, die Daten wie Standort, Bewegung, Beschleunigung, Ausrichtung, Nähe, Umweltbedingungen und mehr registrieren können.*

Natürlich liegt der Vorteil eines Mobilgeräts gerade darin – in seiner Mobilität und der Möglichkeit, außerhalb der Wohnung oder des Büros rasch Orte oder Dinge um Sie herum zu finden, Wegbeschreibungen oder Informationen abzurufen. Mobilgeräte werden aber auch immer häufiger zu Hause während des Fernsehens, auf dem morgendlichen Arbeitsweg oder aus Langeweile in einer Warteschlange eingesetzt. Bei den Mobilgeräten geht es tatsächlich weniger um Mobilität als solche,

sondern eher um die permanente Verfügbarkeit.

Die Mobilsparte verändert sich schneller als jeder andere Marktsektor, denn viele Anwender besorgen sich alle 12 bis 18 Monate ein neues Gerät, und jedes Jahr kommen mehrere hundert (wenn nicht gar tausend) neue Modelle auf den Markt.

Tablet

Tablet-Computer gibt es schon seit Jahren. Doch erst im Jahr 2010 gelang ihnen mit Apples erstem iPad der Sprung vom Büro- zum Freizeit-Accessoire. Seither sind viele weitere Geräte herausgekommen. In der mittelgroßen 7-Zoll-Klasse sind Android-Tablets besonders erfolgreich. und auch Microsoft richtet sein Betriebssystem Windows 8 stark auf Tablets aus. War der Markt für Tablets 2009 noch bestenfalls vernachlässigbar, werden 2015 schätzungsweise bereits etwa 390 Millionen Tablets in Betrieb sein. Diese Zahl ist schier unglaublich, und die Wachstumsraten steigen exponentiell.

»Tragbare« Geräte

Ihre größeren Abmessungen und die Tatsache, dass sie (häufig) auf WLAN-Verbindungen angewiesen sind, machen Tablets eher zu tragbaren als zu echten Mobilgeräten; sie sind damit näher am Laptop als am Mobiltelefon. Meist werden sie zu Hause eingesetzt. Sie sind zwar tragbar, doch in der Regel werden sie nur hervorgeholt, wenn der Anwender sich an einem festen Ort befindet und gerade nicht auf der Straße herumläuft. Trotzdem sieht man sie gelegentlich auch in der Öffentlichkeit als etwas plumpe und sperrige Kameras im Einsatz!

Genau wie Smartphones werden auch Tablets meist mit den Fingern über einen Touchscreen gesteuert (zu einigen gibt es auch einen Eingabestift). Studien haben ergeben, dass sie meist zur Unterhaltung und zum Surfen eingesetzt werden, wenn die benötigte Zeit dabei keine Rolle spielt. Die wichtigsten Browser der Tablets entsprechen im Wesentlichen denen der Mobilgeräte.

Fernseher

Webfähige Fernsehgeräte

Der Fernseher ist zu Hause immer noch der am häufigsten genutzte Bildschirm (allerdings hauptsächlich zum passiven Zusehen und eher selten interaktiv). Das ändert sich jedoch gerade mit dem steigenden Marktanteil der neuen Generation internetfähiger Smart-TVs und Set-Top-Boxen. Weltweite Zahlen sind schwer zu finden, in Europa und Nordamerika können wir Vorhersagen zufolge bis 2016 jedoch von 100 Millionen webfähiger Fernsehgeräte ausgehen.

Viele Web-TVs verwenden zwar eher Applikationen als Websites, doch deren Programmierung basiert häufig auf Webtechnologien, und teilweise ist ein Browsermodus eingebettet. Wenn Sie eine Twitter-App schreiben, sollen die Anwender ja schließlich auch Links öffnen können.

Die Fernbedienung

Der größte Nachteil internetfähiger Fernseher ist die Fernbedienung, mit der das Surfen zum reinsten Horror wird. Verschlimmert wird das noch durch monströse Konstrukte mit unzähligen Knöpfen, welche die für den Webeinsatz nötige Interaktivität bieten sollen – insbesondere auch eine Tastatur. Manche Hersteller setzen auf gesten- oder sprachgesteuerte Fernseher oder integrieren ein Trackpad in die Fernbedienung. Die beste Lösung scheint jedoch, den Fernseher mit einem Mobiltelefon oder Tablet zu koppeln und darüber interaktiv zu steuern.

Ein Gemeinschaftsgerät

Der Fernseher ist stets ein Gemeinschaftsgerät, das viele Leute gleichzeitig nutzen können – ein weiteres Hindernis beim Internet-Surfen, das sonst eher eine sehr persönliche Erfahrung ist. Möchten Sie Ihre Facebook-Seite für jedermann sichtbar auf einem großen Bildschirm übertragen? Wohl eher nicht.

In Zukunft werden Web-TVs voraussichtlich für interaktive Dienste und Video eingesetzt werden, wobei auf dem Mobiltelefon oder Tablet des Anwenders eine zusätzliche soziale Echtzeitkomponente

hinzukommt. Studien zufolge verwenden 77% der Fernsehzuschauer (wenn man sie dann überhaupt noch so nennen kann) währenddessen noch einen zweiten Bildschirm – und von diesen Personen verfolgen nur 5% eine Tätigkeit, die direkt oder indirekt mit dem Bildschirminhalt des Fernsehers in Verbindung steht.

Die anderen

Diese vier großen Kategorien – Desktop/Laptop, Mobiltelefon, Tablet und Fernseher – decken den Markt der internetfähigen Geräte weitgehend ab, jedoch durchaus nicht vollständig. Viele andere Geräte verfügen über Webbrowser, auch wenn diese nicht unbedingt allzu häufig genutzt werden.

Spielekonsolen

Alle aktuellen Spielekonsolen haben einen Webbrowser: Die PlayStation 3 verwendet NetFront (auf WebKit basierend), die Xbox 360 von Microsoft nutzt den Internet Explorer 9, die Nintendo Wii U ein eigenes Build von WebKit. Dasselbe gilt für tragbare Spielekonsolen. In der PlayStation Vita finden wir NetFront, ebenso in neueren Versionen der Nintendo 3DS (ältere Modelle nutzen Opera). Denken Sie daran, dass in jeder Konsole ein etwas unterschiedliches Kontrollsystem verwendet wird. (Mehr hierzu finden Sie in Anna Debenhams beeindruckender Arbeit zu diesem Thema, die in den Literaturempfehlungen auf Seite 12 aufgeführt wird.)

E-Book-Reader

Vergessen wir auch nicht die E-Book-Reader wie den Amazon Kindle, den Kobo, den Nook von Barnes & Noble oder die Gerätereihe von Sony. Mit »E-Book-Reader« meine ich insbesondere »e-ink«-Geräte mit elektronischer Druckfarbe. (Alle Unternehmen bieten zudem auch Versionen mit Farbdisplay an, die eher zur Kategorie der Tablets gehören.) Die Browser der E-Book-Reader basieren meist auf WebKit und sie müssen mit geringer Prozessorleistung, wenig Speicherplatz und sehr langsamen Bildwiederholraten auskommen. Für

viele wären diese Browser nicht die erste Wahl, doch E-Books gewinnen zusehends an Interaktivität und so werden möglicherweise auch diese Browser immer beliebter.

Die »Zwischendinger«

Das iPhone 4 hat eine Bildschirmdiagonale von 3,5 Zoll, das iPhone 5 hat 4 Zoll. Das Samsung Galaxy S III misst 4,8 Zoll quer über den Bildschirm; beim LG Optimus Vu sind es 5 Zoll und beim Samsung Galaxy Note II 5,5 Zoll. Googles Nexus 7 hat eine Bildschirmdiagonale von 7 Zoll und der Kindle HD Fire ist bei Amazon in den Größen 7 Zoll und 8,9 Zoll erhältlich. Beim iPad Mini beträgt das Bildschirmformat 7,9 Zoll, beim iPad sind es 9,7 Zoll.

Vielfalt der Bildschirmgrößen

Mit anderen Worten: Die Vielfalt der Bildschirmgrößen erschwert die Unterscheidung zwischen Mobiltelefon und Tablet. Als einzige Unterscheidungskriterien fallen mir ein, ob die Geräte über eine eingebaute Mobilfunkverbindung verfügen oder nicht, ob sie bequem in eine Tasche passen und ob sie sich komfortabel mit einer Hand bedienen lassen oder ob zwei Hände notwendig sind.

Aber allgemein vermischen sich die Grenzen bei der Computernutzung. Das Surface von Microsoft ist ein Tablet mit optionaler Tastatur in der Abdeckung. Wird die Tastatur mit dem Tablet verbunden, verhält sich es sich wie ein Laptop und sieht auch so aus. Auf dem Ubuntu Phone läuft Android. Wenn Sie es an einen Bildschirm anschließen, steht allerdings ein vollwertiger Linux-Desktop zur Verfügung. Das PadPhone von Asus verwandelt sich in Verbindung mit einem größeren Touchscreen vom Smartphone zum Tablet. Ihr Fernseher wird zur Medienzentrale, wenn Sie Apple TV oder Roku daran anschließen.

Diese Veränderungen werden unsere bestehende Terminologie komplett über den Haufen werfen. In absehbarer Zukunft wird dem

Wort »Tablet« dieselbe Aussagekraft zukommen wie dem Disketten-symbol, das in vielen Softwareanwendungen mit dem Befehl »Speichern« verknüpft ist. Eines Tages wird es nur noch Bildschirme geben.

Die Multiscreen-Welt

Das Konzept des Nutzers als »Mobiltelefonnutzer« oder »Tablet-Nutzer« geht in die falsche Richtung – viel wahrscheinlicher ist es, dass ein Nutzer mehrere Geräte verwendet, und zwar häufig gleichzeitig. Im Jahr 2012 ergaben im Auftrag von Google durchgeführte Studien, dass 81 Prozent der Teilnehmer beim Fernsehen ihr Smartphone verwendeten. 66 Prozent hatten es zur Hand, während sie am Computer saßen. Dieser Einsatz mehrerer Geräte wird als »gleichzeitige Nutzung« bezeichnet.

Sequenzielle Nutzung

Indessen begannen 90 Prozent der Studienteilnehmer eine Aufgabe an einem Gerät und beendeten sie auf einem anderen. Sie surften beispielsweise mit dem Smartphone in Online-Shops und wechselten dann zum Kauf auf einen Laptop. Dieser Wechsel von einem Gerät zum anderen in Abhängigkeit von der Aufgabe wird als »sequenzielle Nutzung« bezeichnet.

Mit anderen Worten werden die Besucher Ihrer Website zwei oder drei verschiedene Geräte verwenden. Und auch wenn sie bei jedem Besuch leicht unterschiedliche Ziele verfolgen, wollen sie doch jeweils auf dieselbe Information zugreifen können.

Kontext: Was wir nicht wissen

Zu Hause, bei der Arbeit, im Bus...?

Denken wir vor allem daran, was wir über die Nutzer unserer Sites nicht wissen. Wir wissen

nicht, wo sie sich aufhalten – damit meine ich
nicht die geografische Position, denn die können wir über GPS ermitteln. Ich meine, ob sie zu Hause, bei der Arbeit oder im Bus sind und so weiter. Wir haben also eigentlich keine Ahnung, in welchem Kontext jemand unsere Site besucht.

Meist wird davon ausgegangen, dass Mobilgerätenutzer unterwegs und in Eile sind und dass ihnen häufig nur 3G-Verbindungen mit geringer Bandbreite zur Verfügung stehen. Das trifft allerdings nicht immer zu: Mobiltelefone kommen häufig zu Hause an einer guten, schnellen WLAN-Verbindung zum Einsatz. Doch auch die Verbindungsart sagt nicht alles aus; vielleicht ist die Signalqualität schlecht oder die Bandbreite wegen überfüllter Kanäle in der Nachbarschaft reduziert. Tatsächlich bieten 3G-Netzwerke (oder in vielen Ländern auch 4G) in der Realität häufig viel bessere Verbindungsgeschwindigkeiten als WLAN.

Wir nehmen außerdem an, dass Mobiltelefone weniger Rechenleistung besitzen als Desktops, doch selbst dies könnte sich bald ändern. Es gibt jetzt schon Smartphones mit Dual- oder Quadcore-CPU, und wer weiß, wie leistungsfähig die Geräte in ein oder zwei Jahren sein werden. Tablet-Laptop-Hybriden, zum Beispiel viele Windows-8-Geräte, sind bereits jetzt leistungsstärker als nur wenige Jahre alte Laptops.

Dann ist da noch das Thema Größe. Wir gehen davon aus, dass die Betrachter eines großen Bildschirms (etwa eines Fernsehgeräts) eine aufwendigere grafische Umgebung wünschen. Ein großer Bildschirm geht aber nicht unbedingt mit einem leistungsfähigen Prozessor oder einer schnellen Internetverbindung einher. Die Prozessoren vieler Web-TVs sind nicht stärker als die eines Smartphones, und für die Verbindungsgeschwindigkeit gelten ebenso viele Unbekannte wie für jedes andere Gerät. Mit der Größe kommt auch die Frage nach der Tragbarkeit auf. Je eher ein Gerät tragbar ist, desto weniger können wir den Nutzungskontext vorhersagen.

Die wichtigste Regel lautet deshalb also: Wir dürfen keinerlei

Annahmen treffen. Kaum habe ich es gesagt, breche ich diese Regel schon wieder:

Einige Kontext-Klischees

Um Wiederholungen im Buch zu vermeiden, zeige ich hier einige typische Beispielszenarien auf. Es sind nicht unbedingt die gängigsten Szenarien, aber ich denke, sie kommen für unseren Zweck oft genug vor.

Mobiles Szenario

Ein Beispiel: Mobiltelefonnutzer haben keine superschnelle Internetanbindung. Häufig nutzen sie ihre Geräte zwar von zu Hause aus mit megaschneller Glasfaseranbindung über ein ungestörtes WLAN-Netzwerk aus einem Meter Entfernung. Aber ebenso häufig trifft das Gegenteil zu: Sie sind nicht zu Hause und bekommen nur ein sehr schwaches 3G-Signal (auch mir passiert das oft). Viele Smartphones sind auf solche Anwendungsfälle ausgelegt; sie beschränken die Anzahl zulässiger Verbindungen, um kein wertvolles Datenvolumen aus den limitierten Tarifen der Nutzer zu verschwenden.

Desktop-Szenario

Ebenso wird ein Desktopnutzer wahrscheinlich über eine direkte Breitbandverbindung verfügen, die schnelle Datentransferraten erlaubt. Natürlich stimmt das nicht immer – in ländlichen Gegenden sind die maximal möglichen Verbindungsgeschwindigkeiten teils extrem eingeschränkt oder die Nutzer sind sogar noch auf Einwahlverbindungen angewiesen –, das erste Szenario ist jedoch so häufig anzutreffen, dass ich es stellvertretend verwenden kann.

Solche Vereinfachungen nutze ich im gesamten Buch, damit ich mich nicht ständig wiederholen und die jeweiligen Szenarien klarstellen muss. Ich kann aber nicht deutlich genug sagen, dass solche Annahmen für Sie beim Planen und Erstellen von Websites und Applikationen nicht an erster Stelle stehen sollten und dürfen.

Nur die Geschwindigkeit zählt

Nur die Geschwindigkeit zählt – und dies ist gleichzeitig eine Herausforderung: Sie kennen Ihre Anwender nicht, wissen nicht, wo sie sind, was sie tun oder welches Gerät sie verwenden. Manches davon (aber nicht alles) können Sie herausfinden. Den gesamten Nutzungskontext können Sie aber nie erfassen; außerdem ändert er sich von Anwender zu Anwender. Als einigermaßen sicher können Sie nur voraussetzen, dass die Nutzer auf Ihr Angebot zugreifen oder aber herausfinden möchten, dass es ihnen eben nicht zusagt. Was immer davon zutrifft: Die Anwender wollen eine schnelle Antwort haben.

Performance ist das einzig wichtige Kriterium. Ob die Anwender in der Rushhour auf dem Smartphone nach der nächsten Bahnverbindung suchen oder zu Hause auf dem Sofa eine Shopping-Website durchstöbern – sie möchten diese Aufgabe so schnell wie möglich abschließen. Wenn sie dabei das ihnen am nächsten gelegene Gerät verwenden, haben sie das Gefühl, effizienter zu sein.

Ihre Site muss sich »schnell« anfühlen.

Ihre Site muss auf jedem möglichen Gerät schnell sein – und sich auch schnell anfühlen. Mit »schnell« ist hier nicht nur die technische Leistung gemeint (die äußerst wichtig ist), sondern auch das Ansprechverhalten der Benutzeroberfläche, wie einfach sich die Seitennavigation dem Benutzer darstellt und wie schnell er sein ursprünglich angestrebtes Vorhaben abschließen kann.

Das werden Sie lernen

Zuerst will ich Ihnen geradeheraus sagen, was ich Ihnen hier nicht beibringen werde: ein Design, das auf die Stärken jeder Plattform abgestimmt ist. Das geht nicht, weil ich Ihnen nichts beibringen kann, wovon ich selbst nichts verstehe. Ich bin aus gutem Grund nicht Designer, sondern Entwickler geworden. Wenn ich Photoshop

verwende, sieht es hinterher so aus, als hätte ich die Maus mit Boxhandschuhen bedient.

Programmiermethoden und Techniken

Bei mir lernen Sie moderne Programmiermethoden und Techniken zur Entwicklung von Websites, die entweder auf mehreren Geräten laufen oder auf eine bestimmte von Ihnen angepeilte Geräteklasse zugeschnitten sind. (Die entsprechenden Technologien werden alle in Kapitel 1 erklärt, darum gehe ich hier nicht näher darauf ein.) Behalten Sie bei der Lektüre dieses Buchs die beiden folgenden, sehr wichtigen Punkte im Hinterkopf:

Es gibt so viele unterschiedliche Technologien, dass ich sie nicht alle behandeln kann. Ich vermittle Ihnen die aus meiner Sicht wichtigsten Techniken und Technologien, die Sie zur Entwicklung von Webprojekten für die gesamte Gerätepalette benötigen.

Nicht alles in diesem Buch wird sich großflächig durchsetzen können – zumindest nicht in der hier gezeigten Form. Das Web entwickelt sich ständig weiter, ein gedrucktes Buch kann immer nur eine einzelne Momentaufnahme darstellen. Manche Dinge werden sich verändern, manche werden verkümmern und wegfallen. Ich versuche dies etwas zu entschärfen, indem ich nur auf offenen Standards basierende Technologien behandle, die auch bereits zu einem gewissen Grad in den Browsern implementiert sind.

Bleiben Sie am Ball!

Für Sie als Webentwickler ist Folgendes wichtig: Informieren Sie sich, bleiben Sie immer am Puls der Webstandard-Entwicklung, seien Sie neugierig und verspielt, halten Sie Augen und Ohren offen. Sie haben das Glück, in einer allein auf Wissensaustausch basierenden Branche zu arbeiten. Behalten Sie also einige der Menschen und Websites, die ich in diesem Buch erwähne, im Blick. Suchen und finden Sie auch Ihre eigenen Quellen, nutzen Sie Twitter, gehen Sie zu Treffen Ihrer lokalen Webentwickler-Community. Bleiben Sie am Ball und seien Sie aktiv. Nie war Webentwicklung spannender als jetzt – Sie müssen allerdings eine Sonderschicht einlegen, um wirklich davon zu

profitieren. Behalten Sie bei jedem Projekt vor allem auch immer den größeren Gesamtzusammenhang im Blick. Wenn Sie eine Website programmieren, entwickeln Sie nicht »eine Website und eine Mobilsite« – streben Sie stattdessen eine Site an, die überall funktioniert. Überlegen Sie sich, wie die Nutzer die Site verwenden werden, was sie sich davon erhoffen und was Sie als Entwickler tun können, um sie bei dieser Zielsetzung zu unterstützen – nicht nur jetzt, sondern auch in Zukunft. In den letzten fünf Jahren haben wir so umwälzende Entwicklungen im Web erlebt – wer kann schon sagen, was wir dort in weiteren fünf Jahren antreffen werden?

Literaturempfehlungen[1]

Die in diesem Kapitel genannten Statistiken stammen aus vielen verschiedenen Quellen, vor allem *The Mobile Industry in Numbers* von VisionMobile auf http://www.visionmobile.com/blog/2012/10/infographic-the-mobile-industry-in-numbers/ sowie »The Internet of Things« von Cisco auf http://blogs.cisco.com/news/the-internet-of-things-infographic/.

Eine gute Einführung in das Internet of Things finden Sie im *The Next Web*-Artikel *Why 2013 will be the year of the Internet of Things*: http://thenextweb.com/insider/2012/12/09/the-future-of-the-internet-of-things/.

David Storey schrieb einen hervorragenden Beitrag über das nicht auf Smartphones ausgerichtete mobile Web: *See your site like the rest of the world does. On the Nokia X2-01*. Sie finden ihn auf http://generatedcontent.org/post/31441135779/mobileweb-row/.

Der beste Artikel, den ich über die Webentwicklung für mobile Geräte gelesen habe und aus dem ich in diesem Kapitel zitiere, ist *The 10*

Principles of Mobile Interface Design von Jonathan Stark: http://www.netmagazine.com/features/10-principles-mobile-interface-design/. Jason Grigsbys hervorragender Artikel *Responsive Design for Apps* ist ein guter Einstieg in die Gestaltung für unterschiedliche Bildschirmabmessungen und -darstellungen: http://blog.cloudfour.com/responsive-design-for-apps-part-1/. Der im *UX Magazine* erschienene Artikel *Five Lessons from a Year of Tablet UX Research* von Brennan Browne enthält herausragende Erkenntnisse über die Nutzung von Tablets: http://uxmag.com/articles/five-lessons-from-a-year-of-tablet-ux-research/.

Ein guter Einstieg für Anna Debenhams Untersuchungen über Spielekonsolen-Browser ist ihr Artikel *Testing Websites in Game Console Browsers* auf *A List Apart*: http://www.alistapart.com/articles/testing-websites-in-game-console-browsers/.

Noch einmal Jason Grigsby: Er hielt den wunderbaren Vortrag *The Immobile Web* über die Entwicklung für TV-Geräte. Das Video finden Sie auf http://vimeo.com/44444464/, die zugehörigen Präsentationsfolien auf http://www.slideshare.net/grigs/the-immobile-web/.

Die komplette Untersuchung zur Multi-Device-Nutzung finden Sie im Google-Blog-Beitrag *Navigating the new multi-screen world* auf http://googlemobileads.blogspot.co.uk/2012/08/navigating-new-multi-screen-world.html.

Es ist immer gut, zukunftsweisende Websites zu gestalten. Mehr auf http://futurefriend.ly/.

Im Anhang B finden Sie die Literaturempfehlungen aller Kapitel zusammengefasst. Auf der dpunkt-Webseite zum Buch (www.dpunkt.de/gasston) finden Sie die Hinweise außerdem online

und mit anklickbaren Links.

1 Die Webplattform

In diesem Kapitel spreche ich die Webtechnologien an, die Sie in diesem Buch kennenlernen werden. Ich gehe auf das notwendige Hintergrundwissen ein, damit Sie das Maximum aus Ihrer Lektüre herausholen können, und ich beschreibe die Anforderungen und Voraussetzungen für die Multi-Device-Entwicklung. In diesem Kapitel möchte ich sozusagen sicherstellen, dass wir uns auch im übertragenen Sinne auf derselben Seite befinden, bevor wir uns in den nächsten Kapiteln wirklich den technischen Details widmen.

Wenn Sie sofort weiterlernen möchten, würden Sie vielleicht am liebsten gleich zu Kapitel 2 blättern. Ich bitte Sie aber, dies nicht zu tun, denn das vorliegende Kapitel 1 enthält einiges an sehr interessantem und nützlichem Hintergrundwissen. Der wenig technische Stoff erlaubt mir zudem, die besten Beispiele meines wundervollen Humors zu präsentieren.

1.1 Ein kurzer Hinweis zu den Begriffen

Websites

Im Buch spreche ich immer wieder von der Website- oder Site-Programmierung. Diese Begriffe sind zwar eine Vereinfachung, helfen mir aber, Wiederholungen zu vermeiden. Das in diesem Buch Gelernte lässt sich auf Websites, Webanwendungen und hybride Apps für Mobilgeräte anwenden, kurz gesagt: auf alles, was HTML, CSS und JavaScript verwendet. Weil das aber ein richtiger Rattenschwanz ist, spreche ich meist von »Websites« – es sei denn, ich muss es präzisieren.

Browser und User Agents

Ich spreche auch wahlweise von »Browsern« oder »User Agents«, wenn ich eine beliebige Softwareinstanz meine, die Websites oder Applikationen wiedergeben kann. Auch hiermit möchte

ich ganz einfach Wiederholungen vermeiden.

1.2 Wer Sie sind und was Sie wissen müssen

Lassen Sie mich vorab klären, welche Annahmen ich über Sie treffe. Außerdem möchte ich Ihnen mitteilen, welches Hintergrundwissen Sie benötigen, um den bestmöglichen Nutzen aus diesem Buch zu ziehen.

HTML, CSS und JavaScript

Sprechen wir zuerst von Ihnen. Egal ob Sie bereits Profi sind, gerne einer werden wollen oder ob Sie einfach nur Spaß daran haben, mit dem Web herumzuspielen: Sie besitzen bereits brauchbare HTML-, CSS-und JavaScript-Kenntnisse. Diese müssen nicht allzu tief gehen oder besonders detailliert sein, aber doch so gut, dass ich Ihnen nicht beibringen muss, was HTML, CSS und JavaScript sind und wie diese Sprachen geschrieben werden.

Website-Programmierung

Vielleicht haben Sie die Website-Programmierung schon vor einiger Zeit erlernt und müssen Ihre Fähigkeiten auf den neuesten Stand bringen; vielleicht lernen Sie Webentwicklung in der Schule und wünschen sich noch einige Extrastunden. Oder Sie arbeiten bereits als Entwickler, haben aber keine Gelegenheit, die neuesten Trends in der Webprogrammierung zu verfolgen. Wenn eine dieser Beschreibungen auf Sie zutrifft, dann möchten Sie sicher gerne lernen, wie Websites auf moderne Weise programmiert werden, sodass sie auch auf unterschiedlichen Geräten laufen und sich den Fähigkeiten und Abmessungen dieser Geräte anpassen – zweifellos haben Sie deshalb zu diesem Titel gegriffen.

Dieses Buch baut auf Ihren Kenntnissen als Webentwickler auf. Es ist kein Leitfaden für Anfänger, aber auch kein Fortgeschrittenenbuch. Es ist vielmehr eine Momentaufnahme der aktuellen, neuen und in naher Zukunft relevanten Funktionen von HTML, CSS, JavaScript und der mit ihnen verwandten Technologien. Ein Schwerpunkt liegt dabei auf

den Techniken, die am besten zur Programmierung von Sites für die vielfältige Gerätewelt geeignet sind.

Entwicklertools

Neben Ihrem Grundwissen müssen Sie sich auch noch mit den Entwicklertools Ihres Browsers auskennen; Sie müssen hier allerdings kein Power-User sein. In einigen JavaScript-Beispielen gebe ich die Ergebnisse in der Entwicklerkonsole eines Tools aus. Dies ist eine Standardmethode, bei der es keine Rolle spielt, ob Sie die integrierten Funktionen von Chrome, Firefox, IE9+, Opera oder Safari oder Drittanbieterlösungen wie Firebug nutzen. Eine meiner Codezeilen könnte etwa so aussehen:

```
console.log('Hello World');
```

Und das Ergebnis wird in der Konsole ausgegeben; Abb. 1–1 zeigt diese Ausgabe in Firebug. Ich verwende die Konsole oder Entwicklertools, wie gesagt, recht selten. Wenn Sie aber nicht damit vertraut sind, sollten Sie sich nun wirklich etwas Zeit nehmen, um den Umgang mit der Konsole zu lernen.

```
console.log('Hello World!');
Hello World!
```

Abb. 1–1
Eine Hello-World-Nachricht als Konsolenausgabe in Firebug

Wenn Sie immer noch weiterlesen, dann haben Sie entweder die nötigen Kenntnisse oder Sie versuchen, sich durchzumogeln. Wie dem auch sei – nun sprechen wir über die Technologie.

1.3 Begriffsklärung

HTML5

Es herrscht einige Unklarheit darüber, was HTML5 überhaupt genau ist. Da ist

einerseits das, was die breite Öffentlichkeit
(und ein Großteil unserer Kunden) darunter versteht, andererseits das wirkliche Wesen von HTML5. Es handelt sich um keine brandneue Plattform zur Programmierung von Websites; es ist keine Rich-Media-Umgebung; es ist keine Funktion, die Sie einschalten können, damit Ihre Websites auf unterschiedlichen Geräten laufen. HTML5 ist im Wesentlichen ein Versuch, das Web so weiterzuentwickeln, dass es unseren heutigen Bedürfnissen entspricht. Denn diese Anforderungen haben sich seit der ersten Implementierung von HTML als simples Netzwerk aus verknüpften Dokumenten dramatisch geändert.

Die »Webplattform«

Für die breite Öffentlichkeit ist HTML5 zu einem Oberbegriff für eine Reihe miteinander verwandter und ineinandergreifender Technologien wie etwa CSS3, SVG, JavaScript-APIs und mehr geworden. Auch wenn einige Entwickler sich gerne diese weiter gefasste Bedeutung zu eigen machen, gefällt mir persönlich die Verschmelzung all dieser Technologien nicht besonders gut. Ich nenne HTML5 daher lieber die »Webplattform«. Im Grunde genommen bevorzuge ich sogar den von Bruce Lawson vorgeschlagenen Begriff »New Exciting Web Technologies (NEWT)«. NEWT ist eine coole Abkürzung, und es gibt außerdem ein niedliches Logo dazu. Allerdings hat er sich zugegebenermaßen leider nie durchgesetzt – also sprechen wir von der Webplattform.

Die Webplattform ist riesig. Um sich ihren Umfang zu verdeutlichen, werfen Sie einen Blick auf *http://platform.html5.org*. Hier werden alle Technologien aufgeführt, die zur Plattform gezählt werden. Die Liste ist wirklich eindrucksvoll lang – sie umfasst viel mehr, als ich je in diesem Buch abzuhandeln wage.

Stattdessen konzentriere ich mich auf den Kern, auf diejenigen Technologien, die mir zur Entwicklung von Websites für unterschiedliche Geräte ausreichend und hilfreich erscheinen: HTML5, CSS3, SVG, Canvas und einige Geräte-APIs. Ich erkläre sie im Verlauf des Buchs jeweils im Einzelnen. Zunächst möchte ich aber genauer

darauf eingehen, was mit HTML5 und CSS3 gemeint ist.

1.4 Das echte HTML5

HTML5 ist eine Weiterentwicklung von HTML 4.01 mit einigen neuen Funktionen, ein paar als veraltet gekennzeichneten oder entfernten Funktionen und einigen bestehenden Merkmalen, deren Funktionsweise geändert wurde. Das Ziel war, die von den Entwicklern jahrelang eingesetzten Hacks und Design-Muster zu standardisieren, um den Ansprüchen des modernen Web gerecht zu werden. Und dabei geht es (mindestens) ebenso sehr um Applikationen wie um Dokumente. Der ursprüngliche Vorschlag für das heutige HTML5 hieß daher auch Web Application 1.0.

Neue Möglichkeiten

Zu den neuen HTML5-Funktionen gehören Möglichkeiten zur Strukturierung von Dokumenten, sodass diese sinnvoller und besser zugänglich werden. Damit beschäftige ich mich in Kapitel 2. HTML5 umfasst zudem eine ganze Palette neuer Möglichkeiten für Formulare und Bedienelemente, mit denen sich Anwendungen leichter entwickeln lassen – diese sehen wir uns in Kapitel 8 an. Und HTML5 beinhaltet auch die Funktion, die viele in erster Linie damit assoziieren: native Videowiedergabe (ohne Plugin). Dies besprechen wir in Kapitel 9.

WHATWG und W3C

Zwei große Gruppen arbeiten an HTML5, und ihre Rollen und Verantwortungsbereiche teilen sich – grob betrachtet – folgendermaßen auf: Das WHATWG (Sie brauchen die Bedeutung dieser Abkürzung nicht zu kennen), ein Konsortium aus Browserherstellern und »interessierten Parteien« erstellt durch den Federführer Ian Hickson eine »Live-Spezifikation« von HTML. Im Grunde ist das eine versionslose Spezifikation, in der ständig neue Funktionen integriert und bestehende aktualisiert werden. Das W3C (World Wide Web Consortium), das Gremium zur Standardisierung des

Web, fertigt Momentaufnahmen dieser Spezifikation an und versieht sie mit Versionsnummern. Dabei wird auch die Kompatibilität der Implementierung durch die Browserhersteller sichergestellt.

In Wirklichkeit ist die Situation etwas komplizierter; es gibt diverse politische Verwicklungen. Dies ist jedoch nur für Standardisierungsspezialisten wichtig und hat für Sie keine praktische Bedeutung.

Das W3C hat vorgeschlagen – zum Zeitpunkt, als dieses Buch entstanden ist, jedoch noch nicht abgesegnet –, dass HTML5 (die Momentaufnahme des W3C) als offizielle Empfehlung im Jahr 2014 veröffentlicht werden, also »fertig« sein soll. HTML5.1 soll 2016 folgen. Außerdem soll HTML5 nicht aus einer einzigen monolithischen Spezifikation, sondern aus verschiedenen Einzelmodulen bestehen. Dadurch kann die Arbeit an Einzelaspekten fortgeführt werden, ohne dass sich dadurch alles verzögern würde. Diese Jahresangaben sind für Sie jedoch überhaupt nicht von Bedeutung, Sie müssen nur wissen, wann HTML5 von den Browsern interpretiert und damit von Ihnen genutzt werden kann.

1.4.1 Das HTML5-Gerüst

Wenn Sie elementare HTML-Kenntnisse haben, sind Sie mit den Grundlagen der Seitenauszeichnung vertraut. In HTML5 gibt es jedoch in dieser Hinsicht einige Änderungen – nicht viele, aber sie sollten dennoch erwähnt werden. Der folgende Codeblock zeigt das grundlegende Gerüst, das ich für alle Beispiele in diesem Buch verwende (Sie finden es auch in der Beispieldatei *template.html*):

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="utf-8">
<title></title>
</head>
<body></body>
```

```
</html>
```

Doctype

Dieser Code sollte Ihnen größtenteils vertraut erscheinen. Ich möchte jedoch auf zwei interessante Punkte hinweisen. Der erste ist der *Doctype*. Dieser ist ein Überbleibsel aus der Zeit, als Sie dem Browser mitteilen mussten, welche Art Dokument Sie erstellten – HTML Strict, HTML Transitional, XHTML1.1 und so weiter. In HTML5 ist das nicht mehr notwendig – es gibt nur eine einzige HTML-Ausprägung. Deshalb ist die Doctype-Deklaration überflüssig geworden – jedenfalls in der Theorie. Moderne Browser haben üblicherweise drei Darstellungsmodi: Der *Quirks-Modus* simuliert die nicht-standardgemäße Darstellung des Internet Explorers 5. Dies ist für die Kompatibilität mit veralteten Webseiten notwendig. Der *Standardmodus* ist für moderne, standardkonforme Webseiten gedacht, und der »Beinahe-Standardmodus« (*Almost Standards Mode)* entspricht dem Standardmodus mit ein paar Verhaltensweisen des Quirks-Modus.

Der Browser findet anhand des Doctype heraus, welchen Modus er verwenden soll. Sie sollten stets den Standardmodus verwenden, und der Doctype ist in HTML5 die schnellste Möglichkeit, den Standardmodus auszulösen:

```
<!DOCTYPE html>
```

Das meta-Element

Der zweite wichtige Punkt und die einzige andere Änderung am Standard-HTML5-Gerüst ist das `meta`-Element. Es definiert, welcher Unicode-Zeichenbereich zur Darstellung des Textes auf der Seite verwendet werden soll – im Web ist UTF-8 der Standard. Also sollten Sie ihn in den meisten Fällen verwenden. Das `meta`-Element verwendet das Attribut `charset`:

```
<meta charset="utf-8">
```

Das war es schon. Wenn ein Kunde Sie bittet, seine Website in HTML5 umzuwandeln, können Sie diese beiden Elemente aktualisieren und ihm

eine dicke Rechnung stellen. (Bitte tun Sie das nicht – dies war nur ein Scherz.)

Ich könnte viele weitere Optionen nennen. Im Sinne der Verständlichkeit und Einfachheit habe ich sie hier weggelassen. Die bekannte Website *HTML5 Boilerplate* bietet ein umfassendes HTML-Gerüst, das in der Dokumentation ausführlich erläutert wird – denken Sie aber bitte daran, dass es als Ausgangspunkt und nicht als unumstößliches Gesetz betrachtet werden sollte.

1.4.2 Neue Standardverfahren

Außer den Änderungen am Grundgerüst gibt es in HTML5 ein oder zwei neue Standardverfahren, die Sie sich möglicherweise zu eigen machen sollten. Da HTML5 den vielen unterschiedlichen Programmierstilen der Entwickler explizit Rechnung trägt, sollten Sie auch diese Verfahren nicht als unumstößliche Regeln betrachten. Meiner Ansicht nach haben Sie es dadurch jedoch leichter, Ihren Code zu schreiben und zu pflegen.

Das type-Attribut

Zunächst sollten Sie sich daran gewöhnen, dass Sie das `type`-Attribut für die meisten externen Quellen nicht mehr brauchen. In HTML 4.01 oder XHTML mussten Sie für jedes `link`-, `script`- oder `style`-Element einen `type` angeben:

```
<link href="foo.css" rel="stylesheet" type="text/css">
<script src="foo.js" type="text/javascript"></script>
```

Im Web sind CSS und JavaScript jedoch die mit diesem Element verwendeten Quasi-Standard-Ressourcen. Deshalb ist es ein wenig redundant, jedes Mal den `type` anzugeben. Aus diesem Grund können Sie ihn nun weglassen. Ihr Code wird dadurch etwas kürzer, kann von den Browsern aber nach wie vor bestens interpretiert werden:

```
<link href="foo.css" rel="stylesheet">
<script src="foo.js"></script>
```

Nur wenn Sie kein Standard-CSS oder -JavaScript verwenden, müssen Sie den `type` dieser Elemente angeben. Beispielsweise enthalten einige Firefox-Versionen experimentelle Implementationen der neuesten JavaScript-Versionen. Aus Sicherheitsgründen ist es in diesem Fall erforderlich, dass Sie das `type`-Attribut mit einem Flag versehen:

```
<script src="foo.js" type="application/javascript;version=1.8">
</script>
```

Nachsichtig bei der Syntax

Auch was die Syntax angeht, ist HTML5 äußerst nachsichtig. Ob Sie nun gerne alles ganz in Kleinbuchstaben schreiben, Ihre Attribute mit Anführungszeichen versehen oder leere Elemente schließen – HTML5 wird Ihren Code problemlos parsen und verstehen. Die folgenden Codezeilen sind daher gleichwertig:

```
<img src=foo.png>

<img src=foo.png />

<IMG SRC="foo.png"/>
```

Hinweis

Attributwerte mit mehreren Werten (etwa einer Folge von Klassennamen) oder mit bestimmten Sonderzeichen benötigen Anführungszeichen.

Manche Attribute, sogenannte *boolesche Attribute*, können nur wahr oder falsch, genauer gesagt: *true* oder *false* sein. Wenn Sie nichts anderes vorgeben, werden sie als *true* betrachtet. Deshalb müssen Sie keinen Wert angeben – es sei denn, Sie verwenden eine XML-ähnliche Syntax, in der Werte vorgeschrieben sind. In diesem Fall nehmen Sie den Namen des Attributs selbst. Das heißt, dass die beiden folgenden Codezeilen gleichwertig sind:

```
<input type="checkbox" checked>
```

```
<input type="checkbox" checked="checked">
```

Ich selbst schreibe gerne alles in Kleinbuchstaben und alles mit Anführungszeichen, und leere Elemente schließe ich nicht:

```
<img src="foo.png">
```

Diesen Stil verwende ich im gesamten Buch, weil ich ihn eleganter finde und weil ich leichter damit arbeiten kann. Mein Texteditor verfügt über eine Syntaxhervorhebung; deshalb kann ich den Code leicht prüfen. Verwenden Sie das von Ihnen bevorzugte System, dieses jedoch konsistent. So lässt sich Ihr Code leichter pflegen.

1.5 CSS3 und mehr

CSS3 verhält sich zu CSS 2.1 wie HTML5 zu HTML 4.01: Es ist ein Evolutionsprozess, der bestimmte vorhandene, aber in den Browsern bisher unterschiedlich implementierte Funktionen standardisiert und außerdem einen komplett neuen Satz Funktionen einbringt. Dadurch ist CSS für eine Welt gerüstet, in der Webbrowser überall eingesetzt werden können.

Präsentation

Die ersten CSS3-Funktionen, die es in die Browser schafften, dienten größtenteils der Präsentation der Seiten und basierten auf Hacks, die die Entwickler schon seit Jahren verwendeten: die Nutzung von Schriften aus jeder beliebigen Quelle, abgerundete Ecken und Schlagschatten für Texte und Rahmen. Es folgten zahlreiche neue Selektoren, um die Auswahl von Elementen sowie dynamische Effekte wie etwa zwei- und dreidimensionale Übergänge und Übergangsanimationen zu vereinfachen. Mehr darüber erfahren Sie bei Bedarf in meinem Buch *The Book of CSS3* (erschienen bei No Starch Press).

Media Queries

Die wirkliche CSS3-Revolution waren jedoch nicht die ganzen glitzernden Effekte,

sondern die *Media Queries* – eine Syntax, mit der Sie den Browsern je nach deren Abmessungen und Möglichkeiten unterschiedliche Stildefinitionen bieten können. Dies war der erste Schritt in Richtung einer echten Multi-Device-Gestaltung. Media Queries sowie verschiedene andere CSS-Eigenschaften, die für die Entwicklung responsiver und adaptiver Websites geeignet sind, behandle ich in Kapitel 3.

Layout

Die nächste große Herausforderung für CSS ist das Layoutproblem – die Gestaltung von Layouts, die für den jeweiligen Browser bzw. User Agent wirklich geeignet sind. Dazu gehören Eigenschaften für dynamische Benutzeroberflächen und CSS-gesteuerte Rastersysteme, über die Sie in Kapitel 4 mehr erfahren werden.

CSS3 ist keine einzelne Spezifikation wie CSS 2.1, das in einem einzigen Dokument vollständig beschrieben werden konnte. Dazu ist CSS3 viel zu umfangreich und komplex. CSS3 ist vielmehr modular – eine Reihe kürzerer, spezifischer Spezifikationen, die von den Browsern auf modulare Weise implementiert werden können. Wie bei HTML5 wäre es ziemlich dumm, zu warten, bis CSS3 »fertig« ist, denn manche Module werden lange vor den übrigen fertig und implementiert sein.

CSS3 steht für »alles, was neuer ist als CSS 2.1«

Die CSS-Module werden durchnummeriert, um anzuzeigen, wie viele Entwicklungsschritte sie schon hinter sich haben. Manche sind bereits bei Version bzw. Level 4 – diese sollten gegenüber solchen mit Level 3 möglichst bevorzugt werden. Das heißt jedoch nicht, dass es irgendwann ein CSS4 geben wird – so wird es nicht sein. CSS3 ist das Kürzel für alles, was »neuer ist als CSS 2.1«. Eines Tages wird es diese Unterscheidung nicht mehr geben, sondern einfach nur noch CSS.

1.5.1 Herstellerspezifische Präfixe

Wenn die Browserhersteller bestimmte Funktionen auf experimentelle

Experimentelle Implementierungen

Art oder vorläufig implementieren, verwenden sie herstellerspezifische Präfixe, um Kompatibilitätsprobleme mit standardisierten Eigenschaftennamen zu vermeiden. Stellen Sie sich beispielsweise vor, dass ein Menschenaffen-CSS-Modul eine neue Eigenschaft namens `gorilla` einführt und dass sowohl Firefox als auch WebKit diese experimentell, aber leicht unterschiedlich implementieren.

Würden beide denselben Eigenschaftsnamen verwenden, hätte dies in jedem Browser unterschiedliche Auswirkung. Deshalb vermeiden Browserhersteller diesen potenziellen Konflikt durch herstellerspezifische Präfixe, die meist als Vendor-Präfixe bezeichnet werden:

```
-moz-gorilla: foo;
-webkit-gorilla: foo;
```

Vom Prinzip her ist dies ein hervorragendes System; in der Realität ist es jedoch etwas verwirrend. Unter anderem wurden manche mit Präfix versehene Eigenschaften von den Entwicklern so stark genutzt, dass einige Browserhersteller das Bedürfnis hatten, die Vendor-Präfixe ihrer Konkurrenten ebenfalls zu implementieren. Das ist zwar verständlich, macht die ganze Sache jedoch sinnlos.

Die Browserhersteller versuchen inzwischen, die Verwendung der herstellerspezifischen Präfixe einzuschränken. Manchmal wird es aber nahezu unvermeidlich sein, Eigenschaften mit Vendor-Präfix zu verwenden. Mit ein paar Ausnahmen enthalten meine Codebeispiele nur Eigenschaften ohne Präfix. In Anhang A finden Sie einen Hinweis, an welchen Stellen Vendor-Präfixe notwendig sind.

1.5.2 CSS-Frameworks und Präprozessoren

Heutzutage sind Hilfsmittel bei der CSS-Entwicklung geradezu unerlässlich, besonders bei der Arbeit in großen Entwicklerteams

und/oder an umfangreichen Projekten. Normalerweise treten diese Helfer in Form von Frameworks oder Präprozessoren oder oft einer Kombination daraus in Erscheinung.

Framework

Ein Framework ist ein Satz vordefinierter CSS-Regeln, die Sie für die schnelle Entwicklung verwenden können; normalerweise beinhalten sie typografische Elemente, Formulare und recht häufig auch Layoutkomponenten. Eines der besseren Frameworks, das in vielen bekannten Websites verwendet wird, ist *Blueprint.css*. Das bekannteste aktuelle Framework ist jedoch *Bootstrap* von Twitter. Es bietet viele vorformatierte Layout-, Typografie- und Formularoptionen, zahlreiche wiederverwendbare Komponenten und auch eine Erweiterbarkeit durch JavaScript.

Präprozessoren

Präprozessoren sind meist serverseitig ausgeführte Programme, die Erweiterungen und Syntaxkürzel in einer CSS-artigen Sprache bieten. Diese werden zur Laufzeit in korrekt formatierte Stylesheets transformiert. Die Erweiterungen enthalten zeitsparende Features wie Variablen und verschachtelte Regeln sowie benutzerdefinierte Funktionen, die dem Anwender fantastische Möglichkeiten bieten. Die Hauptkonkurrenten im Präprozessor-Bereich sind *LESS* und *Sass*. Sass ist der bekanntere der beiden Präprozessoren.

Zwar haben sowohl Frameworks als auch Präprozessoren einen festen Platz in der modernen Webentwicklung, ich beschäftige mich in diesem Buch jedoch nicht damit, da ich Ihnen eher die sprachlichen Grundlagen – auf denen auch die soeben erläuterten Hilfsmittel basieren – nahebringen möchte.

1.6 Browserunterstützung

Unterschiedliche Implementierungen

Nun ist Ihnen sicherlich klar, dass das Multi-Device-Web unglaublich umfang- und

variantenreich ist, dass eine immense Vielzahl
Browser auf den zahlreichen verschiedenen Geräten läuft und dass es sogar innerhalb dieser Browser eine Fülle von verschiedenen Versionen und Implementierungen gibt (ich hoffe es zumindest, denn im größten Teil der Einleitung habe ich versucht, genau dies zu vermitteln). Aus diesem Grund kann es sehr wohl sein, dass manche in diesem Buch besprochenen Funktionen nicht oder etwas anders implementiert sind.

Statt mich im Text mit den Stufen der Implementierung zu beschäftigen, behandle ich alle neuen Funktionen so, als wären sie vollständig implementiert, und weise in Anhang A auf Ungereimtheiten und Kuriositäten in der realen Welt hin.

Außerdem sind bei brandneuen, innovativen Vorschlägen Änderungen stets vorbehalten – selbst wenn mit manchen Browsern bereits experimentelle Implementierungen ausgeliefert wurden (das in Kapitel 4 beschriebene CSS3-Rasterlayout wurde beispielsweise aktualisiert, während ich an diesem Buch arbeitete). Wenn Sie diese Zeilen lesen, könnte also ein Teil der Syntax, die ich hier verwende, bereits überholt sein. Dieses Risiko muss man eben eingehen, wenn man ein auf Papier gedrucktes Buch über Entwicklungsstandards veröffentlicht. Ich versuche jedoch, es zu minimieren, indem ich auf Funktionen hinweise, die möglicherweise noch geändert werden, und indem ich in der Website zum Buch *http://modernwebbook.com* eine Liste mit Errata und Aktualisierungen pflege.[1]

Nutzen Sie Online-Ressourcen!

Sie sollten auf jeden Fall einige Online-Ressourcen nutzen, um sich über den Fortschritt der Implementierung auf dem Laufenden zu halten. Die meisten davon konzentrieren sich allerdings auf Desktop- und Mobilgerätebrowser. *Can I Use…* zeigt die Unterstützung für zahlreiche aktuelle Technologien und kommende Versionen häufig verwendeter Browser, während sich *HTML5 Please* damit beschäftigt, wie sicher die Nutzung innovativer Funktionen ganz allgemein ist, vor allem von CSS3 und JavaScript (aus diesem Grund finde ich den Namen der Site etwas unpassend).

Die Site *The HTML5 Test* informiert Sie darüber, welche Funktionen der HTML5-Spezifikation Ihr Browser unterstützt, führt aber auch sehr sinnvoll den Umsetzungsgrad in vielen unterschiedlichen Browsern und Geräten auf, einschließlich den Browsern von TV-Geräten und Spielekonsolen. Außerdem ermöglicht sie einen direkten Vergleich von bis zu drei verschiedenen Browsern. Allerdings ist die Site nur auf HTML5-Unterstützung beschränkt.

1.7 Testen, testen und nochmals testen

40-50 Prozent der Entwicklungszeit einplanen

Wenn Sie in der vielfältigen Gerätelandschaft sicherstellen möchten, dass die von Ihnen entwickelte Site überall funktioniert, sind Tests die einzige sichere Möglichkeit. Testen Sie zu Projektbeginn, testen Sie am Ende, testen Sie bei jeder Gelegenheit und während der gesamten Projektdauer. Wenn Sie ein Projekt für mehrere Geräte planen, sollten den Tests 40 bis 50 Prozent der Entwicklungszeit eingeräumt werden. Wirklich.

Open Device Labs

Tests auf echten Geräten sind durch nichts zu ersetzen. Bauen Sie sich also ein Arsenal mit so vielen Geräten auf wie möglich. Wenn sich in Ihrer Nachbarschaft weitere Agenturen befinden, sollten Sie darüber nachdenken, die Ressourcen zusammenzulegen. In vielen Städten entstehen Open Device Labs – Geräteparks mit zahlreichen, von lokalen Entwicklern und Firmen zur Verfügung gestellten Geräten, die für jedermann genutzt werden können. Suchen Sie online nach dem nächstgelegenen Open Device Lab oder bauen Sie vielleicht selbst eines in Ihrer Firma oder an Ihrem Arbeitsplatz auf. Und beschränken Sie sich dabei nicht auf mobile und Tablet-Geräte; denken Sie vielmehr auch an Spielekonsolen-Browser, wenn Ihre Sites für ein jüngeres Publikum gedacht sind (Untersuchungen haben ergeben, dass in den Vereinigten Staaten etwa jeder vierte Jugendliche mit einem Spielekonsolen-Browser ins Internet geht) oder an TV-Geräte, wenn Ihre Sites für den

Freizeitmarkt gedacht sind.

Emulatoren und Simulatoren

Wenn Sie keinen Zugang zu aktuellen Geräten haben, gibt es auch spezielle Tools, und die meisten (oder alle) Benutzeroberflächenentwickler und/oder Gerätehersteller bieten kostenfrei herunterladbare Software-Development-Kits (SDKs) mit Geräteemulatoren.

Im mobilen und Tablet-Bereich gibt es SDKs für Android, Windows Phone und Blackberry und viele andere mehr. Das über den App Store erhältliche Xcode von Apple enthält einen iOS-Simulator, in dem Sie zu Testzwecken zwischen Geräte- und OS-Versionen umschalten können.

Wenn sie einmal eingerichtet sind, geben viele dieser SDKs Ihnen auch die Möglichkeit, physische Geräte per USB anzuschließen, um über den Browser zu debuggen; eine einfachere Möglichkeit ist jedoch der Mobile Emulator von Opera. Sobald er geöffnet und mit einer Desktopversion von Opera verbunden ist, können Sie die Entwickler-Tools auf dem Desktop verwenden, um die Seite auf dem Mobilgerät zu debuggen. Wenn Sie WebKit verwenden müssen – und das kann sehr gut sein, weil es die dominierende Multi-Device-Engine ist –, können Sie über die Software *weinre* eine Version von Chrome oder Safari auf dem Desktop mit Android-, iOS- oder Blackberry-Emulatoren/-Simulatoren verbinden.

Adobe bietet eine App namens Edge Inspect, die den Browser Chrome mit jedem Gerät synchronisiert, auf dem Edge Inspect läuft (momentan auf iOS und Android). Dadurch können Sie Ihre Site auf vielen verschiedenen Geräten gleichzeitig betrachten und den Web-Inspector für das Remote-Debugging nutzen.

1.8 Zusammenfassung

Dieses Kapitel liefert Ihnen alle Informationen, die Sie für den Einstieg in die moderne Webentwicklung benötigen. Ich habe die allgemein

übliche Definition von HTML5 enger gesteckt und Sie mit der Webplattform vertraut gemacht. Sie haben gelernt, wozu HTML5 dient und wie Sie es schreiben. Außerdem haben Sie eine kurze Einführung in CSS3 erhalten.

Die Schlüsselaussagen des Kapitels finden Sie den späteren Kapiteln wieder. Erstens: Halten Sie sich stets über den Implementierungsgrad von Webplattform-Funktionen in den verschiedenen Browsern auf dem Laufenden. Zweitens: Testen Sie Ihre Sites, testen Sie sie noch einmal, testen Sie weiter, und wenn Sie denken, dass Sie genug getestet haben, testen Sie erneut. Und dann noch einmal. Nach all diesen Erläuterungen krempeln wir die Ärmel hoch und machen uns an die Arbeit.

1.9 Literaturempfehlungen

Falls Sie es überlesen haben: Die *Liste der Technologien, die die Webplattform ausmachen*, finden Sie auf http://platform.html5.org/. Bruce Lawson schlug den Begriff NEWT in seinem Blog vor: http://www.brucelawson.co.uk/2010/meet-newt-new-exciting-web-technologies/.

Die *HTML5-Spezifikation des W3C* finden Sie auf http://www.w3.org/TR/html5/ und die *Live-Spezifikation des WHATWG* auf http://whatwg.org/html. Es gibt dort auch eine nützliche *Edition for Web Developers*, in der gewisse obskure Sprachelemente weggelassen werden und die deshalb besser lesbar ist: http://developers.whatwg.org/.

Die *vollständige Formulierung des HTML5-Standards* gibt es auf http://html5boilerplate.com/. Denken Sie daran: Verwenden Sie nur die von Ihnen benötigten Teile; Sie müssen nicht alles wörtlich umsetzen.

Mehr über den Implementierungsgrad verschiedener Funktionen bieten

Alexis Deverias Site *Can I Use* auf http://caniuse.com/, die Community-Site *HTML5 Please* auf http://html5please.com/ sowie *The HTML5 Test* auf http://html5test.com/.

Die Website *LabUp!* ist eine Ressource für Open Device Labs: http://lab-up.org/. Der Testleiter der BBC, David Blooman, beschäftigte sich in dem langen und ausführlichen Artikel *Testing for Dummies* damit, wie ein großes Unternehmen Multi-Device-Tests durchführt: http://mobiletestingfordummies.tumblr.com/post/20056227958/testing.

Patrick Meenans Präsentationsfolien für seinen Vortrag *Taming the Mobile Beast* enthalten eine Fülle von Links und Informationen über Tests auf mobilen Geräten: http://www.slideshare.net/patrick-meenan/velocity-2012-taming-the-mobile-beast/. Und Anna Debenhams Artikel für A List Apart *Testing Websites in Game Console Browsers* handelt von ... na ja, der Titel ist ziemlich selbsterklärend: http://www.alistapart.com/articles/testing-websites-in-game-console-browsers/.

Opera bietet eine *detaillierte Anleitung zum Remote-Debugging*: http://www.opera.com/dragonfly/documentation/remote/. *weinre* können Sie sich von http://people.apache.org/~pmuellr/weinre/docs/latest/ herunterladen. Weitere *Informationen über Adobe Edge Inspect* erhalten Sie auf http://html.adobe.com/edge/inspect/.

Anmerkung des Verlags: Bei der vorliegenden Übersetzung haben wir uns zudem bemüht, Änderungen und Neuerungen, die seit dem Druck des englischen Buchs hinzukamen, zu erkennen und in der deutschen Fassung zu berücksichtigen.

2 Struktur und Semantik

Kennen Sie das Gleichnis von dem Mann, der sein Haus auf Sand baute? Oder das von den Schweinen, die ihre Häuser aus Stroh und Stöcken errichteten? Das sind Verlierertypen. Sie haben verloren, weil sie zu wenig Wert auf die Wichtigkeit der Struktur legten.

2.1 Grundsätzliche Anmerkungen

Gute Sites benötigen eine gute Struktur. Im Web ist die Grundlage dafür HTML.

Durch einen entsprechenden Seitencode erhalten Sie nicht nur jetzt, sondern auch in Zukunft eine tragfähige Struktur. Ob Sie nun eine stark interaktive Webapplikation entwickeln, eine Hybrid-App für Mobilgeräte oder eine digitale Broschüre mit einer einzigen Seite – immer hat es oberste Priorität, dass Sie eine solide Struktur erzeugen. Dadurch wird Ihre Site besser zugänglich; Erstellung und Pflege werden vereinfacht. Browser und andere User Agents können den Sinngehalt Ihrer Seiten leichter erfassen. Ein gut strukturiertes DOM kann auch die Performance Ihrer Site stark verbessern, weil der Browser sie leichter parsen kann und der Speicherbedarf sinkt.

Semantischer Gehalt geht über die bloße Struktur hinaus. Wenn Sie dem Inhalt Ihrer Seiten einen solchen zusätzlichen Sinngehalt geben, erzielen Sie einen unmittelbaren Vorteil: Die Suchmaschinen können Ihre Datei leichter durchsuchen und verstehen. Und außerdem könnten sich längerfristig Vorteile auftun, die bisher noch nicht ersichtlich sind.

All dies wird durch HTML5 und verwandte Technologien vereinfacht. Wenn Sie vorhandene und gut implementierte Methoden verwenden, können Sie stabile, aussagekräftige, leistungsstarke und

datenintensive Seiten erstellen.

2.2 Neue HTML5-Elemente

Eine der wichtigsten neuen HTML5-Funktionen sind mehrere semantische Elemente, durch die HTML weit über seinen ursprünglichen Zweck hinaus – die Auszeichnung wissenschaftlicher Dokumente mit Überschriften, Listen und Absätzen – ausgebaut wird. Die meisten dieser neuen Elemente sollen die Seitenstruktur verbessern. Der Entwickler soll damit die Möglichkeit zur Auszeichnung inhaltlicher Elemente erhalten, statt einfach ein `div`-Element in Verbindung einer `id` oder mit Klassen zu verwenden.

Hier ein Beispiel. Bisher haben Sie etwa den folgenden Code verwendet:

```
<div class="article">...</div>
```

In HTML5 können Sie stattdessen die folgende Zeile notieren:

```
<article>...</article>
```

Structural Elements

Die HTML5-Spezifikation des W3C listet neun strukturierende Elemente (*Structural Elements*) auf. Drei davon gibt es bereits in HTML4: `body`, `h1-h6` (wenn wir ein bisschen schummeln und die Überschriftelemente als eine Einheit zählen) sowie `address`.

Sectioning Content

Von den sechs neuen Elementen gehören vier in die Kategorie *Sectioning Content* der HTML5-Spezifikation. Ich erkläre Ihnen gleich, was es damit auf sich hat. Zunächst liste ich diese Sektionselemente aber einfach auf:

- `article` – ein unabhängiger Teil eines Dokuments oder einer Site, zum Beispiel ein Foren- oder Blog-Beitrag oder vom Benutzer übermittelter Inhalt

- `aside` – ein lose mit dem umliegenden Inhalt verbundener Seitenbereich, der aber als getrennte Einheit betrachtet werden sollte, etwa eine seitliche Kolumne in einem Magazinartikel
- `nav` – der Navigationsbereich eines Dokuments, ein Bereich, der Links zu anderen Dokumenten oder zu anderen Bereichen desselben Dokuments enthält
- `section` – eine thematisch zusammenhörige Gruppe von Inhalten, beispielsweise ein Buchkapitel, ein Register in einem durch Tabs unterteilten Dialogfeld oder die Einführung auf der Startseite einer Website

Die anderen zwei strukturierenden Elemente definieren Bereiche innerhalb des Seiteninhalts[1]:

- `footer` – die Fußzeile eines Dokuments oder ein Dokumentbereich mit Metadaten über den Abschnitt, in dem er sich befindet – zum Beispiel Details über den Autor
- `header` – meist die Kopfzeile eines Dokuments; es könnte sich aber auch um den Kopfbereich eines Dokumentteils handeln, im Allgemeinen mit h1- bis h6-Elementen für die Auszeichnung von Überschriften

HTML5 enthält noch weitere neue Elemente, die nichts an der grundlegenden Seitenstruktur ändern; ich bespreche sie bei Bedarf in den übrigen Kapiteln dieses Buchs. Im Moment beschäftigen wir uns damit, warum diese neuen Elemente eigentlich definiert wurden.

2.2.1 Was bringen die neuen Elemente überhaupt?

Das definierte Ziel dieser neuen Elemente besteht darin, eine klare Dokumentstruktur zu schaffen, damit die Dokumente besser vom Browser und anderen Geräten, etwa Screenreadern, geparst werden können. Stellen Sie sich diese Struktur wie eine Landkarte des Dokuments vor, die die Hierarchie der darin enthaltenen Inhalte

darstellen, aufzeigen, welche Überschriften am wichtigsten sind, die Abhängigkeiten der Inhaltsbereiche verdeutlichen usw.

In HTML 4 wurden dazu in erster Linie die Überschriftelemente `h1` bis `h6` verwendet. Die Überschrift `h1` stellte die einzige oder die Hauptüberschrift auf der Seite dar; `h2`-Elemente standen normalerweise in direkter Abhängigkeit von `h1` usw. Typisch war etwa die folgende Struktur:

```
<h1>Menschenaffen</h1>
  <h2>Gorilla</h2>
   <h3>Östlicher Flachlandgorilla</h3>
   ...
   <h3>Westlicher Flachlandgorilla</h3>
   ...
  <h2>Orang-Utan</h2>
  ...
```

Diese Verschachtelung der Überschriften erzeugte die folgende Dokumentstruktur:

1. Menschenaffen
   1.1 Gorilla
      1.1.1 Östlicher Flachlandgorilla
      1.1.2 Westlicher Flachlandgorilla
   1.2 Orang-Utan

Hinweis

Menschenaffen-Fans ist bestimmt aufgefallen, dass ich den Bonobo und den Schimpansen weggelassen habe. Dies geschah nicht aus Voreingenommenheit, sondern aus Platzgründen und zur Verdeutlichung.

Implicit Sectioning

Die von mir erzeugte Struktur ist visuell sinnvoll, und Sie erhalten auf diese Weise eine Dokumentstruktur, die als »implizite

Gliederung« (*Implicit Sectioning*) bezeichnet wird.

Explicit Sectioning

In HTML5 erzeugen Sie mit den weiter vorne besprochenen Sektionselementen die Abschnitte der Gliederung – nicht durch die Überschriften innerhalb dieser Abschnitte. Dies ist eine »explizite Gliederung« (*Explicit Sectioning*). Um also in HTML5 dieselbe Struktur für unsere Menschenaffen zu erzielen, könnten wir Folgendes notieren:

```
<h1>Menschenaffen</h1>
<section>
 <h1>Gorilla</h1>
 <article>
  <h1>Östlicher Flachlandgorilla</h1>
  ...
 </article>
 <article>
  <h1>Westlicher Flachlandgorilla</h1>
  ...
 </article>
</section>
<section>
   <h1>Orang-Utan</h1>
   ...
</section>
```

Sectioning Content

Sie würden dasselbe Ergebnis wie im HTML4-Beispiel erhalten, weil jedes `section-` oder `article-`Element einen neuen Abschnitt in der Gliederung erzeugt. Dies sind die Sektionselemente, mit denen man Inhalte in Abschnitte unterteilt (*Sectioning Content*) und die ich weiter vorne zusammen mit den Elementen `aside` und `nav` erwähnt habe.

Jeder Abschnitt sollte eine Überschrift haben – eine beliebige Überschriftebene ist in Ordnung. In meinem Beispiel habe ich überall `h1-`Überschriften verwendet. In Wirklichkeit ist es aber völlig gleichgültig, welche Überschriftebene ich wähle, weil ich neue Abschnitte per Sectioning Content erzeuge. Genauso gut hätte ich die Überschriftebenen nach dem Zufallsprinzip vergeben können.

Hinweis

Das ist etwas leicht dahingesagt – Sie können (und sollten) trotzdem die Überschriftebenen `h1` bis `h6` auf hierarchische Weise einsetzen, das garantiert eine gute Abwärtskompatibilität und vereinfacht die Gestaltung.

Sectioning Roots

Genau wie eine Überschrift kann jeder Abschnitt einen eigenen `header` und `footer` enthalten, darüber hinaus weitere Abschnitte und sogenannte *Sectioning Roots*. Solche Sectioning Roots wie etwa `blockquote` und `figure` können mit einer eigenen Gliederung versehen sein. Sie ändern jedoch nichts an der Gesamtgliederung des Dokuments.

Wenn Ihnen all dies etwas unklar erscheint, sind Sie in guter Gesellschaft. Die Verwirrung darüber, wozu die einzelnen Sektionselemente gedacht sind, hat so weit geführt, dass der gute HTML5 Doctor ein Schaubild veröffentlicht hat (Abb. 2–1). Dieses Bild soll Ihnen helfen, das jeweils richtige Element für eine bestimmte Aufgabe zu finden. Ein Schaubild! Damit Sie ein Element auswählen können! Wenn Sie etwas zwischen den Zeilen lesen können, ist Ihnen inzwischen wahrscheinlich ganz richtig aufgefallen, dass ich kein Fan der neuen strukturierenden Elemente in HTML5 bin.

2.2.2 Der Nachteil der Sektionselemente in HTML5

article oder section?

Wie Sie an meinem zweifellos frustrierten Unterton im vorigen Abschnitt erkannt haben, kann es recht anspruchsvoll sein, mit manchen dieser neuen Elemente zurechtzukommen. Dies gilt besonders für den Unterschied zwischen `article` und `section`. Zur Wiederholung: Ein `section`-Element kann Artikel und Abschnitte enthalten, und ein `article`-Element kann Abschnitte und Artikel enthalten – und beide erzeugen Abschnitte in der Gliederung. Es gibt zwar einen Unterschied

zwischen den beiden Elementen, aber niemand – noch nicht einmal der Autor der Spezifikation – hat bisher eine Definition liefern können, die so klar und deutlich wäre, dass die Entwickler sie sich leicht merken können.

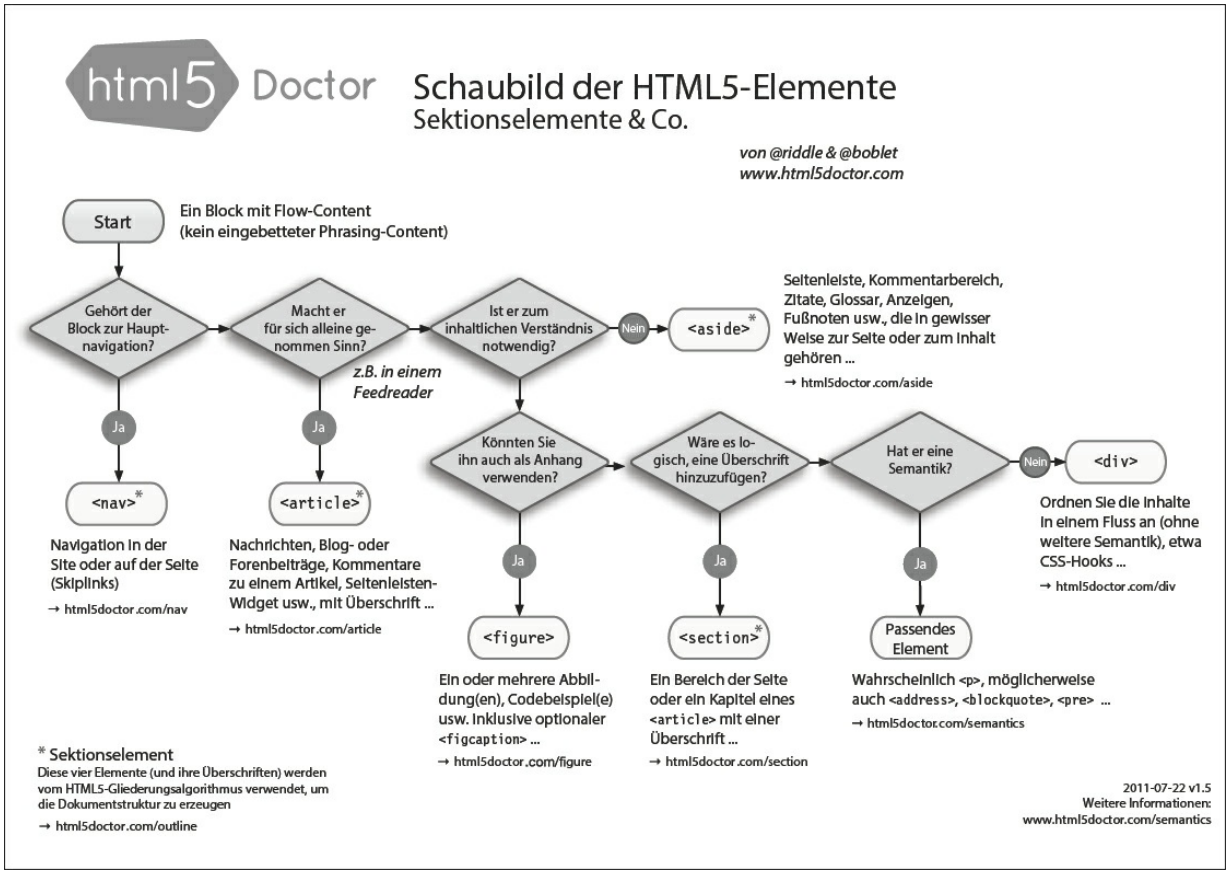


Abb. 2–1
Wenn Sie nicht wissen, welches Sektionselement das richtige ist, liefert der HTML5 Doctor Ihnen die Antwort.

In seinem im Jahr 2012 bei Create Space erschienenen Buch *The Truth About HTML5* schreibt Luke Stevens über die vage Beschreibung des `article`-Elements:

> *Spezifikationen versagen, wenn sie bei Ihnen Fragen offen lassen. Der Sinn einer Spezifikation besteht darin, genau zu spezifizieren, was Sie tun sollen. In diesem Fall lässt sie aber Interpretationsspielraum, bietet keinen klaren Vorteil und wiederholt vorhandene Funktionen ...*

Ich muss ihm beipflichten. Ich sage jetzt schon voraus, dass viele Entwickler mit diesen Elementen üblen Missbrauch treiben werden, bis es klarere Definitionen gibt.

Aus technischen Gründen empfehle ich, die neuen Elemente im öffentlichen Internet nicht zu nutzen: Zum einen werden sie in älteren Versionen des Internet Explorers (bis zur Version 8) einfach nicht unterstützt. Damit diese Browser die neuen Elemente erkennen, müssen Sie sie in JavaScript erzeugen. Dies geht recht problemlos; fügen Sie einfach in einem bedingten Kommentar das bekannte JavaScript-Skript *HTML5-Shiv* ein:

```
<!--[if lt IE 9]>
<script src="html5shiv.js"></script>
<![endif]-->
```

Allerdings machen Sie Ihre Besucher dadurch von JavaScript abhängig: Alle Nutzer, die einen älteren Internet Explorer mit abgeschaltetem JavaScript nutzen, sehen zwar den Inhalt der neuen Elemente, allerdings ist er unformatiert. Sicherlich dürfte es sich dabei nur um einen geringen Nutzeranteil handeln. Barrierefreiheit sollte aber bedeuten, dass *jeder* Ihre Inhalte sehen kann.

Hinzu kommt, dass keiner der aktuellen Browser den neuen Outline-Algorithmus unterstützt (nur der Screenreader JAWS unterstützt ihn, jedoch mit Bugs), sodass all Ihre harte Arbeit in Wirklichkeit kaum Sinn hat – in der Zukunft wird sich das aber aller Voraussicht nach ändern.

Alternativen suchen

Im Endeffekt liegt die Entscheidung natürlich bei Ihnen, ob Sie die neuen strukturierenden Elemente verwenden oder nicht. Sie sind nicht zwingend vorgeschrieben – nach wie vor können Sie mit `div`-Elementen arbeiten. Es fällt mir aber schwer, die neuen Elemente zu empfehlen – es sei denn, Sie sind bereit, die HTML5-Spezifikation zu lesen und genau zu verstehen, wie die strukturierenden Elemente auf die Dokumentstruktur wirken, und vorausgesetzt, Sie arbeiten in einer Umgebung, in der veraltete Browser kein Thema sind. So, wie die Dinge momentan (während ich dieses Buch schreibe) liegen,

würde ich nach einer alternativen Möglichkeit zur Darstellung der Seitenstruktur suchen. Glücklicherweise gibt es diese Möglichkeit in Form von WAI-ARIA.

2.3 WAI-ARIA

Barrierefreiheit

Die *Accessible Rich Internet Applications*-Suite *(WAI-ARIA)* wurde als Antwort auf die mangelnde Barrierefreiheit des Web entwickelt. WAI-ARIA entstand, als im Web nicht mehr nur einfache Dokumente, sondern immer mehr interaktive Anwendungen veröffentlicht wurden. Dazu wurde eine Anzahl HTML-Erweiterungen entwickelt (genauer gesagt Erweiterungen für beliebige DOM-basierte Sprachen wie SVG und XML), sodass die Entwickler Browser und andere Geräte, etwa Screenreader, auf interaktive Inhalte aufmerksam machen konnten.

Wenn Sie beispielsweise einen Link haben, der beim Anklicken JavaScript einsetzt, um einen Dialog zu öffnen, dann haben Sie keine Möglichkeit, den Browser darauf aufmerksam zu machen. Der Code sieht einfach aus wie ein ganz normaler Link:

```
<a href="http://example.com">Launch popup</a>
```

Weil das Event dem Link per Skript zugeordnet ist, hat ein Screenreader keine Informationen darüber, was hier passiert, sodass dem Nutzer diese Funktionalität verborgen bleibt. WAI-ARIA führt genau für diese Situation das neue Attribut `aria-haspopup` ein, sodass der Benutzer weiß, was vor sich geht:

```
<a href="http://example.com" aria-haspopup="true">Launch popup</a>
```

Landmark Roles

Neben zahlreichen anderen Attributen sind sogenannte *Landmark Roles* verfügbar. Diese Attribute informieren Screenreader und andere assistive Navigationsgeräte über die Struktur Ihrer Seite, sodass

sich der Benutzer leicht in Ihrem Dokument zurechtfinden kann. So erfüllt Ihre Seite die strukturellen Aufgaben, für die eigentlich die neuen HTML5-Elemente gedacht sind.

Hinweis

Wie gesagt: Während ich dieses Buch schreibe, parsen manche User Agents und assistiven Geräte die neue HTML5-Dokumentgliederung nicht korrekt. Deshalb könnte es für die Abwärtskompatibilität sinnvoll sein, Landmark Roles zu verwenden.

Landmark Roles werden mit dem `role`-Attribut und einer Reihe vordefinierter Werte zugewiesen. Diese Werte entsprechen nicht direkt den strukturierenden Elementen in HTML5; meist lassen sie sich sehr gut mit diesen vergleichen. Zur Definition des Dokumentbereichs mit den allgemeinen Informationen über die Site, beispielsweise Logo und Slogan, verwenden Sie beispielsweise die `banner`-Role:

```
<div role="banner">...</div>
```

Diese Role entspricht weitgehend dem `header`-Element und wird in den meisten Fällen an dessen Stelle verwendet.

Möchten Sie dem Screenreader mitteilen, wo sich der Hauptinhalt befindet, nehmen Sie die `main`-Role:

```
<div role="main">...</div>
```

Lange Zeit gab es hierfür keine HTML5-Entsprechung, aber im Frühsommer 2013 wurde hierfür das `main`-Element eingeführt.

In der WAI-ARIA-Spezifikation sind acht Landmark Roles definiert:

- `application` definiert einen Seitenbereich, der kein Dokument, sondern eine interaktive Applikation ist
- `banner` definiert, wie erwähnt, allgemeinen Site-Inhalt, wahrscheinlich im Seitenkopf; entspricht in diesem speziellen Zusammenhang dem `header`-Element

- `complementary` definiert Inhalte, die sich zwar auf den Hauptinhalt beziehen, in diesem aber nicht enthalten sind, etwa eine Seitenleiste; entspricht dem `aside`-Element
- `contentinfo` liefert Informationen über das Dokument, zum Beispiel rechtliche Hinweise; häufig in der Fußzeile angesiedelt und deshalb in diesem Zusammenhang dem `footer`-Element entsprechend
- `form` definiert Formulare (außer Suchformulare), beispielsweise ein Kontaktformular
- `main` definiert den Hauptinhalt eines Dokuments
- `navigation` definiert Link-Gruppen für die Navigation im aktuellen Dokument oder in zugehörigen Dokumenten, entspricht dem `nav`-Element
- `search` definiert Formulare, die speziell für die Suche in dieser oder anderen Sites gedacht sind

CSS-Formatierung

Die Landmark Roles sind nicht nur für Navigation und semantische Informationen wichtig, sondern eignen sich auch gut als Ankerpunkte für die Formatierung per CSS. Mithilfe des Attributselektors, der auf einen bestimmten Wert prüft, können Sie beispielsweise dem Seitenkopf problemlos Regeln zuweisen:

```
div[role='contentinfo'] { background-color: blue; }
```

Dieser Selektor ist in so ziemlich jedem (mir bekannten) Browser, der in den letzten zehn Jahren entwickelt wurde, implementiert. Wenn Sie also nicht gerade einen sehr alten oder sehr primitiven Browser verwenden, ist diese Technik sinnvoll – denken Sie jedoch daran, dass komplexe Selektoren einen negativen Effekt auf die Seitenladezeit haben können.

2.4 Die Bedeutung semantischen Markups

Bevor wir uns mit weiteren Möglichkeiten befassen, Ihren Seiten mehr Sinngehalt zu verleihen, legen wir eine Pause ein und stellen uns die

Frage: Warum sollten wir uns überhaupt um die Semantik kümmern? Spricht grundsätzlich etwas dagegen, eine Seite auf die folgende Weise hauptsächlich mit div-Elementen auszuzeichnen?

```
<div class="first">This is the heading.</div>
<div class="main"><b>This is the first sentence.</b><br> This is
the
second sentence.</div>
```

Divya Manian ging darauf in ihrem polemischen Artikel »Our Pointless Pursuit of Semantic Value« ein. Sie stellt darin die These auf, dass es für die meisten Menschen eine Zeitverschwendung ist, zu viel Gewicht auf semantisches Markup zu legen:

> *Das Markup strukturiert den Inhalt; aber welche Elemente wir auswählen, ist längst nicht so wichtig, wie wir gelernt haben ...*

Wartungsfreundlichkeit

Ich finde jedoch, dass es zwei gute Gründe für die Verwendung semantisch korrekter Elemente gibt. Die erste und nüchternste ist, dass Sie auf einen De-facto-Standard mit guter Wartungsfreundlichkeit hinarbeiten. Wenn Sie semantische Elemente verwenden, sind Sie sicher, dass Ihre Kollegen oder eventuellen Nachfolger an Ihrem Code arbeiten können, ohne dass sie die von Ihnen verwendeten Namenskonventionen erlernen müssten. Und auch das Gegenteil trifft zu. Wenn Sie den standardgemäß notierten Code eines anderen Entwicklers übernehmen, wissen Sie genau, was darin vor sich geht.

Aboutness

Ein eher metaphysischer Grund ist, dass die Verwendung semantischer Elemente Ihrem Inhalt verstärkte »Aboutness« verleiht. Einfach ausgedrückt, ist Aboutness ein Maß für die Qualität des Sinngehalts; das Wesen einer Sache wird durch seine Aboutness beschrieben.

Als einfaches Beispiel für dieses Prinzip stellen Sie sich eine Webseite mit W. H. Audens Gedicht »Funeral Blues« vor:

He was my North, my South, my East and West,
My working week and my Sunday rest,
My noon, my midnight, my talk, my song;
I thought that love would last forever: I was wrong.

Zwar wissen *wir*, dass das Gedicht vom Tod handelt; aber das Wort »Tod« taucht nicht in dem Gedicht auf. Wie könnte eine Suchmaschine, die die Seite indiziert, wissen, wovon sie handelt, und diesen Sinngehalt in den Suchergebnissen für das Thema »Tod« zurückgeben? Die Suchmaschine betrachtet den Text der Links zu dieser Seite. Also bringt ein Link mit dem Text »Weiterlesen« in diesem Zusammenhang nicht viel, während ein Link mit dem Text »W.H. Audens Gedicht über den Tod« Aboutness bietet.

Die Verwendung korrekter semantischer Elemente hat denselben Vorteil. Wenn alle Inhalte auf Ihrer Seite mit `div`-Elementen ausgezeichnet sind, bietet der Inhalt keinen Sinnzusammenhang; wenn Sie Ihre Seite semantisch auszeichnen, erhält der Inhalt eine Bedeutung:

```
<h1>This is the heading.</h1>
<p>This is the first sentence.</p>
<p>This is the second sentence.</p>
```

Nun wissen Sie genau, welche Überschrift wichtig ist und was der Hauptinhalt ist. Sie haben den Inhalt mit Aboutness versehen.

Neben der korrekten Verwendung semantischer Elemente zur Auszeichnung Ihres Inhalts können Sie die Aussagekraft Ihrer Dokumente nicht nur für menschliche Benutzer, sondern auch auf unterschiedliche Weise für Maschinen verbessern (man spricht dann von »strukturierten Daten«). Sie können vorhandene Attribute und Elemente mit vordefinierten Mustern (*Mikroformaten*) oder HTML mit neuen Attributen (*RDFa* und *Mikrodaten*) erweitern. Diese Möglichkeiten stelle ich jetzt kurz vor.

2.5 Mikroformate

Keine HTML5-Erweiterung

Mikroformate wurden von Entwicklern »von unten«, d.h. ohne Mitwirkung der bekannten Normungsgremien, definiert. Sie versehen Inhalte durch standardisierte Markup-Muster für vorhandene Attribute mit zusätzlicher Bedeutung. Ihr Hauptvorteil ist, dass sie auf aktuellen Entwicklungsmethoden basieren; sie sind keine HTML-Erweiterung, sondern ein Design-Prinzip bzw. ein Satz von standardisierten Nutzungsmustern.

Rel-Tag

Mikroformate können ziemlich komplex oder auch ganz einfach sein. Hier ein Beispiel für das wahrscheinlich einfachste Mikroformat:

```
<a href="http://flickr.com/photos/tags/gorilla/"
   rel="tag">Gorillas</a>
```

Dies ist das Mikroformat *Rel-Tag*. Das Schlüsselwort `tag` im Attribut `rel` informiert Maschinen darüber, dass die im `a`-Element referenzierte URL eine Seite ist, die durch ein Element beschrieben wird, dessen Name die letzte Pfadkomponente der URL ist – in diesem Fall `gorilla`. An diesem Punkt sollte ich wahrscheinlich darauf hinweisen – falls es Ihnen noch nicht selbst aufgefallen ist –, dass ich Gorillas sehr mag.

hCard

Ein komplexeres, aber genauso geradliniges und nützliches Beispiel ist das Mikroformat *hCard*. Es zeichnet häufig verwendete Personeninformationen mit einer standardisierten »Visitenkarten-Syntax« aus. Mit den folgenden Zeilen könnte ich beispielsweise online zu meinen Kontaktdaten verlinken:

```
<div class="details">
<p><a href="http://about.me/petergasston">Peter Gasston</a>
writes for <a href="http://broken-links.com">Broken Links</a>.</p>
</div>
```

Der Code ist in Ordnung, er hat jedoch nicht viel Aboutness. Der Leser würde verstehen, dass es sich um meinen Namen und um meinen Blog handelt; aber ein Suchmaschinen-Spider wäre wahrscheinlich zu diesem

kognitiven Sprung nicht imstande. Mit dem *hCard*-Muster kann ich den Code mit Attributwerten versehen, die seinen semantischen Gehalt vermehren:

```
<div class="vcard">
<p><a href="http://about.me/petergasston" class="url fn">
Peter Gasston</a>
writes for <a href="http://broken-links.com" class="url org">
Broken Links</a>.</p>
</div>
```

Diese paar standardisierten Klassennamen machen diesen Inhalt für Suchmaschinen sehr viel sinnvoller; ein Web-Crawler, der das Muster *hcard* erkennt, sieht die Klasse `vcard` und weiß, dass deren Inhalt die gesuchten Informationen enthält: Er weiß, dass der erste Link den vollständigen Namen (`fn` = full name) des Kontakts, der zweite Link die Organisation (`org`) enthält, für die der Kontakt arbeitet.

Es gäbe noch mehr über Mikroformate zu sagen; es gibt Mikroformate für Veranstaltungen, Besprechungen, geografische Koordinaten, sogar für Rezepte und noch vieles andere. Sie werden von Suchmaschinen wie Google verwendet, um die Suchergebnisse zu verbessern. Also ist die Verwendung von Mikroformaten in Ihren Seiten durchaus nicht (nur) eine Übung in Entwickler-Esoterik.

2.6 RDFa

Eine HTML5-Erweiterung

Das *Resource Description Format in Attributes (RDFa)* ist eine HTML-Erweiterung, die Inhalte unter Verwendung eines ganzen neuen Satzes maßgeschneiderter Attribute für Maschinen sinnvoll und verständlich macht.

Die Hauptsyntax ist als *RDFa Core* bekannt. Weiterhin ist die einfachere Untermenge *RDFa Lite* verfügbar. Beide beschreiben Inhalte mithilfe vordefinierter Schemata (Datenbeschreibungen).

Statt zu erläutern, was damit gemeint ist, zeige ich es Ihnen (die Jahre an der Schule für Drehbuchautoren waren nicht umsonst!). Ein typisches Beispiel für Daten im Web, vor allem in Blogs und News-Sites, ist das Datum. Recht häufig wird ein Datum etwa folgendermaßen ausgezeichnet:

```
<p class="date">2013-04-01</p>
```

property

Dieser Code ist funktional und unkompliziert, aber der einzige semantische Sinngehalt wird durch den Klassennamen vermittelt. Mit RDFa Lite können Sie seinen Kontext verbessern, sodass Maschinen erkennen können, dass es sich um ein Datum handelt. Dazu verwenden Sie das neue Attribut `property`:

```
<p property="http://purl.org/dc/elements/1.1/date">2013-04-01</p>
```

Der Wert des Attributs ist die URL der Beschreibung des Begriffs `date` in einem Schema, das zum standardisierten Vokabular *Dublin Core* gehört.

Sie haben wahrscheinlich bemerkt, dass das von mir verwendete Datum kein besonders leserfreundliches Format hat. Dies ist der Nachteil von RDFa Lite: Alle Inhalte müssen auf eine streng maschinenlesbare Weise formatiert werden. Damit der Inhalt für menschliche Betrachter besser geeignet ist, müssen sie RDFa Core verwenden. Mit RDFa Core kann ich mithilfe des Attributs `content` einen Satz Informationen für Maschinen und einen anderen für Menschen liefern:

```
<p property="http://purl.org/dc/elements/1.1/date"
content="2013-04-01">April 1</p>
```

Die Leser sehen den Inhalt des Elements; die Maschine liest den Wert des Attributs. Dazu ist Zusatzcode notwendig, aber alle sind zufrieden. Wie bei den Mikroformaten suchen manche Suchmaschinen nach RDFa-Mustern, um ihre Suchergebnisse zu verbessern. Auf Seite 45 im Abschnitt »Rich Snippets« gehe ich ins Detail.

2.7 Mikrodaten

Microdata

HTML5 begegnet dem Semantikproblem mit einer einfachen Syntax, die *Mikrodaten* (*Microdata*) genannt wird. Prinzipiell handelt es sich dabei um eine Reihe von Name-Wert-Paaren, die sinnvolle maschinenlesbare Daten liefern. Auch hier versuche ich nicht, es zu erklären, sondern zeige Ihnen lieber, wie es funktioniert:

```
<p itemscope>I live in <span itemprop="city">London</span></p>
```

Item

Dieser Code erzeugt ein einzelnes Element *(Item)*. Das Attribut `itemscope` wird für das Container-Element verwendet, um den Geltungsbereich (*Scope*) dieses speziellen Items zu definieren.

Innerhalb des Containers definieren Sie das Name-Wert-Paar, das P*roperty* genannt wird. Der Wert des Attributs `itemprop` ist der Name – in diesem Beispiel `city` –, und der Inhalt des Elements ist der Wert – in diesem Beispiel `London`. Das Ergebnis ist ein Element mit einer einzelnen Eigenschaft:

```
city: 'London'
```

Sie sind jedoch nicht auf eine einzelne Eigenschaft beschränkt; Sie können so viele Eigenschaften definieren, wie Sie möchten:

```
<p itemscope>Hello, my name is <span itemprop="given-name">Peter
</span> and I'm <span itemprop="role">a developer</span> from
<span itemprop="city">London</span>.</p>
```

In diesem Fall sieht die Eigenschaftenliste des Elements folgendermaßen aus:

```
given-name: 'Peter'
role: 'a developer'
city: 'London'
```

Wie Sie erkennen können, ist dieser Code durchaus RDFa-ähnlich, und genau wie bei diesem Format können Sie Maschinen und Menschen

unterschiedliche Werte liefern. Betrachten Sie das folgende Beispiel, in dem ich das Attribut `datetime` verwende:

```
<p itemscope>My birthday this year is on <span itemprop="birthday"
datetime="2013-12-14">December 14</span>.</p>
```

Und Sie können – wie bei RDFa – Inhalte mit vordefinierten Schemata beschreiben, indem Sie sie mit dem Attribut `itemtype` verknüpfen:

```
<p itemscope itemtype="http://example.org/birthday">My birthday
this year is on <span itemprop="birthday" datetime="2013-12-
14">December 14</span>.</p>
```

Dabei können Sie ein Schema wie den zuvor erwähnten Dublin Core verwenden – oder sogar ein von Ihnen selbst erdachtes Schema, wie der vorige Codeblock gezeigt hat.

2.7.1 Die API für Mikrodaten

Mikrodaten haben eine zugehörige DOM-API, die zur Extraktion von Daten aus der Seite geeignet und bereits recht umfassend in den modernen Browsern implementiert ist. Der Schlüssel ist die Methode `getItems()`, die eine `NodeList` mit allen Elementen auf der Seite zurückgibt:

```
var items = document.getItems();
```

Von hier aus können Sie ein einzelnes Element auswählen und beispielsweise mit dem `properties`-Objekt prüfen, wie viele Eigenschaften enthalten sind:

```
var firstItemLen = items[0].properties.length;
```

Oder Sie können den Wert einer dieser Eigenschaften herausfinden:

```
var itemVal = items[0].properties['name'][0].itemValue;
```

Ich demonstriere Ihnen dies in der Beispieldatei *microdata-api.html*[2]. Die Ergebnisse habe ich an die Konsole ausgegeben; öffnen Sie also Ihren

bevorzugten Browser und sehen Sie nach. Experimentieren Sie ruhig selbst damit. Falls Sie gerade keinen Browser zur Hand haben, zeigt Ihnen Abb. 2-2, wie die Ergebnisse in Firebug aussehen.

```
Items NodeList[p]                                  microd...pi.html (line 12)
1 Properties HTMLPropertiesCollection[span]        microd...pi.html (line 13)
Value Peter                                        microd...pi.html (line 14)
```

Abb. 2–2
Das Ergebnis einiger einfacher Experimente mit der Mikrodaten-API in der Konsole

2.7.2 Mikrodaten, Mikroformate und RDFa

Wenn Sie finden, dass maschinenlesbare semantische Daten für Ihre Seiten das Richtige sind, stellt sich die Frage: Welches Format sollen Sie wählen? Die Antwort lautet wie immer: Es kommt darauf an. Prüfen Sie Ihre Inhalte, informieren Sie sich über die Stärken und Schwächen der einzelnen Datentypen und entscheiden Sie, welcher für Sie am besten geeignet ist.

Eine Mischung

Meiner persönlichen Meinung nach wird es in Zukunft meistens eine Mischung von Mikrodaten und einfachen Mikroformaten geben. An Mikrodaten ist besonders interessant, dass sie beide ihrer Zeitgenossen in ihre eigene, flexible Syntax aufnehmen können. `hCard` wird beispielsweise mithilfe von Mikrodaten folgendermaßen notiert:

```
<div itemscope itemtype="http://microformats.org/profile/hcard">
<p><a href="http://about.me/petergasston" itemprop="url fn">
Peter Gasston</a> writes for <a href="http://broken-links.com"
itemprop="url org">Broken Links</a>.</p>
</div>
```

Ebenso können Sie problemlos das RDFa-Datenschema verwenden:

```
<p itemscope itemtype="http://purl.org/dc/elements/1.1/date"
datetime="2013-04-01">April 1</p>
```

Mikrodaten

Meiner Meinung nach wird die Flexibilität von Mikrodaten dazu führen, dass sie immer häufiger eingesetzt werden. Dennoch sind Mikrodaten keine perfekte Lösung für alles; manche Mikroformate – etwa *Rel-Tag* – sind so einprägsam und leicht anwendbar, dass es kaum sinnvoll wäre, sie zu ersetzen.

2.7.3 Schema.org

Durch Mikrodaten wird Ihr Inhalt von Suchmaschinen und Portalen erkannt und besser herausgestellt. Auch dies ist ein Grund, warum Mikrodaten meiner Ansicht nach die Mikroformate und RDFa ausstechen werden.

Google, Microsoft, Yahoo! und Yandex

Im Jahr 2011 brachten vier große Web-Riesen – Google, Microsoft, Yahoo! und Yandex – die Website Schema.org heraus. Mit dieser führten sie gemeinsame Vokabulare ein, um gängige Muster mit Mikrodaten auszuzeichnen. Dazu gehören häufig im Web verwendete Elemente wie *Reviews*, *Events*, *Places*, *Items* und *Objects*.

Beispiel Buchrezension

Zur Verdeutlichung nehmen wir an, dass Sie eine Buchrezension für Ihre Website schreiben (ich habe wahllos ein Buch herausgegriffen und eine ganz wertfreie Rezension geschrieben):

```
<div class="review">
   <h1>The Book of CSS3 by Peter Gasston</h1>
   <p>What an amazing book! 5 stars!</p>
</div>
```

Diese Rezension enthält zwei Elemente: die Buchdetails und die Rezension. Schema.org enthält zwei Vokabulare, die Sie zur Auszeichnung dieser Rezension verwenden können: *Book* und *Review*. Ein Besuch der entsprechenden Seiten auf Schema.org zeigt mir, welche Mikrodaten-muster ich verwenden soll. Anschließend kann ich meinen

Code folgendermaßen optimieren:

```
<div class="review" itemscope itemtype="http://schema.org/Review">
   <h1><span itemprop="itemReviewed">The Book of CSS3</span>,
   by <span itemprop="creator">Peter Gasston</span></h1>
   <p><span itemprop="reviewBody">What an amazing book!</span>
   <span itemprop="reviewRating">5</span> stars!</p>
</div>
```

Der Code ist jetzt zwar komplexer geworden, aber auch sinnvoller. Jedes der von mir verwendeten Vokabulare wird im Attribut `itemtype` durch einen Link zum relevanten Schema definiert, und die Elemente sind mit vordefinierten `itemprop`-Werten ausgezeichnet.

Interessant an Schema.org ist die Tatsache, dass bestimmten Schemata die Eigenschaften von übergeordneten Schemata vererbt werden. *Book* beispielsweise besitzt Eigenschaften aus seinem eigenen Schema, dem umfassenderen Vokabular *CreativeWork*, und zudem aus dem Vokabular der höchsten Ebene *Thing* (toller Name!) mit seinen allgemeingültigen Eigenschaften.

Wenn ich meinen Inhalt mit Schema.org-Mustern auszeichne, erkennen alle Crawler auf meiner Seite den Autor und Titel des Buchs, die Tatsache, dass ich es rezensiere und dass ich ihm eine Fünf-Sterne-Bewertung gegeben habe. Wenn jemand nach dem Buch sucht, könnte meine Rezension in den Suchergebnissen erscheinen oder mit anderen Rezensionen gemeinsam präsentiert werden, sodass der Leser eine sinnvolle Übersicht erhält.

2.7.4 Rich Snippets

Google

Google bezeichnet die Technik, den Suchmaschinen zusätzliche Informationen zu liefern, als *Rich Snippets*. Rich Snippets ordnen einer Suchabfrage einen Kontext zu. Dadurch kann der Nutzer die Relevanz des Ergebnisses besser beurteilen, ohne dass er sich durch die Seite klicken muss. Ein Beispiel für ein Rich Snippet sehen Sie in Abb. 2–

3.

The Book of CSS3: A Developer's Guide to the Future ... - **Goodreads**
www.goodreads.com › Computer Science › Programming
★★★★☆ Rating: 4.0 - 37 votes
May 13, 2011 – **The Book of CSS3** has 37 ratings and 11 reviews. Tami said: CSS has such potential. Even in its infancy, it's absolutely revolutionized the way ...

Abb. 2–3
Ein Beispiel für ein Rich Snippet, das auf der Google-Suchergebnisseite Zusatzinformationen liefert

Rich Snippets funktionieren mit Mikroformaten und RDFa; die bevorzugte Syntax sind jedoch Mikrodaten. Entwickler finden auf den Google-Webmaster-Seiten zahlreiche Informationen und Dokumentationen. Hier steht auch ein nützliches Tool zur Verfügung, mit dem Sie die korrekte Formatierung Ihrer Mikrodaten prüfen können. Abb. 2–4 zeigt die Daten, die dieses Tool aus der Buchrezension im vorigen Abschnitt extrahiert hat.

Item

type:	http://schema.org/review
property:	
itemreviewed:	The Book of CSS3
creator:	Peter Gasston
reviewbody:	What an amazing book!
reviewrating:	5

Abb. 2–4
Mit dem Rich-Snippet-Testwerkzeug aus der ausgezeichneten Buchrezension extrahierte Daten

2.8 Datenattribute

HTML5

Eine weitere HTML5-Möglichkeit zur Erweiterung des Sinngehalts von Elementen sind Datenattribute. Es handelt sich dabei um benutzerdefinierte Eigenschaften, deren Werte Informationen zu einem Element liefern sollen, ohne dass sich aber dessen Sinngehalt für Maschinen oder Menschen erhöht. Dies möchte ich etwas detaillierter erläutern.

Nehmen wir an, Sie möchten einen Satz Daten ausgeben. Jedes Datenelement hat zwei Werte – einen Namen und einen Zahlenwert (beispielsweise eine eindeutige Datenbank-ID). Der Name soll im Dokument angezeigt werden; Sie möchten aber auch den Zahlenwert für Skripte verfügbar machen. Momentan ist kein relevantes Attribut verfügbar; Sie müssten wahrscheinlich eine Klasse verwenden:

```
<p class="id-123">Peter</p>
```

data-...

Genau aus diesem Grund wurden Datenattribute geschaffen: Sie sollen Daten zuordnen. Sie können damit solche Zusatzinformationen speichern, ohne ihnen irgendeine zusätzliche Bedeutung zu verleihen, wie es bei der Verwendung von Klassen der Fall wäre. Jedes Datenattribut beginnt mit dem Wort `data`. Darauf folgt ein benutzerdefinierter, eindeutiger Schlüssel. In unserem Beispiel könnte das folgendermaßen aussehen:

```
<p data-id="123">Peter</p>
```

Das Datenattribut *id* ist nun dem Wert *Peter* zugeordnet. Obwohl das Element dadurch keine zusätzliche semantische Bedeutung erhält, ist das Attribut vorhanden und kann von anderen Prozessen genutzt werden: Möglicherweise befinden sich die Informationen über diese Daten in einer zugehörigen JSON-Datei, sodass Sie mithilfe von JavaScript darauf zugreifen können.

2.8.1 Die API für Datenattribute

dataset

Damit Skripte leichter auf diese Daten zugreifen können, ist eine einfache DOM-API verfügbar, die die `dataset`-Eigenschaft nutzt. Um den Wert eines Datenattributs zu ermitteln, verwenden Sie diese Eigenschaften mit dem Schlüssel des Attributs, das Sie abfragen:

```
var el = document.querySelector('p');
var id = el.dataset['id'];
```

Hier würde das Ergebnis *123* zurückgegeben. Sie können Datenattributwerte mit `dataset` auch aktualisieren:

```
el.dataset['id'] = 100;
```

Hier ein praktisches Beispiel:

```
var el = document.querySelector('p');
console.log('The ID is',el.dataset['id']);
el.dataset['id'] = 100;
console.log('Now the ID is',el.dataset['id']);
```

In diesem Beispiel führe ich drei Operationen aus: Zuerst ermittle ich die *id*-Daten, dann setze ich sie auf 100, dann ermittle ich sie erneut, und jedes Mal gebe ich die Ergebnisse an die Konsole aus. Das Ergebnis zeigt Ihnen Abb. 2–5.

```
The ID is 123          data-a...es.html (line 11)
Now the ID is 100      data-a...es.html (line 13)
```

Abb. 2–5
Das Ergebnis der Manipulation des Datenattributs mit der API in der Konsole

2.8.2 jQuery und Datenattribute

Wenn Sie jQuery nutzen, ist der Umgang mit Datenattributen noch einfacher (wenn Sie nicht wissen, was jQuery ist, erfahren Sie es in Kapitel 5; Sie können auf diesen Abschnitt zurückkommen, nachdem Sie

das Kapitel 5 gelesen haben).

data()

Nutzen Sie die Methode `data()`, um Datenwerte zu ermitteln und zu setzen:

```
var id = $(el).data('id');
```

Dieser Code entspricht dem im vorigen Abschnitt und gibt denselben Wert zurück: *123*.

Es gibt jedoch einen großen Vorteil gegenüber `dataset`, bei dem alle Ergebnisse als Zeichenkette zurückgegeben werden: Die Methode `data()` parst auch den Wert des Attributs und konvertiert ihn in den korrekten Typ. Im vorigen Beispiel wäre der Typ ein Zahlenwert. Würden Sie den Code jedoch folgendermaßen ändern ...

```
<p data-name="Peter">123</p>
```

... und wieder die Methode `data()` verwenden, ...

```
var name = $(el).data('name');
```

... wäre der Wert der Variablen *name* nun *Peter* und der Typ ein String.

Ein praktisches Beispiel zeigt Ihnen die Datei *data-attributes-jquery. html.* Darin habe ich die beiden unterschiedlichen Datenattribute in demselben Markup kombiniert:

```
<p data-id="123" data-name="Peter">Gasston</p>
```

Per jQuery habe ich den Typ jedes Datenattributs mit dem JavaScript-Operator `typeof` an die Konsole ausgegeben:

```
var el = $('p');
console.log('ID:',typeof el.data('id'));
console.log('Name:',typeof el.data('name'));
```

Die resultierende Ausgabe sehen Sie in Abb. 2–6.

ID: number	**data-a...ry.html (line 13)**
Name: string	**data-a...ry.html (line 14)**

Abb. 2–6

Den Ergebnistyp mithilfe der jQuery-data()-Methode mit Datenattributen ermitteln

2.8.3 Datenattribute in natura

Datenattribute sind so nützlich, dass manche Unternehmen bereits ausgiebig Gebrauch davon machen. Twitter hat sie schnell als Möglichkeit übernommen, Webseiten einen Tweet-Button hinzuzufügen. Bestimmte inhaltliche Parameter werden in einem Satz vordefinierter Attribute gespeichert:

```
<a href="https://twitter.com/share" class="twitter-share-button"

data-url="http://broken-links.com" data-
via="stopsatgreen">Tweet</a>
```

Das Twitter-JavaScript wird an beliebiger Stelle der Seite aufgerufen, und dieses Element wird durch einen Tweet-Button mit Verwendung der bereitgestellten Daten ersetzt. Auch viele andere Social-Media-Plattformen, etwa Facebook, Google+ und LinkedIn, nutzen Datenattribute auf dieselbe Weise.

2.9 Webkomponenten: Die Zukunft der Auszeichnungssprachen?

Web Components

Ein spannender neuer Ansatz zur Erweiterung von HTML befindet sich in der Vorschlagsphase für die sogenannten *Webkomponenten* (*Web Components*), ein Sammelname für eine Gruppe von Technologien, die die Entwicklung von reichhaltigen Benutzeroberflächen für Webanwendungen mit CSS und Markup-Code vereinfachen sollen.

Während ich dieses Buch schreibe, befindet sich die Spezifikation im

Entwurfsstadium und wird sehr wahrscheinlich noch geändert werden. Deshalb beschäftige ich mich nicht hier mit den Webkomponenten, sondern gehe in Kapitel 11 ausführlicher darauf ein.

2.10 Zusammenfassung

In diesem Kapitel haben wir uns mit einer Kernfunktion einer guten Website bzw. Webanwendung beschäftigt: der zugrunde liegenden Struktur. Wenn Sie den Code korrekt und sinnvoll auszeichnen, legen Sie den Grundstein für alles Weitere, das in diesem Buch noch besprochen wird. Diese Vorgehensweise ist äußerst wichtig, damit Ihre Sites gut zu pflegen und skalierbar sind.

Ich bin ein bisschen harsch mit den strukturierenden Elementen in HTML5 umgegangen – ob Sie sie aber einsetzen oder nicht: Sie sollten auf jeden Fall darüber nachdenken, in Ihrem Projekt WAI-ARIA zu nutzen. Die Zugänglichkeit von Content ist das Fundament des Web – und selbst wenn Sie ausschließlich Roles nutzen und sonst nichts, ziehen Sie und die Nutzer Ihrer Seiten Vorteile daraus.

Sie haben weiterhin einen Blick darauf geworden, wie Sie Ihre Sites mithilfe semantischer und strukturierter Daten sinnvoller gestalten, Sie haben die Wichtigkeit von Aboutness kennengelernt. Welchen Ansatz Sie wählen, hängt natürlich vom jeweiligen Kontext ab. Zum Aufbau einer großen, datenbankgesteuerten Website mit zahlreichen Inhalten für Endverbraucher sind wahrscheinlich tiefergehende Überlegungen zur Semantik nötig, um Suchmaschinen und Crawler bestmöglich zu bedienen. In diesem Fall sollten Sie über den Einsatz von RDFa, Mikroformaten und/oder Mikrodaten nachdenken. Wenn Sie eine Hybrid-App für Mobilgeräte aufbauen, sind diese Überlegungen hingegen viel weniger wichtig.

2.11 Literaturempfehlungen

HTML5 Doctor ist die beste Quelle für Informationen über die meisten HTML5-Themen. Sie finden beispielsweise im folgenden *Artikel von Mike Robinson* die meiner Meinung nach beste Definition der neuen Gliederungsalgorithmen: http://html5doctor.com/outlines/. Das *in Abb. 2–1 gezeigte Schaubild* können Sie von http://html5doctor.com/resources/#flowchart/ herunterladen.

Lesen Sie auch *Derek Johnsons Artikel* im Smashing Magazine: http://coding.smashingmagazine.com/2011/08/16/html5-and-the-document-outlining-algorithm/.

Für tiefere Einblick in das Problem der strukturierenden Elemente in HTML5 (Sectioning Elements) empfehle ich Luke Stevens' Buch *The Truth About HTML5*. Sie finden es auf http://www.truthabouthtml5.com/. Wenn Sie die *vollständige HTML5-Spezifikation* studieren möchten, um selbst zu entscheiden, rate ich Ihnen zur Entwicklerversion auf http://developers.whatwg.org/sections.html.

Die *vollständige WAI-ARIA-Spezifikation* finden Sie auf http://www.w3.org/TR/wai-aria/. Für Informationen über die Barrierefreiheit von HTML5 lohnt es sich, den *Paciello Group Blog* zu lesen. Wichtig ist in diesem Zusammenhang der Beitrag http://www.paciellogroup.com/blog/2010/10/using-wai-aria-landmark-roles/.

Divya Manians Artikel über Semantik erschien im Smashing Magazine auf http://coding.smashingmagazine.com/2011/11/11/our-pointless-pursuit-of-semantic-value/. Wenn Sie mehr über Aboutness und die Bedeutung der Semantik lesen möchten, empfehle ich unbedingt das Buch *Ambient Findability: What We Find Changes Who We Become* von Peter Morville (O'Reilly, 2005). Die Website http://webdatacommons.org/ liefert *Informationen und Statistiken über Sites, die strukturierte Daten nutzen.*

Lesen Sie *alles über Mikroformate* auf http://microformats.org/. Eine Überarbeitung der Syntax, Microformats 2.0, begann im Jahr 2010 und ist immer noch in Entwicklung begriffen; erfahren Sie mehr darüber auf http://microformats.org/wiki/microformats-2.

Möchten Sie mehr über das RDFa-Format lesen, hat das W3C eine exzellente *Einführung* veröffentlicht: http://www.w3.org/TR/xhtmlrdfa-primer/. Die beste *Ressource über Mikrodaten* stammt ebenfalls vom HTML5 Doctor: http://html5doctor.com/microdata/. Sind Sie ganz masochistisch veranlagt und möchten Sie die *Spezifikation im Ganzen* lesen, finden Sie diese auf http://www.w3.org/TR/microdata/.

Weitere *Informationen über Schema.org* erhalten Sie auf http://schema.org/, und die *Google-Dokumentation über Rich Snippets* befindet sich auf http://support.google.com/webmasters/bin/answer.py?hl=en&answer=99170. Das *Test-Tool* finden Sie auf http://www.google.com/webmasters/tools/richsnippets/.

John Resig schrieb in seinem Blog http://ejohn.org/blog/html-5-data-attributes/ eine einprägsame *Einführung zu den Datenattributen*, und die `data()`-Methode wird vollständig auf der jQuery-Website auf http://api.jquery.com/data/ dokumentiert.

Das ursprünglich dazugehörige Element hgroup sollte zur Gruppierung mehrerer Überschriftebenen dienen, beispielsweise einer Unterüberschrift oder eines Slogans. Es wurde aber im April 2013 wieder gestrichen (http://lists.w3.org/Archives/Public/public-html-admin/2013Apr/0003.html).

Alle Beispieldateien finden Sie über die dpunkt-Webseite zum Buch (www.dpunkt.de/gasston) oder direkt auf der Website der Originalausgabe (modernwebbook.com).

3 Mediengerechtes CSS

Sobald Sie eine solide Struktur erzeugt haben, sollten Sie sich Gedanken über die Darstellung der Inhalte machen. Es gibt hier zahlreiche Ansätze. In der Entwicklergemeinde sind momentan die adaptive und die responsive Entwicklung von Websites besonders populär.

Beide erkläre ich weiter hinten in diesem Kapitel. Im Moment müssen Sie nur wissen, dass sie sich grundsätzlich ähneln und dass es nur einen einzigen Unterschied zwischen ihnen gibt. Dieses Kapitel soll Ihnen vor allem die grundlegenden Techniken näherbringen, mit denen Sie – gleich mit welchem Ansatz – großartige Websites entwickeln können.

Eine wichtige Technik haben adaptive und responsive Methoden gemeinsam, und diese ist für die neue Welt der Multidevice-Webentwicklung absolut notwendig: Medienabfragen, meist »Media Queries« genannt.

3.1 Media Queries

Meiner Meinung nach sind Media Queries seit Jahren die größte treibende Kraft hinter den Veränderungen im Webdesign – vielleicht seit CSS selbst in der Webentwicklung populär wurde. Durch die allgemeine Unterstützung von CSS in den Browsern konnten Sie die starren Einschränkungen tabellenbasierter Websites überwinden, und Media Queries gehen einen Schritt weiter – Sie können Ihre Seiten so gestalten, dass sie sich an die Merkmale jedes einzelnen Anzeigegeräts anpassen.

Media Query und Media Features

Eine *Media Query* ist eine logische Anweisung: Ist die Abfrage wahr (*true*), werden die Stildefinitionen in der Anweisung angewandt, ist sie falsch (*false*), werden die Definitionen übersprungen. Die Parameter der Anweisung werden

Media-Eigenschaften (*Media Features*) genannt. Die heute am häufigsten verwendeten Media Queries betreffen die Abmessungen des Geräts oder des Browserfensters. Bevor wir jedoch detailliert auf Media-Eigenschaften eingehen, beschäftigen wir uns mit ihrer grundsätzlichen Funktionsweise.

Media-Types-Syntax

Media Queries erweitern die in CSS 2.1 und HTML 4.01 verwendete Media-Types-Syntax. Zur Erinnerung: Dies ist die Syntax, mit der Sie medienabhängige Stildefinitionen abrufen können, zum Beispiel beim Verlinken eines externen Stylesheets:

```
<link rel="stylesheet" href="foo.css" media="screen">
```

Dieser Code wendet das externe Stylesheet `foo.css` nur dann an, wenn das Ausgabegerät ein Bildschirm ist – also kein anderer Medientyp wie etwa ein Drucker. Sie erweitern diese Syntax, indem Sie einfach das Wort `and` hinzufügen und die Abfrage in Klammern setzen:

```
<link rel="stylesheet" href="foo.css" media="screen and (query)">
```

Dieser modifizierte Code enthält zwei Bedingungen: Der Medientyp sollte ein Bildschirm sein und die Media Query sollte *true* sein. Treffen beide Bedingungen zu, wird das Stylesheet `foo.css` angewandt.

Mit Media Queries können Sie auch externe Stylesheets aus anderen Stylesheets einschließen. Dazu verwenden Sie die @-Regel `@import`. Der folgende Code entspricht von der Logik her dem vorhergehenden Codeblock; er kann aber innerhalb von style-Elementen oder in einem externen Stylesheet verwendet werden:

```
@import url('foo.css') screen and (query);
```

Achtung

Wenn Sie @import auf diese Weise anwenden, kann es zu Performance-Problemen kommen. Deshalb sollten Sie hier Vorsicht walten lassen. Weitere Informationen erhalten Sie in Steve Souders' Blog-Beitrag

»Don't Use @import«
(http://www.stevesouders.com/blog/2009/04/09/dont-use-import/).

Schließlich können Sie Media Queries auch inline, innerhalb des CSS-Codes verwenden. Das ist praktisch, wenn Sie einen Satz von Regeln anwenden wollen und keine externen Stylesheets laden möchten.

```
@media screen and (query) { ... }
```

Nachdem Sie nun wissen, wie Media Queries funktionieren und wie Sie sie in Ihre Seiten einfügen, beschäftigen wir uns mit den Media-Eigenschaften selbst. Wir beginnen mit den am häufigsten verwendeten – den Media-Eigenschaften für die Abmessungen Ihres Ausgabegeräts oder Anzeigefensters.

3.1.1 Media-Eigenschaften für die Abmessungen

Am häufigsten verwendet man Media Queries heutzutage, um die Abmessungen des verwendeten User Agent zu ermitteln und dann die passenden Regeln für die Anzeige darauf bereitzustellen: großen Text und große Bilder für große Monitore, kleinen Text und ein einspaltiges Layout für Smartphones und Ähnliches. Natürlich steckt hinter den Media Queries mehr als solche einfachen Anwendungsfälle; aber Sie wissen jetzt, was ich meine.

Geräteabmessung und Viewport

Sie müssen zwei verschiedene Arten von Abmessungen unterscheiden: zunächst die Abmessungen des Ausgabegeräts selbst und dann den Viewport, also die Anzeigefläche des User Agent (bei den meisten Nutzern ist das ein Webbrowser; es könnte sich aber auch um ein App-Fenster handeln). Wenn Ihre Site auf einem gigantischen Breitbildfernseher betrachtet wird, die dazu genutzte App aber nur ein Viertel der Bildschirmfläche einnimmt, spielt die schiere Größe des Geräts nur eine geringe Rolle. Auf bestimmten Geräten sind die beiden Abmessungen identisch: Auf den meisten

Smartphones und Tablets entspricht beispielsweise die Breite des Browserfensters stets der des Displays des Ausgabegeräts.

Hinweis

Der Viewport selbst kann sich auf manchen Geräten ändern – mehr darüber erfahren Sie auf Seite 61 im Abschnitt »Geräteanpassung«.

height und width

Die wichtigsten Abmessungen sind wahrscheinlich die des Viewports. Die relevanten Media-Eigenschaften sind `height` und `width`. Die Features für die Abmessungen des Viewports akzeptieren eine einzelne Längenangabe als Argument. Wenn die fragliche Abmessung der jeweiligen Längenangabe entspricht, wird *true* zurückgegeben und die Regeln werden angewandt. In der folgenden Abfrage wird die Regel in den geschweiften Klammern auf das `body`-Element angewandt, wenn der Viewport genau 480 Pixel breit ist:

```
@media screen and (width: 480px) {
body { background-color: #00f; }
}
```

Hinweis

Meist arbeitet man mit der Breite. Deshalb verwende ich in den Beispielen in diesem Abschnitt `width`. Die erläuterten Techniken treffen aber ebenso auf `height` (die Höhe) zu.

Zwar verwende ich in diesem Beispiel die Einheit `px`; es sind jedoch alle Längeneinheiten möglich. Egal, welche Einheit Sie verwenden: Für die meisten Zwecke ist ein exakter Wert wahrscheinlich zu spezifisch. Zwei Ergänzungen machen die Funktion jedoch flexibler:

max-... und min-...

Viele Media-Eigenschaften, so auch `width`, erlauben vor dem Feature-Namen die Präfixe `max-` und `min-`. Diese stehen – wenig

überraschend – für »Maximum« bzw. »Minimum«. In der Praxis bedeutet dies »nicht mehr als« und »nicht weniger als«. Mit `max-width` können Sie beispielsweise allen Browsern mit einer Höchstbreite von 480 Pixel bestimmte Stildefinitionen zuweisen und mit `min-width` allen Browsern mit einer Mindestbreite von 480 Pixel andere Definitionen:

```
@media screen and (max-width: 480px) { ... }
@media screen and (min-width: 480px) { ... }
```

In der Datei `mq-width.html` finden Sie ein einfaches Beispiel für den Einsatz dieser Präfixe. Die Seite enthält drei `iframe`-Elemente mit den Breiten 190 Pixel, 200 Pixel und 210 Pixel und jeweils einem eindeutigen `id`-Wert:

```
<iframe src="mq-hello.html" width="190" id="a"></iframe>
<iframe src="mq-hello.html" width="200" id="b"></iframe>
<iframe src="mq-hello.html" width="210" id="c"></iframe>
```

Weil ein `iframe` einen neuen Viewport erstellt, können Sie den einzelnen Viewport-Größen mit der Media-Eigenschaft `width` Regeln zuweisen. Die Seite `mq-hello.html`, die in den Inline-Frames angezeigt wird, enthält im Kopfbereich einen CSS-Block, der den Elementen je nach den zutreffenden Media-Eigenschaften unterschiedliche `border-style`-Formatierungen zuweist. Diese Aufgabe wird mit den folgenden drei Media Queries gelöst:

```
❶ @media screen and (width: 200px) {
    h1 { border-style: solid; }
  }
❷ @media screen and (min-width: 205px) {
    h1 { border-style: dotted; }
  }
❸ @media screen and (max-width: 195px) {
    h1 { border-style: dashed; }
  }
```

Die Regeln werden folgendermaßen zugewiesen: ❶ iframe `#b` hat eine Breite von 200 Pixel; ❷ iframe `#c` ist mindestens 205 Pixel breit, ❸ iframe `#a` ist höchstens 195 Pixel breit. Das Ergebnis sehen Sie in Abb. 3–1.

Abb. 3–1
Mithilfe von Media Queries werden unterschiedlich breiten Viewports unterschiedliche Regeln zugewiesen.

Sie brauchen sicherlich nicht viel Fantasie, um sich den Nutzen des `width`-Features vorzustellen: Sie könnten beispielsweise auf kleineren Bildschirmen kleineren Text verwenden und auf größeren Bildschirmen größeren Text.

```
@media screen and (max-width: 480px) {
  h1 { font-size: 2em; }
}
@media screen and (min-width: 481px) {
  h1 { font-size: 3.6em; }
}
```

Hinweis

Die hier verwendeten Abfragen dienen nur zur Verdeutlichung. Es gibt eine bessere Möglichkeit, mit der wir uns im Abschnitt »Mobile First und Breakpoints für den Inhalt« auf Seite 73 beschäftigen.

device-width und device-height

Wenn Sie sicher sind, dass Sie nicht mit den Abmessungen des Viewports, sondern mit denen des Geräts arbeiten möchten, können Sie die Features `device-width` und `device-height` verwenden – wobei ich, wie zuvor gesagt, der Ansicht bin, dass die Breite als Maß häufiger verwendet wird als die Höhe. Also konzentriere ich mich auf das Feature `width`. Der Einsatz des Features `device-width` entspricht in der Praxis der Verwendung des `width`-Features – es arbeitet nur mit einer anderen Metrik. Wie `width` können Sie auch `device-width` durch die Präfixe `max-` und `min-` erweitern:

```
@media screen and (max-device-width: 799px) { ... }
@media screen and (device-width: 800px) { ... }
@media screen and (min-device-width: 801px) { ... }
```

Aspect Ratio

Eine andere Möglichkeit nutzt das Seitenverhältnis des Viewports oder des Geräts. Die *Aspect Ratio* ist das Verhältnis von `width` zu `height` (oder `device-width` zu `device-height`); ein 1024 × 768 großer Monitor hat beispielsweise das bei älteren Bildschirmen übliche Seitenverhältnis von 4/3 – vier Pixel in der Waagerechten auf drei Pixel in der Senkrechten. Neuere Breitbildgeräte (etwa das iPhone 5) haben üblicherweise ein Seitenverhältnis von 16/9.

Hinweis

Auf Seite 57 im Abschnitt »Ein kleiner Exkurs: Alles über Pixel« gehe ich näher auf Pixel ein.

Um einen Monitor mit dem Seitenverhältnis 4/3 anzusprechen, verwenden Sie die Eigenschaft `device-aspect-ratio`:

```
@media screen and (device-aspect-ratio: 4/3) { ... }
```

Die Präfixe `max-` und `min-` gibt es bei diesen Features ebenfalls. Für eine Abfrage, mit der Sie nur Viewports – keine Geräte – mit einem Seitenverhältnis von mindestens 8/5 ansprechen, verwenden Sie also das Feature `min-aspect-ratio`:

```
@media screen and (min-aspect-ratio: 8/5) { ... }
```

So können Sie schnell ermitteln, welches Seitenverhältnis größer ist: Teilen Sie die erste Zahl durch die zweite. Das Seitenverhältnis 4/3 ergibt 1,333, während das Seitenverhältnis 8/5 1,6 ergibt. Je größer das Ergebnis, desto breiter das Seitenverhältnis. Das hier gezeigte Beispiel wird angewandt, wenn diese Zahl mindestens 1,6 beträgt.

orientation

Wichtiger als der ganz exakte Wert eines Seitenverhältnisses ist wahrscheinlich die

Ausrichtung des Viewports – Hochformat
oder Querformat. Sie fragen dies mit der Eigenschaft `orientation` ab. Dieses Feature ist besonders bei Mobilgeräten wie Handys und Tablets nützlich, bei denen der Anwender die Ausrichtung häufig ändern kann:

```
@media screen and (orientation: portrait) { ... }
```

Die Präfixe `max-` und `min-` gibt es bei diesem Feature aus (hoffentlich) offensichtlichen Gründen nicht.

3.1.2 Media Queries kombinieren und negieren

Durch die wiederholte Verwendung des `and`-Schlüsselworts können Sie eine Folge logischer Anweisungen mit zusätzlichen Media Queries hinzufügen. Mit `and` werden die Regeln dann nur angewandt, wenn die *gesamte* Logik der Media Query zutrifft. Möchten Sie beispielsweise prüfen, ob ein Viewport das Querformat hat *und* mindestens 800 em breit ist, verwenden Sie den folgenden Code:

```
@media all and (orientation: landscape) and (min-width: 800em)
{...}
```

Dies lässt sich noch weiter vereinfachen: Der Medientyp `all` ist der Standardzustand; also können Sie ihn aus der Abfrage herauslassen, wodurch das erste Vorkommen von `and` überflüssig wird:

```
@media (orientation: landscape) and (min-width: 800em) { ... }
```

Sie können eine Abfolge von Media Queries erzeugen und die Regeln zuweisen, wenn *irgendeine* davon zutrifft. Dazu nutzen Sie den Komma-Separator so, dass er wie ein `or`-Operator wirkt. In diesem Beispiel werden die Regeln angewandt, wenn sich das Gerät entweder im Querformatmodus befindet *oder* die Breite des Viewports mindestens 800 em beträgt:

```
@media (orientation: landscape), (min-width: 800em) {}
```

Schließlich können Sie die Logik einer Media Query auch umkehren,

indem Sie den `not`-Operator am Beginn der Abfrage hinzufügen. Dieser Operator bedeutet, dass die Regeln angewandt werden, wenn die Logik der Abfrage `false` ist. Mit der folgenden Zeile werden die Regeln allen Geräten zugewiesen, die nicht das Seitenverhältnis 8/5 haben:

```
@media not all and (device-aspect-ratio: 8/5) {}
```

Die richtige Wahl der Media Queries für Ihre Site kann recht komplex sein und viele unterschiedliche Variablen beinhalten. Sie können damit jedoch äußerst gute Ergebnisse erzielen. Ich empfehle Ihnen unbedingt, sich Zeit zu nehmen und sich mit der Funktionsweise von Media Queries vertraut zu machen. Zoe Mickley Gillenwater verfasste zu dem Thema den hervorragenden Beitrag »*Essential Considerations for Crafting Quality Media Queries*«. Sie schreibt:

> *Die Entwicklung von Weblayouts mit Media Queries ist ein* Prozess. *Sie müssen sie von Anfang an berücksichtigen und an bestimmten Punkten Entscheidungen treffen, wie Sie sie so integrieren, dass es für Ihre Site am sinnvollsten ist. Es gibt nur wenige grundsätzlich richtige oder grundsätzlich falsche Antworten.*

Diesen hervorragenden Beitrag sollten Sie im Ganzen lesen und überdenken.

3.1.3 Ein kleiner Exkurs: Alles über Pixel

Im Multidevice-Zeitalter sollten wir nicht nur die Abmessungen eines Anzeigegeräts berücksichtigen, sondern auch seine Auflösung. Viele neue Geräte, besonders Smartphones, besitzen mittlerweile hochauflösende Displays. Wenn Sie also für jeden Anwender eine optimale User Experience bieten möchten, müssen Sie überlegen, wie Bilder und andere Objekte auf diesen Geräten angezeigt werden. Bevor ich mich detailliert damit beschäftige, möchte ich kurz auf Pixel eingehen.

Physische Pixelanzahl

Zunächst muss ich hierzu einige Begriffe definieren, wobei ich voraussetze, dass der

Leser weiß, was ein Pixel ist. Die *physische Pixelanzahl* eines Monitors ist die Anzahl der Pixel, die in der Breite auf den Bildschirm passen. Zum Beispiel hat ein Monitor mit einer Auflösung von 1280 × 800 Pixel eine physische Pixelanzahl von 1280 entlang seiner breiten Seite. Das iPhone 3GS hat in seiner standardmäßigen (querformatigen) Ausrichtung eine physische Pixelbreite von 320, das iPhone 4 und höher mit dem »Retina«-Display hat eine Breite von 640 Pixel.

Pixeldichte

Die *Pixeldichte* eines Monitors wird üblicherweise in *Pixel pro Zoll (Pixels per Inch = PPI)* gemessen – wenn Sie das metrische System bevorzugen, können Sie auch *Pixel pro Zentimeter (Pixels per Centimeter = PPC)* verwenden. Ich bleibe hier jedoch beim angelsächsischen Maßsystem. Dabei wird die Maßeinheit auch manchmal – selbst in bekannten Bildbearbeitungsprogrammen – als *Dots per Inch (DPI)* angegeben; die beiden Begriffe PPI und DPI können als austauschbar betrachtet werden. Auf jeden Fall ist die Pixeldichte ein Maß für die Anzahl der Pixel, die auf einer Strecke von einem Zoll auf dem Display des Ausgabegeräts Platz finden.

Der überwiegende Anteil der momentan genutzten Monitore hat nach wie vor die PPI-Auflösung 96. Mit dem Aufkommen der Smartphones hat sich die PPI-Anzahl jedoch stetig erhöht; das erste iPhone und die Folgeversionen bis zum 3GS haben eine Auflösung von 163 ppi. Beim iPhone ab Version 4 wurde die Auflösung auf unglaubliche 326 ppi erhöht (auch wenn sich die Größe des Displays selbst nicht verändert hat – gleich erfahren Sie, warum). So weit, so einfach – aber das ist nicht alles. Für die Mehrzahl der hochauflösenden Displays gibt es nun eine zusätzliche, virtuelle Pixeleinheit: *Density Independent Pixel (DIP)* oder *CSS-Pixel*. Es handelt sich dabei um eine relative Einheit: Ein physisches Pixel entspricht einer beliebigen Anzahl DIPs. Damit können Sie Ihre Websites hoch- und hinunterskalieren, sodass Sie insgesamt eine bessere Usability auf kleineren Anzeigegeräten erzielen.

Device Pixel Ratio

Das Verhältnis von virtuellen zu physischen Pixeln wird *Device Pixel Ratio* genannt. Ich verwende dafür die Abkürzung *DPR*, wobei man auch von *DPPX (dots per pixel)* spricht. Geräte ohne DIPs haben eine DPR von 1 – das heißt, ein virtuelles Pixel entspricht einem physischen Pixel. Während dieses Buch entstand, hatten die am weitesten verbreiteten Geräte (zum Beispiel das Samsung Galaxy S III oder das iPhone 5) eine DPR von 2 – auf ein physisches kommen zwei virtuelle Pixel.

Betrachten Sie dazu Abb. 3–2. Links sehen Sie einen Block aus 2×2 physischen Pixeln. In der Mitte erkennen Sie, dass auf einem Display mit einer DPR von 1,5 9 DIPs denselben Platz einnehmen. Rechts sehen Sie ein Display mit einer DPR von 2 – hier passen 16 DIPs auf denselben Raum.

Abb. 3–2
Der Vergleich unterschiedlicher DPR-Werte: 1 (links), 1,5 (Mitte) und 2 (rechts)

Per JavaScript können Sie die DPR jedes Geräts herausfinden, da die meisten Browser mittlerweile die DOM-Eigenschaft `devicePixelRatio` des `window`-Objekts unterstützen. Um die DPR des von Ihnen verwendeten Geräts herauszufinden, geben Sie einfach in das Entwicklertool Ihres Browsers Folgendes ein:

```
window.devicePixelRatio;
```

Wenn Ihr Browser kein solches Tool beinhaltet, nutzen Sie das Skript in der Beispieldatei *mq-dpr.html*.

Bis hierhin war alles ziemlich technisch und, um ehrlich zu sein, habe ich trotzdem gerade einmal an der Oberfläche gekratzt. Ich hoffe jedoch, dass Sie nun genug von Auflösung und Pixeln verstehen, um aus dem Rest dieses Kapitels maximalen Nutzen zu ziehen.

3.1.4 Media Queries für die Bildschirmauflösung

Wie Sie gerade gesehen haben, werden viele Geräte mittlerweile mit sehr hochauflösenden Displays ausgeliefert. Sicherlich ist dies positiv – jedoch haben diese Displays auch den einen oder anderen Nachteil, vor allem hinsichtlich der Darstellung von Grafiken. Bestimmt wissen Sie, dass es zwei Arten von Grafiken gibt: *Vektorgrafiken*, beispielsweise im SVG-Format, bestehen aus einer Reihe von Koordinaten, aus denen auflösungsunabhängige Formen konstruiert werden (mehr über SVG erfahren Sie in Kapitel 7). *Bitmap-* oder *Pixelbilder*, beispielsweise in den Formaten JPG oder PNG, bestehen aus einer Anzahl unterschiedlich gefärbter Quadrate, und ihre Darstellung ist deshalb auflösungsabhängig.

Der praktische Unterschied zwischen beiden Grafikarten ist, dass Pixelbilder eine bestimmte PPI-Auflösung haben und dass ein Pixelbild, das auf einem Gerät mit einer niedrigen DPR wunderbar aussieht, auf einem Gerät mit einer hohen DPR ziemlich lausig wirkt. Also müssen Sie für Displays mit höherer Auflösung höchstwahrscheinlich Pixelbilder mit höherer PPI verwenden. Und genau zu diesem Zweck gibt es zwei Media-Eigenschaften.

-webkit-device-pixel-ratio

Die erste und wegen der Popularität von WebKit auf Mobilgeräten am meisten verwendete Media-Eigenschaft ist `-webkit-device-pixel-ratio`. Im Zusammenspiel mit den Präfixen `max-` und `min-` können Sie damit die DPR eines Geräts ansprechen. Hier sehen Sie beispielsweise, wie Sie ein hochaufgelöstes Bild für Geräte mit einer DPR von mindestens 2 hinzufügen:

```
E { background-image: url('foo.png'); }
@media screen and (-webkit-min-device-resolution: 2) {
```

```
  E { background-image: url('foo-hi-res.png'); }
}
```

Achtung

Denken Sie daran, dass ein Gerät mit hoher Auflösung nicht automatisch mit einer guten Internetverbindung genutzt wird. Der Anwender wird Ihnen nicht gerade dankbar sein, wenn er ein 600-KB-Bild über eine langsame 3D-Verbindung herunterladen muss.

resolution

Alles ziemlich einfach, nicht wahr? Nein. `-webkit-device-pixel-ratio` ist, wie erwähnt, kein Teil der Media-Queries-Spezifikation, sondern eine WebKit-proprietäre CSS-Erweiterung. Die Spezifikation enthält jedoch eine ähnliche Abfrage: `resolution`. Sie ist `-webkit-device-pixel-ratio` überlegen, weil hinsichtlich der möglichen Werte flexibler. Sie können beispielsweise die Einheit `dpi` verwenden. Zur Erinnerung: Ein Standardmonitor hat eine DPI (oder PPI) von 96; vor diesem Hintergrund können Sie diese Monitore mit dem folgenden Code ansprechen:

```
@media screen and (resolution: 96dpi) {
  E { background-image: url('foo.png'); }
}
```

Sie können aber auch Bildschirme mit höherer Auflösung ansprechen, indem Sie die Standardauflösung 96 mit der DPR multiplizieren. Zum Beispiel gilt die folgende Abfrage für Geräte mit einer DPR von 2 oder mehr, genau wie im `-webkit-device-pixel-ratio`-Beispiel ein paar Absätze früher:

```
@media screen and (min-resolution: 192dpi) {
 E { background-image: url('foo-hires.png'); }
}
```

dppx

Es gibt jedoch eine noch bessere Möglichkeit: Verwenden Sie die Einheit `dppx`. Diese

entspricht der DPR, sodass sich eine verbesserte Version des vorigen Beispiels folgendermaßen präsentiert:

```
@media screen and (min-resolution: 2dppx) {
  E { background-image: url('foo-hires.png'); }
}
```

Alles zusammen

`-webkit-device-pixel-ratio` gibt es seit der Einführung des iPhones im Jahr 2006 – und es sind nach wie vor noch sehr viele ältere Geräte im Einsatz, sodass diese Erweiterung nicht so schnell überflüssig werden wird. Vor diesem Hintergrund sollten Sie beide Möglichkeiten implementieren, indem Sie die Media Query erweitern. Eine kommaseparierte Liste von Media Queries weist – wie erwähnt – die Regeln zu, wenn eine von ihnen *true* ist.

```
@media screen and (min-resolution: 2dppx),
      screen and (-webkit-device-pixel-ratio: 2)  {
  E { background-image: url('foo-hires.png'); }
}
```

Diese Logik testet, ob das Gerät eine DPR von mindestens 2 hat, und zwar sowohl in WebKit- als auch in standardkonformen Browsern.

Hinweis

Noch komplizierter wird die Sache dadurch, dass der Anwender in manchen Browsern – etwa Opera Mobile – die Pixeldichte ändern kann.

3.1.5 Geräteanpassung

Mobilbrowser können Websites für die bessere Darstellung auf kleinen Geräten skalieren. Dabei wird vorausgesetzt, dass die meisten Websites für Desktop-Browser gedacht sind. Diese Annahme bedeutet, dass die Sites für eine bestmögliche Anpassung standardmäßig ausgezoomt/herunterskaliert werden. Man spricht hier vom Layout-

Viewport, und dieser hat den Nachteil, dass Sites, die von vornherein für Mobilgeräte optimiert sind, in der ausgezoomten Ansicht sehr klein dargestellt werden (Sie erkennen dies im linken Teil von Abb. 3–3).

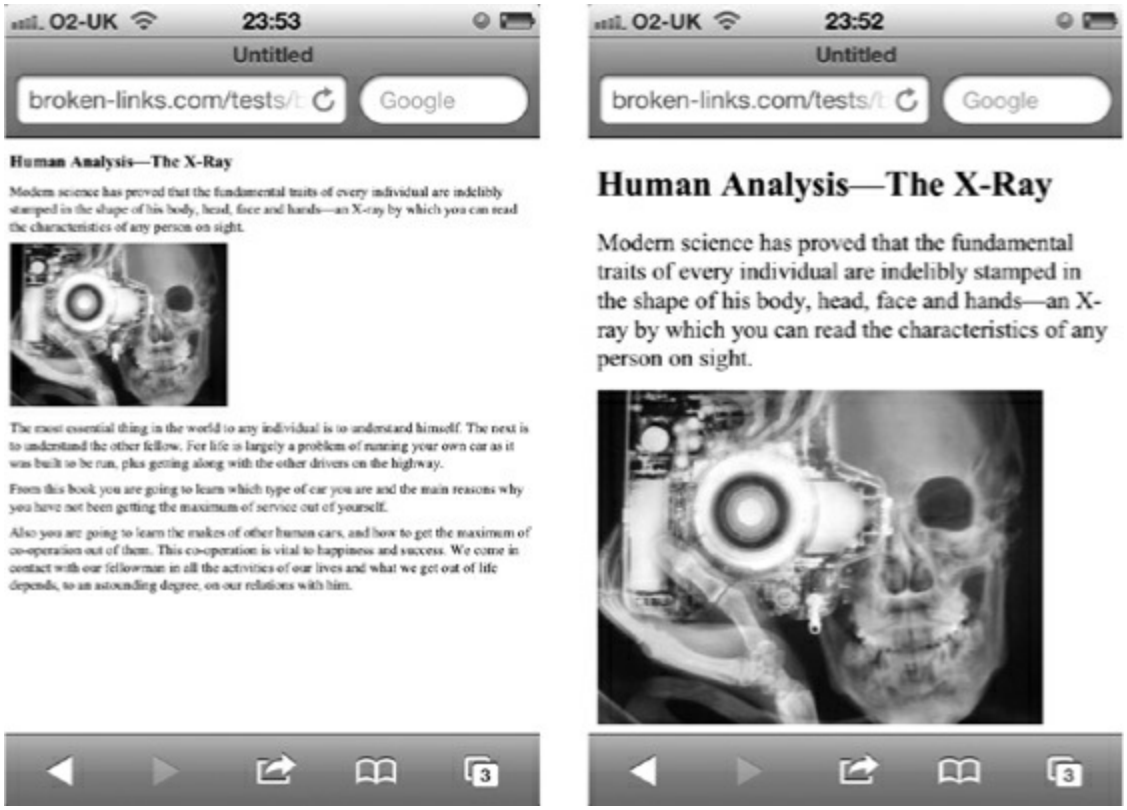


Abb. 3–3
Dieselbe Seite auf einem iPhone vor (links) und nach (rechts) der Anwendung des Meta-Tags viewport

viewport-Meta-Tag

Als Apple Safari für iPhone entwickelte, stellte die Firma den Entwicklern ein spezielles `viewport`-*Meta-Tag* zur Verfügung, das in den Head-Bereich der Dokumente eingefügt werden sollte. Mit diesem Tag werden die Abmessungen und die Skalierung des Browser-Viewports kontrolliert. Hier ein Beispiel:

```
<meta name="viewport" content="width=device-width">
```

Im `content`-Attribut können Sie eine Anzahl unterschiedlicher Werte verwenden. Die Werte in der obigen Codezeile bedeuten: »Gleiche die

Breite dieses Dokuments mit der Breite dieses Geräts ab.« Den Unterschied erkennen Sie deutlich in Abb. 3–3. Sie zeigt dasselbe Dokument vor und nach dem Zuweisen dieses Tags.

Im linken Screenshot hat der Browser den Haupttext automatisch so angepasst, dass er auf das Display passt; Überschrift und Bild sind deshalb sehr klein geworden. Im rechten Screenshot mit der Anwendung des Meta-Viewport-Tags passen Überschrift und Bild von der Größe her besser zum Display.

Die Breite eines Geräts variiert, wird aber auch von seiner DPR bestimmt. Der Bildschirm des iPhone 3GS hat, wie erwähnt, eine Breite von 320 Pixel. Weil jedoch das »Retina«-Display des iPhone eine physische Pixelanzahl von 640 px sowie eine DPR von 2 hat, ist es nur 320 px breit – die DIPs –, wenn das Viewport-Meta-Tag mit dem Wert `device-width` verwendet wird. Andere Parameter für dieses Tag kontrollieren die initiale Zoomstufe der Seite und ob der Nutzer selbst zoomen kann. Im folgenden Beispiel ist die initiale Zoomstufe 1,5-fach, und der Nutzer kann dies nicht ändern:

```
<meta name="viewport" content="initial-scale=1.5,user-
scalable=no">
```

Achtung

Die fehlende Möglichkeit zum Auszoomen kann für den Nutzer sehr ärgerlich sein. Wägen Sie sorgfältig ab, bevor Sie diese Option anwenden.

Obwohl das Viewport-Meta-Tag seine Hauptfunktion gut erfüllt, hat es einen Nachteil: Die Verwendung eines Meta-Tags bedeutet, dass Sie nur einen Satz Parameter pro Seite definieren können. Beispielsweise könnten Sie statt der standardmäßigen Gerätebreite eine feste Größe definieren:

```
<meta name="viewport" content="width=480">
```

Das Problem ist, dass nun alle Geräte, die das Viewport-Tag erkennen, ihr Zoom-Level auf 480 px setzen. Bei kleineren Mobilgeräten ist dies sinnvoll, bei größeren Tablets eher nicht.

Die Lösung besteht darin, die Viewport-Parameter in CSS festzulegen. Sie verwenden dazu die @-Regel `@viewport`. Während das Viewport-Meta-Tag verschiedene Parameter im `content`-Attribut akzeptiert, nutzt `@viewport` die CSS-Syntax, um dieselben Funktionen mithilfe einer Untermenge spezialisierter Eigenschaften auszuführen. Ich illustriere dies durch eine Modifikation des vorigen Beispiels:

```
@viewport { width: 480px; }
```

Diese Lösung ist deshalb sinnvoller, weil Sie `@viewport` mit Media Queries kombinieren können, um – abhängig vom Gerät – unterschiedliche Viewport-Parameter zu erzielen. Nehmen wir das folgende Beispiel:

```
@media screen and (max-width: 480px) {
  @viewport {
   width: 480px;
  }
}
@media screen and (min-width: 481px) {
  @viewport {
   width: device-width;
  }
}
```

Auf Monitoren mit einer Breite von 480 Pixel oder weniger wird der Viewport mit 480 Pixel angezeigt und auf größeren Bildschirmen in seiner natürlichen Größe.

Device Adaptation

Dieser Ansatz wird *Device Adaptation (Geräteanpassung)* genannt und hat den Vorteil, dass Sie nun viel einfacher Änderungen in der gesamten Site vornehmen können: Sie müssen nur eine einzige CSS-Regel ändern und nicht möglicherweise zahlreiche Einzelseiten.

3.1.6 Media-Eigenschaften für die Benutzerinteraktion

Ich habe nun sehr viel über die Abmessungen oder die Auflösung des Displays berichtet. Allerdings sind noch weitere Schlüsselfaktoren wichtig – beispielsweise die Art der Benutzerinteraktion mit dem Gerät. Viele Jahre lang war die Maus der dominante Eingabemechanismus. In der neuen Gerätelandschaft sind jedoch auch Berührung, Stift und Stimme wichtig geworden – und wer weiß, was die Zukunft in dieser Hinsicht noch bringen wird.

Wegen der vielfältigen Eingabemechanismen wurden in der Spezifikation Media Queries Level 4 zwei neue Media-Eigenschaften vorgeschlagen. Mit diesen können Sie Stilregeln aufgrund der Art der Benutzerinteraktion mit Ihren Seiten zuweisen.

pointer

Das erste ist das `pointer`-Feature mit drei verfügbaren Parametern: Von den heute am häufigsten genutzten Eingabemechanismen hat ein Gerät mit Tastatur oder Stimmkontrolle keinen Zeiger – also wird es als `none` klassifiziert. Fingerbasierte Touchscreens wie ein Tablet oder Smartphone werden am wahrscheinlichsten als `coarse` klassifiziert und eine Maus oder ein Stift, die sehr feine Kontrolle ermöglichen, als `fine`. Vor diesem Hintergrund eine Beispielabfrage, die Hyperlinks auf einem Touchscreen oder anderen wenig präzisen Geräten mit zusätzlichem Abstand versieht:

```
@media screen and (pointer: coarse) {
  a { padding: 1em; }
}
```

hover

Den Hover-Zustand gibt es nur bei Geräten mit Mauseingabe. Er entdeckt, wenn ein Bildschirmzeiger über einem Element positioniert wird. Logischerweise existiert diese Funktion nicht für Berührung oder Stimmeingabe. Dazu ist ein `hover`-Feature verfügbar, das das Vorhandensein eines Hover-Zustands abfragt. Für die boolesche Hover-Abfrage wird kein zweiter Wert benötigt:

```
@media screen and (hover) {}
```

Wenn Sie Regeln für Geräte ohne Hover-Zustand definieren möchten, sollten Sie entweder den Wert 0 oder den `not`-Operator verwenden. Deshalb sind die folgenden beiden Regeln gleichwertig:

```
@media screen and (hover:0) {}
@media not screen and (hover) {}
```

Achtung

Dies sind nur Vorschläge. Während ich dies schreibe, sind sie zwar bereits in WebKit implementiert, können sich aber noch ändern.

3.1.7 Weitere Media-Eigenschaften

monochrome und color

Es gibt eine Handvoll weiterer Media-Eigenschaften. Bei vielen davon geht es um Geräteeigenschaften, die für die meisten Entwickler nicht wichtig sind, bei manchen etwa um die Farbe. Wenn Sie beispielsweise für E-Book-Reader entwickeln, die dank der Anzeigetechnik des »elektronischen Papiers« teilweise ein graues Display haben, sollten Sie sich mit der Abfrage `monochrome` und ihrem Gegenstück `color` beschäftigen:

```
@media screen and (color) {}
@media screen and (monochrome) {}
```

Ich muss dazu sagen, dass sich in meinen (begrenzten) Tests die Browser auf Graustufen-Displays nicht immer als monochrom ausweisen, selbst wenn diese Features vollständig unterstützt werden. Auch hier gilt also: Führen Sie Tests auf möglichst vielen Geräten durch.

Bezüglich der übrigen Features in der Media-Queries-Spezifikation müsste ich einfach nur wiederholen, was in der Dokumentation steht – sie sind stark auf bestimmte Geräte ausgerichtet, auf die ich keinen Zugriff habe. Sie sollten unbedingt selbst einen Blick in die Spezifikation

werfen und entscheiden, ob diese Features für Ihr Projekt relevant sind.

script

Im Arbeitsentwurf der Media Queries Level 4 finden Sie das nützliche Feature `script`. Dieses testet, ob der Browser JavaScript (oder korrekter ECMAScript) unterstützt und ob dies aktiviert ist. `script` ist ein Boolean, Sie können also ohne einen zusätzlichen Wert herausfinden, ob Skripte unterstützt werden:

```
@media screen and (script) {}
```

Während ich dieses Kapitel schreibe, ist dieses Feature ein Vorschlag und in keinen Browser implementiert. Deshalb könnte es jederzeit wieder verschwinden. Weil es jedoch sehr nützlich sein könnte, erwähne ich es aber trotzdem.

3.2 Media Queries mit JavaScript

Die Vorteile von Media Queries in CSS liegen auf der Hand. Es verwundert also wenig, dass sie auch in JavaScript übernommen wurden. Sie können also je nach Anzeigegerät bestimmte Funktionen ausführen oder externe Skripte laden. Dadurch richten Sie nicht nur die Präsentation, sondern auch das Verhalten Ihrer Website ganz auf das Anzeigegerät aus.

matchMedia()

Mit der Methode `matchMedia()` führen Sie die Abfragen aus, die in einem String als Argument übergeben werden. Führen Sie die folgende Zeile in Ihrer Entwicklerkonsole aus:

```
window.matchMedia('screen and (min-width: 800px)');
```

Das Ergebnis ist ein `MediaQueryList`-Objekt. Wie es in Ihrer Konsole erscheint, hängt vom Browser und den verwendeten Tools ab; Abb. 3–4 zeigt das Ergebnis in Firebug.

Abb. 3–4
Das Ergebnis einer matchMedia()-Abfrage in der Firebug-Konsole

An diesem Objekt interessiert mich der Wert der Eigenschaft `matches`, die je nach der Abfragelogik entweder `true` oder `false` zurückgibt. Mit diesem Wert können Sie problemlos eine `if`-Anweisung erstellen und je nach dem Ergebnis bestimmte Aktionen ausführen. Im folgenden Beispiel wird die Funktion `foo()` ausgeführt, wenn der Viewport mindestens 800 px breit ist:

```
var mq = window.matchMedia('screen and (min-width: 800px)');
if (mq.matches) { foo(); }
else { // do something else }
```

In der Datei *mq-matches.html* sehen Sie ein kurzes Skript, das ich auf Grundlage dieser Logik geschrieben habe. Es gibt einfach eine Meldung aus, die Ihnen mitteilt, ob Ihr Browserfenster mindestens 800 Pixel breit ist. Ändern Sie die Größe Ihres Browserfensters und laden Sie die Seite neu, erhalten Sie ein anderes Ergebnis.

Wegen der dynamischen Natur von JavaScript können Sie dies durch einen Listener erweitern, der reagiert, sobald sich die Eigenschaften des Ausgabegeräts ändern – wenn der Nutzer beispielsweise die Größe seines Browserfensters oder die Ausrichtung seines Geräts ändert. Ich kann den vorherigen Code durch die neue Funktion `widthWatch()` erweitern. Diese enthält die `if-else`-Logik, die aufgerufen wird, sobald sich der Viewport ändert und der Listener deshalb reagiert:

```
function widthWatch(mq) {
  if (mq.matches) { foo(); }
```

```
  else { // do something else }
}
var mq = window.matchMedia('screen and (min-width: 800px)');
mq.addListener(widthWatch);
widthWatch(mq);
```

In der Beispieldatei `mq-widthwatch.html` sehen Sie die Ergebnisse eines solchen Skripts; es gibt dieselben Ergebnisse wie im vorigen Beispiel zurück. Sie müssen die Seite aber nicht aktualisieren, um ein anderes Ergebnis zu erzielen – ändern Sie einfach die Browserfenstergröße.

3.3 Adaptives oder responsives Webdesign?

Wie zu Beginn dieses Kapitels erwähnt, werden für die moderne Website-Entwicklung vor allem zwei Methoden verwendet, die als adaptiv und responsiv bekannt sind. Beiden gemeinsam ist das Konzept der `Breakpoints`. Dies sind durch Media Queries definierte Punkte, an denen sich das Layout der Site ändert.

Der Unterschied zwischen der adaptiven und der responsiven Technik ist das Verhalten der Site zwischen diesen Breakpoints. Bei der *adaptiven* Technik verwenden Sie grundsätzlich mehrere Layouts mit fixer Breite, während Sie bei der *responsiven* Technik flexible Abmessungen verwenden, sodass sich die Site sogar zwischen den Breakpoints an eine geänderte Display-Größe anpasst.

Fluid Design

Meiner Meinung nach ist es in Anbetracht der enormen Variabilität der Bildschirmgrößen wahrscheinlich vergebliche Mühe, eine Reihe von Seiten mit fester Breite zu entwickeln, um wenigstens die häufigsten Konfigurationen abzudecken. Der bessere Ansatz und der Schlüssel zur responsiven Webentwicklung ist ein sogenanntes flüssiges Design (*Fluid Design*). Dabei werden die Längenmaße beispielsweise in Prozent angegeben, sodass die Abmessungen der Seitenelemente relativ zur Viewport-Größe dargestellt werden.

Früher war die Entwicklung einer Seite mit einer festen Breite von 960 Pixel akzeptabel, weil Sie davon ausgehen konnten, dass Ihre Besucher vor allem Desktop- oder Laptop-Computer mit mindestens 1024 Pixel breiten Monitoren nutzten. Diese Tage sind jedoch vorbei, und wenn Sie Ihre Websites heutzutage auf diese Weise gestalten, erhalten Benutzer mit Mobilgeräten einen herunterskalierten Bildschirm, den sie ein- und auszoomen und auf dem sie umherscrollen müssen, damit sie alles erkennen können. Kein Weltuntergang – aber auch nicht gerade ideal.

Die Verwendung von Prozentwerten statt fester Werte bedeutet, dass Ihre Seitenelemente je nach Größe des Viewports größer oder kleiner werden. Dadurch umbricht der Inhalt innerhalb der Monitorgrenzen – deshalb der Ausdruck »Fluid« (flüssig). Wenn Sie dies mit Media Queries für den Inhalt oder die Geräte kombinieren, haben Sie damit die Grundlage für ein responsives Design und eine auf den Benutzer zugeschnittene User Experience geschaffen – gleichgültig, welches Gerät er verwendet.

Problem: Längeneinheiten mischen

Die Verwendung von Prozentwerten ist nicht ohne Tücken. Ein Problem ist die Schwierigkeit, Längeneinheiten zu mischen. Stellen Sie sich beispielsweise eine Seite mit drei Spalten vor: einer mittleren Spalte mit einer Breite von 50 Prozent und rechts und links je einer Spalte mit einer Breite von 25 Prozent:

```
E, G { width: 25%; }
F { width: 50%; }
```

Nehmen wir nun an, dass Sie links und rechts von der mittleren Spalte jeweils 20 px Spaltenzwischenraum hinzugeben möchten. Dieser Wert soll aber nicht variabel sein – es sollen immer exakt 20 px sein:

```
F {
  padding: 0 20px;
  width: 50%;
}
```

Vielleicht erkennen Sie das Problem bereits. Wie Sie wissen, werden im

CSS-Box-Modell Innenabstand (padding), Rahmen und Ränder den Abmessungen des Inhalts (width) hinzugefügt. Also haben diese Elemente nun eine Gesamtbreite von 100 Prozent plus 40 px und sind dadurch breiter als ihr Container.

Problem: Elemente verschachteln

Ein weiteres Problem bei der Nutzung prozentualer Breiten: Die Verschachtelung von Elementen führt zu recht seltsamen Zahlenwerten. Stellen Sie sich beispielsweise ein Element vor, das 85 Prozent der Viewport-Breite einnehmen soll. Darin befindet sich ein untergeordnetes Element, das 55 Prozent der Viewport-Breite einnehmen soll. Da Prozentwerte relativ sind, können Sie die Breite leider nicht einfach auf 55 Prozent setzen; vielmehr müssen Sie 55 Prozent von 85 Prozent berechnen und erhalten Folgendes:

```
E { width: 85%; }
F { width: 64.70588%; }
```

Sie sehen, dass ich nach der fünften Dezimalstelle aufgehört habe.

Wenn Sie also auf diese Weise arbeiten, müssen Sie möglicherweise eine Menge Berechnungen anstellen. Noch schlimmer: Die verschiedenen Browser runden diese Werte auf unterschiedliche Weise nach oben oder unten, sodass keine Pixelgenauigkeit garantiert werden kann.

Allein dieser Nachteil könnte ein Grund sein, auf responsive Layouts zu verzichten. Ein paar Features in CSS3 sollten Ihnen jedoch helfen, diese Probleme zu lösen.

3.3.1 Die Eigenschaft »box-sizing«

Einheiten mischen

Bis zu einem gewissen Punkt lässt sich das gerade erwähnte Problem der gemischten Einheiten mit der CSS-Eigenschaft `box-sizing` umgehen. Nehmen wir wieder das aktuelle Codebeispiel: Ich habe

drei Spalten mit einem Abstand von 20 px – die Gesamtbreite ist größer als die des übergeordneten Elements:

```
E, G { width: 25%; }
F {
  padding: 0 20px;
  width: 50%;
}
```

Mit der Eigenschaft `box-sizing` können Sie die Begrenzung ändern, ab der die Breite gemessen wird – also enthält die festgelegte Breite den Rahmen und den zusätzlichen Abstand (*Padding*). In diesem Beispiel kann ich mit dem Wert `border-box` für `box-sizing` das Element `F` so ändern, dass seine Breite den Innenabstand beinhaltet.

```
F {
  box-sizing: border-box;
  padding: 0 20px;
  width: 50%;
}
```

Nun ist das gesamte Element `F` einschließlich des zusätzlichen Abstands 50 Prozent breit. Die Gesamtbreite der Elemente beträgt also wieder 100 Prozent. Dadurch passen sie gut in ihren Container.

Manche Entwickler finden `box-sizing` so praktisch, dass sie es mithilfe des Universal-Selektors für jedes Element verwenden:

```
* { box-sizing: border-box; }
```

Meiner Meinung nach ist das übertrieben und könnte ungewollte Abweichungen hervorrufen. Ich wende es lieber nur an, wenn ich es tatsächlich benötige:

```
div, .etc { box-sizing: border-box; }
```

Die Verwendung von `box-sizing` hat ihre Grenzen: Zum einen werden die Ränder (*Margin*) dadurch überhaupt nicht beeinflusst. Deshalb wäre eine Möglichkeit zur dynamischen Berechnung von Längenmaßen besser.

3.3.2 Längenmaße dynamisch berechnen

calc()

Alle im vorigen Abschnitt genannten Probleme, die bei der Kombination unterschiedlicher Einheiten auftreten, lassen sich durch eine neue Wertfunktion beheben. Mit der Funktion `calc()` können Sie einfache Berechnungen mit beliebigen numerischen Längeneinheiten durchführen. In einfachsten Fall sind die an die Funktion übergebenen Argumente zwei Zahlenwerte und ein mathematischer Operator:

```
E { height: calc(10px + 10px); }
```

Für ein praxisnahes Beispiel kehren wir wieder zu dem einfachen Beispiel mit den drei Spalten zurück. Nehmen wir an, Sie möchten jetzt einen 4 px breiten Rahmen auf der linken und der rechten Seite der mittleren Spalte sowie die 10 px Innenabstand und 20 px Rand. Sie könnten wie zuvor `box-sizing` verwenden, um den Rahmen und den Innenabstand zu erzeugen, dieses Mal aber auch die `calc()`-Funktion nutzen, um die rechten und linken Ränder von der Breite abzuziehen:

```
F {
  border: 4px solid black;
  border-width: 0 4px;
  box-sizing: border-box;
  margin: 0 20px;
  padding: 0 10px;
  width: calc(50% - 40px);
}
```

Statt `box-sizing` könnten Sie die Breite außerdem mit `calc()`-Werten einstellen und die Werte mit unterschiedlichen Längeneinheiten sogar ein wenig stärker mischen. Im folgenden Beispiel verwende ich `em`-Einheiten für die Ränder und dann eine weitere `calc()`-Funktion, um die kombinierten Werte für border, padding und margin zu berechnen, bevor ich alles vom Gesamtwert abziehe:

```
F {
  border: 4px solid black;
```

```
  border-width: 0 4px;
  margin: 0 20px;
  padding: 0 1em;
  width: calc(50% - calc(48px + 2em));
}
```

Sie können `calc()` jedoch nicht nur für `width` oder `height` verwenden, sondern für jede Eigenschaft oder Funktion, die Längenwerte erlaubt.

3.4 Längenangaben relativ zum Viewport

vh und vw

Wie zuvor erwähnt, sind Prozentsätze für Breitenangaben deshalb problematisch, weil sie relativ zu den übergeordneten Elementen sind. Das mag zunächst unproblematisch erscheinen, kann jedoch schnell zu ziemlicher Komplexität führen. Ein besserer Ansatz ist die Verwendung einer Einheit, die relativ zu einem festen Wert statt zu einem vom übergeordneten Element vererbten Wert ist. Dazu dienen die Einheiten `vh` und `vw`.

`v` steht für »Viewport«, `h` für »height« und `w` für »width«. Mit anderen Worten: Diese Einheit ist relativ zu den Abmessungen des Viewports. So erzeugen Sie ein Element, das die halbe Höhe des Viewports aufweist:

```
E { height: 50vh; }
```

Und weil die Längenangaben stets relativ zum Viewport (statt zum übergeordneten Element) sind, gibt es keine langen Dezimalstellen. Nehmen wir wieder das vorherige Beispielelement mit 85 Prozent der Viewport-Breite und einem untergeordneten Element mit 55 Prozent der Viewport-Breite. Nun sind die Werte viel einfacher:

```
E { width: 85vw; }

F { width: 55vw; }
```

Zusätzlich sind die beiden Einheiten `vmax` und `vmin` verfügbar. Bei der

ersten wird die längere Seite des Viewports verwendet – egal ob es sich dabei nun um die Höhe oder die Breite handelt. Bei der zweiten wird die kürzere Seite verwendet. Bei einem Viewport mit der Auflösung 800×600 würde `vmax` also `vw` entsprechen; `vmin` würde `vh` entsprechen.

3.4.1 Einheiten relativ zum HTML-Wurzelelement (root)

root em (rem)

Einheiten, die relativ zu den Abmessungen sind, sind zwar nützlich – wenn Sie jedoch etwas mehr Kontrolle brauchen, können Sie eine Einheit verwenden, die relativ zu einem von Ihnen festgelegten Wert ist. *root em* (`rem`) ist eine typografische Einheit, die wie `em` auf der Breite eines großen *M* basiert. Die genaue Definition ist jedoch nicht wichtig – Sie müssen nur wissen, dass Sie `rem` für relative Größen verwenden können.

Der Unterschied zwischen `rem` und `em` ist, dass die zuletzt genannte Einheit ererbt wird, während die zuerst genannte feststehend ist. Nehmen wir beispielsweise an, Sie setzen eine Schriftgröße im HTML-Wurzelelement (*root*) auf 10 px:

```
html { font-size: 10px; }
```

Wenn Sie ein p-Element mit einer Schriftgröße von 12 px erzeugen wollen, können Sie beide Möglichkeiten nutzen:

```
p { font-size: 1.2em; }
p { font-size: 1.2rem; }
```

So weit, so identisch. Nehmen wir aber nun an, dass sich innerhalb dieses `p`-Elements ein `b`-Element befindet – zum Beispiel:

```
<p>This is <b>bold</b>.</p>
```

Sie möchten, dass das `b`-Element 13 px groß ist. Weil das `em` ererbt wird, müssten Sie die größere Zahl durch die kleinere teilen, um die neue Schriftgröße zu erhalten:

```
p b { font-size: 1.08333em; }
```

Die Verwendung von `em` hat die beiden bereits weiter vorne in diesem Kapitel genannten Nachteile: lange, sperrige Zahlen, mit denen es sich immer schwerer arbeiten lässt, und Rundungsunterschiede der unterschiedlichen Browser. Mit der Einheit `rem`, die immer relativ zum Root ist, gibt es diese Probleme nicht. Deshalb können Sie viel einfacher damit arbeiten:

```
p b { font-size: 1.3rem; }
```

Zwar handelt es sich bei `rem` um eine typografische Einheit, trotzdem können Sie sie – genau wie `em` – für Längenmaße aller Art nutzen.

Hinweis

Ich verwende 10 px nur zur Veranschaulichung als Root-Schriftgröße. Im wirklichen Leben sollten Sie dies vermeiden, weil Sie damit die Schrifteinstellungen des Benutzers außer Kraft setzen. Verwenden Sie besser 62,5 Prozent. Bei den meisten Nutzern entspricht dies 10 px (16 px ist die Standard-Schriftgröße, 62,5 Prozent davon sind 10 px), gibt Benutzern mit Seheinschränkungen aber dennoch die Möglichkeit, die Schrift anzupassen.

3.4.2 »Mobile First« und Breakpoints für den Inhalt

Für die Entwicklung responsiver Websites erscheinen mir zwei Systeme besonders sinnvoll: *Mobile First* und *Breakpoints für den Inhalt* (*Content Breakpoints*). Dies sind keine in Stein gemeißelten Regeln – Sie müssen sie nicht nutzen, wenn Sie den Eindruck haben, dass sie in Ihrem Projekt keinen Sinn ergeben. Bei der Entwicklung browserbasierter Websites kann ich mir jedoch nicht vorstellen, ohne sie auszukommen.

Mobile First

Der Mobile-First-Ansatz spricht zuerst die kleineren (und höchstwahrscheinlich

leistungsschwächeren) Geräte an und wird dann stufenweiser komplexer, um auch größere Geräte anzusprechen. Im Prinzip bedeutet dies: Beginnen Sie mit einem für alle Geräte geeigneten Stilgerüst– etwa für Farbe, Typografie, Symbole usw. – sowie minimalen Layoutregeln, die für ein Gerät mit kleinem Bildschirm benötigt werden.

Der nächste Schritt ist eine Media Query, die weitere Regeln für größere User Agents wie etwa Tablets hinzufügt. Für alle Nutzer mit noch größeren Geräten fügen Sie weitere Regeln hinzu usw., bis Sie die typischen Geräte abgedeckt haben. Wenn Sie davon ausgehen, dass Ihre Site hauptsächlich mit Mobiltelefon, Tablet- und Desktop-PC betrachtet werden, haben Sie zum Schluss etwa die folgenden Stilregeln:

```
<link rel="stylesheet" href="base.css" media="screen">
<link rel="stylesheet" href="tablet.css"
      media="screen and (min-width: 481px)">
<link rel="stylesheet" href="desktop.css"
      media="screen and (min-width: 801px)">
```

Welche Breakpoints Sie tatsächlich verwenden, hängt komplett von Ihnen ab bzw. von den Geräten, für die Sie Ihre Site oder Anwendung optimieren möchten. Dies sollte möglichst auf einer Analyse der Zielgruppe basieren.

Die hier verwendeten Beispiele sind äußerst einfach und dienen lediglich zur Darstellung der »Mobile First«-Methode. Luke Wroblewski ist einer ihrer führenden Verfechter. In der Einführung zu seinem Buch *Mobile First* (A Book Apart, 2011), schreibt er:

> *Wenn Sie in erster Linie für Mobilgeräte entwickeln, eröffnen Sie sich nicht nur neue Wachstumschancen – diese Vorgehensweise kann vielmehr auch zu einer insgesamt besseren User Experience für eine Website oder Anwendung führen.*

Breakpoints für Inhalte

Auch ein anderer Ansatz ist möglich: Sie setzen keine Breakpoints für Geräte, sondern für Inhalte.

Bei der großen Vielfalt der momentan (und in Zukunft) erhältlichen

webfähigen Geräte verschwimmen die Grenzen zwischen Telefon, Tablet, Laptop und Desktop völlig. Mein eigenes Mobiltelefon hat ein 4,65-Zoll-Display, das Galaxy Note von Samsung ein 5,3-Zoll-Display. Das Display des Google Nexus 7 ist 7 Zoll groß. Ab wann ist ein Gerät kein Mobiltelefon mehr, sondern ein Tablet?

Meiner Meinung nach werden Begriffe wie »Mobiltelefon« und »Tablet« (und vor allem Kunstwörter wie *Phablet*) bald überholt sein. Weil sich die Funktionalität sämtlicher internetfähiger Geräte einander annähert, müssen wir ein neues Vokabular finden, um diese Gegenstände zu beschreiben. (Dabei will ich nicht behaupten, dass ich weiß, wie dieses neue Vokabular aussehen wird.)

Mir geht es darum, dass Breakpoints auf der Grundlage von Geräteabmessungen in ein aussichtsloses Unterfangen ausarten könnten. Statt darüber nachzudenken, wie breit der Inhalt auf einem Tablet sein sollte, sollten Sie lieber überlegen: »Wie breit kann dieser Inhalt werden, bevor er unleserlich wird?«

In der Praxis bedeutet dies, dass Sie den Inhalt mit Media Queries nicht dann ändern, wenn eine willkürlich festgelegte Gerätegröße erreicht ist, sondern sobald er nicht mehr gut aussieht. Um dies zu verdeutlichen, nehmen wir an, dass ich breiten Viewports mithilfe einer Media Query ein Stylesheet zugewiesen habe:

```
<link rel="stylesheet" href="foo.css"
      media="screen and (min-width: 1000px)">
```

Innerhalb dieses Stylesheets setze ich die Breite eines `article`-Elements auf 60 Prozent der Viewport-Breite und die Schriftgröße auf 120 Prozent der Root-Schriftgröße.

```
article { width: 60vw; }
article p { font-size: 1.2rem; }
```

Zahlreiche Untersuchungen haben gezeigt, dass die maximale Anzahl der Zeichen in einer Textzeile für eine optimale Lesbarkeit 45 bis 75 Zeichen beträgt, wobei 66 als ideal betrachtet werden. Wenn der Nutzer jedoch ein Gerät mit einem sehr breiten Viewport hat, könnte eine Breite von

60 vw schnell doppelt so lange Zeilen ergeben. Wenn Sie dem Ansatz der inhaltlichen Breakpoints folgen, könnten Sie das Stylesheet für das größte Gerät mit einer zusätzlichen Regel versehen, sodass der Text auf sehr breiten Geräten vergrößert und die Lesbarkeit so gut wie möglich wiederhergestellt wird:

```
@media screen and (min-width: 1200px) {
  p { font-size: 1.4rem; }
}
```

Diese Arbeitsweise ist nicht gerade leicht – im Gegenteil. Sie werden viele Tests und Analysen durchführen und Entscheidungen aufgrund bewährter Praktiken und Ihrem eigenen Instinkt treffen müssen.

Manche Entwickler, etwa Thierry Koblentz in seinem Artikel »Device-Agnostic Approach to Responsive Web Design«, plädieren für die ausschließliche Verwendung von inhaltlichen Breakpoints und beschäftigen sich lieber überhaupt nicht mit Geräten:

Wenn wir bedenken, dass es nur auf den Inhalt ankommt, dann ist es nur logisch, diesen als den Eckpfeiler der Entwicklungslösung zu betrachten. Mit anderen Worten: Wir sollten unsere Breakpoints für den *Inhalt* statt für die Geräte setzen.

Meiner Meinung nach ist eine Kombination aus beidem am sinnvollsten. Wie immer sollten Sie experimentieren und Ihr eigenes System finden.

3.5 Responsives Design und Objektersetzung

Ein Stolperstein, den ich bisher noch nicht angesprochen habe, sind die Objekte auf Ihren Seiten – vor allem Bilder, aber auch Plugins wie Video und Flash. Mit den zuletzt Genannten können Sie nicht viel anfangen (vor allem, weil Flash auf vielen mobilen Geräten nicht unterstützt wird), aber Bilder stellen eindeutig ein Hindernis beim responsiven Design dar. In Kürze komme ich darauf zurück.

Die Skalierung der meisten Objekte mit Prozentsätzen (oder auch Viewport-relativen Einheiten) ist kein Problem; Sie könnten problemlos die Eigenschaft `max-width` auf 100 Prozent setzen und damit verhindern, dass das Objekt jemals breiter werden kann als sein Container:

```
img {
  height: auto;
  max-width: 100%;
}
```

Beachten Sie, dass ich auch den `auto`-Wert der `height`-Eigenschaften verwendet habe, um das ursprüngliche Seitenverhältnis des Objekts zu bewahren – wenn Sie eine feste Höhe und eine dynamische Breite (oder umgekehrt) angeben, wird das Objekt möglicherweise verzerrt.

Wenn man die Dimensionen des Objekts beibehält, erhält man manchmal horizontale oder vertikale Leerräume, weil das Objekt zu klein für seinen Container ist (ein Beispiel dafür sehen Sie in Abb. 3–5). In diesem Fall können Sie mit den Eigenschaften `object-fit` und `object-position` besser kontrollieren, wie das Objekt in seinem übergeordneten Element dargestellt wird.

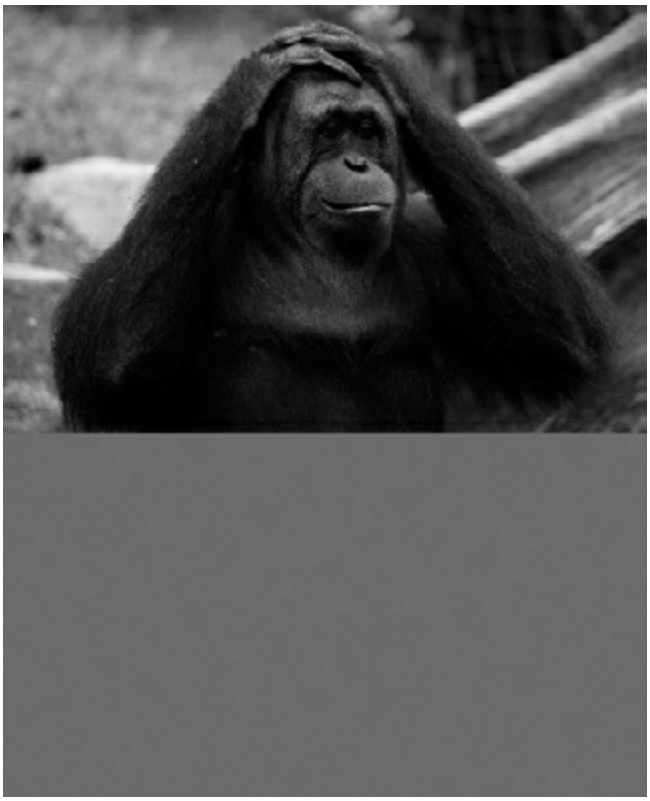


Abb. 3–5
Dieses Bild wurde so skaliert, dass seine Proportionen erhalten wurden. Dadurch entstand ein Leerraum um das Bild.[1]

object-fit

Die Eigenschaft `object-fit` definiert, wie das Objekt innerhalb seines Containers skaliert wird. Mit dem Schlüsselwortwert `contain` wird das Objekt so skaliert, dass es im Ganzen innerhalb des Containers angezeigt wird. Das Seitenverhältnis bleibt erhalten, und es entsteht – falls nötig – horizontal oder vertikal ein Leerraum. Alternativ verwenden Sie das Schlüsselwort `cover`. Damit wird die kürzeste Seite des Objekts der kürzesten Seite des Containers gleichgesetzt, sodass die längste Seite über den Container hinausragt. Mit dem Schlüsselwort `fill` passen Sie das Objekt exakt in den Container ein.

In der Beispieldatei *object-fit.html* sehen Sie einen Vergleich dieser drei Schlüsselwörter. Das Beispiel enthält drei `div`-Elemente mit je einem `img`. Jedes davon besitzt einen eindeutigen `id`-Wert. Diesem

Markup füge ich den folgenden Code hinzu:

```
img {
  height: 100%;
  width: 100%;
}
#obj-fill { object-fit: fill; }
#obj-contain { object-fit: contain; }
#obj-cover { object-fit: cover; }
```

Abb. 3–6 zeigt das Ergebnis: Das erste Element, *#obj-fill*, wird auf dieselben Abmessungen skaliert wie sein übergeordnetes Element; es wird also horizontal gequetscht und vertikal gestreckt. Diese Verzerrung sieht natürlich nicht besonders gut aus. Das Element in der Mitte, *#obj-contain*, behält sein ursprüngliches Seitenverhältnis, wird aber so skaliert, dass das gesamte `img` in seinen Container passt. Dadurch ergibt sich oben und unten ein Leerraum. Mit *#obj-cover* schließlich bleibt das Seitenverhältnis ebenfalls erhalten; das Objekt wird aber so skaliert, dass es den gesamten Container ausfüllt. Dazu wird das Bild in diesem Fall größer skaliert und reicht über das übergeordnete Element hinaus (die überstehenden Teile habe ich in diesem Beispiel ausgeblendet).

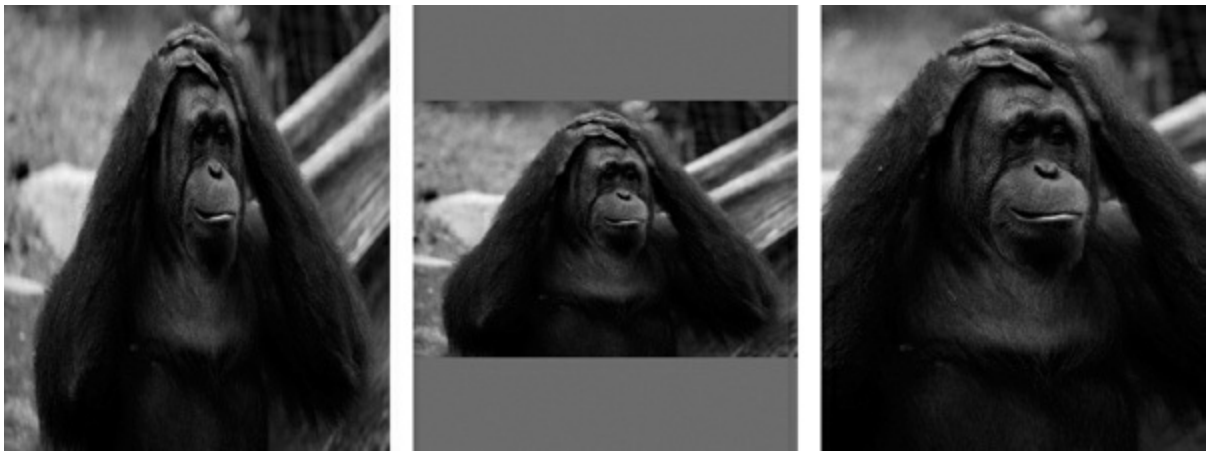


Abb. 3–6
Unterschiedliche Werte für die Eigenschaft object-fit (von links nach rechts): fill, contain, cover

Wenn Sie `object-fit` zuweisen, sitzt das

object-position

Objekt standardmäßig genau in der Mitte seines Containers; Sie können dies aber mit der Eigenschaft `object-position` ändern. Diese Eigenschaft funktioniert gewissermaßen wie die Eigenschaft `background-position`, weil Sie entweder mit zwei Werten einen Versatz von der linken oberen Containerecke aus festlegen oder zusätzliche Schlüsselwörter verwenden können (`top`, `right`, `bottom`, `left`, `center`), um den Versatz von einer anderen Seite aus zu definieren. Ein Beispiel:

```
E { object-position: bottom 10px right 2em; }
```

Die Beispieldatei *object-position.html* zeigt drei verschiedene Werte für die Eigenschaft `object-position`. Das Markup entspricht grundsätzlich dem vorigen Beispiel; der CSS-Code wurde jedoch verändert:

```
img {
  height: 100%;
  width: 100%;
  object-fit: contain;
}
#obj-1 { object-position: center top; }
#obj-2 { object-position: center bottom; }
#obj-3 { object-position: right; }
```

Das Ergebnis sehen Sie in Abb. 3–7. Allen drei `img`-Elementen wurde der Wert `contain` der Eigenschaft `object-fit` zugewiesen – dies ist jedoch nicht erforderlich, sondern dient nur zur Verdeutlichung. Das erste Bild, *#obj-1*, ist horizontal zentriert in seinem übergeordneten Element und an der vertikalen Oberkante ausgerichtet, das nächste, *#obj-2*, ist ebenfalls horizontal zentriert, aber nun an der vertikalen Unterkante ausgerichtet; das letzte Bild, *#obj-3*, wurde auf das Querformat zugeschnitten, damit ich besser demonstrieren kann, wie es rechts an seinem übergeordneten Element ausgerichtet ist.

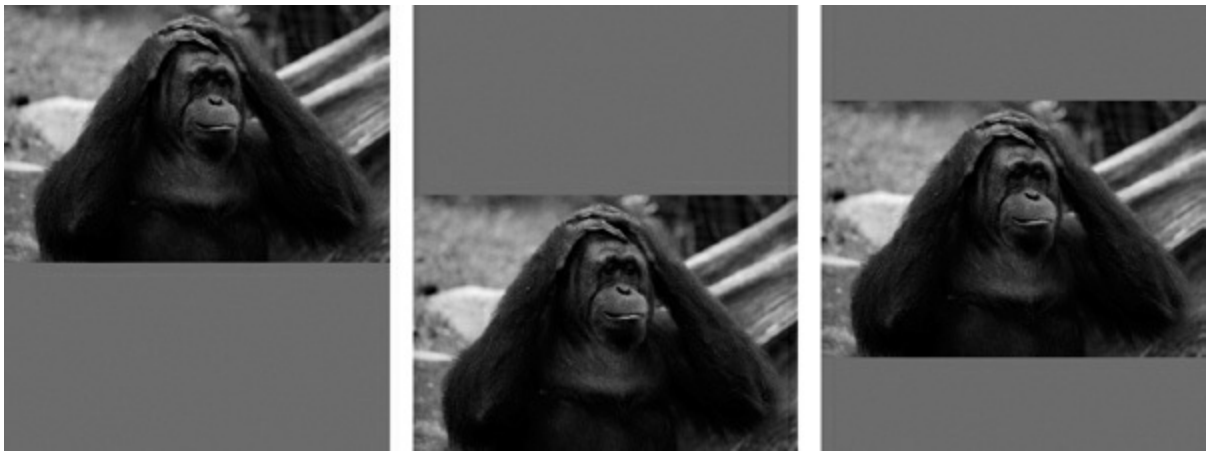


Abb. 3–7
Unterschiedliche Werte für die Eigenschaft object-position

3.5.1 Das Bildproblem

Sie haben nun gesehen, dass sich Objekte mit responsiven Techniken recht einfach skalieren und positionieren lassen. Trotzdem sind Bilder im responsiven Design aus verschiedenen Gründen eine größere Problemquelle. Einer der wichtigsten ist die Frage der Dateigröße. Oft nutzen Anwender mit Mobilgeräten lediglich 3G oder 4G und verfügen deshalb über eine im Vergleich zum Nutzer eines Desktop-PCs reduzierte Bandbreite. Solche Nutzer sollten nicht mit einem großen Bild konfrontiert werden, das sie mit ihrer eingeschränkten Verbindung herunterladen müssten.

Durch die immer höher auflösenden Displays wird dies noch verschärft. Standardmäßig aufgelöste (1DPR-)Bitmap-Bilder (wie JPG und PNG) können auf höher aufgelösten (2DPR-)Displays recht niedrig aufgelöst wirken. HTML bietet momentan keine Möglichkeit, für bestimmte Geräte höher aufgelöste Bilder zur Verfügung zu stellen – und selbst wenn dies so wäre, gäbe es trotzdem noch das Bandbreitenproblem. Was ist die Lösung?

3.5.2 Responsive HTML5-Bilder als Lösung

Während ich dies schreibe, gibt es leider keine native Lösung. Zwar wurden mehrere Vorschläge unterbreitet; aber keiner davon wurde bisher offiziell abgesegnet.

picture

Der populärste Vorschlag ist momentan, ein neues `picture`-Element zu verwenden, etwa so:

```
❶ <picture alt="Description of image subject.">
❷   <source srcset="small.jpg 1x, small-highres.jpg 2x">
❸   <img src="small.jpg" alt="Description of image subject.">
  </picture>
```

Sie erkennen hier die folgenden drei Komponenten: ❶ ist das neue `picture`-Element mit dem `alt`-Attribut, das das Bild beschreibt (hier könnten auch andere Standardattribute verwendet werden); ❷ ist das `source`-Element, das unterschiedliche Quellenalternativen mit dem `srcset`-Attribut auflistet – hier sehen Sie die beiden alternativen Bildquellen, eine für Standardbildschirme und eine für höher aufgelöste Bildschirme (unter Verwendung der früher in diesem Kapitel erläuterten DPR); ❸ ist schließlich das `img`-Element als Fallback für ältere Browser, die `picture` noch nicht implementiert haben.

Achtung

Denken Sie daran, dass dies nur der bekannteste Vorschlag ist; er stellt möglicherweise nicht die endgültige Syntax dar.

Diese Syntax ermöglicht es also, die Auflösung des Geräts zu prüfen; aber was ist mit anderen Media Queries? Sie können diese mit weiteren `source`-Elementen hinzufügen:

```
<picture alt="Description of image subject.">
  <source srcset="small.jpg 1x, small-hi-res.jpg 2x">
  <source media="(min-width: 481px)"
          srcset="med.jpg 1x, med-hi-res.jpg 2x">
  <img src="small.jpg" alt="Description of image subject.">
</picture>
```

In diesem Beispiel erkennen Sie ein zusätzliches `source`-Element mit dem `media`-Attribut. Dadurch wird eine Media Query erzeugt und ein anderer Satz Bilder zur Verfügung gestellt, wenn die Abfrage true ist. Ich muss sagen, dass ich diese Syntax nicht besonders mag, auch wenn sie funktioniert – zunächst einmal ist sie repetitiv, und wenn Sie sie auf einer Seite mit mehreren Bildern verwenden, wird die Wartung der Seite erschwert. Trotzdem: Es wurde noch kein einfacherer Vorschlag eingereicht.

Der aktuelle Vorschlag des WHATWG besteht ebenfalls darin, das Attribut `srcset` zu verwenden, aber nur für das `img`-Element. Mit dieser Option können Sie Bilder abhängig von der Bildschirmauflösung zur Verfügung stellen, aber ohne eine andere Media Query. Ich möchte nachdrücklich betonen, dass dies nicht »offiziell«, sondern nur ein Vorschlag ist.

Von dritter Seite wurden viele Lösungen erarbeitet. Hier sollen vor allem Matt Wilcox' adaptive Bilder genannt werden. Dabei werden Bilder mit PHP und Apache auf dem Server skaliert und den Nutzern je nach den Eigenschaften ihrer Geräte zur Verfügung gestellt. Diese Möglichkeit funktioniert zwar gut, sie erfordert jedoch eine spezifische Serverkonfiguration, die nicht für jeden verfügbar ist, und außerdem JavaScript. Dieses Problem muss nativ gelöst werden.

3.6 Zusammenfassung

In diesem Kapitel habe ich zahlreiche Themen besprochen: Ich habe Media Queries erläutert, die Geräteauflösung, die Unterschiede zwischen physischen und virtuellen Pixeln, responsives Webdesign und das Problem der responsiven Bilder. Wenn Sie geräteübergreifend entwickeln, sollten diese Informationen zumindest nützlich, wenn nicht sogar essenziell sein.

Dabei habe ich eine detaillierte Erörterung des tatsächlichen Seitenlayouts, der bestmöglichen Präsentation der Inhalte,

ausgeklammert. Hier hat sich in den vergangenen Jahren viel getan, und im nächsten Kapitel gehe ich darauf ein.

3.7 Literaturempfehlungen

Die erste Anlaufstelle, wenn Sie mehr über Media Queries erfahren möchten, sollte Zoe Mickley Gillenwaters Post *Essential Considerations for Crafting Quality Media Queries* sein: http://zomigi.com/blog/essential-considerations-for-crafting-quality-media-queries/.

Die maßgebliche Autorität für Mobilgeräte ist Peter-Paul Koch. Wenn Sie mehr über virtuelle und physische Pixel erfahren möchten, empfehle ich Ihnen für den Einstieg seinen Artikel *A Pixel Is Not a Pixel Is Not a Pixel*: http://www.quirksmode.org/blog/archives/2010/04/a_pixel_is_not.ht Wikipedia bietet eine *Auflistung der gängigen Geräteauflösungen und Pixeldichten*: http://en.wikipedia.org/wiki/List_of_displays_by_pixel_density.

Patrick Lauke schrieb einen Artikel über benutzerkontrollierte DPR: *devicePixelRatio in Opera Mobile*. Sie finden ihn auf http://my.opera.com/ODIN/blog/2012/07/05/devicepixelratio-in-opera-mobile. Matt Wilcox's Artikel *The Responsive Design Process* enthält ein gutes Glossar mit Schlüsselbegriffen sowie zahlreiche praktische Design-Hinweise: http://mattwilcox.net/archive/entry/id/1078/.

Mehr darüber, wie unterschiedliche Browser Dezimalstellen runden, erfahren Sie in John Albin Wilkins' Beitrag *Responsive Design's Dirty Little Secret*: http://www.palantir.net/blog/responsive-design-s-dirty-little-secret/.

Paul Irishs Blogbeitrag *box-sizing: border-box FTW* legt seine Gründe dar, warum diese Eigenschaft global gesetzt werden sollte: http://paulirish.com/2012/box-sizing-border-box-ftw/.

Luke Wroblewskis Buch *Mobile First* ist bei A Book Apart erschienen: http://www.abookapart.com/products/mobile-first/.

Mehr über Inhalts-Breakpoints erfahren Sie in dem Artikel des australischen Webdesign-Studios Jordesign (http://www.jordesign.com/blog/responsive-breakpoints-from-the-content-out/) und dem *Artikel des Entwicklers Thierry Koblentz* (http://coding.smashing-magazine.com/2012/03/22/device-agnostic-approach-to-responsive-web-design/).

Die *Geschichte des momentan wichtigsten Vorschlags für responsive Bilder und Informationen zu seinem aktuellen Status* finden Sie auf der Website der Responsive Images Community Group: http://www.w3.org/community/respimg/.

Matt Wilcox' *Tool für adaptive Bilder* finden Sie auf http://adaptive-images.com/.

Bildnachweis: *Evolution of Expression!* von kabils: http://www.fotopedia.com/items/flickr-2680214376/, Creative-Commons-Lizenz.

4 Neue CSS-Konzepte

Sobald Ihr Markup fertig ist und Sie überlegt haben, wie Sie Ihre Websites oder -anwendungen am besten für die in Frage kommenden Geräte optimieren, können Sie mit der Gestaltung des eigentlichen Layouts beginnen.

In Zeiten von CSS2.1 waren die Weblayouts recht beschränkt und formelhaft. Auch wenn clevere Designer und Entwickler Floats und Positionierungen bestmöglich zur Gestaltung des Layouts nutzten, beschränkten sich die meisten Website-Layouts auf eine Variante des dreispaltigen Rasters. Dieses Muster wird eher von den Einschränkungen der verfügbaren Technologie diktiert statt von den inhaltlichen Ansprüchen, die die Inhalte dieser Seiten stellen.

Erstmals mit CSS3 gibt es neue Eigenschaften speziell für die Erzeugung variabler und flexibler Layouts. Diese basieren auf den seit Jahrhunderten für die Gestaltung geschriebenen und gedruckten Materials gesammelten Kenntnissen, sind gleichzeitig aber auf die Möglichkeiten elektronischer Anzeigegeräte zugeschnitten. Mit diesen neuen Features lassen sich Inhalte bestmöglich präsentieren, und die Benutzeroberfläche der Site oder Anwendung kann besser auf die Anzeigegeräte reagieren.

Achtung

Obwohl viele der neuen Funktionen in diesem Kapitel bereits in den Browsern implementiert worden sind, sollten sie als experimentell und vorläufig betrachtet werden, bis das W3C den Standardisierungsprozess abgeschlossen hat. Detaillierte Informationen über die aktuelle Browserunterstützung finden Sie in Anhang A und den entsprechenden Online-Updates.

4.1 Mehrspalten-Layout

Die Wurzeln von HTML liegen in der Auszeichnung wissenschaftlicher Dokumente. Deshalb wurden Text auf Websites auf relativ einfache, formelhafte Weise dargestellt: Alles basiert auf einer einzigen, ununterbrochenen Textspalte – wie bei einem Dokument in einem Textverarbeitungsprogramm oder einem Text-Editor. Der Grund dafür ist größtenteils die dynamische Natur des Web mit variablen Schriftgrößen und variabler Zeichenzahl, die eine exakte Positionierung erschweren. Printmedien mit ihren festen Buchstabengrößen und festgelegter Zeichenzahl ermöglichen eine viel flexiblere Textgestaltung.

Nehmen Sie einfach ein beliebiges gedrucktes Magazin oder eine gedruckte Zeitung (jüngere Leser: Fragt eure Eltern, was das ist!) und Sie werden auf jeden Fall Beispiele für mehrspaltigen Text finden – häufig in zwei Spalten, manchmal auch in drei oder mehr. Durch diese Präsentationsform lässt sich bei unverändert guter Lesbarkeit mehr Text auf einer Seite unterbringen.

CSS-Modul: Multi-column Layout

Bis vor Kurzem war es nicht möglich, solchen Spaltensatz ohne JavaScript im Web zu realisieren. Diese Einschränkung lässt sich mit dem Modul *Multi-column Layout* überwinden. Ehrlich gesagt: Meiner Meinung nach ist Spaltensatz für Printmedien sehr viel besser geeignet als für Bildschirmmedien, denn es kann lästig sein, wenn man auf- und ab-scrollen muss, um die Textspalten zu lesen. In manchen Fällen ist der Spaltensatz jedoch definitiv die bessere Lösung.

column-count

Mit den im Multi-column-Layout-Modul zur Verfügung stehenden Eigenschaften können Sie problemlos Inhalte in Spalten anordnen. Weisen Sie dem Elternelement der gewünschten Elemente die Eigenschaft `column-count` mit einem ganzzahligen Wert für die Anzahl der Spalten zu. Mit dem folgenden Code würden Sie den Inhalt beispielsweise in drei Spalten umbrechen:

```
E { column-count: 3; }
```

column-width

Der Inline-Inhalt befindet sich nun in drei Spalten, die jeweils durch einen Abstand von 1 em voneinander getrennt sind. Wenn es Ihnen lieber ist, die Spaltenbreite genau vorzugeben, könnten Sie stattdessen mit der Eigenschaft `column-width` und einer Längenangabe die Spaltenbreite einstellen:

```
E { column-width: 120px; }
```

Dieser Code erzeugt so viele Spalten mit 120 px Breite – zuzüglich des 1-em-Abstands zwischen jeder Spalte –, dass sie genau in das Elternelement passen. Gehen wir beispielsweise davon aus, dass der Container eine Breite von 600 px hat. Dann entstehen vier Spalten mit 120 px Breite und einem Gesamtabstand von 3 em sowie einem verbleibenden Leerraum. Dieser wird gleichmäßig den einzelnen Spalten zugeschlagen, sodass diese breiter werden als 120 px. Auf diese Weise nehmen die Spalten die volle Breite des Elternelements ein – der bei `column-width` angegebene Wert ist also kein absoluter, sondern ein Minimalwert.

In Abb. 4–1 vergleiche ich die beiden Ansätze miteinander. Im oberen Beispiel habe ich mit `column-count` drei identische Spalten erzeugt. Sie füllen das Elternelement in der Breite vollständig aus. Das untere Beispiel hat eine `column-width` von 120 px. Die tatsächliche Breite der Spalten weicht jedoch etwas ab, weil diese so skaliert werden, dass sie besser in den Container passen.

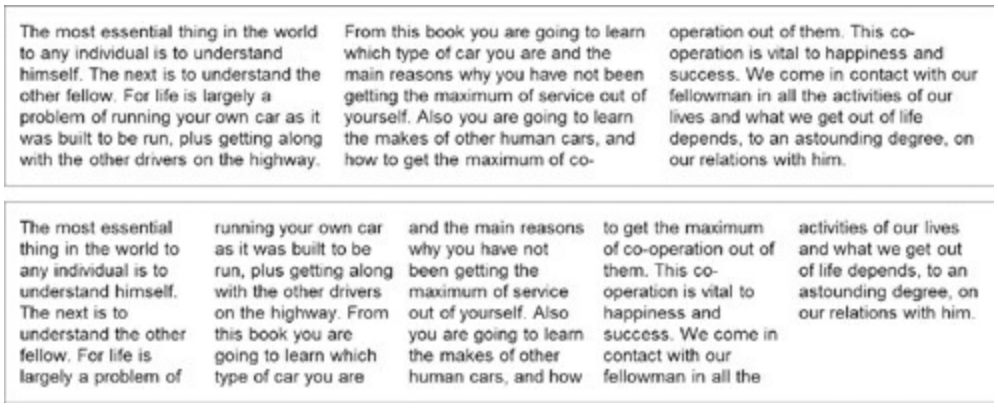


Abb. 4–1
Die oberen Spalten wurden mit der Eigenschaft column-count erzeugt, die unteren mit der Eigenschaft column-width.

Sie können auch `column-width` und `column-count` auf dasselbe Element anwenden. Dann wird der Wert `column-count` zur maximalen Spaltenanzahl. Die Logik lautet in diesem Fall: »Erzeuge höchstens drei 120 px breite Spalten; skaliere sie bei Bedarf, bis das Elternelement damit ausgefüllt ist.« Dazu könnten Sie auch die Abkürzung `columns` mit den beiden Werten `column-width` bzw. `column-count` verwenden:

```
E { columns: 120px 3; }
```

column-fill

Wenn die Inhalte in einem Container ohne feste Höhe in Spalten umbro-chen werden, verteilt der Browser die Anzahl der Zeilen bestmöglich in jeder Spalte, wobei am unteren Rand des Elternelements möglicherweise ein Leerraum verbleibt. Hat der Container jedoch eine feste Höhe, können Sie entscheiden, wie der Text umbrochen wird. Dazu verwenden Sie die Eigenschaft `column-fill`.

Der Standardwert für `column-fill` ist `balance`, womit die Zeilen gleichmäßig verteilt werden. Als Alternative füllen Sie die Spalten nacheinander bis zu ihrer maximalen Höhe. Dadurch bleibt höchstwahrscheinlich eine letzte Spalte mit sehr viel weniger Zeilen und dadurch viel ungenutztem Platz übrig. Wenn Sie dies vorziehen, verwenden Sie den Schlüsselwortwert `auto`.

4.1.1 Spaltenabstände und -linien

column-gap

Wie bereits erwähnt, beträgt der Abstand zwischen den einzelnen Spalten standardmäßig 1 em. Sie können diesen Abstand mit der Eigenschaft `column-gap` verändern. Diese akzeptiert einen einfachen Längenwert und erhöht den Abstand zwischen den

einzelnen Zeilen auf das angegebene Maß. Hier ein Beispiel:

```
E { column-gap: 2em; }
```

column-rule

Außerdem können Sie zwischen den Spalten Linien hinzufügen. Verwenden Sie dazu die Eigenschaft `column-rule`. Das Ergebnis ist grundsätzlich dasselbe wie bei Verwendung der CSS2.1-Eigenschaft `border`. `column-rule` erfordert auch dieselben drei Werte: einen für die Breite, einen für den Stil und einen für die Farbe; es entsteht aber nur eine einzige vertikale Linie, während `border` Linien an allen vier Seiten erzeugt. Möchten Sie also zwischen den Spalten eine gepunktete graue, 2 Pixel breite Linie hinzufügen, verwenden Sie den folgenden Code:

```
E { column-rule: 2px dotted gray; }
```

Mit `column-gap` wird die Abstandsbreite gleichmäßig auf beiden Seiten der Linie verteilt. In dem in Abb. 4–2 gezeigten Beispiel erhalten Sie einen Abstand von 2 em: 1 em Abstand auf jeder Seite der 2 px breiten Linie.

The most essential thing in the world to any individual is to understand himself. The next is to understand the other fellow. For life is largely a problem of running your own car as it was built to be run, plus getting along with the other drivers on the highway. From this book you are going to learn which type of car you are and the main reasons why you have not been getting the maximum of service out of yourself. Also you are going to learn the makes of other human cars, and how to get the maximum of co-operation out of them. This co-operation is vital to happiness and success. We come in contact with our fellowman in all the activities of our lives and what we get out of life depends, to an astounding degree, on our relations with him.

Abb. 4–2
Den Abstand und die Linie zwischen den Spalten kontrollieren

4.1.2 Spaltenspanne und Umbrüche

Spalten sind gut geeignet, wenn Sie den Fluss von Text und anderen Inline-Inhalten steuern möchten. Andere Objekte oder Blockelemente

eignen sich weniger gut für den Spaltensatz. Hierfür gibt es einige verwandte Eigenschaften.

column-span

Die erste ist `column-span`. Sie ist gut geeignet, wenn Sie den Fluss der Spalten mit einem neuen Element unterbrechen möchten, das sich über alle Spalten erstreckt. `column-span` ist eine boolesche Eigenschaft – ein Element erstreckt sich entweder über alle Spalten oder über keine. Deshalb sind die möglichen Werte entweder `all` oder `none`:

```
F { column-span: all; }
```

Mit dem Wert `all` wird der Fluss des Inline-Textes vor dem Element mit der Eigenschaft column-span unterbrochen und danach fortgesetzt (siehe Abb. 4–3).

break-after, break-before und break-inside

Bei im Raster angeordneten Elementen kann es passieren, dass das Ende einer Spalte einen Umbruch in der Mitte des Elements erzeugt. Wenn Sie beispielsweise eine Unterüberschrift haben, die über zwei Spalten umbrochen werden könnte, ist dies nicht ideal. Um das zu vermeiden, können Sie die Umbrüche mit den Eigenschaften `break-after`, `break-before` und `break-inside` kontrollieren. Sie funktionieren alle mehr oder weniger auf dieselbe Weise:

From this book you are going to learn which type of car you are and the main reasons why you have not been getting the maximum of service out of yourself. Also you are going to learn the makes of other human cars, and how to get the maximum of co-operation out of them. This co-operation is vital to happiness and success. We come in contact with our fellowman in all the activities of our lives and what we get out of life depends, to an astounding degree, on our relations with him.

Reaction to Environment

The greatest problem facing any organism is successful reaction to its environment. Environment, speaking scientifically, is the sum total of your experiences. In plain United States, this means fitting vocationally, socially and maritally into the place where you are. If you don't fit you must move or change your environment to fit you. If you can't change the environment and you won't move you will become a failure, just as tropical plants fail when transplanted to the Nevada desert.

Learn From the Sagebrush

But there is something that grows and keeps on growing in the Nevada desert—the sagebrush. It couldn't move away and it couldn't change its waterless environment, so it did what you and I must do if we expect to succeed. It adapted itself to its environment, and there it stands, each little stalwart shrub a reminder of what even a plant can do when it tries!

Abb. 4–3
Die zweite Unterüberschrift umspannt alle Spalten, wodurch der Fluss des Inhalts unterbrochen wird.

Ich demonstriere die Funktionsweise anhand von `break-before`. Die möglichen Werte für diese und die anderen beiden Eigenschaften definieren, ob ein Spaltenumbruch vor einem Element auftritt. Der Wert `column` erzwingt bei Bedarf einen Umbruch vor dem Element. Der Wert `avoid-column` (oder einfach `avoid` – ein Indiz, dass diese Eigenschaft auch für andere CSS-Features verfügbar ist) untersagt dem Browser einen Umbruch vor dem Element, es sei denn, dies ist für den Seitenfluss absolut notwendig. Der Standardwert `auto` lässt den Browser entscheiden, wie die Spalten am besten dargestellt werden.

Wenn dies immer noch ein wenig undurchsichtig klingt, betrachten Sie die Beispieldatei *column-breaks.html*, die den Einsatz des Werts `column` demonstriert. Hier der Code für dieses Beispiel:

```
h2 { break-before: column; }
```

Die Ergebnisse sehen Sie in Abb. 4–4. Im oberen Beispiel bedeutet der Standardwert `auto`, dass die Unterüberschriften im Textfluss erscheinen. Der Wert `column` weist den Browser im zweiten Beispiel an, vor jeder h2-oder Unterüberschrift einen Spaltenumbruch einzufügen.

Abb. 4–4
Die Auswirkungen des Werts column bei der Eigenschaft break-before zeigen sich deutlich im unteren Beispiel.

Die Eigenschaften `break-after` und `break-inside` funktionieren auf dieselbe Weise. Der Unterschied ist die Stelle, an der der Umbruch eingefügt wird. (Muss ich Ihnen sagen, wo das ist, oder möchten Sie lieber selbst darauf kommen?)

4.2 Flexbox

CSS-Modul: Flexible Box Layout

Wenn Sie nicht nur eine inhaltsreiche Website entwickeln wollen, sondern eine richtige App, deren Benutzeroberfläche viele Buttons, Formularelemente und interaktive Bereiche enthält, dann ist das CSS-Layoutmodul *Flexible Box Layout*, kurz: *Flexbox*, äußerst nützlich.

Die Wurzeln von Flexbox liegen in der Sprache XUL, mit der das Layout des Firefox-Browsers entwickelt wurde. Schon daran erkennen Sie, dass Flexbox für Benutzeroberflächen gedacht ist. Flexbox sorgt für

eine flexible Skalierung von Elementen, sodass diese besser in den vorhandenen Raum auf dem Bildschirm passen und neu angeordnet oder von der Ausrichtung her geändert werden können.

Die Flexbox-Syntax hat in den vergangenen Jahren eine Menge Änderungen erfahren; sie ist nun jedoch ziemlich stabil und gut implementiert, sodass wir sie ohne Sorge besprechen können!

4.2.1 Das Flexbox-Modell deklarieren

Der Flex-Container

Der erste Schritt bei der Verwendung von Flexbox besteht darin, den *Flex-Container* zu erzeugen. Dieser ist das Elternelement, das alle *flex*-Elemente (*flex items*) enthält (ich komme gleich darauf zu sprechen). Um einen Flex-Container zu deklarieren, verwenden Sie einfach einen neuen Wert für die `display`-Eigenschaft:

```
E { display: flex; }
```

Kindelemente

Nun haben Sie Ihren Flex-Container – aber was können Sie damit anfangen? Das wird deutlich, wenn Sie dem Container Kindelemente hinzufügen. In der Beispieldatei *flexbox.html* gibt es zwei Kindelemente:

```
<div id="container">
  <div id="a">...</div>
  <div id="b">...</div>
</div>
```

Beachten Sie, dass die beiden Elemente in Abb. 4–5 mit identischen Breiten horizontal im Container positioniert sind, obwohl ich keinerlei Floats oder Positionierungseigenschaften angegeben habe. Dieses Verhalten ist der Standard für Flex-Elemente (Kindelemente eines Flex-Containers): Sie sind innerhalb des Containers in einer Zeile angeordnet. Anders ausgedrückt: Sie sind in der Richtung der Dokumentsprache angeordnet – in Sprachen von links nach rechts (wie etwa Deutsch) also

in einer Zeile von links nach rechts:

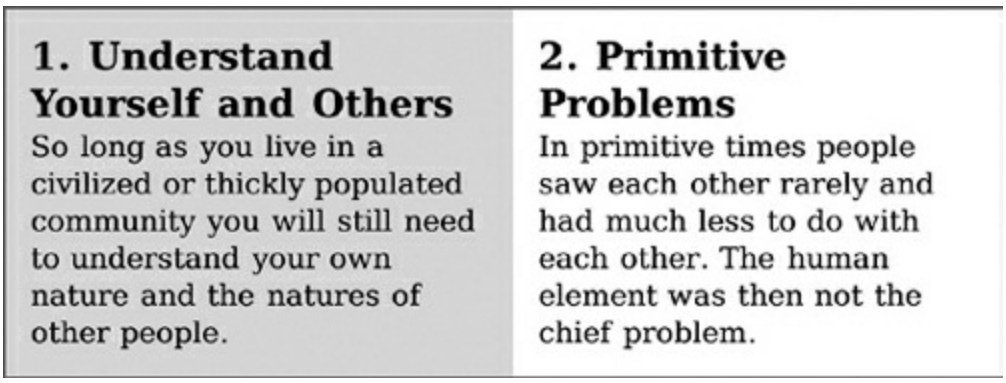


Abb. 4–5
Die Kindelemente eines Flex-Containers werden automatisch horizontal angeordnet.

flex-direction

Sie können diese Richtung ändern, indem Sie die Eigenschaft `flex-direction` für den Container verwenden. Ihr Standardwert ist `row`, der das soeben erläuterte Verhalten ergibt, während ein Wert von `column` die Elemente im rechten Winkel zur Textrichtung anordnet – in diesem Fall in einer Spalte (es gibt noch ein paar weitere Werte, auf die ich in Kürze eingehen werde):

```
E {
  display: flex;
  flex-direction: column;
}
```

Diese Syntax sorgt dafür, dass Ihre Flex-Elemente dem standardmäßigen Webseiten-Layout entsprechen – die Elemente folgen in vertikaler Ausrichtung aufeinander.

4.2.2 Die Reihenfolge der Inhalte ändern

Eine weitere großartige Möglichkeit von Flexbox ist die Möglichkeit, die Reihenfolge der Elemente ungeachtet ihrer Reihenfolge im DOM

schnell zu ändern. Beispielsweise hatte ich im vorigen Abschnitt zwei Elemente in einer Zeile angeordnet – was aber, wenn ich *#b* vor *#a* anzeigen möchte?

Reihenfolge umkehren

Verwenden Sie dazu einfach die soeben erläuterte Eigenschaft `flex-direction` mit dem neuen Wert `row-reverse`. Diese Eigenschaft kehrt die Reihenfolge um, in der die Flex-Elemente angezeigt werden (siehe Abb. 4–6).

```
E { flex-direction: row-reverse; }
```

Ähnlich der Wert `column-reverse`: Damit werden in Spalten angeordnete Elemente in umgekehrter Reihenfolge vertikal angeordnet.

Experimentieren Sie mit den Werten von `flex-direction` in der Beispieldatei, um die unterschiedlichen Auswirkungen zu prüfen.

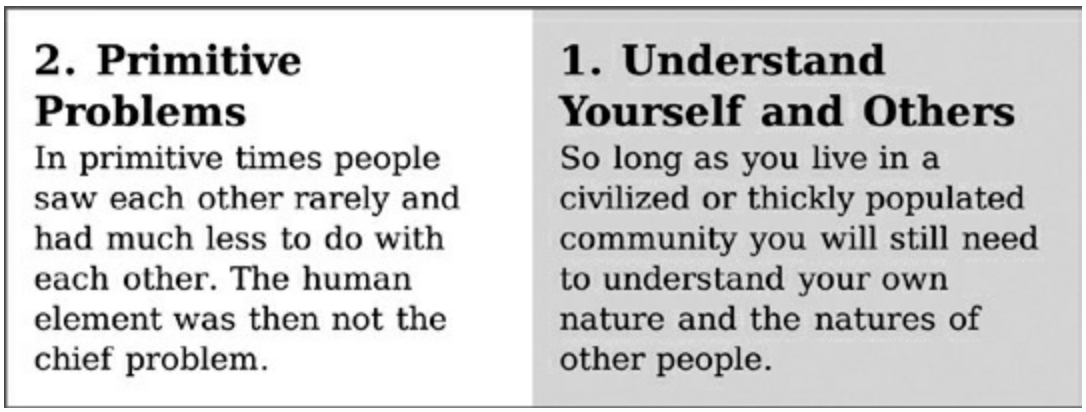


Abb. 4–6
*Der Wert row-*reverse *kehrt die Reihenfolge der Flex-Kindelemente schnell um.*

Reihenfolge festlegen

Mit der Eigenschaft `flex-order` können Sie nicht nur die Reihenfolge umkehren, sondern auch komplett maßgeschneiderte Abfolgemuster festlegen. Diese Eigenschaft wird nicht dem Container, sondern den darin enthaltenen Flex-Elementen zugewiesen. Der Wert ist eine Zahl, die *Ordinalgruppe* (*ordinal group*) – Elemente mit demselben

Zahlenwert werden gruppiert. Sämtliche Elemente in der Gruppe mit der niedrigsten Nummer kommen zuerst, gefolgt von allen Elementen in der Gruppe mit der zweitniedrigsten Zahl usw. Elemente ohne einen definierten Wert werden zuerst angezeigt – sie haben den Standardwert 0.

Elemente mit der gleichen Ordinalgruppe werden nacheinander in der Reihenfolge angezeigt, in der sie im DOM erscheinen. Dieses Prinzip erscheint möglicherweise zunächst verwirrend. Deshalb verdeutliche ich es anhand der folgenden vier Flex-Elemente:

```
<div id="container">
  <div id="a">...</div>
  <div id="b">...</div>
  <div id="c">...</div>
  <div id="d">...</div>
</div>
```

Wenn keine expliziten Werte gesetzt sind, werden die Kindelemente in der Reihenfolge angezeigt, in der sie im DOM erscheinen: *#a*, *#b*, *#c*, *#d*. Nun wollen wir sie aber mit unterschiedlichen Werten für die Eigenschaft `flex-order` neu anordnen:

```
#a { flex-order: 2; }
#b, #d { flex-order: 3; }
#c { flex-order: 1; }
```

Jetzt werden die Elemente in der folgenden Reihenfolge angeordnet: *#c*, *#a*, *#b*, *#d*. Element *#c* kommt zuerst, weil seine Ordinalgruppe den niedrigsten Zahlenwert hat, gefolgt von *#a* mit der nächsthöchsten Zahl und dann *#b* und *#d* – beide befinden sich in der Ordinalgruppe 3 –, und zum Schluss folgt *#d*, weil es auch weiter hinten in der DOM-Reihenfolge erscheint.

4.2.3 Anordnung innerhalb des Containers

Flex-Elemente mit expliziten Abmessungen können kleiner sein als ihr Container. Ein weiterer Vorteil von Flexbox ist die Möglichkeit der

exakten Kontrolle über Ausrichtung und Platzierung.

Haupt- und Querachse

Bevor wir uns damit beschäftigen, erkläre ich rasch die beiden unterschiedlichen Achsen für die Ausrichtung. Die *Hauptachse* (*main axis*) geht in die Richtung, in der die Elemente platziert sind: Hat `flex-direction` den Wert `row`, ist die Hauptachse horizontal; lautet der Wert `column`, ist die Hauptachse vertikal. Die *Querachse* (*cross axis*) steht im rechten Winkel dazu: Hat `flex-direction` den Wert `row`, ist die Querachse vertikal, lautet der Wert `column`, ist die Querachse horizontal. Den Unterschied verdeutlicht Abb. 4–7.

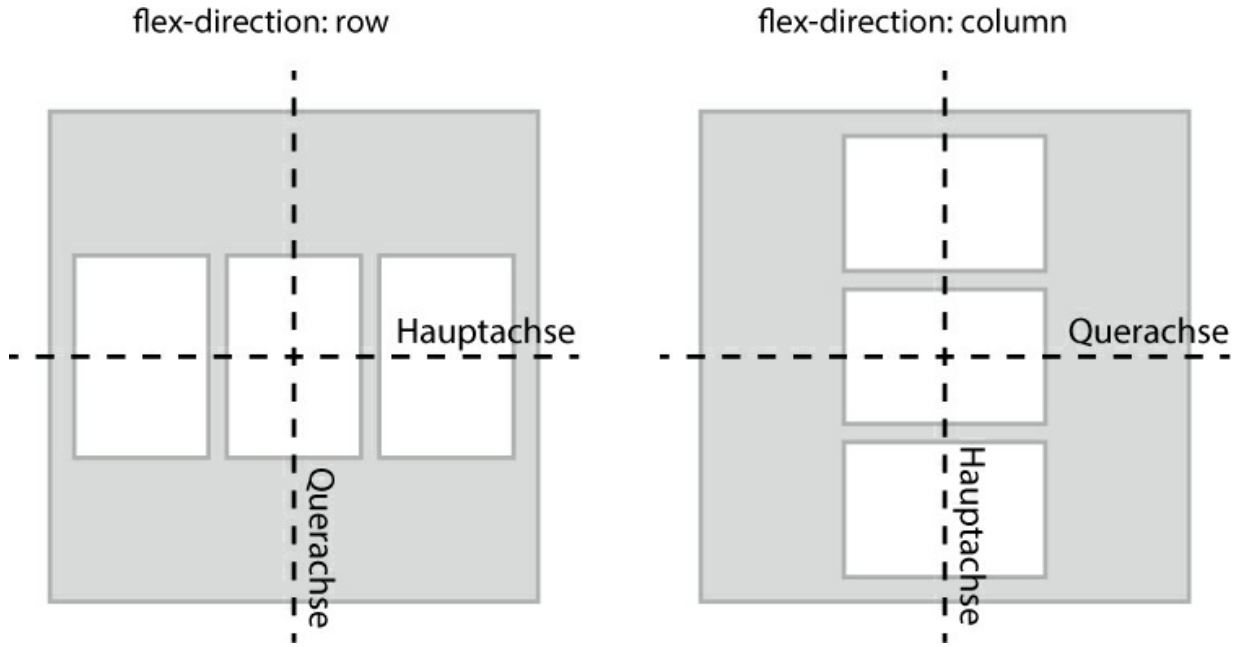


Abb. 4–7
Die Richtung der Haupt- und der Querachse hängt davon ab, ob die Flex-Kindelemente in Zeilen oder Spalten angeordnet sind.

Nun zu den Ausrichtungseigenschaften. Nehmen wir an, Sie haben einen Flex-Container mit einer Breite von 600 px mit drei Flex-Elementen, von denen jedes 150 px breit ist. Standardmäßig werden die drei Elemente in einer Zeile angezeigt; sie sind links ausgerichtet und nach allen folgen 150 px Leerraum (wie in der ersten Zeile von Abb. 4–8 gezeigt).

justify-content

Sie können diesen nicht genutzten Raum mit der Eigenschaft `justify-content` neu verteilen. Diese Eigenschaft akzeptiert eine Abfolge von Schlüsselwortwerten je nach der Richtung des übergeordneten Flex-Elements (Zeile, Spalte, umgekehrte Anordnung und so weiter). Zu Demonstrationszwecken gehe ich vom westlichen Standard mit einem Textfluss von links nach rechts aus. Der Standardwert ist `flex-start`. Er richtet alle Flex-Elemente links am Elternelement aus, wobei der ungenutzte Raum – wie im vorigen Absatz erwähnt – den verbleibenden horizontalen Raum zur Rechten einnimmt.

Die alternativen Werte sind: `flex-end`, wodurch die Elemente rechts am Container mit dem ungenutzten Raum zur Linken ausgerichtet werden, `center`, wodurch der ungenutzte Raum auf beiden Seiten der im Container zentrierten Elemente verteilt wird, `space-between`, wodurch ein gleicher Abstand zwischen den einzelnen Elementen entsteht (vor dem ersten und nach dem letzten Element aber keiner). Mit `space-around` wird auf beiden Seiten jedes Elements ein gleichmäßiger Abstand hinzugefügt.

Öffnen Sie *flexbox-alignment.html* und ändern Sie den Wert von `justify-content`, um die Auswirkungen zu prüfen. Abb. 4–8 zeigt verschiedene Werte zum Vergleich.

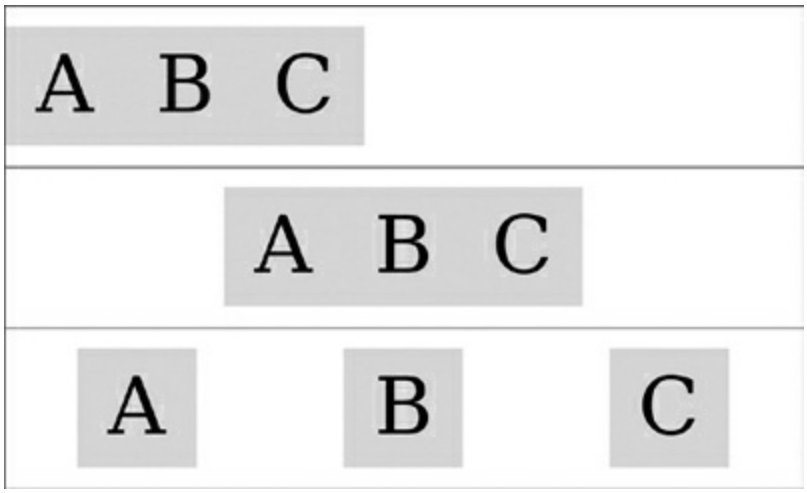


Abb. 4–8

Unterschiedliche Werte für die Eigenschaft justify-content: flex-start (oben), center (Mitte) und space-around (unten)

Wie sieht es jedoch mit der Ausrichtung auf der Querachse aus? Schließlich gibt es Fälle, in denen Ihre Flex-Elemente niedriger sind als der Flex-Container. Damit ich mich nicht wiederhole, gehe ich weiterhin davon aus, dass wir die Elemente in einer Zeile anordnen.

align-items

Die entsprechende Eigenschaft ist `align-items`; ihr Standard lautet `stretch`. Damit erhält das Element dieselbe Höhe wie das Elternelement. Dieser Wert funktioniert nur, wenn für die Elemente keine Höhe festgelegt wurde. Sonst ist der Standardwert `flex-start`, wodurch die Elemente an der Oberkante des Containers ausgerichtet werden. `flex-end` richtet die Elemente am unteren Rand des Containers aus, `center` in der vertikalen Mitte, mit gleichmäßigem Leerraum darüber und darunter; `baseline` richtet die Elemente an der Grundlinie der ersten Inhaltszeile aus.

Sehen Sie sich die Datei *flexbox-alignment.html* an und ändern Sie den Wert selbst, um die Auswirkungen zu betrachten. Abb. 4–9 zeigt verschiedene Werte.

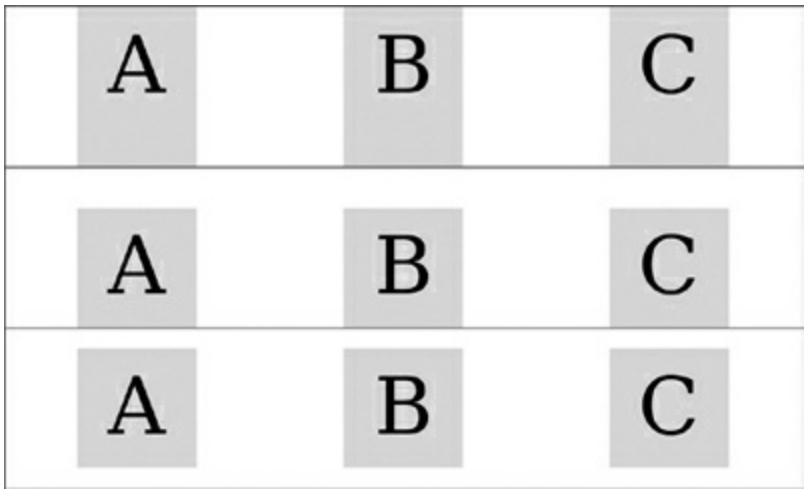


Abb. 4–9
Die Ausrichtung auf der Querachse wird durch unterschiedliche Werte für align-items bestimmt: stretch (oben), flex-end (Mitte) und center

(unten).

Zur Kontrolle der Ausrichtung einzelner Elemente verwenden Sie die Eigenschaft `align-self`. Diese Eigenschaft bezieht sich nicht auf den Container, sondern auf das Element selbst. Die Werte sind jedoch mit denen für `align-items` identisch und haben dieselben Auswirkungen (gelten aber nur für das ausgewählte Element – Geschwisterelemente bleiben unverändert).

4.2.4 Mehr Flexibilität

flex

Sie haben gesehen bis hierher, wie Sie per Flexbox Elemente ausrichten und anordnen können. Was ist jedoch mit der Flexibilität, auf die der Name hindeutet? Diese kommt durch die Möglichkeit ins Spiel, Elemente dynamisch wachsen oder schrumpfen zu lassen, sodass sie sich besser in den Container einpassen lassen. Hierzu gibt es die neue Eigenschaft `flex`. Diese Eigenschaft wird nicht auf den Container, sondern vielmehr auf Elemente angewandt und kann drei Werte annehmen:

```
E { flex: 1 2 150px; }
```

`flex` ist ein Eigenschaftskürzel. Jeder Wert steht für eine weitere Eigenschaft – der Reihe nach: `flex-grow`, `flex-shrink` und `flex-basis`. Wenn Sie alle drei Eigenschaften ausschreiben, sieht diese Regel folgendermaßen aus:

```
E {
  flex-grow: 1;
  flex-shrink: 2;
  flex-basis: 150px;
}
```

flex-basis

Was bewirken diese Eigenschaften eigentlich? Ich erläutere zunächst `flex-basis`. Dieses verhält sich grundsätzlich wie der (und hat

Vorrang vor dem) Wert der Eigenschaft `width`. Die Eigenschaft `flex-basis` können Sie sich wie eine Eigenschaft »bevorzugte Breite« vorstellen. Das bedeutet, dass die Elemente zwar flexibel sind, aber auf Grundlage dieses Werts.

Im bisher verwendeten Beispielcode sehen Sie drei Flex-Elemente mit einer `flex-basis` von 150 px und in einem Flex-Container mit einer Breite von 600 px, wobei eine ungenutzte Breite von 150 px verbleibt. Und jetzt kommen die anderen Eigenschaften ins Spiel.

flex-grow

Die Eigenschaftenwerte `flex-grow` und `flex-shrink` sind Verhältniswerte. Ignorieren Sie `flex-shrink` für einen Augenblick – jedes Element hat den `flex-grow`-Wert 1. Dieser bedeutet, dass die 150 px ungenutzter Breite im Verhältnis 1:1:1 aufgeteilt werden – also gleichmäßig. Jedem Element werden 50 Pixel zugeschlagen, sodass sie 200 px breit werden und damit den Container ausfüllen. Was passiert aber, wenn ich den `flex-grow`-Wert für eines der Kindelemente ändere?

```
#a, #b { flex: 1 2 150px; }
#c { flex: 3 2 150px; }
```

Nun hat sich das Verhältnis in 1:3:1 geändert. Das bedeutet, dass die 150 px in diesem Verhältnis auf die Elemente aufgeteilt werden. Für jedes Pixel, das den Elementen *#a* und *#b* hinzugefügt wird, werden dem Element *#c* 3 px hinzugefügt. Deshalb ist *#c* nun 240 px breit; die Elemente *#a* und *#b* sind 180 px breit. Wenn Sie den Wert `flex-grow` ändern, verändern Sie dieses Verhältnis und die Flexibilität der Elemente.

Abb. 4–10 zeigt beispielsweise, was passiert, wenn Sie `flex-grow` ändern. Im oberen Beispiel haben alle Elemente identische Werte von 1; im unteren Beispiel hat Element *B* den Wert 3: Es wird proportional vergrößert.

Abb. 4–10
Ein größerer flex-grow-Wert bedeutet, dass das Element B im unteren Beispiel proportional größer wird als die benachbarten Elemente.

flex-shrink

Die Eigenschaft `flex-shrink` funktioniert genau umgekehrt. Nehmen wir wieder dieselben Elemente, reduzieren die Breite des Containers aber auf 300 px. Dann ergibt sich durch die kombinierte Breite (oder `flex-basis`) der Elemente ein Überschuss von 50 px, der von den einzelnen Elementen im von `flex-shrink` vorgegebenen Verhältnis 2:2:2 abgezogen wird. Dies ist ein gleiches Verhältnis, entspricht also im Grunde genommen dem Verhältnis 1:1:1 oder 5:5:5. Das Ergebnis ist dasselbe: Die Breite jedes Elements wird um 50 px verringert, damit es in den Container passt.

Was passiert, wenn Sie den Wert `flex-shrink` eines Elements ändern?

```
#a, #b { flex: 1 2 150px; }
#c { flex: 3 3 150px; }
```

Für jede 2 px, die von den Elementen *#a* und *#b* abgezogen werden, verkleinert sich Element *#c* nun um 3 px. Es erhält damit eine Breite von 86 px, während seine benachbarten Elemente 107 px breit sind. Wenn Sie den Wert `flex-grow` ändern, verändern Sie dieses Verhältnis und die Flexibilität der Elemente.

Diese Flexibilität ist äußerst praktisch, wenn Sie eine Abfolge von Elementen in unterschiedlich große Container einpassen möchten. Ein Beispiel ist die Benutzeroberfläche einer App, die auf verschiedenen Geräten funktionieren soll. Flexbox ist auch gut geeignet, wenn Sie nicht wissen, wie viele GUI-Elemente es gibt, diese aber auf jeden Fall alle

proportional zueinander bleiben sollen.

4.2.5 Wrap und Flow

flex-wrap

Trotz der mit Flexbox erzielten Flexibilität kommt es vor, dass die Anzahl Ihrer Elemente einfach nicht mehr in eine Zeile (oder Spalte) eines Containers passt. In diesem Fall können Sie die Elemente mithilfe der Eigenschaft `flex-wrap` in mehrere Zeilen umbrechen. Der Standardwert ist `nowrap`: Alle Elemente bleiben in derselben Zeile. Mit dem Wert `wrap` werden die Elemente bei Bedarf in einer eigenen Zeile dargestellt:

```
E { flex-wrap: wrap; }
```

Der Wert `wrap` erzeugt unter der ersten Zeile (oder bei der Anordnung in Spalten rechts davon) neue Zeilen. Der Alternativwert `wrap-reverse` ändert die Richtung der Querachse, sodass neue Zeilen stattdessen darüber (bzw. links davon) angezeigt werden. Abb. 4–11 vergleicht die beiden unterschiedlichen Werte.

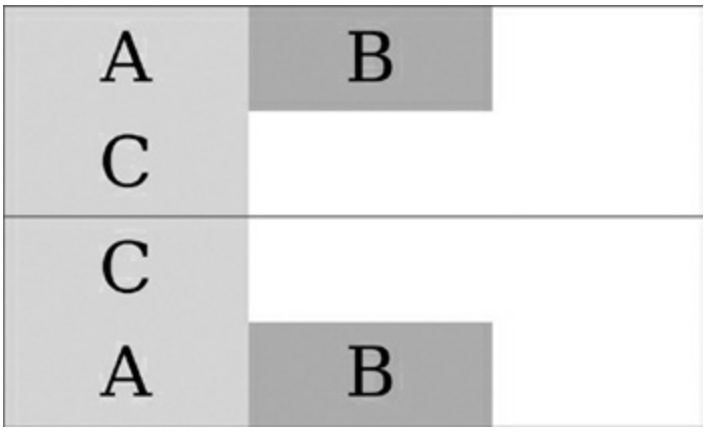


Abb. 4–11
Vergleich der Werte für die Eigenschaft flex-wrap. Das obere Beispiel hat den Wert wrap, also erscheint das Element C in einer neuen Zeile darunter, während der Wert im unteren Beispiel wrap-reverse lautet,

sodass Element C in einer neuen Zeile darüber erscheint.

Sie können `flex-wrap` mit `flex-direction` im Eigenschaftskürzel `flex-flow` kombinieren. So richten Sie eine Spalte mit mehreren Zeilen und einer umgekehrten Querachse ein:

```
E { flex-flow: column wrap-reverse; }
```

flex-flow

align-content

Wenn Elemente über mehrere Zeilen umbrochen werden, können Sie ihre Ausrichtung mit der Eigenschaft `align-content` kontrollieren. Diese Eigenschaft funktioniert wie die Eigenschaft `justify-content`, aber auf der Querachse. Hier gibt es dieselben möglichen Werte und zusätzlich noch den Wert `stretch`, mit dem die Elemente so skaliert werden, dass die Leerräume komplett gefüllt werden. Abb. 4–12 vergleicht zwei unterschiedliche Werte.

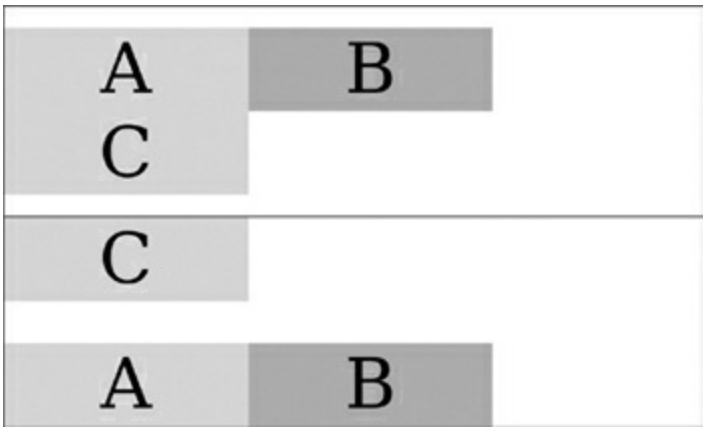


Abb. 4–12
Die Ausrichtung auf der Querachse beim Umbrechen der Flex-Elemente wird mit align-content kontrolliert: center (oben) und space-between (unten).

Flexbox kann zur Entwicklung ganzer Seitenlayouts dienen, ist aber eigentlich nicht dafür gedacht. Wahrscheinlich müssten Sie einige Tricks anwenden, um exakt das gewünschte Layout zu erzielen. Eine bessere Option ist die Verwendung von speziellen Seitenlayout-Eigenschaften.

Hier kommt das Modul Grid Layout ins Spiel.

4.3 Rasterlayouts

Raster (*grids*) sind ein grundlegendes Design-Element. Schon im Mittelalter verwendeten die Buchmaler einfache Raster. Das moderne typo-grafische Raster ist seit der zweiten Hälfte des 20. Jahrhunderts im Einsatz. Seit ein paar Jahren gibt es Bemühungen, rasterbasierte Layouts mithilfe von Kästen, die mit Floats, Abständen und Rändern versehen sind, ins Web zu übertragen und so die Möglichkeiten des Print-Designs nachzuempfinden.

CSS-Modul: Grid Layout

In den neueren Browsern wurde mit der Implementierung eines nativen CSS-Rasterlayout-Systems begonnen. Dies bietet eine Reihe von Eigenschaften, die speziell zur Darstellung von Rastern auf dem Bildschirm dienen. Das bedeutet, dass der Entwickler diese nicht mehr aus vorhandenen Eigenschaften und Verhalten zusammensetzen muss. Bevor ich jedoch mit der neuen Raster-Syntax beginne, erläutere ich die Terminologie, die im Modul *Grid Layout* verwendet wird. Selbst wenn Sie als Designer mit der Gestaltung von Layoutrastern (*grid layouts*) vertraut sind, sollten Sie etwas Zeit in das Verständnis dieser Begriffe investieren. Denn die Terminologie für die Entwicklung von CSS-Rastern unterscheidet sich deutlich von der Terminologie für typografische Raster.

Hier eine Definition der Schlüsselbegriffe dieses Moduls:

- *Grid-Element* (*grid element*) – das Container-Element, das die Begrenzung des Rasters darstellt und dessen Abmessungen definiert
- *Rasterelemente* (*grid items*) – die einzelnen im Raster platzierten Kindelemente
- *Rasterlinien* (*grid lines*) – die Trennlinien zwischen den Zeilen und Spalten; es handelt sich nicht um sichtbare, sondern lediglich um

gedachte Linien

- *Raster-Tracks* (*grid tracks*) – Die einzelnen Spalten oder Zeilen im Raster werden *Tracks* genannt.
- *Rasterbereiche* (*grid areas*) – Jeder Schnittpunkt einer Zeile und einer Spalte erzeugt einen Rasterbereich – entsprechend den Zellen in einer Tabelle.

Um ein Raster zu erzeugen, legen Sie zuerst die Anzahl der Zeilen im Grid-Element fest. Somit erzeugen Sie eine Abfolge von Tracks. Anschließend werden die Rasterelemente im Raster positioniert, wobei die Tracks als Koordinaten dienen. Sie werden entweder in einen Rasterbereich eingepasst oder überspannen mehrere Rasterbereiche (siehe Abb. 4–13).

Wenn Sie schon einmal mit einem Tabellenkalkulationsprogramm wie Excel gearbeitet haben, können Sie sich ein Raster leicht vorstellen. Die Rasterlinien – die gedachten Linien zwischen den Zellen – definieren eine nummerierte Anzahl von Zeilen und Spalten. (Wenn Sie noch keine Tabellenkalkulationssoftware verwendet haben oder nichts über typografische Raster wissen und nichts mit alldem anfangen können – machen Sie sich nichts daraus; ich erkläre es Ihnen Schritt für Schritt.)

Hinweis

Die in diesem Abschnitt verwendete Syntax ist ein Bestandteil des Internet Explorer 10. Die Spezifikation wurde jedoch in der Zwischenzeit mehrfach geändert. Mehr über die Weiterentwicklung der Syntax erfahren Sie im Abschnitt »Die Grid-Layout-Syntax« auf Seite 104.

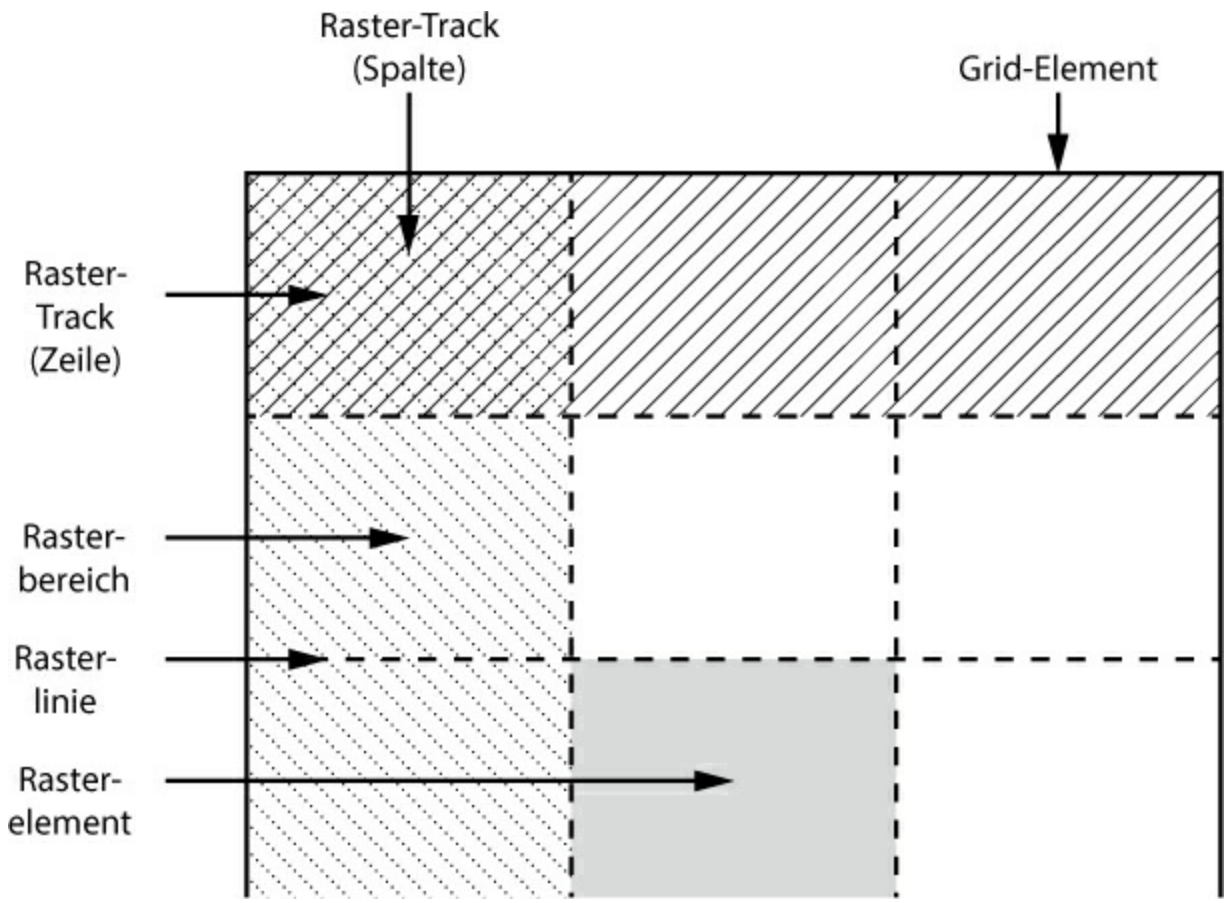


Abb. 4–13
Ein einfaches 3×3-Raster zeigt die Hauptbegriffe der CSS-Rasterlayout-Syntax.

4.3.1 Das Raster deklarieren und definieren

display

Der erste Schritt besteht in der Deklaration des Rasters. Wie in der Einführung zu diesem Abschnitt erläutert, ist dies das Container-Element, dessen Abmessungen die Rasterbegrenzungen darstellen und dem alle Rastereigenschaften zugewiesen werden. Es wird unter Verwendung des neuen Werts `grid` mit der herkömmlichen `display`-Eigenschaft definiert:

```
E { display: grid; }
```

grid-columns

Als Nächstes definieren Sie die Raster-Tracks mit den Eigenschaften `grid-columns` und `grid-rows`. Der Wert für die einzelnen Eigenschaften ist eine Längeneinheit, die die Breite der Spalte oder die Höhe der Zeile bestimmt. Als einfaches erstes Beispiel erfahren Sie nun, wie Sie zwei Spalten erzeugen – eine davon hat 75 Prozent von der Breite des Rasters, die andere 25 Prozent.

```
E { grid-columns: 75% 25%; }
```

Sie sind hier nicht auf Prozentwerte festgelegt; vielmehr sind alle Längeneinheiten möglich. Außerdem wurde speziell für die Definition von Rastern die neue Längeneinheit, *fr* (*fraction* = Bruchteileinheit), ins Leben gerufen. Mit fr weisen Sie den Zeilen oder Spalten einen entsprechenden Anteil des verfügbaren Raums zu. Im vorigen Beispiel sollten Sie anstelle von Prozentsätzen die Einheit fr verwenden; Sie erhalten dann dasselbe Ergebnis:

```
E { grid-columns: 3fr 1fr; }
```

Der wahre Vorteil von Bruchteileinheiten zeigt sich jedoch, wenn diese mit anderen Einheiten vermischt werden. Nehmen wir beispielsweise an, Sie möchten eine Spalte mit fester Breite und zwei identische und dynamische Spalten als Teil eines responsiven Layouts erzeugen. Die Maßeinheit fr eignet sich dazu bestens. Mit dem folgenden Code werden drei Spalten erzeugt: eine mit einer Breite von 15 em und die verbleibenden beiden jeweils mit der Hälfte der verbleibenden Breite:

```
E { grid-columns: 15em 1fr 1fr; }
```

grid-rows

Zeilen fügen Sie mit denselben Einheiten und auf dieselbe Weise hinzu. Nehmen wir beispielsweise an, dass ich drei Zeilen benötige: Die erste ist 100 px hoch, die zweite hat den Wert `auto` (sie ist also so hoch wie ihr Inhalt), die dritte ist 15 em hoch. Diese Zeilen erzeugen Sie mit dem folgenden Code:

```
E { grid-rows: 100px auto 15em; }
```

Wenn Sie alle diese Eigenschaften zusammensetzen, können Sie Ihr Raster definieren. Als einfaches Beispiel erzeugen wir ein grundlegendes Raster aus drei Spalten und drei Zeilen, insgesamt also neun Zellen. Hier ist der Code dafür:

```
E {
  display: grid;
  grid-columns: 1fr 4fr 1fr;
  grid-rows: 100px auto 15em;
}
```

Die Spalten dieses Rasters werden im Verhältnis 1:4:1 verteilt. Die obere Zeile ist 100 px hoch, die untere 15 em, die mittlere hat eine automatische Höhe, die sich an die darin enthaltenen Inhalte anpasst.

Dieses Raster sehen Sie in Abb. 4–14.

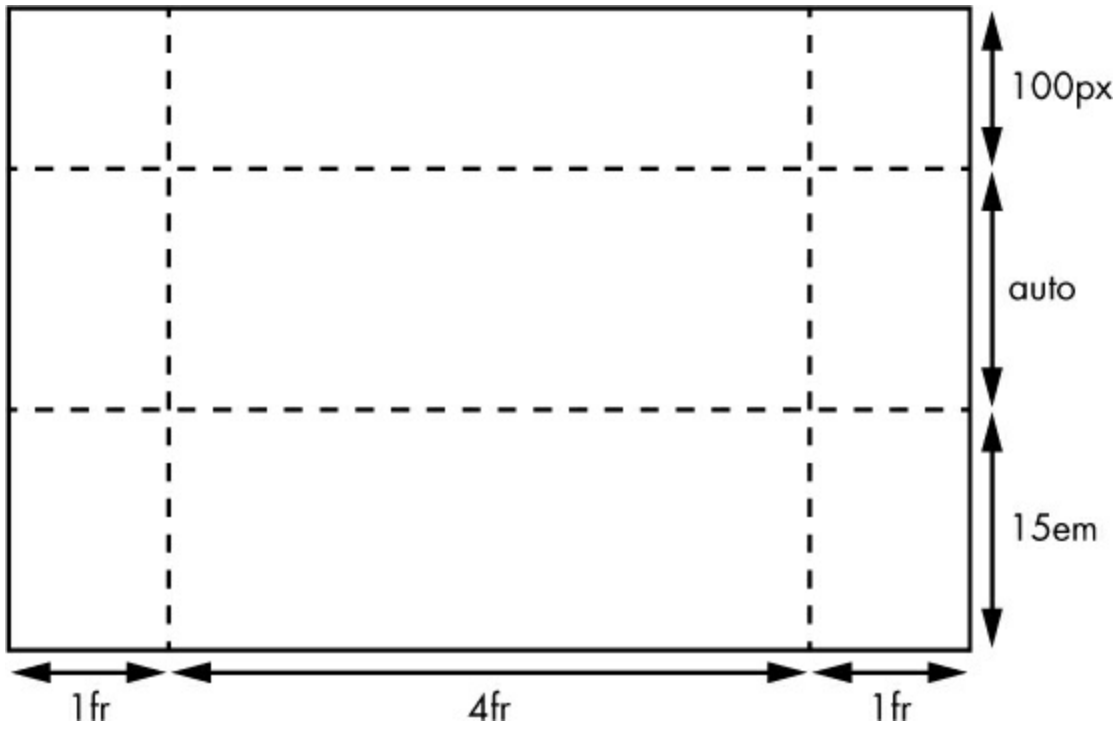


Abb. 4–14
Ein einfaches 3x3-Raster

4.3.2 Rasterlinien wiederholen

Die einfachen, in diesem Kapitel definierten Raster eignen sich zwar gut zum Lernen und auch für manche realen Situationen. Häufig benötigen Sie jedoch für eine angemessene Kontrolle über die Inhalte recht komplexe Raster. Viele Raster enthalten 12 oder auch 16 Spalten, wobei sich zwischen jeder Spalte normalerweise ein Steg (Leerraum) befindet. Mit der Rasterlayout-Syntax kann selbst ein sechsspaltiges Raster mit Stegen recht redundant wirken:

```
E { grid-columns: 1fr 10px 1fr 10px 1fr 10px 1fr 10px 1fr 10px
1fr; }
```

repeat()

Stellen Sie sich nun vor, wie redundant der Code wäre, wenn Sie 16 Spalten erzeugen wollten! Sie können dieses Problem mit der `repeat()`-Funktion angehen. Diese Funktion kennt zwei Werte: eine Ganzzahl, die die Anzahl der Wiederholungen festlegt, und die Rasterlinienwerte, die wiederholt werden sollen:

```
E { grid-columns: 1fr repeat(5, 10px 1fr); }
```

Damit definieren Sie ein Track mit einer Breite von einer Bruchteileinheit und wiederholen dann ein Muster, das fünfmal zwischen 10 px und 1 fr abwechselt.

Der Internet Explorer 10 verwendet die Syntax einer älteren Version der Spezifikation, in der die Anzahl der Wiederholungen nach den runden in eckige Klammern gesetzt wird. Wenn Sie Apps für die Windows 8-UI erzeugen, verwenden Sie die folgende Syntax:

```
E { -ms-grid-columns: 1fr repeat(10px 1fr)[5]; }
```

4.3.3 Elemente im Raster erzeugen

grid-column und grid-row

Um ein Element im Raster zu platzieren, weisen Sie ihm unter Verwendung der Eigenschaften `grid-column` und `grid-row` einen Zellverweis zu. Jede Eigenschaft akzeptiert eine einzelne Ganzzahl als

Wert, die sich auf ein Track bezieht (entweder eine Spalte oder eine Zeile). Möchten Sie beispielsweise ein Element in der Zelle auf dem Schnittpunkt der zweiten Spalte und der zweiten Zeile platzieren, notieren Sie den folgenden Code (das Ergebnis sehen Sie in Abb. 4–15):

```
F {
  grid-column: 2;
  grid-row: 2;
}
```

Der Standardwert von `grid-column` und `grid-row` ist jeweils 1. Wenn Sie eines davon weglassen, wird das Element in der ersten Zeile bzw. Spalte platziert. Der folgende Code platziert das Element demnach in der Zelle auf dem Schnittpunkt der zweiten Spalte und der ersten Zeile (siehe Abb. 4–16).

```
F { grid-column: 2; }
```

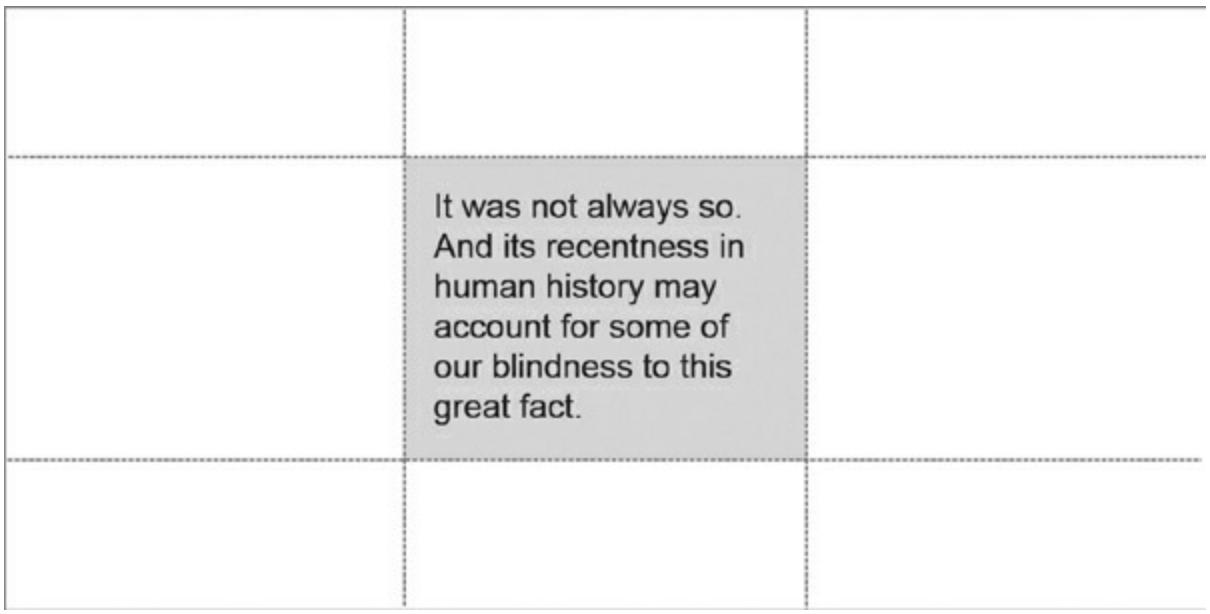


Abb. 4–15
Ein Element, das im Raster der zweiten Spalte und der zweiten Zeile platziert wurde (zur Verdeutlichung wurden Rasterlinien hinzugefügt)

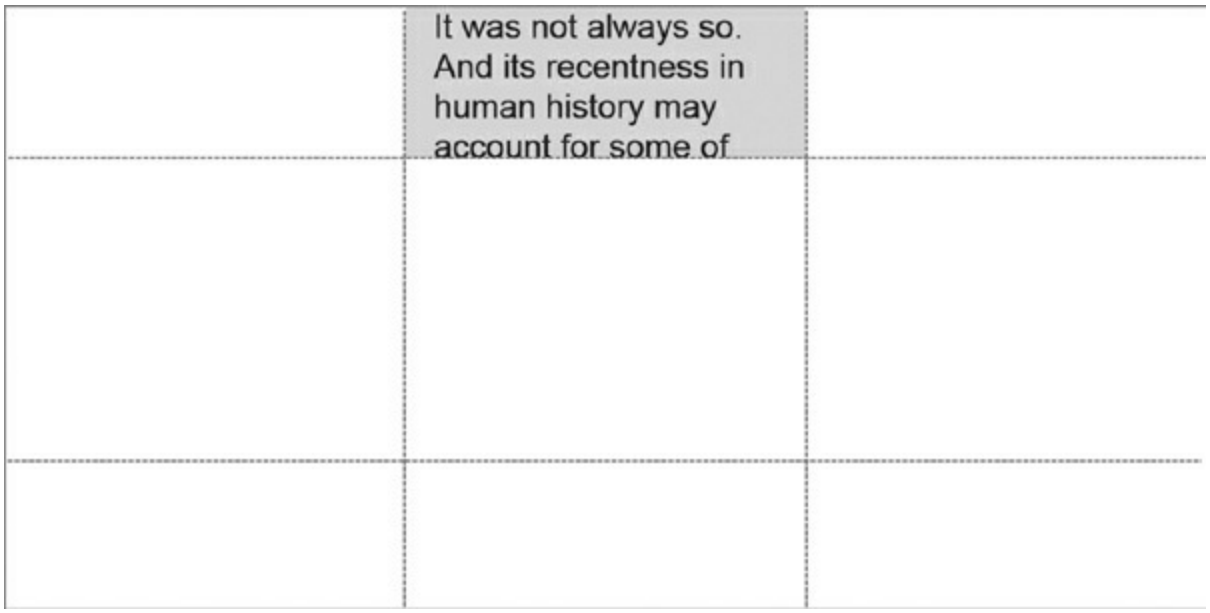


Abb. 4–16
Ein Element, das im Raster in der zweiten Spalte und der ersten Zeile platziert wurde (die Rasterlinien wurden zur Verdeutlichung hinzugefügt)

grid-column-span und grid-row-span

Standardmäßig wird das Element in die angegebene Zelle eingepasst; Sie können das Element jedoch vergrößern, sodass es sich horizontal oder vertikal über mehrere Zellen erstreckt. Dazu nutzen Sie die Eigenschaften `grid-column-span` und `grid-row-span`, von denen jede wieder einen einzelnen Ganzzahlwert erwartet. Dieser bestimmt, über wie viele Tracks sich das Element erstrecken soll. Das folgende Element erstreckt sich beispielsweise über die ersten drei Zeilen der ersten Spalte:

```
F { grid-row-span: 3; }
```

Erinnern wir uns: Sie müssen `grid-column` und `grid-row` in diesem Fall nicht definieren, weil beide den Standardwert 1 haben. Das Element wird in der ersten Spalte der ersten Zeile platziert – genau dort, wo Sie es haben wollen –, und es reicht, wie in Abb. 4–17 gezeigt, über alle drei Zeilen.

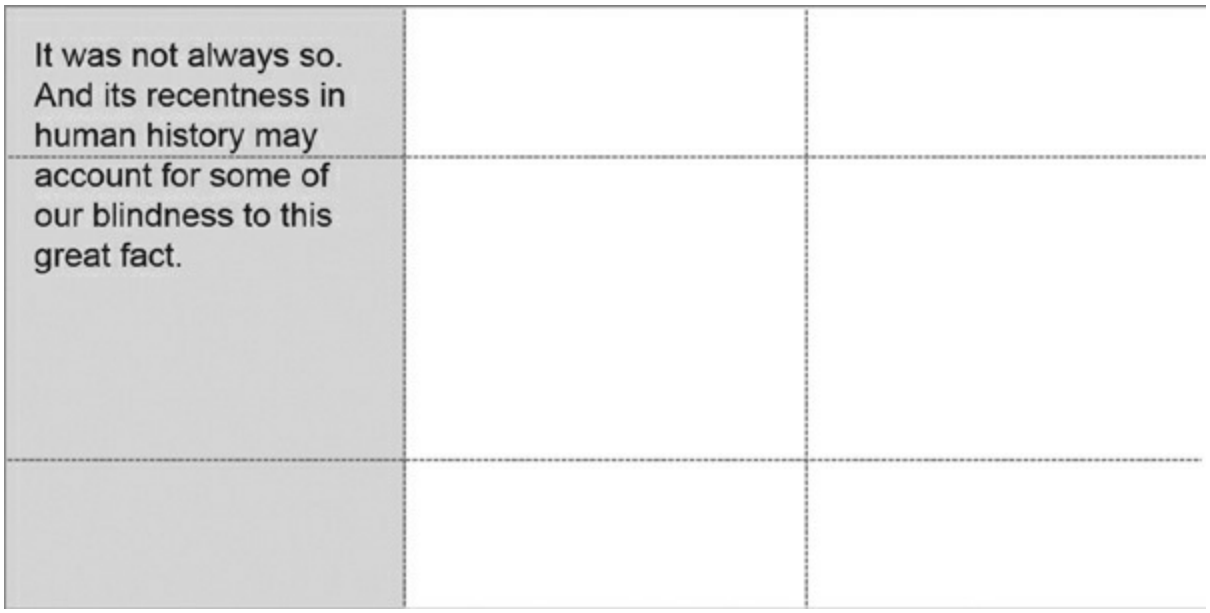


Abb. 4–17
Ein Element, das sich im Raster über drei Zeilen der ersten Spalte erstreckt (zur Verdeutlichung habe ich Rasterlinien hinzugefügt)

4.3.4 Ausrichtung und Stapelfolge

grid-column-align und grid-row-align

Standardmäßig werden die im Raster platzierten Elemente so positioniert, dass sich ihre linke obere Ecke in der linken oberen Ecke der festgelegten Zelle befindet. Sie können diese Standardeinstellung aber mit den Eigenschaften `grid-column-align` und `grid-row-align` ändern. Diese Eigenschaften erwarten jeweils eine Abfolge von Schlüsselwortwerten (`center`, `end`, `start` und `stretch`). Diese Werte legen die Ausrichtung des Elements innerhalb seiner Zelle fest.

Der Wert `center` richtet das Element am horizontalen und vertikalen Zentrum der Zelle aus, `start` und `end` hingegen hängen von der Schreibrichtung in Ihrem Dokument ab. Wenn Sie mit einer von links nach rechts gelesenen Sprache wie Deutsch, Englisch usw. arbeiten, richtet `start` das Element links oder oben an der Zelle aus, `end` rechts

oder unten. Durch den Wert `stretch` vergrößert sich das Element so, dass es die gesamte Zelle in ganzer Breite oder Höhe ausfüllt.

Eine Demonstration dieser Werte finden Sie in der Beispieldatei *grid-align.html*. Sie enthält ein Raster aus drei gleichen Zellen, von denen jede ein identisches Element enthält. Ich habe die einzelnen Elemente mit dem folgenden Code unterschiedlich positioniert:

```
#a { grid-row-align: end; }
#b { grid-row-align: stretch; }
#c { grid-row-align: center; }
```

Wie Sie erkennen können, sind die drei Elemente unterschiedlich in der Zeile ausgerichtet: *#a* ist an der Unterkante ausgerichtet, *#b* wird so vergrößert, dass die ganze Zelle ausgefüllt ist und *#c* ist vertikal zentriert. Sie können die Ergebnisse in Abb. 4–18 vergleichen.

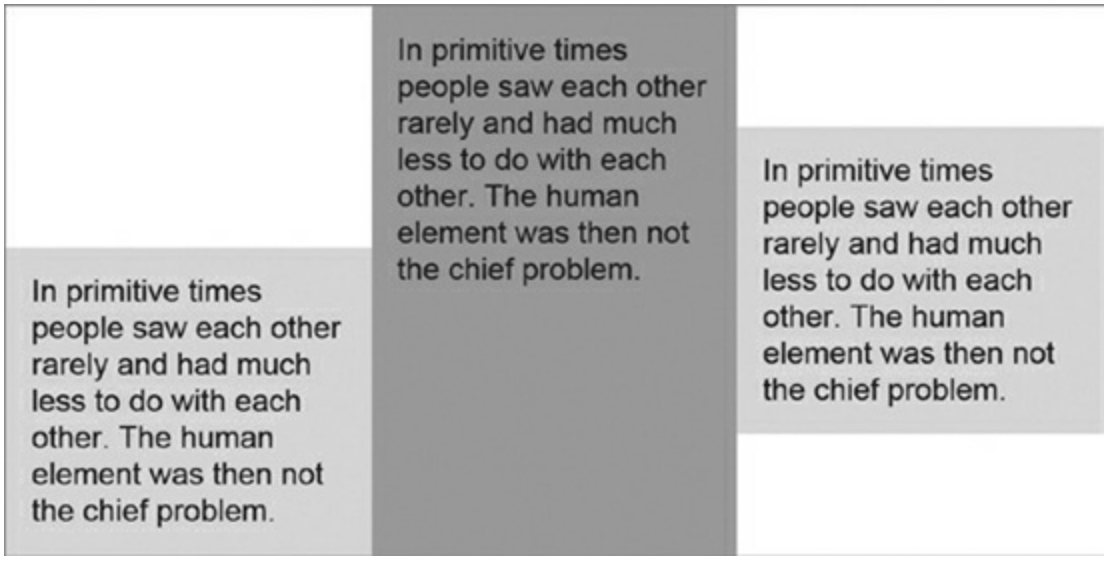


Abb. 4–18
Die Elemente werden innerhalb der Zellen mit der Eigenschaft grid-row-align unterschiedlich ausgerichtet.

Bei der Platzierung von Elementen im Raster werden Sie feststellen, dass bestimmte Elemente sich in derselben Zelle überlappen. Standardmäßig werden die Elemente in der DOM-Reihenfolge gestapelt, wobei später im DOM auftauchende Elemente über frühere Elemente gelegt werden.

Der folgende Code verdeutlicht dies:

```
<div id="a">...</div>
<div id="b">...</div>
```

Wären die `div`-Elemente *#a* und *#b* in derselben Zelle platziert, würde *#b* über *#a* liegen, weil es sich in der DOM-Reihenfolge weiter hinten befindet. Sie können diese Reihenfolge mit der Eigenschaft `z-index` ändern (die Sie von der absoluten Positionierung in CSS2.1 her kennen). Soll Element *#a* über *#b* erscheinen, gehen Sie folgendermaßen vor:

```
#a { z-index: 10; }
```

Einen anderen Ansatz zum Umgang mit Stapelkonflikten bespreche ich im Abschnitt »Exclusions und Grids« auf Seite 208.

4.3.5 Die Grid-Layout-Syntax

Wie bereits erwähnt, wurde die Syntax des Grid-Layout-Moduls aktualisiert, nachdem sie in den Internet Explorer 10 implementiert wurde. Die aktualisierte Spezifikation befindet sich immer noch im Entwurfsstadium und ist noch weit von einer offiziellen Empfehlung entfernt. Während ich dies schreibe, muss die neue Syntax noch in den Browsern implementiert werden.

Die gute Nachricht ist, dass die Grundprinzipien unverändert bleiben. Die Änderungen betreffen vor allem neue und aktualisierte Eigenschaftennamen. Bei der Definition des Rasters heißen beispielsweise die Eigenschaften `grid-columns` und `grid-rows` jetzt `grid-definition-columns` und `grid-definition-rows`. Das ist noch recht einfach; beim nächsten Teil müssen Sie etwas aufpassen:

Die Eigenschaften `grid-column` und `grid-row` werden nun durch `grid-column-position` und `grid-row-position` ersetzt, wobei die früheren Eigenschaften nach wie vor als Kürzel vorhanden sind: `grid-column` ist das Kürzel für `grid-column-position` und `grid-column-span`. `grid-row` ist das Kürzel für `grid-row-position` und `grid-row-span`.

Es gibt noch weitere Kürzel: `grid-position` ist das Kürzel für `grid-column-position` und `grid-row-position`, `grid-span` steht für `grid-column-span` und `grid-row-span`, `grid-area` für (tief einatmen) `grid-column-position`, `grid-row-position`, `grid-column-span` und `grid-row-span` (oder `grid-position` und `grid-span`, wenn Sie es bevorzugen, Kürzel noch abzukürzen).

Achtung

Die Syntax hat sich seit Erscheinen der englischen Auflage (erneut) leicht geändert. Die aktuellste Version finden Sie direkt in der Spezifikation (siehe »Literaturempfehlungen« am Ende dieses Kapitels).

4.3.6 Über die Terminologie des Rasterlayout-Moduls

Die Terminologie des Grid-Layout-Moduls unterscheidet sich deutlich von der für typografische Raster verwendete Terminologie. Zu dieser Tatsache haben sich einige Designer recht deutlich geäußert. Mark Boulton, einer der wichtigsten Verfechter von Rastersystemen auf Webseiten, schrieb zur im Rasterlayout-Modul verwendeten Terminologie einen offenen Brief an das W3C:

> *Ich bin verwirrt, weil ich für Raster, die es ja seit Jahrhunderten gibt, neue Begriffe erfinden muss. Es stört mich auch ein wenig, dass das mentale Modell, von dem diese Terminologie abgeleitet ist, eher auf Tabellen und Tabellenkalkulationsprogrammen aufbaut […] als auf Rastern und Layouts.*

Tab Atkins von Google und der CSS Working Group äußerte sich mit deutlichen Worten darüber, warum CSS-Raster offensichtlich mehr Gemeinsamkeiten mit Tabellen als mit typografischen Rastern haben:

> *Weil es sich beim Rasterlayout im Grunde genommen um den guten Teil der Tabellenlayouts handelt, ohne die besch*** Teile, haben wir*

dieselbe Terminologie verwendet.

4.3.7 Rasterbereiche

Das Rasterlayout-Modul bietet noch eine andere Möglichkeit, um Raster zu definieren. Während ich dies schreibe, hat noch kein Browserhersteller Rasterbereiche (*grid templates*)implementiert oder macht Anstalten dazu. Trotzdem bin ich überzeugt, dass sie eine kurze Erklärung verdient haben, weil sie ein intelligenter und andersartiger Ansatz sind.

grid-template

Mit der Eigenschaft `grid-template` können Sie mithilfe einer Anzahl eindeutiger Bezeichner in Textstrings Rasterbereiche definieren. Jeder Bezeichner stellt einen Bereich dar, jeder String eine Zeile. Ein einzeiliges Raster mit drei Spalten ließe sich beispielsweise folgendermaßen erzeugen:

```
E {
  display: grid;
  grid-template: 'a b c';
}
```

Für mehrere Zeilen, weitere Strings und Zellen, die sich über mehrere Zeilen und Spalten erstrecken, fügen Sie eine zusätzliche Instanz des Buchstabens in der gewünschten Richtung hinzu. Im folgenden Code beispielsweise reicht Zelle *a* über zwei Zeilen und Zelle *b* über zwei Spalten. Ich habe die Strings formatiert, sodass Sie sich Ihr Aussehen leichter vorstellen können:

```
E {
  display: grid;
  grid-template:
   'a b b'
   'a c d';
}
```

Um ein Element in einem Raster zu platzieren, das mit einem

Rasterbereich deklariert ist, verwenden Sie einfach den Buchstaben der Zelle als Wert für die Eigenschaft `grid-area`. Der folgende Code platziert beispielsweise ein Element in Zelle *b*:

```
F { grid-area: 'b'; }
```

Wie erwähnt, habe ich keine Ahnung, ob Rasterbereiche jemals das Licht der Welt in einem Browser erblicken werden, aber sie bieten eine interessante Möglichkeit, die Erzeugung einfacher Raster anzugehen.

4.4 Die ferne Zukunft

Die drei in diesem Kapitel behandelten CSS-Module – Multi-column Layout, Flexible Box Layout und Grid Layout – stellen hinsichtlich der Gestaltung von Webseiten in der nahen Zukunft geradezu eine tektonische Verschiebung dar. Wenn wir aber noch weiter vorausblicken, entdecken wir weitere unglaubliche Module, die als Vorschläge eingereicht wurden. Weil sie sich in unterschiedlichen Vorschlags- und Stabilitätsstadien befinden, habe ich beschlossen, sie nicht in diesem Kapitel abzuhandeln. Ich beschäftige mich aber im Abschnitt »Layouts: Noch weiter in die Zukunft geblickt« auf Seite 249 damit. Wenn reichhaltige, dynamische Weblayouts Sie begeistern (und warum sollte dies nicht der Fall sein?), dann wird Ihnen dieser Abschnitt sicherlich gefallen.

4.5 Zusammenfassung

Dieses Kapitel hat sich mit zahlreichen neuen Ansätzen zur Entwicklung von Weblayouts beschäftigt. Von der einfachen Rastererzeugung bis hin zu DOM-unabhängigen UI-Layouts über dynamischen Textumfluss sollen die neuen Methoden die alten dreispaltigen Layouts zu einer überholten Technik machen. Wenn Sie die neuen Methoden mit den leistungsfähigen, in Kapitel 3 vorgestellten Media Queries kombinieren,

können Sie flexible, dynamische Layouts und Benutzeroberflächen für die verschiedensten Geräte erzeugen.

Die Zukunft von CSS-Layouts ist jedoch noch aufregender. Während ich diese Zeilen verfasse, sind zwar viele Vorschläge und Spezifikationen noch völlig im Fluss – die neuen Möglichkeiten bedeuten aber, dass wir in fünf Jahren vollständig neue Werkzeuge zur Verfügung haben und dass dadurch die Entwicklung von Websites und Anwendungen völlig umgekrempelt wird.

4.6 Literaturempfehlungen

Das stets zuverlässige MozDev enthält eine klare Einführung zu mehrspaltigen Layouts, *Using CSS multi-column layouts:* https://developer.mozilla.org/en-US/docs/CSS/Using_CSS_multi-column_layouts/.

Die Flexbox-Syntax hat sich so oft geändert, dass es quasi keine aktuelle Online-Ressource mehr gibt! Auch wenn sich Stephen Hays Artikel *Learn You a Flexbox for Great Good!* auf http://www.the-hay-stack.com/2012/01/04/learn-you-a-flexbox/ auf eine veraltete Syntax bezieht, sollten Sie ihn lesen, denn Stephens Kenntnisse des CSS-Layouts sind unvergleichlich.

Die beste Erklärung des Grid-Layout-Moduls, zumindest hinsichtlich der Implementierung im Internet Explorer 10, finden Sie im *»Internet Explorer 10 Guide for Developers«*: http://msdn.microsoft.com/en-us/library/ie/hh673533%28v=vs.85%29.aspx/.

Die *aktuellste Version des Grid-Layout-Moduls* finden Sie unter http://dev.w3.org/csswg/css-grid.

Lesen Sie *Mark Boultons offenen Brief* zur Grid-Layout-Terminologie auf

http://www.markboulton.co.uk/journal/comments/open-letter-to-w3c-css-working-group-re-css-grids.

5 Modernes JavaScript

Nach Markup und CSS-Formatierung ist der dritte Eckpfeiler bei der Entwicklung von Web-Frontends das Verhalten, und das wird im Web durch JavaScript umgesetzt. Die Aktivitäten rund um HTML5 und die dazugehörigen APIs haben auch dieser Skriptsprache viele neue Funktionen und (darf ich das sagen?) Verbesserungen gebracht.

Für Frontend-Entwickler wird es damit einfacher, hin und wieder JavaScript einzusetzen, und wer wirklich tief in die Funktionsweise dieser Skriptsprache einsteigen möchte, bekommt jetzt mehr Leistung geboten. Im ersten Abschnitt dieses Kapitels werde ich einige dieser Funktionen betrachten.

Die größte Veränderung in der JavaScript-Welt betrifft aber wohl nicht die Sprache selbst, sondern die Anzahl der Bibliotheken und Frameworks, in denen sie verwendet wird. Es gab bei den Bibliotheken sozusagen eine kambrische Explosion, und im zweiten Abschnitt des Kapitels präsentiere ich Ihnen meinen Burgess-Schiefer – ein kleines, aber erhellendes Streiflicht auf die aktuellen Entwicklungen.

5.1 Neu in JavaScript

JavaScript hat sich in den vergangenen drei bis vier Jahren kräftig verändert, von vielen neuen Möglichkeiten der DOM-Traversierung bis hin zu einer ganzen Reihe neuer Ereignisse. Wenn Sie der JavaScript-Community nicht aktiv folgen und auf dem neuesten Stand bleiben, geht dies schnell an Ihnen vorbei.

In diesem Abschnitt gehe ich einige der neuen Funktionen von JavaScript durch. Meine Auswahl treffe ich dabei extrem unwissenschaftlich und allein aufgrund meiner ganz persönlichen Einschätzung ihres Nutzens. Dieser Abschnitt enthält längst nicht alle

Neuerungen der Sprache – lediglich eine Handvoll Methoden, Attribute und Ereignisse, die Sie häufig in Ihren tagtäglichen Webprojekten einsetzen könnten.

5.1.1 Die Attribute async und defer

Standardmäßig lädt ein Browser alle Seitenelemente nacheinander und führt die dabei vorgefundenen JavaScript-Skripte sofort aus. Durch Letzteres wird das Parsen des HTML-Codes behindert. Sites mit langsamer Anbindung oder solche mit viel ausführbarem Skript können dabei sehr langsam werden. Daher finden sich Links zu Skriptdateien in der Praxis idealerweise eher am Ende des Dokuments, wenn der Rest der Seite bereits geladen wurde.

defer

Es wurde jedoch eine Lösung für dieses Blockierungsproblem entwickelt. Wenn das `defer`-Attribut zum `script`-Element für externe Dateien hinzugefügt wird (wenn also das `src`-Attribut vorhanden ist), werden Skripte geladen, ohne den Parser anzuhalten – allerdings werden sie erst dann ausgeführt, nachdem der HTML-Code vollständig geladen wurde. Da es sich bei `defer` um ein boolesches Attribut handelt, wird kein Wert benötigt:

```
<script src="foo.js" defer></script>
```

async

HTML5 bietet eine weitere neue Option, nämlich die asynchrone Ausführung der Skripte. Wie beim verzögerten Laden werden externe Skripte auch hier heruntergeladen, ohne den Parser anzuhalten; der Unterschied ist aber, dass der Parser diese Skripte dann so bald wie möglich ausführt. Damit erhalten Sie sozusagen »das Beste von beidem«: der Parser wird während des Downloads nicht angehalten, Sie brauchen aber mit der Ausführung auch nicht bis zum Schluss zu warten. Für diese Option benötigen Sie das `async`-Attribut, das ebenso vom Typ Boolean ist:

```
<script src="foo.js" async></script>
```

Als Nachteil ergibt sich, dass die Skripte nicht unbedingt in der Reihenfolge ausgeführt werden, in der sie im Dokument aufgerufen werden. Sie können sich also nicht auf Abhängigkeiten zwischen den Skriptdateien verlassen. Ist die Reihenfolge jedoch nicht von Belang und sind alle Skripte eigenständig, könnte diese Möglichkeit die beste Wahl sein.

Sie sehen die Funktionsweisen der unterschiedlichen Ansätze in Abb. 5–1. Die erste Zeile zeigt das Standardverhalten, bei dem das HTML-Parsing während des Ladens und der Ausführung der Skripte unterbrochen wird. Die zweite Zeile zeigt die verzögerte Ausführung mittels `defer`. Die Skripte werden hier geladen, ohne den Parser anzuhalten, und dann ausgeführt, sobald das DOM bereit ist. In der letzten Zeile schließlich läuft der Parser beim Laden des Skripts ebenfalls weiter, hält allerdings bei dessen Ausführung an.

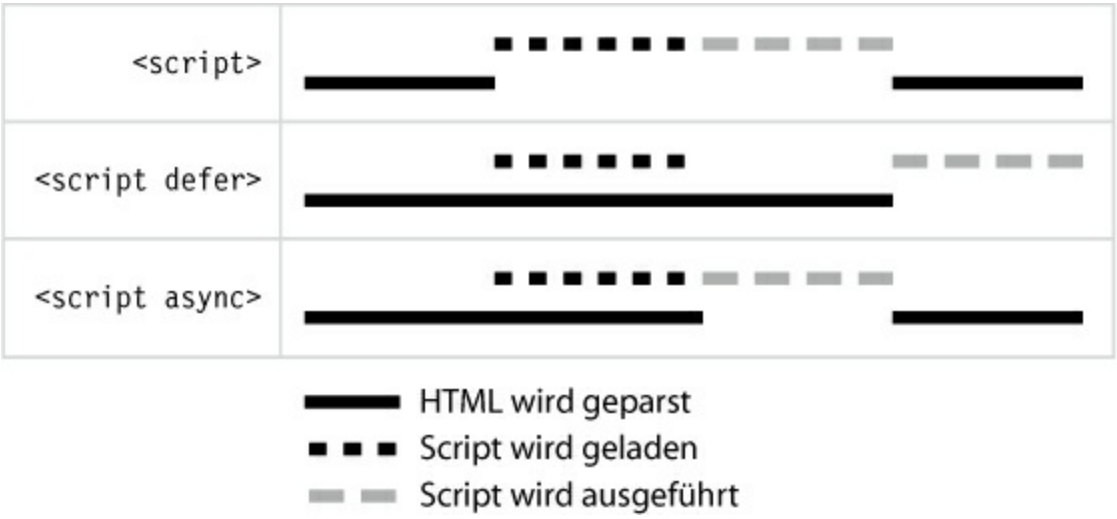


Abb. 5–1
So wirken sich verzögert und asynchron geladene JavaScript-Dateien aus.

5.1.2 Die addEventListener-Methode

Die Standardmethode zum Hinzufügen von Event-Handlern zu Elementen waren bisher die verschiedenen Event-Eigenschaften wie etwa `element.onclick` oder `element.onmouseout`. Das war alles so weit ganz gut, doch leider erlaubten sie nur einen dieser Event-Handler pro Element. Das W3C legte mit der `addEventListener()`-Methode einen flexibleren Ansatz vor.

Diese Methode ist, ehrlich gesagt, nicht ganz neu, doch erst seit 2011 bietet `addEventListener()` mit der Implementation in IE9 einen guten Unterstützungsgrad über die Browser hinweg. Als generische Methode verwendet `addEventListener()` zwei erforderliche und einen (semi-)optionalen Parameter:

```
el.addEventListener(type,listener,useCapture);
```

Der `type`-Parameter gibt den Typ des Ereignisses an, das überwacht werden soll; der `listener` ist das Objekt, das benachrichtigt wird, wenn das Ereignis eintritt – meist ist das eine Funktion; `useCapture` ist schließlich ein boolescher Parameter für das sogenannte Event-Bubbling. Auf diesen werde ich nicht näher eingehen, denn Event-Bubbling geht über den Umfang dieses Buchs hinaus. Doch obwohl es sich hierbei eigentlich um einen optionalen Parameter handeln soll, müssen Sie ihn verwenden, da einige Browser Ihnen ansonsten einen Fehler auswerfen. Wenn Sie ihn nicht brauchen oder nicht wissen, was er bewirkt, setzen Sie `useCapture` auf `false` und kümmern sich nicht weiter darum.

Ein Beispiel

Hier ist ein kleines Beispiel für die einfache Verwendung von `addEventListener()`; dieser Code wartet auf ein Click-Ereignis und führt eine anonyme Funktion aus, um beim Auftreten des Ereignisses eine Nachricht in der Konsole auszugeben:

```
el.addEventListener('click', function () {
  console.log('Click!');
},false);
```

Selbst auf dieser einfachen Ebene ist `addEventListener()` bereits sehr nützlich. Die Methode ist jedoch noch wertvoller, als sie vielleicht

scheint, denn Sie können damit mehrere Ereignisse gleichen Typs zu einem Element hinzufügen. Sie können also Skripte aus unterschiedlichen Quellen verwenden, ohne sich Sorgen machen zu müssen, versehentlich einen Event-Listener zu überschreiben. Das nächste Beispiel zeigt, wie zwei benannte Funktionen (*named functions*) auf dasselbe Element angewendet werden; jede von ihnen könnte sich in einer komplett anderen Skriptdatei befinden und doch würde sie beim Anklicken des Elements ausgeführt:

```
el.addEventListener('click', foo, false);
el.addEventListener('click', bar, false);
```

Der Einsatz benannter Funktionen anstelle von anonymen Funktionen (wie in diesem Beispiel) ist eine gute Idee. Benannte Funktionen lassen sich einfach besser an anderer Stelle Ihrer Skripte verwenden. Und das ist ganz wesentlich, wenn Sie Event-Handler loswerden möchten, wie ich es in »Event-Listener entfernen« auf Seite 113 beschreibe.

Das event-Objekt

Die Methode `addEventListener()` gibt nach dem Auslösen des Ereignisses ein Objekt mit dessen Details zurück. Sie können auf dieses Objekt mit einer anonymen Funktion mit einem eindeutigen Bezeichner als Argument zugreifen. Dieser Bezeichner ist der Name des Objekts. In diesem Beispiel nenne ich es `ev` und gebe es in der Konsole aus, damit Sie das Ergebnis sehen können:

```
el.addEventListener('click', function (ev) {
  console.log(ev);
},false);
```

Abb. 5–2 zeigt das Ereignis-Objekt als Ausgabe in Firebug. Sie sehen einige verschiedene Eigenschaft-Wert-Paare, die recht ausführliche Informationen über das eingetretene Ereignis geben (es gibt noch mehr Informationen, die nicht angezeigt werden).

Diese Methode funktioniert auch mit benannten Funktionen; Sie

müssen der Definition der Funktion nur den eindeutigen `id`-Wert als Argument übergeben:

```
var foo = function (ev) {
  console.log(ev);
};
el.addEventListener('click', foo, false);
```

currentTarget

Zu den nützlicheren Eigenschaften des Ereignis-Objekts gehört `currentTarget`. Dieses Objekt enthält Informationen über das Element, für welches das Ereignis eingetreten ist. Sie können diesem Element dann problemlos Verhalten zuweisen, so wie in diesem Code, der die Farbe des angeklickten Elements auf `#F00` setzt:

```
el.addEventListener('click', function (ev) {
  ev.currentTarget.style.color = '#F00';
},false);
```

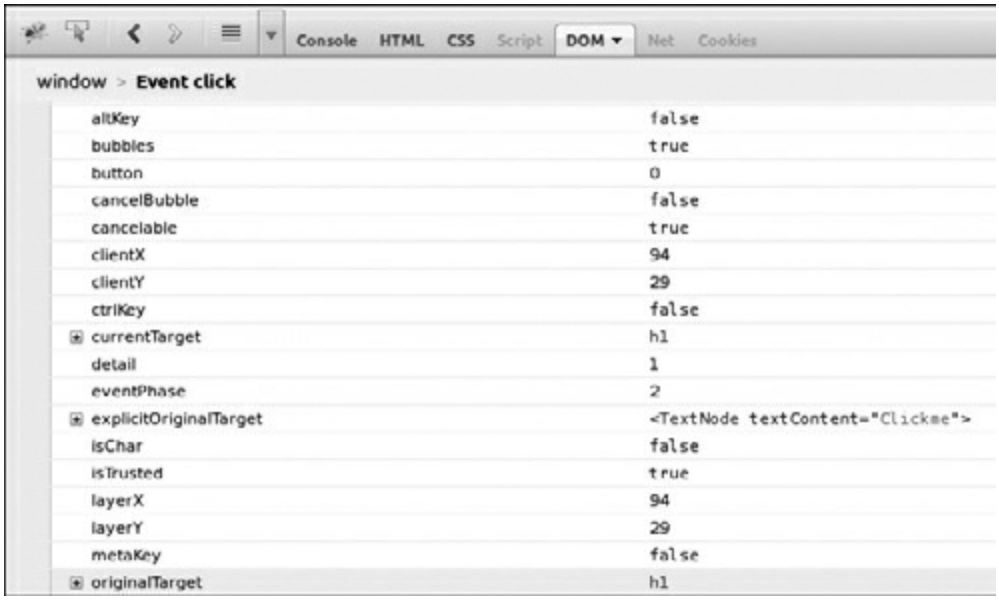


Abb. 5–2
Einige der im vom Klick-Ereignis erzeugten Objekt enthaltenen Informationen

Das Ereignis-Objekt ist auch nützlich, wenn Sie verhindern wollen, dass eine Standardaktion für ein Element ausgeführt wird. Zum Beispiel

können Sie an ein `a`-Element mit enthaltenem Link ein Ereignis anfügen, wenn der Browser dem Link nicht folgen soll, bevor die dem Event zugewiesene Funktion ausgeführt wurde. Verwenden Sie dazu die Methode `preventDefault()`:

```
el.addEventListener('click', function (ev) {
  ev.preventDefault();
  ...
},false);
```

Event-Listener entfernen

removeEventListener()

Manchmal möchten Sie einen Event-Handler von einem Element entfernen, damit er nicht erneut auslöst. Dazu verwenden Sie die `removeEventListener()`-Methode, die die gleichen drei Parameter aufweist wie `addEventListener()`:

```
el.removeEventListener(type,listener,useCapture);
```

Zum Entfernen eines Ereignisses müssen Sie `removeEventListener()` exakt die gleichen Parameter übergeben, die auch von der `addEventListener()`-Methode verwendet werden. Daher hatte ich zuvor so nachdrücklich für den Einsatz von benannten statt von anonymen Funktionen geworben. Der nachfolgende Code zeigt, was ich meine; zuerst wird ein Event-Listener hinzugefügt, um die Funktion `foo` auszuführen, dann wird er mit genau denselben Parametern wieder entfernt:

```
el.addEventListener('click',foo,false);
el.removeEventListener('click',foo,false);
```

Sie können auch anonyme Funktionen aus Event-Handlern entfernen, aber diese Methoden gelten seit den letzten JavaScript-Revisionen als veraltet; daher meiden Sie sie am besten.

5.1.3 Das DOMContentLoaded-Ereignis

Die alte Methode zum Starten von Skripten nach dem Laden einer Seite war der Einsatz des `load`-Ereignisses. Dieser Weg hat allerdings den Nachteil, dass das Ereignis erst auslöst, wenn wirklich alle Ressourcen – Bilder, Stylesheets und so weiter – vollständig geladen wurden, und das konnte manchmal eine deutliche Verzögerung bedeuten. Die Entwickler haben dann ein eigenes Ereignis, `domready`, erstellt. Dieses tritt ein, wenn das DOM fertig geladen wurde, aber noch vor dem Laden der Ressourcen.

Das Ereignis `domready` wurde rasch in zahlreiche JavaScript-Bibliotheken übernommen; in den Browsern wurde es nach und nach als `DOMContentLoaded` nativ implementiert, und jetzt wurde es in HTML5 stan-dardisiert. `DOMContentLoaded` wird für das `document`-Objekt etwa so ausgelöst:

```
document.addEventListener('DOMContentLoaded', function () {...},
false);
```

Insbesondere die Verhaltensebene der Site kann viel früher initialisiert werden, sodass der Benutzer früher mit der Erkundung Ihrer Site beginnen kann – das ist besonders wichtig bei langsamen Internetverbindungen oder für Seiten mit vielen Bildern, die etwas mehr Ladezeit benötigen könnten. Falls Sie Skripte haben, die erst nach dem Laden sämtlicher Ressourcen ausgeführt werden sollen, können Sie weiterhin `load` verwenden.

5.1.4 Eingabe-Events

Um der stark ansteigenden Zahl von Geräten mit Touch-Eingabe (egal ob mittels Finger, Eingabestift oder Würstchen) Rechnung zu tragen, wurde eine Reihe entsprechender neuer Ereignisse eingeführt. Bei der Touch-Eingabe werden zwar auch weiterhin Klickereignisse ausgelöst, aber darauf alleine ist kein Verlass – es gibt grundlegende Unterschiede in der Funktionsweise von Maus- und Touch-Geräten. Drückt ein Benutzer zum Beispiel die Maustaste, kommt es zu einer kurzen Pause, in

der der Browser noch wartet, ob es sich bei der Eingabe um einen Einfach- oder Doppelklick handelt. Die Verzögerung ist kurz, auf Touch-screens ist sie aber spürbar, und es entsteht der Eindruck, das Gerät reagiere schlecht.

Touch Events und Pointer Events

Das W3C überdenkt momentan zwei verschiedene Eingabeereignismodelle: *Touch Events* zielen direkt auf die Touch-Eingabe ab. Seit einiger Zeit geht die Entwicklung daran aktiv voran, und sie wurden auch bereits in einigen Browsern implementiert. Das Modell wurde jedoch mancherorts kritisiert, und es könnte auch an einigen bestehenden Patenten scheitern. *Pointer Events* sind ein neuer Vorschlag für ein einheitliches Eingabemodell. Er wird vom W3C geprüft, wurde jedoch noch nicht formal bestätigt oder implementiert.

Welches dieser beiden Modelle – wenn überhaupt – zum Standard wird, bleibt abzuwarten. Es gibt jedoch Hinweise darauf, dass Pointer Events bevorzugt werden. Wegen der Unwägbarkeiten gehe ich nur kurz auf die Funktionsweise der beiden Modelle ein.

Touch Events

Die Level-1-Spezifikation der Touch Events weist vier Ereignisse aus:

- `touchstart` löst aus, wenn ein Finger (ich verwende ab jetzt dieses Wort, um nicht ständig »Würstchen« zu wiederholen) den Bildschirm berührt.
- `touchend` löst aus, wenn der Finger vom Bildschirm gehoben wird.
- `touchmove` löst aus, wenn ein Finger sich zwischen zwei Bildschirmpunkten bewegt.
- `touchcancel` löst aus, wenn der User Agent eine Berührung unterbricht oder der Finger den aktiven Bereich verlässt.

Die Level-2-Spezifikation der Touch Events ist seit Oktober 2013 als

Recommendation bei der W3C (läuft aber immer noch nicht überall), und es kommen damit zwei weitere Ereignisse hinzu:

- `touchenter` löst aus, wenn sich ein bereits auf dem Bildschirm befindlicher Finger über das Zielelement bewegt.
- `touchleave` löst aus, wenn der Finger den Zielbereich, nicht aber den Bildschirm verlässt.

Als Analogie zum Mausklick eignet sich wohl am besten das `touchend`-Ereignis. Es löst aus, wenn der Finger vom Bildschirm gehoben wird und der Benutzer damit seine Aktion abgeschlossen hat:

```
el.addEventListener('touchend', function () {...}, false);
```

Immer wenn ein Touch-Ereignis ausgelöst wird, erzeugt es ein `TouchEvent`-Objekt, das weitere Informationen über das Ereignis enthält. Zu diesen Informationen gehört auch `TouchList`, ein Objekt mit dem Unterobjekt `touches`. Dieses wiederum enthält Informationen darüber, wie viele Finger den Bildschirm berühren – das ist natürlich für Multitouch-Ereignisse hilfreich. Bedenken Sie, dass jede Berührung ein Touch-Ereignis erzeugt. Beim Einsatz von Multitouch enthält daher jedes Touch-Ereignis auch alle vorhergehenden Bildschirmberührungen. Das beim Berühren des Bildschirms mit dem ersten Finger erzeugte `TouchList`-Objekt enthält also Informationen über Berührung 0; die vom zweiten Finger erzeugte `TouchList` enthält Informationen über Berührung 0 und 1; die `TouchList` des dritten Fingers über Berührung 0, 1 und 2 und so weiter.

Pointer Events

Wie bereits geschildert, stellen Pointer Events einen einheitlichen Ansatz zur Erstellung vor Ereignissen unabhängig von der tatsächlichen Eingabemethode dar – der Zeiger kann ein Finger, eine Maus, ein Eingabestift oder anderes sein. In einer Hinsicht gleicht das Konzept den Mausereignissen im DOM: Die einzelnen Schritte der Benutzeraktion

werden aufgezeichnet, dabei werden aber `pointer*`-Events verwendet: `pointerdown`, `pointerup`, `pointercancel`, `pointermove`, `pointerover` und `pointerout`. Um zum Beispiel ein Ereignis auszulösen, wenn der Zeiger freigegeben wird, verwenden Sie `pointerup`:

```
el.addEventListener('pointerup', function () {...}, false);
```

Jedes Ereignis erzeugt beim Auslösen eine `pointerEvent`-Schnittstelle, die zusätzlich zu den Eigenschaften aus der `MouseEvent`-Schnittstelle – `currentTarget`, Koordinaten, wo das Ereignis eintrat, und so weiter – eine Reihe neuer Eigenschaften bereitstellt. Dazu gehören die Art des verwendeten Zeigers (`pointerType`), der angewendete Druck und der Winkel des Zeigers (falls relevant).

Ohne Implementierung kann ich Ihnen kein Beispiel für den Einsatz dieser Ereignisse geben, aber die Spezifikation ist sicherlich sinnvoll. Sie wirkt wie eine logische Erweiterung bestehender Ereignisse und sie ist allgemeiner – und damit nützlicher – als reine Touch-Ereignisse. Während ich dies schreibe, wurde es zwar noch nicht bestätigt, aber scheinbar sind Pointer Events die Spezifikation, die das W3C weiter entwickeln wird. In der Zwischenzeit füllen einige in der Community entstandenen Bibliotheken die funktionale Lücke: Unter all der Vielfalt sollten Sie einen genaueren Blick auf das PointerEvents-Polyfill werfen.

5.1.5 CSS-Selektoren in JavaScript

Es war immer ein wenig problematisch, auf Elemente im DOM zuzugreifen. Es gab `getElementById()` und `getElementsByTagName()`; doch gar zu häufig erforderte es einiges an DOM-Traversierung oder -Filterung, um genau das gewünschte Element zu erhalten. Bibliotheken wie jQuery lösten dieses Problem mit Methoden, die CSS-Selektoren als Argumente akzeptieren:

```
var foo = $('p.foo');
```

Dieser eindeutige Vorteil floss schnell auch

querySelector() und

querySelectorAll()

über zwei native Methoden zurück in JavaScript ein: `querySelector()` und `querySelectorAll()`. Beide funktionieren ähnlich: Sie übernehmen CSS-Selektoren als Argumente. Der Unterschied ist, dass `querySelector()` nur das erste passende Element anspricht. Das Ergebnis ist also ein einzelner Knoten, während `querySelectorAll()` alle Instanzen des Elements erfasst und das Resultat daher als `NodeList` zurückliefert:

```
var foo = document.querySelector('p.foo');
var foo = document.querySelectorAll('p.foo');
```

In Abb. 5–3 sehen Sie die Ergebnisse der unterschiedlichen Selektoren als Ausgabe in der Konsole. `querySelector()` ist das erste Ergebnis, `querySelectorAll()` das zweite.

Abb. 5–3
Vergleich der Ergebnisse von querySelector() und querySelectorAll()

Sie können mit diesen beiden Methoden jeden komplexen Selektor oder mehrere Selektoren verwenden; unzulässig sind nur relative Selektoren – sie dürfen also nicht mit einem Kombinator wie > oder + beginnen. Folgendes Argument ist daher ungültig:

```
var el = document.querySelectorAll('+ h2');
```

find() und findAll()

Ein Vorschlag zur Lösung dieses Problems sind zwei ähnliche Methoden namens `find()` und `findAll()`, die in etwa dieselbe Funktionsweise haben, jedoch auch Selektoren relativ zum Ziel zulassen. Eine gültige Version der vorherigen Codes sieht so aus:

```
var foo = el.findAll('+ h2');
```

Diese Methoden sind zwar nicht überlebensnotwendig, sie vereinfachen aber die Traversierung des DOM-Baums ein wenig. Ich hoffe daher, dass sie bald auf breiter Basis übernommen werden.

5.1.6 Die Methode getElementsByClassName

Eine recht große (und überraschende) Lücke beim Zugriff auf das DOM mit JavaScript war eine Methode zum Ansprechen von Elementen über ihren Klassennamen. Wie im Fall von `DOMContentReady` führte auch das wieder dazu, dass Entwickler ihre eigenen Lösungen in Skripte fassten, von denen sich viele um den Namen `getElementsByClassName()` herum verdichteten. Diese Lösung wurde ebenfalls in HTML5 standardisiert, und die Methode ist nun nativ in den modernen Browsern anzutreffen. `getElementsByClassName()` übernimmt einen einzelnen Zeichenkettenwert und liefert eine `NodeList` aller Elemente zurück, deren Klassenname mit der Zeichenkette übereinstimmt:

```
var el = document.getElementsByClassName('foo');
```

Sie können auch nach mehreren Klassennamen suchen, indem Sie sie in der Zeichenkette mit Leerzeichen voneinander abtrennen. Der nachfolgende Code wählt alle Elemente mit den Klassennamen `foo` oder `bar` aus.

```
var el = document.getElementsByClassName('foo bar');
```

5.1.7 Interaktion mit Klassen

classList

Mit der Einführung des `classList`-Objekts hat sich in JavaScript auch die Interaktion mit Klassennamen von Elementen vereinfacht. Jedes Element im DOM hat eine zugehörige `classList`, und hier können Sie sich auf Ihr Sprachgefühl verlassen: Das ist eine Liste aller Klassen

für dieses Element. So erhalten Sie die `classList` eines Elements und geben sie in der Konsole aus:

```
var el = document.querySelector('.foo');
console.log(el.classList);
```

Das `classList`-Objekt enthält eine Reihe von Eigenschaften und Methoden zur Manipulation von Klassen. Sie können mit der `contains()`-Methode prüfen, ob eine Klasse vorhanden ist: Bei einem Rückgabewert von `true` ist die angegebene Klasse vorhanden, andernfalls wird `false` zurückgeliefert. Eine Klasse hinzufügen können Sie mit der `add()`-Methode; zur Entfernung einer Klasse verwenden Sie `remove()`. Der nachfolgende Code überprüft den Klassennamen `foo`, fügt ihn hinzu, falls er noch nicht vorhanden ist, und löscht ihn, falls es ihn bereits gibt:

```
if (el.classList.contains('foo') {
  el.classList.remove('foo');
} else {
  el.classList.add('foo');
}
```

Dasselbe erreichen Sie schneller mit der `toggle()`-Methode; `toggle()` entfernt einfach eine vorhandene Klasse oder fügt eine nicht vorhandene hinzu:

```
el.classList.toggle('foo');
```

5.2 JavaScript-Bibliotheken

JavaScript-Bibliotheken sind Sammlungen vorgefertigter Skripte, die den Aufwand bei der Entwicklung von Anwendungen verringern sollen. In den letzten Jahren ist ihre Anzahl explodiert, und Sie werden heute kaum noch an einem großen Projekt arbeiten, das keine Bibliotheken verwendet. Die Bandbreite ist erstaunlich und reicht von kompakten Bibliotheken, die nur einem Zweck dienen, bis hin zu ausgedehnten Frameworks. Was immer Sie mit JavaScript anstellen möchten – die Chancen stehen gut, dass irgendjemand bereits eine Bibliothek dafür

geschrieben hat.

Manchmal ist es am besten, seine Skripte von Grund auf selbst zu schreiben, besonders dann, wenn Sie die beträchtlichen Dateigrößen einiger Bibliotheken umgehen möchten. Für die Arbeit in großen Teams ist es jedoch hilfreich, wenn sich alle an dieselben Standards halten – hier empfiehlt sich der Einsatz von Bibliotheken. Das gilt auch, wenn Sie ein Projekt schnell in die Produktionsphase bringen müssen – und in der Regel spricht noch vieles mehr für Bibliotheken.

Eine kleine Auswahl

Ich stelle hier kurz vier JavaScript-Bibliotheken vor, die für Frontend-Entwickler von Nutzen sind. Denken Sie daran, dass ich hier kaum an der Oberfläche des tatsächlich Verfügbaren kratze. Fast jeder Entwickler, mit dem Sie sich unterhalten, wird seine eigenen Vorlieben haben (und ihnen vielleicht ohne Scheu erklären, warum sie die weit bessere Alternative darstellen).

Zum Einsatz einer Bibliothek müssen Sie sie in den meisten Fällen nur herunterladen oder verlinken und allen Code, der von der Bibliothek abhängt, in den späteren Dateien unterbringen. Erinnern Sie sich dabei auch an die Warnung zur Verwendung des `async`-Attributs. Falls die Installation komplizierter abläuft, finden Sie die vollständigen Anweisungen in der Dokumentation der Bibliothek.

WARNUNG

Der Einsatz von JavaScript-Bibliotheken kann den Code der Seite deutlich aufblähen und ihre Ladezeit negativ beeinflussen. Überlegen Sie genau, ob Sie sie verwenden möchten, besonders wenn Sie Sites entwickeln, die auch auf Mobilgerätnutzer abzielen. In diesem Fall kann die Performance ein wichtiger Faktor sein.

5.2.1 jQuery

Bestimmt haben Sie schon einmal von jQuery gehört, das sich schnell zum De-facto-Standard bei der Webentwicklung gemausert hat. Im August 2012 setzten bereits geschätzte 50 Prozent der einen Million Top-Websites darauf. Wenn Sie mit jQuery vertraut sind und all seine Fähigkeiten kennen, überspringen Sie diesen Abschnitt ruhig. Allen anderen gebe ich hier einen kurzen Überblick.

`jQuery` ist ein JavaScript-Framework, das das Schreiben von Skripten vereinfacht. Es abstrahiert häufig verwendete Funktionen und bietet eine einheitliche Erfahrung auf allen Browsern. jQuery basiert auf einer Reihe von Methoden, die mit einfachen Argumenten vom Autor einige recht komplexe Aufgaben ausführen.

Ein Beispiel

Hier ein einfaches Beispiel, das ich gleich mit Ihnen durchgehen werde:

```
$(document).ready(function () {
  $('h1').addClass('foo');
});
```

Die erste Zeile ist zur Verwendung von jQuery erforderlich. Sie erfüllt praktisch die Funktion des `DOMContentLoaded`-Ereignisses, indem sie die anonyme Funktion ausführt, sobald das DOM bereitsteht. Die zweite Zeile enthält einen einfachen jQuery-Ausdruck: Der erste Teil ist ein CSS-Selektor, der einen Knoten oder eine Knotengruppe auswählt, worauf die Aktion angewendet werden soll, und der zweite Teil ist eine Methode, die die anzuwendende Aktion beschreibt. In diesem Fall werden alle `h1`-Elemente um eine Klasse `foo` erweitert.

Der Einsatz des Selektors zu Beginn des Ausdrucks sorgt immer wieder für Verwirrung, da wir unsere Sätze im Englischen (und auch im Deutschen) andersherum aufbauen; es ist so, als würden wir sagen: »Apfel ich werde essen.« Falls es dadurch für Sie einfacher wird, stellen Sie sich vor, wie Meister Yoda es ausdrücken würde.

Elemente mit Ereignissen versehen

Beim Einsatz von jQuery sind Sie nicht auf die Traversierung oder Manipulation des DOM-Baums beschränkt; es leistet noch viel mehr. Zum Beispiel können Sie Elemente

mithilfe der `on()`-Methode mit Ereignissen versehen – `on()` ist wie `addEventListener`, bietet aber ein paar zusätzliche Vorteile. Dazu gehört etwa, dass Sie ein einzelnes Element mit mehreren Event-Listenern versehen können. In diesem Beispiel wird eine anonyme Funktion ausgeführt, sobald ein `h1`-Element angeklickt oder berührt wird:

```
$('h1').on('click touch',function () { ... });
```

Lange Ausdrücke

Ein weiteres tolles Feature von jQuery ist die Fähigkeit, Methoden aneinanderzuhängen und aus Selektoren und Aktionen lange Ausdrücke zu erstellen. Der nachfolgende Code ist etwas komplizierter, als Sie es bisher von mir gewohnt waren. Finden Sie seine Funktionalität selbst heraus, bevor ich ihn erkläre?

```
$('.foo').on('click', function (ev) {
  $(ev.currentTarget).find('.bar').css('background-color','#f00');
});
```

Hier ist der Ablauf: Zuerst wird allen Elementen mit der Klasse `foo` der Event-Listener Click hinzugefügt, als Nächstes wird eine anonyme Funktion beim Auslösen dieses Ereignisses ausgeführt und das EreignisObjekt der Variablen `ev` zugewiesen; in dieser Funktion: Suche alle Elemente mit der Klasse `bar`, die Unterelemente des Elements sind, für welches das Ereignis ausgelöst wurde, und ändere deren Hintergrundfarbe in `#f00`.

Die Bandbreite unterschiedlicher Methoden in jQuery ist weit größer, als dass ich hier alles aufzählen könnte. Ich schlage daher vor, einen Blick in die – besonders für ein Open-Source-Projekt – absolut vorbildliche Dokumentation zu werfen.

jQuery Mobile

jQuery wird von *jQuery Mobile* begleitet. Das klingt zunächst wie eine für Mobilgeräte optimierte Version der Bibliothek. Ganz so einfach ist es aber nicht: Tatsächlich ist jQuery Mobile eine Erweiterung von jQuery, die plattformübergreifende Widgets und Stile bereitstellt, ebenso wie neue Ereignisse und Methoden, welche die neuen

Fähigkeiten der mobilen Geräte ausnutzen. Als Grundlage wird die jQuery-Bibliothek benötigt, Ihre Seiten werden dadurch also noch größer.

Zepto.js und jQTouch

Wenn Sie eine leichte und für den mobilen Gebrauch optimierte Bibliothek suchen, sollten Sie eine Alternative in Betracht ziehen. Hier bietet sich zum Beispiel Zepto.js an. Sie bietet eine zu jQuery kompatible API, ist dabei aber nur ein Viertel so groß. Die Bibliothek jQTouch bietet viele der Funktionen von jQuery Mobile, bleibt dabei aber viel leichter, und sie ist kompatibel zu jQuery und Zepto. Allerdings könnte die Browserunterstützung etwas magerer ausfallen als für jQuery Mobile. Nehmen Sie am besten alle Alternativen einmal genau unter die Lupe – so finden Sie heraus, welche für Ihre Zwecke am besten geeignet ist.

5.2.2 YepNope

Ich habe bereits in Kapitel 3 den Einsatz von Media Queries in CSS und JavaScript zum Laden von Ressourcen in Abhängigkeit von der Gerätegröße angesprochen. Doch was ist mit all den anderen unterschiedlichen Fähigkeiten und Funktionen, die Sie überprüfen könnten? Vielleicht wollen Sie das Laden bestimmter Ressourcen davon abhängig machen, ob ein Browser die Konsole oder eine bestimmte API unterstützt (mehr darüber erfahren Sie in Kapitel 6). Sie könnten maßgeschneiderte Funktionen zur Prüfung jeder kritischen Abhängigkeit schreiben, der Einsatz eines JavaScript-Loaders wie YepNope.js könnte jedoch die bessere Wahl sein.

Die Idee hinter YepNope ist unglaublich einfach: Sie übergeben eine Bedingung, die überprüft werden soll, und geben dann eine Ressource an, die in Abhängigkeit vom Ergebnis geladen werden soll. Als einfaches Beispiel wollen wir testen, ob der Browser über eine Konsole verfügt. Falls nicht, laden wir eine virtuelle Konsole:

```
yepnope({
```

```
    test: window.console,
    nope: 'foo.js'
  });
```

Sie erkennen recht schnell, was hier vor sich geht. Die Funktion `yepnope` wird aufgerufen. Sie besitzt zwei Eigenschaften: `test`, das ist die Bedingung, die entweder `true` oder `false` zurückliefern soll, und `nope`, eine Ressource, die für den Testwert `false` geladen werden soll. Wird `window. console` also unterstützt, tue nichts, ansonsten lade `foo.js`.

Es gibt noch einige weitere Eigenschaften, zum Beispiel `yep`, das eine Ressource angibt, die für den Wert `test` gleich `true` geladen werden soll. `both` lädt die Ressource unabhängig vom Wert `test`, und `callback` führt eine Funktion aus, nachdem der Test abgeschlossen wurde. Erweitern wir damit das vorhergehende Codebeispiel ein wenig:

```
yepnope({
  test: window.console,
  yep: 'bar.js',
  nope: 'foo.js',
  both: 'foobar.css',
  callback: function () { ... }
});
```

Ich führe hier dieselbe Überprüfung durch wie zuvor, doch nun lade ich `bar.js` für `test` gleich `true`, `foo.js` für `false` und `foobar.css` unabhängig vom Ergebnis. Wenn der Test abgeschlossen ist, wird außerdem noch die anonyme Funktion ausgeführt.

Jeder Test, der `true` oder `false` als Ergebnis zurückliefert, kann in `test` ausgeführt werden. Seine volle Leistungsfähigkeit entfaltet YepNope.js aber in Kombination mit Modernizr – der nächsten Bibliothek auf unserer Liste.

5.2.3 Modernizr

Bei experimentellen Funktionen

Die Browserunterstützung für experimentelle Funktionen, egal ob in HTML, CSS oder JavaScript, kann sehr unterschiedlich sein.

Sichere Alternativlösungen für den Fall, dass eine Funktion nicht unterstützt wird, sind nicht immer einfach umzusetzen. Modernizr widmet sich diesem Problem. Dazu führt es eine Reihe von Überprüfungen auf von Ihnen angegebene Funktionen aus und liefert dann `true` oder `false` als Ergebnis zurück.

CSS

Modernizr lässt sich auf zwei verschiedenen Wegen einsetzen. Der erste führt über CSS. Sagen wir zum Beispiel, Sie wollen überprüfen, ob der Browser des Anwenders Flexbox-Eigenschaften unterstützt. Zunächst erzeugen Sie mit dem Build-Tool eine eigene Modernizr-Version, wobei Sie darauf achten, die `flexbox`-Option anzuklicken. Dann fügen Sie in den Head-Bereich Ihres Dokuments einen Link zu der erstellten Datei ein. Wenn das Dokument fertig geladen wurde, wird dem `html`-Element je nach Testergebnis die Klasse `flexbox` oder `no-flexbox` hinzugefügt.

Diese Klasse könnte dann zur Formatierung der Seite in Abhängigkeit von der vorhandenen Flexbox-Unterstützung dienen; zum Beispiel könnten Sie ein Element standardmäßig als Block-Element anzeigen, in entsprechenden Browsern aber als Flex-Container:

```
.foo { display: block; }
.flexbox .foo { display: flex; }
```

JavaScript

Der zweite Einsatzbereich für Modernizr ist das bedingte Laden von JavaScript. Jeder Test, den Sie ausführen, erzeugt eine Eigenschaft für das `Modernizr`-Objekt mit den Werten `true` oder `false` für bedingte Anweisungen. In diesem Beispiel wird der in geschweiften Klammern stehende Code ausgeführt, wenn Flexbox unterstützt wird:

```
if(Modernizr.flexbox) { ... }
```

Wenn Sie dazu noch externe Ressourcen laden möchten, können Sie die `Modernizr.load()`-Methode verwenden. Wenn Sie bisher gut aufgepasst haben, wird sie ihnen vielleicht ein wenig vertraut vorkommen:

```
Modernizr.load({
```

```
    test: Modernizr.flexbox,
    nope: 'foo.js'
  });
```

Genau – sie verwendet YepNope.js als Grundlage. Alle Eigenschaften von YepNope können in Modernizr verwendet werden. Sie können `Modernizr.load()` alleine einsetzen oder, falls Sie bereits YepNope verwenden, das `Modernizr`-Objekt als Test nutzen:

```
yepnope({
  test: Modernizr.flexbox,
  nope: 'foo.js'
});
```

Beides funktioniert genauso. Was Sie bevorzugen, hängt von Ihrer Konfiguration ab. Doch egal, für welche Variante Sie sich entscheiden – mit Modernizr sind Sie ungemein flexibel bei der Entwicklung komplexerer Anwendungen mit eleganten Fallback-Lösungen.

Das in Modernizr verwendete Konzept der bedingten Formatierung ist so nützlich, dass es mit der @-Regel `@supports` Einzug in CSS selbst gefunden hat. `@supports` funktioniert ähnlich wie eine Medienabfrage, es werden damit aber keine Media Features, sondern CSS-EigenschaftWert-Paare abgeprüft. Um beispielsweise einen Browser auf Flexbox-Unterstützung zu testen, verwenden Sie folgende Regel:

```
@supports (display: flex) { ... }
```

Die Deklarationen in den geschweiften Klammern werden für jeden Browser angewendet, der für die `display`-Eigenschaft den Wert `flex` unterstützt. Mehr über `@supports` und die zugehörige API lesen Sie im Abschnitt »Feature Queries« auf Seite 251.

5.2.4 Mustache

Wenn Sie ohne Server-Backend entwickeln oder einfach nur schnell ein paar statische Vorlagen erstellen möchten, könnte ein clientseitiges Template-System für Sie interessant sein. Unter diesen sticht besonders

Mustache hervor, eine »Logik-lose«-Syntax, die ihre Popularität durch Portierungen in nahezu alle großen Webprogrammiersprachen bewiesen hat, darunter natürlich auch – und für unsere Zwecke besonders relevant – JavaScript.

Kein if, kein else, kein or

Mustache kommt ohne logische Ausdrücke aus – kein `if`, `else` oder `or`. Stattdessen gibt es ein System mit Tags, die durch Werte ersetzt werden können. Hier ist ein einfaches Beispiel, in dem ein einzelnes Tag durch einen Wert ersetzt wird:

```
var view = {
  Name: 'Bonobo'
}
var output = Mustache.render('The {{Name}} is funny.', view);
```

Sie sehen, dass ich ein Objekt namens `view` erzeugt habe. Es besitzt nur eine Eigenschaft, `Name`, und diese hat den Wert `'Bonobo'`. In der Variablen `output` verwende ich für die Tag-Ersetzung die Methode `Mustache.render()`. Diese Methode hat zwei Argumente. In diesem Beispiel ist das erste eine Zeichenkette, die die Eigenschaft `Name` in doppelten geschweiften Klammern enthält (oder Mustache = »Schnurrbärtchen«) enthält – dies ist die Syntax für ein Tag. Die darin enthaltene Variable, `key` genannt, wird durch den Eigenschaftswert mit demselben Namen aus dem Objekt, das das zweite Argument der Methode darstellt, ersetzt – in diesem Fall `view`. Als Ausgabe ergibt sich schließlich der Text `'The Bonobo is funny.'`, wie Sie in Beispiel 1 in der Datei *mustache.html* sehen können.

Sektionen

Mithilfe von Mustache können Sie auf einfache Weise sich wiederholende Inhaltsabschnitte erzeugen, die als *Sektionen* bekannt sind. Sektionen sind beispielsweise nützlich, um Wiederholungen beim Anlegen von Listen oder Tabellen zu vermeiden. Als ersten Schritt definieren Sie im `view`-Objekt die Eigenschafts-Werte-Paare:

```
var view = {
```

```
    'apes' : [
     {'Name':'Bonobo'},
     {'Name':'Chimpanzee'},
     {'Name':'Gorilla'},
     {'Name':'Orangutan'}
    ]
  }
```

Die Syntax einer Sektion lässt sich mit einem Tag vergleichen – zum Öffnen und Schließen der Tags kommen aber `#` bzw. `/` zum Einsatz. Im folgenden Code gebe ich die Sektion `apes` aus, die den zu wiederholenden Text oder HTML-Code enthält:

```
var templApe = '{{#apes}}The {{Name}} is funny.{{/apes}}';
```

Hier habe ich eine neue Variable namens `templApe` angelegt, welche die zu wiederholende Sektion enthält. Ich hätte dies auch direkt in `Mustache.render()` tun können, aber ich vereinfache mir die Aufgabe, indem ich meine Template-Regeln voneinander getrennt halte. Jedenfalls habe ich nun meinen Inhalt und mein Template, also gebe ich nun eine Liste aus, die behauptet, jeder der Menschenaffen sei lustig:

```
var output = Mustache.render(templApe,view);
```

Das Ergebnis sehen Sie in Beispiel 2 der Datei `mustache.html`.

Templates

Templates sind besonders nützlich, wenn Ressourcen von mehreren Seiten gemeinsam genutzt werden. Am besten ruft man hierzu eine externe Datei auf. In Mustache klappt das ziemlich einfach; Sie laden dazu lediglich die Ergebnisse einer JSON-Datei in die Variable `view`:

```
var request = new XMLHttpRequest();
request.open('GET', 'apes.json', true);
request.onreadystatechange = function () {
  if (request.readyState != 4) { return; }
  var view = JSON.parse(request.responseText);
❶ var templApe = '{{#apes}}The {{Name}} is funny.{{/apes}}';
❷ var output = Mustache.render(templApe,view);
};
```

```
request.send();
```

❶ und ❷ entsprechen im Wesentlichen dem vorhergehenden Beispiel; als einziger Unterschied dienen als Quelle von `view` nun Daten, die aus der Datei `apes.json` geparst wurden. Diese enthält dieselbe Information wie im letzten Beispiel, wird aber extern vorgehalten. Die Ausgabe sehen Sie in Beispiel 3 von *mustache.html*.

Wegen des zum Laden der externen Daten mit einem `XMLHttpRequest` erforderlichen Skripts sieht das Beispiel vielleicht kompliziert aus. Wenn Sie lieber jQuery verwenden, erreichen Sie mit weniger Zeilen dasselbe Ergebnis:

```
$.getJSON('json/apes.json', function (view) {
  var templApe = '{{#apes}}The {{Name}} is funny.{{/apes}}';
  var output = Mustache.render(templApe,view);
});
```

Die Funktion ist dieselbe, jedoch hat jQuery den `XMLHttpRequest()` und die JSON-Datei wegabstrahiert, wodurch alles deutlich vereinfacht und browserübergreifend kompatibel wird – das kann jQuery am besten.

Wenn Sie Templates nicht gerne im Inhalt, sondern lieber im Skript unterbringen möchten, dann können Sie auch Templates im Markup einsetzen. Nutzen Sie dazu Inline-`script`-Tags vom Typ `text/template`, in denen sich das Markup des Templates befindet. So könnte es zum Beispiel aussehen:

```
<div id="ape_area">
  <script id="apeTpl" type="text/template">
  <ul id="apes">
   {{#apes}}<li>The {{Name}} is funny.</li>{{/apes}}
  </ul>
  </script>
</div>
```

Das nächste Skript entspricht dem vorherigen Beispiel. Der Unterschied ist, dass Sie nun die ID des `script`-Tags, `#apeTpl` verwenden, um das Template zu deklarieren, und das Inline-Skript zum Zeitpunkt der Darstellung überschreiben, sodass es aus dem Markup entfernt wird:

```
$.getJSON('json/data.json', function (view) {
  var templApe = document.getElementById('apeTpl').innerHTML;
  var output = Mustache.render(templApe,view);
  document.getElementById('ape_area').innerHTML = output;
});
```

Beispiel 4 in `mustache.html` zeigt, wie es funktioniert. Es liegt an Ihnen, ob Sie Inline-Templates verwenden möchten. Mustache bietet in jedem Fall den Vorteil, Ihnen diese verschiedenen Alternativen zugänglich zu machen.

5.3 Polyfills und Shims

Codeschnnipsel für experimentelle Funktionen

Ein *Shim* (manchmal auch als *Shiv* bezeichnet) ist ein Codeschnipsel, der einen API-Aufruf abfängt und eine Abstraktionsebene bereitstellt. Ein *Polyfill* ist eine besondere Art von Shim. Dabei werden neue oder experimentelle Funktionen auch in solchen Browsern verfügbar gemacht, die dafür keine native Unterstützung bieten. Viele ältere Browser unterstützen zum Beispiel keinen CSS-Spaltensatz; ein Polyfill für den Spaltensatz würde diese Funktionalität für solche Browser also in JavaScript replizieren.

JavaScript-Abhängigkeiten

Dabei soll das Polyfill möglichst unsichtbar bleiben; der Autor muss nur noch einen Link auf die Polyfill-Datei setzen und sonst nichts (oder nur sehr wenig) tun. Im Fall des mehrspaltigen Textes sollte schon der CSS-Code für den Spaltensatz ausreichen. Natürlich entsteht bei so einer Umsetzung mehrspaltigen Textes in älteren Browsern eine Abhängigkeit von JavaScript. Also müssen Sie über eine gangbare Alternativlösung nachdenken – aber das sollte ohnehin zu Ihrem täglichen Arbeitsablauf gehören.

Ganz ehrlich, es gibt viel zu viele Polyfills, die viel zu viele Funktionen abdecken. Ich kann also nicht einmal daran denken, hier eine entsprechende Liste zu veröffentlichen. Im Modernizr-Wiki finden Sie

eine regelmäßig aktualisierte Liste, die zu Ihrer ersten Anlaufstelle werden sollte. Überlegen Sie sich einfach, für welche Funktion Sie ein Polyfill nutzen möchten, und sehr wahrscheinlich werden Sie es auf dieser Seite finden.

In der Tat sind Modernizr und YepNope perfekt geeignet, um Polyfills nur solchen Browsern bereitzustellen, die sie auch benötigen. Dadurch wird die Performance jener Browser verbessert, die diese Funktionen nativ unterstützen.

Um nochmals auf das Beispiel mit dem Mehrspaltensatz zurückzukommen – so könnten Sie das Polyfill nur für Browser ohne eigene Unterstützung für das Feature laden:

```
Modernizr.load({
  test: Modernizr.csscolumns,
  nope: 'css3-multi-column.js'
});
```

Performance

Polyfills wirken sich kritisch auf die Performance aus und sie schaffen eine Abhängigkeit von JavaScript. Sie sollten sich also sorgfältig mit den möglichen Nachteilen auseinandersetzen, bevor Sie sich für deren Nutzung entscheiden.

5.4 Testen und Debuggen

JS Bin und JS Fiddle

Wenn Sie Ihre Skripte in einer sicheren Umgebung testen und ausprobieren möchten, sind Ressourcen wie JS Bin und JS Fiddle ideal geeignet. Beide funktionieren im Wesentlichen gleich, aber da ich eher zu JS Bin tendiere, werde ich es in diesem Abschnitt behandeln.

JS Bin bietet Ihnen bis zu fünf Spalten, in denen Sie Ihre Skripte ausführen können: Die ersten drei Spalten sind für HTML, CSS und JavaScript, das Sie für Ihre Tests verwenden möchten; zur Ausgabe von Ergebnissen und Informationen steht eine Konsole zur Verfügung; ein

Ausgabebereich zeigt zudem die Ergebnisse der Tests.

Ein Drop-down-Menü enthält eine Reihe populärer JavaScript-Bibliotheken, und wenn Sie eine davon auswählen, wird sie automatisch im Head Ihres Beispielcodes eingebunden. Sie können auch beliebig Bibliotheken von Hand einbinden, wenn sie nicht in der Liste enthalten sind (wie etwa YepNope).

Die Möglichkeit, Ihre Tests mit Versionsnummern zu versehen und *Snapshots* (sogenannte »milestones«) aller Spalten anzulegen, vereinfacht die Verzweigung Ihrer Tests und das Ausprobieren unterschiedlicher Ansätze. Und das Beste ist, dass Sie Ihre Tests auch mit anderen Nutzern teilen können, die dann wieder in einer separaten Version ihre eigenen Änderungsvorschläge einbringen können. Die Zusammenarbeit wird also wirklich vereinfacht. Sie können Ihre Tests teilen, um die Ergebnisse zu präsentieren, und wenn Sie ein Benutzerkonto anlegen, können Sie ihre Tests dort speichern und später wieder einsehen.

5.5 Zusammenfassung

In diesem Kapitel habe ich die Grundlagen neuer DOM- und API-Funktionen in JavaScript notgedrungen im Schnelldurchlauf abgehandelt. Es sind einfach viel zu viele, um sie alle zu besprechen. Also habe ich mich auf die meiner Ansicht nach nützlichsten beschränkt. Dazu gehören die Funktionen, die das Laden von Skripten verbessern, die die Traversierung und Manipulation des DOM-Baums vereinfachen und die die Funktion von Ereignissen verbessern, besonders auf Touch-Geräten.

Ich habe auch einige populäre und nützliche JavaScript-Bibliotheken behandelt, von der Leistungsfähigkeit des allgegenwärtigen jQuery bis hin zum Nutzen von YepNope, Modernizr und Mustache. Ich empfehle Ihnen wirklich, Ihr Wissen über die vielen unterschiedlichen Bibliotheken zu vertiefen und die besten für sich selbst und Ihre Anwendung herauszufinden.

5.6 Literaturempfehlungen

Die *Illustration in Abb. 5–1* stammt aus Peter Beverloos Blog: http://peter.sh/experiments/asynchronous-and-deferred-javascript-execution-explained/.

Christian Heilmann hat eine fundierte *Einführung in JavaScript-Ereignisse* für das Smashing Magazine geschrieben: http://coding.smashingmagazine.com/2012/08/17/javascript-events-responding-user/. Die *PointerEvents-Bibliothek* finden Sie bei GitHub unter https://github.com/toolkitchen/PointerEvents/.

Die jQuery-Website http://jquery.com/ liefert *Hinweise für den Einstieg*, die ausgezeichnete Dokumentation finden Sie unter http://docs.jquery.com/Main_Page. Die statistischen Erhebungen zur Verwendung von jQuery entstammen dem Blogbeitrag *jQuery Now Runs on Every Second Website* unter http://w3techs.com/blog/entry/jquery_now_runs_on_every_second_

Alle *Bibliotheken für Mobilgeräte* sind vollständig dokumentiert: jQuery Mobile unter http://jquerymobile.com/, Zepto.js unter http://zeptojs.com/ und jQTouch unter http://jqtouch.com/.

YepNope.js finden Sie unter http://yepnopejs.com/ und eine gute *Einführung* unter http://net.tutsplus.com/tutorials/javascript-ajax/easy-script-loading-with-yepnope-js/.

Die Modernizr-Website http://modernizr.com/ bietet eine *vollständige Dokumentation* und ein *konfigurierbares Build-Tool*. Den Leitfaden *The All-In-One Entirely-Not-Alphabetical No-Bullshit Guide to HTML5 Fallbacks* (ich habe den Titel nicht erfunden) finden Sie unter https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills/.

Christopher Coenraets hat ein hervorragendes *Einstiegstutorial für Mustache* geschrieben, allerdings hat sich die Syntax geringfügig geändert: http://coenraets.org/blog/2011/12/tutorial-html-templates-with-mustache-js/. Die *vollständige Dokumentation zu Mustache.js* befindet sich hier: https://github.com/janl/mustache.js/.

Es gibt zahlreiche *Entwicklertools zum Experimentieren und Debuggen*, ganz hervorragend sind sowohl http://jsbin.com/ als auch http://jsfiddle.net/.

6 Geräte-APIs

In den vorhergehenden Kapiteln bin ich auf einige der zahlreichen APIs eingegangen, die als Teil von HTML5 eingeführt wurden, zum Beispiel Microdata und Touch Events. Es gibt aber noch eine Reihe weiterer APIs – sie sind zwar nicht Teil der Spezifikation, spielen aber dennoch eine wichtige Rolle. Diese APIs bieten für Entwickler im Multi-Screen-Umfeld eine ganz besondere Attraktion: den Zugriff auf das Gerät selbst.

In diesem Kapitel sehen wir uns einige Geräte-APIs an – angefangen von den neuen Möglichkeiten zur Ermittlung des Standorts und der Bewegung portabler Geräte bis hin zu Datei- und Speicheroptionen in den meisten modernen Browsern. Natürlich werden nicht alle APIs auf jedem Gerät zur Verfügung stehen – Sie müssen ja nicht unbedingt die Position eines Fernsehers im dreidimensionalen Raum kennen. Doch viele APIs sind für eine breite Palette von User Agents nützlich.

Ich habe eine Liste von APIs zusammengestellt, die meiner Meinung nach den größten praktischen Nutzen bringen werden. Ihre Vorstellung fällt aus Platzgründen teilweise recht knapp aus; die APIs sind häufig deutlich umfangreicher als von mir geschildert. Ich werde es anmerken, wenn für Sie aus meiner Sicht noch Spielraum vorhanden ist, mehr zu lernen – und ich empfehle Ihnen, unbedingt auch selbst die Möglichkeiten und Fähigkeiten des Gerätezugriffs über JavaScript zu ergründen.

Hinweis

Die Beispiele und Demonstrationen in diesem Kapitel sind interaktiv; in einigen Fällen habe ich Screenshots und Illustrationen beigefügt. Wenn es aber ein Kapitel gibt, für das Sie sich unbedingt die Beispieldateien herunterladen sollten, dann ist es dieses.

6.1 Geolokalisierung

Geolocation-API

Ortsabhängige Dienste sind in vielerlei Hinsicht praktisch – Mobilgerätenutzer finden besser ans Ziel und erhalten maßgeschneiderte Informationen über die Region, in der sie sich aufhalten. Die Geolocation-API greift auf die Ortungsdienste eines Geräts zu, die mittels GPS, WLAN oder Mobilfunkmast Informationen über den Standort des Geräts liefern. Diese können Sie dann wieder in Ihren standortbezogenen Apps verwenden.

Ganz offensichtlich ist das Thema Privatsphäre bei der Geolokalisierung nicht ganz unwichtig. In den meisten (wenn nicht gar allen) Browsern müssen die Anwender daher ihre ausdrückliche Zustimmung zum Zugriff auf diese Informationen geben. Meist erfolgt das über eine Bildschirmnachricht, in der sie sich dann für oder gegen die Preisgabe ihres Standorts entscheiden können (Abb. 6–1).

Abb. 6–1 *Chrome für Android fordert Ihre Zustimmung zur Geolokalisierung an.*

getCurrentPosition()

Die Daten befinden sich im `geolocation`-Objekt, einem Unterobjekt von `window.navigator`, auf das Sie über die `getCurrentPosition()`-Methode zugreifen können:

```
navigator.geolocation.getCurrentPosition(function(where){
  // Do something
});
```

Ein Callback liefert ein Objekt zurück (ich habe es *where* genannt), in dem sich ein `coords`-Unterobjekt befindet. Dieses Unterobjekt hat eine

Reihe von Eigenschaften, die die Nutzerposition betreffen, zum Beispiel die Höhe und Ausrichtung. Ich bin aber vor allem an den Koordinaten in `latitude` und `longitude` interessiert. Im folgenden Code greife ich auf diese Eigenschaften zu und zeige sie in einer Bildschirmmeldung an:

```
navigator.geolocation.getCurrentPosition(function(where){
  alert(where.coords.latitude + ',' + where.coords.longitude);
});
```

Sie können es selbst in der Datei *position-current.html* ausprobieren. Meine Ergebnisse sehen Sie in Abb. 6–2.

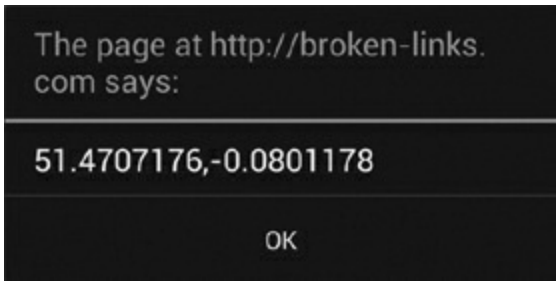


Abb. 6–2
Vom geolocation-Objekt gelieferte Koordinaten meiner Straße. Bitte stalken Sie mich jetzt nicht.

Hin und wieder wird bei der Positionsabfrage ein Fehler zurückgemeldet; ein optionales Fehler-Callback kann dies überprüfen. Der nachfolgende Code verwendet zwei Funktionen, eine für eine erfolgreiche Verortung und eine für den Fehlerfall:

```
var geo = navigator.geolocation,
   lcn_success = function(where) { ... },
   lcn_error = function() { ... };
geo.getCurrentPosition(lcn_success, lcn_error);
```

watchPosition()

Manchmal braucht ein GPS-Gerät einige Zeit, bis es die exakte Position des Anwenders bestimmt hat, und natürlich kann dieser auch in Bewegung sein. Statt sich auf eine einzelne Positionsangabe zu verlassen, können Sie die Position des Anwenders also auch wiederholt

abfragen und bei einem Ortswechsel die Daten entsprechend aktualisieren. Das erledigen Sie mit der `watchPosition()`-Methode, ebenfalls über das `geolocation`-Objekt. Sie funktioniert genauso wie `getCurrentPosition()`:

```
navigator.geolocation.watchPosition(function(where){
  console.log(where.coords.latitude,where.coords.longitude);
});
```

clearWatch()

Wenn Sie die Position des Anwenders nicht mehr überwachen möchten, verwenden Sie die `clearWatch()`-Methode mit dem von der `watchPosition()`-Methode erzeugten eindeutigen Bezeichner; in diesem Beispiel endet der Vorgang, sobald der Anwender den *#stop*-Link anklickt:

```
var geo = navigator.geolocation,
  watchID = geo.watchPosition( ... ),
  endWatch = document.getElementById('stop');
endWatch.addEventListener('click', function () {
  geo.clearWatch(watchID);
}, false);
```

In *position-watch-clear.html* sehen Sie eine Demonstration dieser Aktion. Öffnen Sie die Seite auf einem Mobilgerät und setzen Sie sich in Bewegung. Sobald das Gerät Ihre Position gut erfassen kann, sollten Sie die aktualisierten Positionsdaten sehen können.

6.2 Orientierung und Bewegung im Raum

Orientation-API

Die Orientation-API detektiert Änderungen der Geräteposition im dreidimensionalen Raum – also Bewegungen nach oben und unten, links und rechts und Drehungen im oder gegen den Uhrzeigersinn. Diese Bewegungen werden mit einem Beschleunigungssensor gemessen, der am ehesten in einem tragbaren Gerät verbaut ist: Mobiltelefone und Tablets werden häufig bewegt, also

besitzen sie wahrscheinlich einen solchen Sensor. Laptops werden auch hin und wieder bewegt, vielleicht ist also ebenfalls ein Beschleunigungssensor verbaut. Desktop-Computer und Fernseher werden hingegen so selten bewegt, dass sie wohl kaum einen Beschleunigungssensor oder Zugriff auf diese API bieten.

Durch den Einsatz von Orientation-Events öffnen sich ganz neue Möglichkeiten der Interaktion und Gerätesteuerung. In manchen Apps ist es bereits möglich, das Scrollen auf einer Seite durch Neigung des Geräts nach vorne oder hinten zu steuern, Titel oder Seiten lassen sich durch Neigung nach rechts oder links wechseln.

Exkurs:x-,y- und z-Achse

Ehe ich genauer auf die API eingehe, sollte ich die dreidimensionalen Achsen besprechen (Sie können diesen Absatz überspringen, wenn Sie bereits mit ihnen vertraut sind). Jede dreidimensionale Bewegung verläuft in drei Richtungen (oder entlang dreier *Achsen*), die meist mit x, y und z bezeichnet werden. Wenn Sie jetzt ein Gerät vor sich halten (oder sich dies vorstellen), dann verläuft die x-Achse von links nach rechts und die y-Achse von oben nach unten. Die z-Achse läuft auf Sie zu und von Ihnen weg, wie in Abb. 6–3 dargestellt.[1]

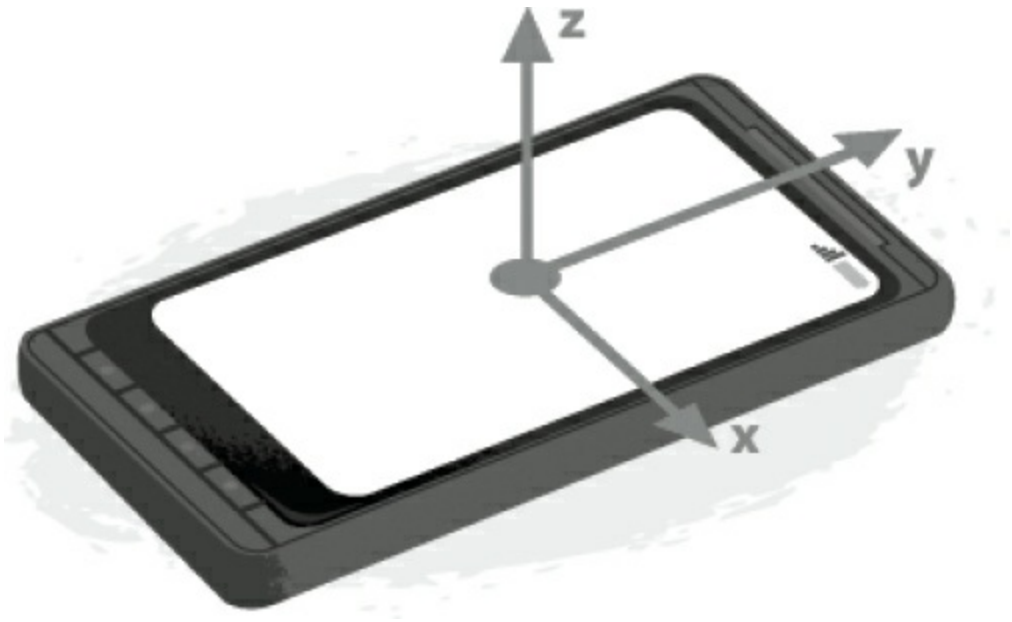


Abb. 6–3 *Bewegung entlang der dreidimensionalen Achsen*

Die Bewegung entlang dieser Achsen wird von der Gerätemitte aus gemessen, und sie kann entweder positiv oder negativ sein: Bringen Sie das Gerät näher zu Ihrem Körper, bewegt es sich entlang der positiven z-Achse; halten Sie es weiter von sich weg, entspricht die Bewegung einem negativen z-Wert. Bewegen Sie das Gerät nach unten in Richtung Ihrer Füße, verschieben Sie es negativ an der y-Achse entlang, eine Bewegung nach rechts folgt der positiven x-Achse.

deviceorientation

Um die Bewegung entlang jeder Achse zu erfassen, verwenden Sie das `deviceorientation`-Ereignis des `window`-Objekts. Dieses Ereignis wird ausgelöst, sobald sich das Gerät bewegt. Es liefert ein Objekt zurück, das eine Reihe nützlicher Ausrichtungseigenschaften mitbringt:

```
window.addEventListener('deviceorientation',function (orientData)
{
 ...
}, false);
```

Drehung um die Achsen

Die drei bewegungsbezogenen Schlüsseleigenschaften sind `alpha`, `beta` und

`gamma`. Jede wird als Zahlenwert gemessen, der
den Drehwinkel beschreibt; einige davon sind jedoch auf einen bestimmten Bereich beschränkt.

- `alpha` misst die Drehung um die z-Achse (nicht die Bewegung an der Achse entlang). Läge das Gerät flach auf dem Tisch, würde dies einer Drehung im Uhrzeigersinn oder entgegen des Uhrzeigersinns entsprechen. Der Wert für `alpha` ist eine Zahl zwischen 0 und 360.
- `beta` ist die Drehung um die *x*-Achse. Stellen Sie sich dazu vor, Sie neigen die Oberkante des Geräts zu sich hin oder von sich weg. `beta` hat einen Wertebereich von–180 (zu Ihnen hin geneigt) bis 180 (von Ihnen weg geneigt).
- `gamma` ist die Drehung um die *y*-Achse, die einer seitlichen Neigung des Geräts entspricht. Der Wert für `gamma` reicht von –90 (Neigung nach links) bis 90 (Neigung nach rechts).

Als sehr einfaches Beispiel spürt der Code im nachfolgenden Skript mit `deviceorientation` Änderungen in der Geräteausrichtung auf. Alle drei Werte werden dann in der Konsole ausgegeben:

```
window.addEventListener('deviceorientation',function (orientData)
{
  console.log(orientData.alpha,orientData.beta,orientData.gamma);
}, false);
```

In der Beispieldatei *orientation.html* sehen Sie eine leicht abgeänderte Version, die beim Ändern der Geräteausrichtung den Seitentext aktualisiert. Öffnen Sie sie auf einem Mobilgerät oder Tablet und bewegen Sie das Gerät umher, damit die Seite aktualisiert wird.

6.3 Vollbild

Wie wir alle wissen, ist das Web eine gewaltige Ablenkungsmaschinerie. Manchmal ist es daher hilfreich, eine Option anzubieten, um sich nur auf den gegebenen Inhalt zu konzentrieren. Diese Möglichkeit bietet die Fullscreen-API, mit deren Hilfe Sie ein beliebiges Element auf den

gesamten Bildschirm des Geräts ausweiten können, statt es nur im Darstellungsbereich des Browsers anzuzeigen. Das ist bei Geräten mit großem Bildschirm besonders nützlich. Sie können etwa bei der Videowiedergabe ein echtes »Fernsehgefühl« vermitteln.

fullScreenEnabled

Ehe Sie dieses Skript einrichten, überprüfen Sie, ob der Browser einen Vollbildmodus hat. Das gelingt mit dem booleschen Attribut `full-ScreenEnabled`:

```
if (document.fullScreenEnabled) { ... }
```

requestFullScreen()

Der Vollbildmodus wird mit der Methode `requestFullScreen()` aufgerufen. Dies stellt ein potenzielles Sicherheitsrisiko dar (als Beispiel könnte Ihnen ein bösartiges Skript eine Website vorgaukeln und dann Ihre Tastatureingaben aufzeichnen). Daher bieten viele Geräte eine Sicherheitsabfrage, die sicherstellt, dass Sie Ihre Erlaubnis zum Wechsel in den Vollbildmodus geben. Gewähren Sie diese, wird das Element, für das die Methode aufgerufen wurde, auf 100% der Bildschirmhöhe und -breite des Geräts vergrößert.

Im nächsten Codeabschnitt wird ein Click-Event-Listener auf das Element *#trigger* angewandt. Wenn es auslöst, versetzt es *.target* in den Vollbildmodus, falls die erforderliche Erlaubnis dafür vorhanden ist. Sehen Sie es sich selbst in der Datei *fullscreen.html* an, zu der auch Abb. 6–4 gehört.

```
var el = document.querySelector('.target'),
   launch = document.getElementById('trigger');
launch.addEventListener('click', function () {
  el.requestFullScreen();
}, false);
```

exitFullScreen()

Der Browser sollte eine Möglichkeit zum Verlassen des Vollbildmodus anbieten, Sie können jedoch mit der `exitFullScreen()`-Methode auch selbst eine definieren. Der nächste Codeabschnitt zeigt eine Funktion, die den Vollbildmodus mithilfe dieser Methode beim

Drücken der `Enter`-Taste beendet. Beachten Sie im Code zwei weitere Dinge: Zunächst sehen Sie das `fullscreenchange`-Ereignis, das immer dann auslöst, wenn ein Element in den Vollbildmodus versetzt wird oder wenn es diesen verlässt. Zweitens kommt ein `if`-Ausdruck mit `fullScreenElement`-Attribut zur Anwendung. Dieser liefert entweder Informationen über das im Vollbildmodus befindliche Element zurück – oder `null`, falls sich kein Element im Vollbildmodus befindet.

```
document.addEventListener('fullscreenchange', function () {
  if (document.fullScreenElement !== null) {
    document.addEventListener('keydown', function (e) {
     if (e.keyCode === 13) {
      document.exitFullScreen();
    }
   }, false);
 }
}, false);
```

Abb. 6–4
Ein in den Vollbildmodus versetztes Element mit entsprechender Warnmeldung auf dem Bildschirm in Firefox für Android

Für die Vollbildanzeige eines Elements möchten Sie vielleicht dessen Stil (oder den seiner Unterelemente) ein wenig verändern. Ein Vorschlag

lautet, dies mit einer neuen, speziell dafür vorgesehenen CSS-Pseudoklasse zu realisieren, die entweder `:fullscreen` oder `:full-screen` genannt wird:

```
.target:full-screen {}
```

6.4 Vibration

Die Vibration-API lässt ein Gerät vibrieren und liefert dem Anwender damit eine haptische Rückmeldung. Diese API hieß ursprünglich Vibrator-API – ich muss Ihnen sicher nicht erklären, warum der Name sehr schnell wieder geändert wurde. Natürlich können nicht alle Geräte vibrieren, besonders größere nicht, und diese API ist daher eindeutig eher für Mobilgeräte von Nutzen.

vibrate()

Sie ist extrem einfach gehalten. Sie benötigt nur die vibrate()-Methode für das navigator-Objekt. Der Wert für vibrate() ist eine Zahl, die die Vibrationsdauer des Geräts in Millisekunden angibt. Damit das Gerät zum Beispiel eine Fünftelsekunde lang vibriert, nachdem der Anwender ein touchend-Event abgeschlossen hat, verwenden Sie folgenden Code:

```
document.addEventListener('touchend', function () {
  window.navigator.vibrate(200);
});
```

Sie können auch eine Abfolge von Werten verwenden, die das Gerät zwischen Vibration und Vibrationspausen umschalten; die Werte an ungeraden Positionen sind dann die Vibrationen, die geradzahlig platzierten Werte die Pausen. In diesem Beispiel vibriert das Gerät für 200 ms, macht 200 ms Pause und vibriert dann für 500 ms:

```
document.addEventListener('touchend', function () {
  window.navigator.vibrate([200,200,500]);
});
```

Batterieverbrauch!

Vibration erhöht den Batterieverbrauch, also setzen Sie diese API mit Bedacht ein. Sie können eine Vibration von Hand stoppen, indem Sie eine 0 oder einen leeren Zahlenwert verwenden. In diesem Code vibriert das Gerät 5 Sekunden lang, wenn das Touch-Ereignis beginnt, und hört dann auf, wenn das Ereignis endet:

```
document.addEventListener('touchstart', function () {

window.navigator.vibrate(5000);
});
document.addEventListener('touchend', function () {
  window.navigator.vibrate(0);
});
```

Probieren Sie die API selbst in der Datei *vibration.html* aus. Natürlich benötigen Sie dazu ein Mobilgerät mit Vibrationsmotor, wenn Sie die Vibrationen auch tatsächlich spüren möchten. Falls Sie keines zur Hand haben, verdeutlicht Abb. 6–5 diese Erfahrung.

Abb. 6–5 *Die Vibration-API in Aktion*

6.5 Batteriestand

Battery-Status-API

Von besonderer Bedeutung beim Umgang mit modernen Geräten ist die Kenntnis des Akkuladezustands. Mobilgeräte holen dürftige sieben oder acht Stunden Laufzeit aus einem vollen Akku heraus, bei

einem Laptop kann man sich schon über drei bis vier Stunden freuen. Es kann wichtig sein, den Batteriestand zu kennen, ehe Sie leistungshungrige Prozesse starten oder mit dem Herunterladen großer Dateien beginnen.

charging

Informationen über den Batteriestand erhalten Sie mit der Battery-Status-API, die im `navigator.battery`-Objekt eine Reihe von Attributen mitbringt. Möchten Sie beispielsweise herausfinden, ob die Batterie derzeit geladen wird, können Sie das `charging`-Attribut verwenden, das als Wert entweder `true` oder `false` zurückliefert:

```
var batteryStatus = navigator.battery.charging;
```

level

Zur Abfrage des aktuellen Batteriestands verwenden Sie das `level`-Attribut, das einen Wert von 0 (leer) bis 1 (voll geladen) zurückgibt. Der nachfolgende Code ist eine einfache Demonstration dieser Aktion: Der Batteriefüllstand wird ausgelesen und sein Wert als Wert für ein `meter`-Element genutzt (das im Kapitel 8 vollständig vorgestellt wird). Darunter wird zudem der aktuelle Ladestatus (»*Charging*« oder »*Discharging*«) ausgegeben. In der Beispieldatei *battery.html* können Sie es selbst ausprobieren.

```
var el = document.getElementById('status'),
   meter = document.querySelector('meter'),
   battery = navigator.battery,
   status = (battery.charging) ? 'Charging' : 'Discharging';
meter.value = battery.level;
meter.textContent = battery.level;
el.textContent = status;
```

Das Ergebnis sehen Sie in Abb. 6–6.

Abb. 6–6

Ein meter-Element zeigt den verbleibenden Batteriezustand sowie den Ladestatus meines Geräts.

chargingTime und dischargingTime

Das `battery`-Objekt hat zwei weitere Attribute: `chargingTime` und `dischargingTime`. Beide liefern einen Wert in Sekunden zurück, der die verbleibende Zeit bis zur vollständigen Ladung beziehungsweise Entla-dung der Batterie angibt.

Ereignisse

Die Battery-Status-API verfügt außerdem über eine Reihe von Ereignissen, die ausgelöst werden, wenn sich eines der Attribute verändert: `chargingchange`, `chargingtimechange`, `dischargingtimechange` und `levelchange`. Der nachfolgende Code verwendet `chargingchange`, um eine Veränderung des Ladestatus des Geräts zu überwachen. Ändert er sich, wird eine Warnmeldung ausgegeben:

```
var status,
    battery = navigator.battery,
    chargeStatus = function () {
      (battery.charging) ? status = 'Charging' : status =
'Discharging';
      return status;
};
battery.addEventListener('chargingchange', function (){
  window.alert(chargeStatus());
}, false);
window.alert(chargeStatus());
```

Sie können es selbst in der Datei *battery-event.html* ausprobieren – stecken Sie Ihr Handy ans Ladegerät und ziehen Sie es wieder ab, um die Statusaktualisierung zu erhalten.

6.6 Verbindungsinformation

Network-Information-API

Die Kenntnis der aktuellen

Verbindungsqualität eines Geräts mit dem Internet ist äußerst nützlich: So könnten Sie für Geräte mit wenig Bandbreite Bilder mit geringerer Auflösung bereitstellen oder Anwendern abhängig von ihrer Internetverbindung unterschiedliche Videostreams anbieten. Ebenso möchten Sie vielleicht die Ausführung von Hintergrundprozessen aufschieben, wenn der Anwender gerade einen Tarif mit Volumenbegrenzung oder -abrechnung nutzt.

bandwidth und metered

Die Network-Information-API besteht aus zwei Attributen für das `connection`-Objekt: Das erste ist `bandwidth`, eine Zahl, die die *geschätzte* Bandbreite der aktuellen Verbindung in Megabytes/Sekunde (MBps) angibt (der Wert ist 0, wenn das Gerät offline ist, und `infinity`, wenn das Ergebnis unbekannt ist). Das zweite Attribut wird `metered` genannt und ist noch ein boolescher Parameter – er liefert `true` zurück, wenn der Tarif nach Volumen abgerechnet wird (wie bei vielen Prepaid-Tarifen der Fall).

Der nachfolgende Code zeigt eine Funktion, die beide Attribute verwendet: `bandwidth`, um die Bandbreite der aktuellen Verbindung zurückzuliefern, und `metered`, um eine zusätzliche Statusmeldung auszugeben, falls es sich um keinen Flatrate-Tarif handelt.

```
var status,
   connection = navigator.connection,
   showStatus = function () {
    status = connection.bandwidth + ' MB/s';
    if (connection.metered) {
     status += ' (metered)';
    }
  alert(status);
   };
  showStatus();
```

Die Network-Information-API hat zudem einen Event-Handler, `change` für das `connection`-Objekt. Dieser löst aus, sobald sich der Verbindungsstatus verändert. Hierdurch können Sie bei Bedarf sehr einfach einen zusätzlichen Funktionsaufruf hinzufügen:

```
connection.addEventListener('change', showStatus, false);
```

In der Datei *network.html* können Sie beides im Einsatz sehen. Probieren Sie, sich mit Ihrem WLAN-Netz zu verbinden und die Verbindung wieder zu trennen, um das `change`-Event auszulösen.

6.7 Kamera und Mikrofon

Kameras und Mikrofone an Desktop- und Laptop-Computern gibt es schon lange, und mit dem Aufkommen der Mobilgeräte sind sie inzwischen äußerst verbreitet – fast schon allgegenwärtig. Jedoch mussten wir jahrelang auf Plugins von Drittanbietern zurückgreifen – etwa Flash und Java –, um Audio- und Videoeingaben im Web zu erhalten. Eine native Eingabemethode ist also schon längst fällig.

getUserMedia()

Diese kommt nun in Form der `getUserMedia()`-Methode daher. Sie ist Teil des in Kapitel 9 noch näher erläuterten WebRTC-Projekts. Die `getUserMedia()`-Methode des `navigator`-Objekts nimmt bis zu drei Argumente entgegen: Das erste für Optionen zur Stream-Ausgabe: Werden nur Audio oder Video oder beides zugelassen? Die zweite ist ein Rückruf, der beim erfolgreichen Verbindungsaufbau ausgelöst wird. Das dritte (optionale) Argument ist eine Rückruffunktion zur Fehlerbehandlung:

```
navigator.getUserMedia({options}, success, failure);
```

Wie schon die Nutzung der Geolocation- und Fullscreen-APIs stellt auch der Zugriff auf Kamera und Mikrofon des Anwenders einen Eingriff in die Privatsphäre dar. Viele Browser holen daher die Erlaubnis des Nutzers für den Gerätezugriff per Bildschirmabfrage ein. Auf Geräten mit mehreren Kameras bieten manche User Agents einen nativen Umschalter zwischen beiden an.

Gestreamte Medien benötigen zur Anzeige ein spezielles Element, entweder das neue `video`- oder das `audio`-Element in HTML5 (je nach

Inhalt des Streams). Ich stelle diese neuen Elemente in Kapitel 9 ausführlich vor. Für die nachfolgende Demonstration, in der ein Videostream eingesetzt wird, müssen Sie zunächst einmal irgendwo auf der Seite dieses Markup unterbringen:

```
<video autoplay></video>
```

Wenn der erfolgreiche Callback von `getUserMedia()` ausgelöst wird, wird der Medienstram mit (von Ihnen vorgegebenem) eindeutigem Bezeichner zurückgeliefert. Dieser wird dann dem `video`-Element zugeführt. Der nachfolgende Code zeigt ein einfaches Beispiel, das ich dann gleich erkläre:

```
❶ navigator.getUserMedia({video:true}, function (stream) {
❷ var video = document.querySelector('video');
❸ video.src = window.URL.createObjectURL(stream);
  });
```

In Zeile ❶ liefere ich der `getUserMedia()`-Methode zwei Argumente: Das erste sind die Stream-Optionen, und ich gebe hier an, dass ich nur Video und kein Audio erfassen möchte. Das zweite ist die Rückruffunktion, in der ich dem Ergebnis die eindeutige ID *stream* zuweise. In der nächsten Zeile ❷ habe ich `querySelector()` verwendet, um das `video`-element der *video*-Variablen zuzuweisen, damit ich in Zeile ❸ mithilfe der `createObjectURL()`-Methode *stream* in eine URL umwandeln und als `src`-Attribut des `video`-Elements einsetzen kann. Hier gibt es keine Rückruffunktion zur Fehlerbehandlung.

Um dies selbst auszuprobieren, sehen Sie sich die Datei *getusermedia.html* an – Sie brauchen ein Gerät mit Videokamera, um die Datei in Aktion zu erleben.

6.8 Web Storage

Informationen über vorhergehende Aktivitäten werden meist mithilfe

Web-Storage-API

von Cookies aufgezeichnet. Einer der Nachteile dabei ist, dass Sie nur relativ wenige Daten speichern können. Die Web-Storage-API wurde erstellt, damit User Agents mehr Daten auf dem Gerät des Anwenders speichern können. Diese Daten werden bis zum Schließen des Browsers behalten (*session storage*), oder bis der Anwender oder ein Skript sie aktiv löscht (*local storage*). Beide Speicheroptionen funktionieren im Wesentlichen gleich, bis auf einen entscheidenden Unterschied: die Dauerhaftigkeit.

localStorage und sessionStorage

Die Daten werden als Schlüssel-Wert-Paare gespeichert, genau wie Sie es bereits von Cookies her kennen. Allerdings kann jetzt eine größere Datenmenge gespeichert werden. Die API verfügt über zwei entscheidende Objekte, die einfach zu verstehen und merken sind: `localStorage` für die lokale Speicherung und `sessionStorage` für die sitzungsbezogene Speicherung.

Hinweis

In den Beispielen dieses Abschnitts verwende ich `sessionStorage`; Sie können dies jedoch mit `localStorage` austauschen, falls Sie eine dauerhaftere Speicherung bevorzugen. Die Syntax bleibt dabei gleich.

setItem()

Die Syntax für Web Storage ist recht flexibel, es gibt drei verschiedene Möglichkeiten, ein Element zu speichern: mit der `setItem()`-Methode, über die Notation mit eckigen Klammern oder mit Punkt. Als einfaches Beispiel zeigt der nächste Codeschnipsel, wie Sie den Namen des Autors speichern könnten. Zum Vergleich präsentiere ich die drei verschiedenen Speicherungsmöglichkeiten, die alle einwandfrei und gültig sind.

```
sessionStorage.setItem('author','Peter Gasston');
sessionStorage['author'] = 'Peter Gasston';
sessionStorage.author = 'Peter Gasston';
```

Mithilfe einiger Entwicklertools können Sie die gespeicherten Inhalte untersuchen. Abb. 6–7 zeigt das Ergebnis dieses Codes, unabhängig von der Variante, für die Sie sich entscheiden.

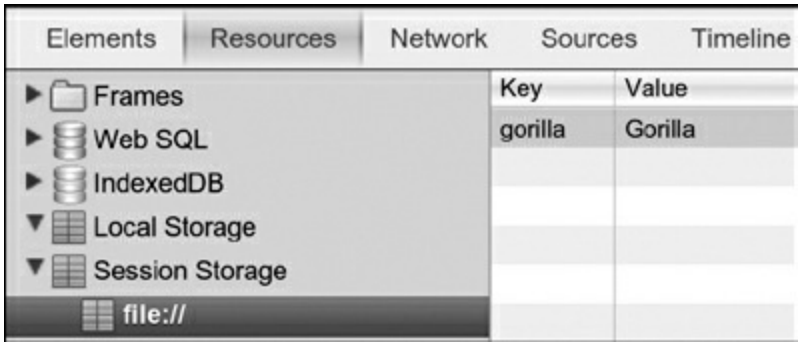


Abb. 6–7
Ein im Browser gespeichertes Schlüssel-Wert-Paar wird im WebKit Web Inspector angezeigt.

getItem()

Das Abrufen der gespeicherten Daten geht ebenso flexibel; Sie können die `getItem()`-Methode verwenden, die nur den Namen des entsprechenden Schlüssels als Argument akzeptiert, oder die Notation mit eckigen Klammern oder Punkt ohne zusätzlichen Wert. Im nächsten Codebeispiel sind die drei gleichwertigen Techniken dargestellt:

```
var author = sessionStorage.getItem('author');
var author = sessionStorage['author'];
var author = sessionStorage.author;
```

Hinweis

Ich speichere hier in diesen Beispielen nur sehr einfache Werte. In den meisten Browsern können Sie aber bis zu 5 MB Daten je Subdomain speichern. Dieser Wert wird in der Spezifikation empfohlen, er ist aber nicht bindend.

removeItem()

Ein einzelnes gespeichertes Element entfernen Sie mit der `removeItem()`-Methode.

Genau wie `getItem()` akzeptiert auch sie einen
Wert als Argument und löscht das entsprechende Element mit passendem Schlüssel aus dem Speicher:

```
sessionStorage.removeItem('author');
```

In der Datei *storage.html* habe ich eine einfache Demonstration zusammengestellt, die Elemente in den Speicher schreibt und wieder daraus entfernt. Damit Sie das Ergebnis sehen können, brauchen Sie Entwicklerwerkzeuge, die den Inhalt des Speichers anzeigen, wie etwa im Ressourcen-Register des WebKit Web Inspector. Die Inhalte werden nicht in Echtzeit aktualisiert. Sie müssen die Anzeige also manuell aktualisieren, um die Änderungen zu sehen.

clear()

Zur gleichzeitigen Löschung aller gespeicherten Daten (allerdings natürlich nur für die entsprechende Domain, für die sie gespeichert werden) nutzen Sie die `clear()`-Methode:

```
sessionStorage.clear();
```

Ereignisse

Ein `storage`-Ereignis wird für `localStorage` ausgelöst, sobald sich etwas am Speicherinhalt ändert. Dadurch wird ein Objekt zurückgeliefert, das einige nützliche Eigenschaften hat: etwa `key`, das den Namen des veränderten Schlüssels enthält, `oldValue` und `newValue` mit den alten und neuen Werten des veränderten Elements. Beachten Sie, dass dieses Ereignis nur für andere geöffnete Instanzen derselben Domain (Tabs oder Fenster) ausgelöst wird und nicht für die aktive Instanz. Der Zweck besteht darin, Veränderungen zu überwachen, falls der Anwender beispielsweise mehrere Tabs geöffnet hat.

Der nächste Codeabschnitt enthält eine Funktion, die ausgelöst wird, sobald sich der Speicher verändert. Daraufhin wird ein Eintrag in der Konsole vorgenommen. Sie können es selbst in der Datei *storage-event.html* ausprobieren. Allerdings müssen Sie die Datei und die Entwicklerkonsole in zwei unterschiedlichen Tabs öffnen, damit Sie die Veränderungen beobachten können. Denken Sie daran, dass

Veränderungen im anderen Fenster dargestellt werden und nicht in dem, wo geklickt wurde.

```
window.addEventListener('storage', function (e) {
  var msg = 'Key ' + e.key + ' changed from ' + e.oldValue + ' to
' +
  e.newValue;
  console.log(msg);
}, false);
```

IndexedDB-API

Die Speicherung wird mit der Entwicklung der Indexed-Database-(IndexedDB-)API sogar noch weiter getrieben. Mit dieser API wird die Erstellung einer kompletten Datenbank zur Speicherung im Browser angestrebt, die dann über JavaScript zugänglich ist. Viele Browser haben hierzu bereits einen Versuch unternommen; die Hersteller konnten sich jedoch nicht auf ein einheitliches Format einigen. IndexedDB ist ein unabhängig entwickelter Standard, der alle glücklich machen soll. Dieses sehr technische Thema kann ich im vorliegenden Buch nicht behandeln. Behalten Sie es aber im Hinterkopf, falls Sie komplexere Speichermöglichkeiten benötigen.

6.9 Drag & Drop

Eine Website zum »Anfassen«, auf der die Benutzer Elemente über den Bildschirm schieben können, ist eine nette Option. Dieses »Drag & Drop«-Verhalten ist besonders auf Geräten mit Touch-Bedienung vorteilhaft.

Die Drag-and-Drop-API ist wahrscheinlich das älteste Feature, das ich in diesem Buch behandle. Die erste Implementierung fand sich 1999 im Internet Explorer 5 (also vor rund 150 Internetjahren). Andere Browser folgten, Anstrengungen zur Standardisierung gab es aber erst im Zuge der HTML5-Bewegung. Leider ist Drag and Drop inzwischen etwas angestaubt, und zu Beginn wirkt es recht undurchsichtig und wenig intuitiv.

a- und img-Elemente verschieben

Standardmäßig können die `a`- und `img`-Elemente auf dem Bildschirm verschoben werden (auf weitere Elemente gehe ich gleich noch ein). Sie müssen allerdings eine *Drop Zone* einrichten, also einen Bereich, in den die Elemente gezogen werden können. Sie entsteht durch Anhängen der beiden Ereignisse `dragover` und `drop` an ein Element. Bei `dragover` ist es lediglich erforderlich, sein Standardverhalten zu unterdrücken (aus einem der obskuren Gründe, die ich vorher angesprochen hatte, um die Sie sich aber nicht kümmern müssen). Die harte Arbeit übernimmt dann das `drop`-Ereignis.

Weil das eventuell etwas verwirrend klingt, sehen Sie sich am besten dieses einfache Beispiel an: Dem *#target*-Element werden die Event-Listener `dragover` und `drop` angehängt, die Rückruffunktion `dragover` unter-drückt mit `preventDefault()` das Standardverhalten und die eigentliche Aktion findet in der Rückruffunktion des `drop`-Events statt.

```
var target = document.getElementById('target');
target.addEventListener('dragover', function (e) {
  e.preventDefault();
}, false);
target.addEventListener('drop', function (e) {
  // Do something
}, false);
```

Alle Ereignisse in der Drag-and-Drop-API erzeugen ein Objekt namens `dataTransfer`, das über eine Reihe wichtiger Eigenschaften und Methoden verfügt. Diese sind für Sie interessant, wenn das `drop`-Event ausgelöst wird. Bei `img`-Elementen ist die URL des Elements wichtig. Sie verwenden also die Methode `getData()` mit dem Wert *URL*; anschließend können Sie damit weiterarbeiten. In diesem Beispiel erzeuge ich ein neues `img`-Element und übergebe die URL an das `src`-Attribut, wobei ich eine Kopie davon mache:

```
target.addEventListener('drop', function (e) {
  e.preventDefault();
  var newImg = document.createElement('img');
  newImg.setAttribute('src', e.dataTransfer.getData('URL'));
  e.currentTarget.appendChild(newImg);
```

```
}, false);
```

Beachten Sie wieder den Einsatz von `preventDefault()` innerhalb der `drop`- Rückruffunktion. Dies ist sehr wichtig, weil die meisten (wenn nicht sogar alle) Browser nach dem Ablegen eines Elements in einer Drop Zone standardmäßig versuchen, ihre URL zu öffnen. Das gehört zu den Undurchsichtigkeiten im Verhalten von Drag and Drop. Sie brauchen sich nur zu merken, dass Sie dieses Verhalten mit `preventDefault()` erfolgreich unterbinden können.

Ein einfaches Beispiel auf der Grundlage des vorhergehenden Codes finden Sie in der Datei *drag-drop.html* – ziehen Sie das Bild lediglich von seiner Startposition in den Kasten.

Weitere Elemente verschieben

Ich hatte zuvor erwähnt, dass standardmäßig nur `a`- und `img`-Elemente verschoben werden können. Sie können dieses Verhalten aber in zwei Schritten für alle Elemente freigeben. Zunächst wenden Sie den Wert `true` auf das `draggable`-Attribut des fraglichen Elements an:

```
<div draggable="true" id="text">Drag Me</div>
```

Zweitens geben Sie einen Datentyp für das Element an. Das erledigen Sie mit dem `dragstart`-Ereignis. Verwenden Sie die `setData()`-Methode von `dataTransfer`, um einen MIME-Typ (in diesem Fall `text/plain`) und einen Wert (in diesem Fall den Textinhalt des Elements) anzuwenden:

```
var txt = document.getElementById('txt');
  txt.addEventListener('dragstart', function (e) {
  e.dataTransfer.setData('text/plain',
e.currentTarget.textContent);
}, false);
```

Den Typ der abgelegten Datei können Sie mit der Methode `contains()` abfragen. Sie ist ein Unterelement des `types`-Objekts, das selbst wieder ein Unterelement des in der Rückruffunktion des `drop`-Ereignisses angelegten `dataTransfer`-Objekts ist. Die Methode liefert `true` oder `false` zurück, wenn die Zeichenkette im Argument zu einem Wert in `types`

passt. Um beispielsweise zu ermitteln, ob ein abgelegtes Element dem Typ »einfacher Text« entspricht, würden Sie so vorgehen:

```
var foo = e.dataTransfer.types.contains('text/plain');
```

Mithilfe der `contains()`-Methode können Sie unterschiedliche Aktionen auf unterschiedlichen Dateitypen ausführen.

Die Beispieldatei *drag-drop-2.html* zeigt zwei Elemente, ein `img`-Element und ein `p`-Element. Werden sie in die markierte Drop Zone gezogen, entsteht jeweils eine Kopie. Der nachfolgende Code zeigt, wie es funktioniert: Die `contains()`-Methode ermittelt, ob das gezogene Element eine URL beinhaltet. Ist dies der Fall, muss es sich um ein `img` handeln, also wird ein neues `img`-Element mit der URL des abgelegten Elements im `src`-Attribute erstellt. Ist keine URL enthalten, muss es sich um Text handeln, also wird ein neuer Textknoten angelegt und mit dem Text des abgelegten Elements gefüllt.

```
target.addEventListener('drop', function (e) {
  var smth;
  e.preventDefault();
  if (e.dataTransfer.types.contains('text/uri-list')) {
    smth = document.createElement('img');
    smth.setAttribute('src', e.dataTransfer.getData('URL'));
  } else {
    smth =
document.createTextNode(e.dataTransfer.getData('Text'));
  }
  e.currentTarget.appendChild(smth);
}, false);
```

Die Erläuterungen in diesem Abschnitt sind mehr als ausreichend, um Ihnen die Nutzung der Drag-and-Drop-API zu ermöglichen. Allerdings ist die API noch deutlich umfangreicher. Falls Sie interessiert sind – es gibt noch eine Reihe zusätzlicher Ereignisse: `dragenter` und `dragleave` sind Ereignisse für die Drop Zone, und `dragend` und `drag` lösen für das gezogene Element aus.

6.10 Mit Dateien arbeiten

File-API

Häufig wird mit verschiedenen Dateien gearbeitet – insbesondere auf Desktop- und Laptop-Computern, weniger auf Mobilgeräten. Daher gibt es auch eine API für die Arbeit mit Dateien im Web. Die File-API ist ziemlich einfach aufgebaut. Sie erhalten damit Informationen über Dateien und können auf deren Inhalte zugreifen. Es gibt zusätzlich einige komplexere APIs, die ich in diesem Zusammenhang noch erwähnen werde.

Drag-and-Drop-API

Zum Zugriff auf Dateien können Sie diese entweder mit dem `file`-Eingabeelement auswählen oder sie mit der Drag-and-Drop-API aus einem Verzeichnis auf Ihrem System (in Abhängigkeit vom verwendeten System) herausziehen. Diesen Ansatz werden wir uns hier ansehen.

Das `dataTransfer`-Objekt, das ich erst im letzten Abschnitt besprochen habe, enthält ein `files`-Unterobjekt mit einer Liste aller in der Drop Zone abgelegten Dateien. Jede Datei hat drei Eigenschaften – `name`, `size` und `type` –, und deren Bedeutung sollte recht offenkundig sein.

Der folgende Beispielcode zeigt eine Funktion, die die Namen der in der Drop Zone abgelegten Dateien auflistet. Das gelingt Ihnen mit einer `for`-Schleife, die das `files`-Objekt durchläuft und jedes Mal die `name`-Eigenschaft ausgibt. Probieren Sie es selbst in der Beispieldatei *files.html* aus.

```
target.addEventListener('drop', function (e) {
  var files = e.dataTransfer.files,
      fileNo = files.length;
  e.preventDefault();
  for (i = 0; i < fileNo; i++) {
    var el = document.createElement('li'),
      smth = document.createTextNode(files[i].name);
    el.appendChild(smth);
    e.currentTarget.appendChild(el);
  }
```

```
}, false);
```

Wenn Sie mehr als reine Dateiinformationen benötigen, erhalten Sie mit der `FileReader`-Schnittstelle den Inhalt als Textdatei oder Data-URL (so vorhanden). Der nachfolgende Codeabschnitt zeigt ein einfaches Beispiel mit einer Bilddatei als Quelle. Weil die Syntax etwas kompliziert ist, habe ich den Code mit Anmerkungen versehen und werde ihn nachfolgend erklären.

```
target.addEventListener('drop', function (e) {
  e.preventDefault();
  var files = e.dataTransfer.files[0],
❶ reader = new FileReader();
❷ reader.addEventListener('load', function (evt) {
      var img = document.createElement('img');
❸ img.src = evt.target.result;
      target.appendChild(img);
   }, false);
❹reader.readAsDataURL(files);
❺reader.addEventListener('error', function (evt) {
       console.log(evt.target.error.code)
     }, false);
  }, false);
```

In ❶ wird ein neues `FileReader`-Objekt erzeugt und der Variablen *reader* zugewiesen. Zu diesem Objekt wird ein neuer Event-Listener hinzugefügt ❷. Dieser wird ausgelöst, wenn die Datei fertig geladen wurde. Dadurch wird eine Funktion ausgeführt, die ein neues `img`-Element mit dem Inhalt der hochgeladenen Datei anlegt. Die Quelle (`src`) für das `img`-Element wird in ❸ mithilfe des `result`-Attributs von `target`, einem Unterobjekt des Ereignisses, erhalten. Der Ergebnistyp wird in ❹ mithilfe der `readAsDataURL()`-Methode festgelegt, die den Dateiinhalt als 64-Bit-Datenstring codiert. Schließlich wird dem Objekt in ❺ ein Event-Listener zur Fehlerbehandlung hinzugefügt, der das `code`-Attribut des `error`-Objekts des `target`-Objekts des `event`-Objekts (puh!) verwendet, um eine Fehlermeldung in der Konsole auszugeben.

Probieren Sie es selbst in der Datei *file-2.html*: Ziehen Sie (falls möglich) ein Bild aus einem lokalen Verzeichnis auf die Seite, damit es

dort angezeigt wird. Neben `readAsDataURL()` existieren noch einige weitere Methoden: `readAsText()` liefert den Dateiinhalt als reinen Text zurück; `readAsArrayBuffer()` liefert den Inhalt als unformatierten Puffer von Binärdaten zurück (besonders hilfreich für Bilder).

File-Writer-API und File-System-API

Sie können eine Reihe weiterer APIs verwenden, um noch tiefergehende Dateioperationen durchzuführen: Mit der File-Writer-API können Sie die Inhalte einer Datei verändern. Die File-System-API geht noch weiter und stellt ein voll funktionsfähiges Dateisystem auf dem Gerät des Anwenders bereit. Diese APIs sind spannend, aber doch zu technisch für das vorliegende Buch.

6.11 Mozillas Firefox OS und WebAPIs

Ein potenziell sehr interessanter Neuzugang im Markt der mobilen Betriebssysteme kommt von Mozilla, dem Entwickler des Firefox-Webbrowsers. Mozilla hat ein von Grund auf neues Betriebssystem entwickelt, das komplett auf offenen Webstandards basiert – HTML, CSS, JavaScript und weiter. Das OS erschien im Februar 2013 in der Version 1.0, im Oktober darauf folgte 1.1. Im Gegensatz zu iOS oder Android kommt es ohne Middleware-Schicht, und die System-APIs verwenden alle JavaScript zum direkten Ansprechen der Hardware.

Als Teil dieser Anstrengungen musste Mozilla auch viele der vorhandenen Geräte-APIs entwickeln und zahlreiche neue hinzufügen. Dieses Projekt nennt sich WebAPI. Viele darin enthaltenen APIs wurden bereits in diesem Kapitel abgedeckt, einige sind aber momentan einzigartig in Firefox OS.

Hier sind einige der neuen APIs: *WebTelephony* für ausgehende und eingehende Anrufe; *WebSMS* zum Versenden, Empfangen und Verwalten von Textnachrichten; *Contacts* zum Zugriff auf und zur Verwaltung des Adressbuchs sowie *Device Storage* zum Zugriff auf

gemeinsam genutzte Dateien oder Verzeichnisse wie etwa die Bildergalerie auf vielen Mobiltelefonen.

Das weitere Schicksal von Firefox OS und die Implementierung dieser APIs auf breiterer Basis stehen noch in den Sternen, aber ich bin schon sehr gespannt auf den mittelfristigen Erfolg dieses Smartphone-Betriebssystem, das ausschließlich auf offenen Webtechnologien basiert – und auf die vielen Möglichkeiten, die sich dadurch für Webentwickler hinsichtlich der Geräteinteraktion ergeben.

6.12 PhoneGap und native Wrapper

Hybride Apps

Wenn Sie tiefergehenden Zugriff auf Geräte-APIs benötigen, aber dennoch bei der Entwicklung auf Webtechnologien setzen möchten, überlegen Sie sich den Einsatz eines nativen Wrappers für Ihre App. Diese Wrapper fungieren als eine Art Zwischenschicht zwischen Ihrer Webanwendung und dem fraglichen Gerät. Dazu bieten sie Hooks in die API, verwenden aber keinen nativen Code zur Bildschirmanzeige. Beim Einsatz eines nativen Wrappers entsteht eine sogenannte *Hybrid-App*.

Welchen Wrapper Sie verwenden, hängt größtenteils von Ihren Zielgeräten ab (ich werde darauf in Kapitel 10 noch näher eingehen). Ein perfektes Beispiel für die Arbeitsweise von Wrappern liefert Phone-Gap. Dieser Wrapper für Mobiltelefon-Apps bietet eine API für Entwickler zur Erstellung von Hybrid-Apps, die auf iOS, Android, Windows Phone, Blackberry und weiteren Systemen laufen.

6.13 Zusammenfassung

Notgedrungen konnte ich nur einige der vielen APIs ansprechen, aus denen sich die Webplattform zusammensetzt. Darunter befinden sich die

APIs zur Orts- und Bewegungsbestimmung, zum Ermitteln von Batteriestand und Internetverbindung, für den Zugriff auf Mikrofon und Kamera, lokale Datenspeicherungsmöglichkeiten, Drag & Drop mit Dateien und Elementen sowie für den Zugriff auf Informationen über den Dateiinhalt. Ich hoffe, dass ich mit diesem Überblick und den Beispieldateien die kreativen Möglichkeiten wenigstens andeuten konnte, die sich beim Gerätezugriff mittels JavaScript eröffnen.

6.14 Literaturempfehlungen

Dive Into HTML5 liefert unter http://diveintohtml5.info/geolocation.html eine ausführliche *Erklärung der Geolocation-API*, während der MozDev-Artikel *Orientation and Motion Data Explained* einen guten Überblick zur dreidimensionalen Ausrichtung und Bewegung gibt: https://developer.mozilla.org/en-US/docs/DOM/Orientation_and_motion_data_explained/.

Die Fullscreen-API wird im Sitepoint-Artikel *How to Use the HTML5 Full-Screen API* von Craig Buckler erklärt, allerdings wurde die API geringfügig überarbeitet, während ich dies schrieb; einige Objektnamen oder Eigenschaften könnten sich also geändert haben. Den Artikel finden Sie unter http://www.sitepoint.com/html5-full-screen-api/.

Die *Battery-Status-API* wird von David Walsh unter http://davidwalsh.name/battery-api/ gut erklärt, und eine *Diskussion der früheren und neu aktualisierten Network-Information-API* findet sich hier: http://nostrongbeliefs.com/a-quick-look-network-information-api/.

HTML5 Rocks bietet im Artikel *Capturing Audio & Video in HTML5* die beste Erklärung von `getUserMedia()`: http://www.html5rocks.com/en/tutorials/getusermedia/intro. Die

vollständige *Zielsetzung des WebRTC-Projekts* ist unter http://www.webrtc.org/aufgelistet. (Wieder einmal) MozDev liefert eine prägnante *Einführung in die Web-Storage-API:* https://developer.mozilla.org/en-US/docs/DOM/Storage/.

Der am besten verständliche *Leitfaden zur Drag-and-Drop-API*, den ich finden konnte, stammt von den HTML5 Doctors unter http://html5doctor.com/native-drag-and-drop/, während das fünfteilige *Working with Files in Java Script* von Nicholas Zakas eine exzellente Ressource zur File-API darstellt: http://www.nczonline.net/blog/2012/05/08/working-with-files-in-javascript-part-1/.

Die *APIs, aus denen das Firefox-OS-Projekt besteht*, sind unter https://wiki.mozilla.org/WebAPI/ aufgeführt, und die Folien der Präsentation *WebAPIs and Apps* von Robert Nyman liefern einen tollen Überblick darüber: http://www.slideshare.net/robnyman/webapis-apps-mozilla-london/. *Are We Mobile Yet?* zeigt aktuell und auf einen Blick, wie es um die Implementierung von APIs bestellt ist: http://arewemobileyet.com/.

Dieses Bild wurde dem Artikel »Orientation and Motion Data Explained« aus dem Mozilla Developer Network [MDN] entnommen: http://developer.mozilla.org/en-US/docs/DOM/Orientation_and_motion_data_explained/. Es unterliegt einer Creative-Commons-Lizenz.

7 Bilder und Grafiken

Viele Jahre lang konnten Bilder im Web nur als JPG- und GIF-Dateien angezeigt werden. Erstere wurden hauptsächlich für Fotos und Letztere für Symbole und Grafiken eingesetzt. Zu diesen Formaten gesellte sich später das PNG-Format, welches das GIF-Format weitgehend verdrängte (zumindest bei Icons, nicht bei animierten Katzen).

Dies sind jedoch alles reine Bildformate; außer für einfache Bildfür-Bild-Animationen eignen sie sich nicht für dynamische oder interaktive Bilder oder Grafiken. Dafür waren meist Drittanbieter-Plugins, wie zum Beispiel Flash, zuständig.

Moderne Webbrowser bieten hingegen den Luxus zweier neuer Grafikformate: Scalable Vector Graphics (SVG), ein skalierbares Format, das im DOM verankert und mittels CSS und JavaScript verändert werden kann, und canvas, ein HTML5-Element, auf dem Sie mithilfe einer API zeichnen können. Mit diesen beiden eröffnet sich eine völlig neue Palette grafischer Möglichkeiten, die von interaktiven Schaubildern bis hin zur unmittelbaren Bildbearbeitung reicht.

In diesem Kapitel bringe ich Ihnen die Grundlagen beider Formate näher. Dabei zeige ich Ihnen ein paar gängige Beispiele und liefere Ihnen ausreichend Informationen, damit Sie selbst entscheiden können, wann für Sie welches Format in Frage kommt. Bevor ich beginne, erkläre ich jedoch zunächst kurz einige grundlegende Begriffe und Konzepte.

7.1 Vektor- und Pixelgrafiken im Vergleich

Vektorgrafiken: SVG

*Vektor*grafiken bestehen aus geometrischen Formen, die innerhalb eines Koordinatensystems angeordnet sind; eine

Reihe von Punkten wird auf ein Gitter gelegt und dazwischen werden Linien gezogen. Durch diesen Aufbau sind Vektorgrafiken skalierbar. Vergrößere ich mein Gitter, werden die Formen auf dieselben Koordinaten gezeichnet; die Koordinaten liegen jetzt aber weiter auseinander und meine Form vergrößert sich.

Pixelgrafiken: JPG, PNG, GIF

*Pixel*grafiken bestehen aus einer Ansammlung farbiger Pixel, die zusammen auf einem Raster ein Bild ergeben. Wenn Sie ein Bild in Photoshop (oder einem ähnlichen Programm) öffnen und hineinzoomen, können Sie dieses Pixelraster mit eigenen Augen sehen. Alle gängigen Webgrafikformate (JPG, PNG, GIF) sind Pixelbildformate. Ändern Sie die Größe eines solchen Pixelrasters, muss der Computer einen Skalierungsalgorithmus bemühen, um die Bildgröße an die neue Fläche anzupassen. Je nachdem, ob Sie dabei vergrößern oder verkleinern, müssen neue Pixel hinzugerechnet oder bestehende Pixel entfernt werden. Diese Algorithmen liefern in der Regel beeindruckende Ergebnisse. Meist gelingt es ihnen aber besser, Pixel wegzulassen als neue hinzuzufügen. Ein verkleinertes Pixelbild sieht also meist viel besser aus als ein vergrößertes.

Abb. 7–1 zeigt einen Vergleich der beiden unterschiedlichen Grafikformate beim Heranzoomen an ein Bild. Das Vektorbild ist scharf und deutlich, das Pixelbild wirkt hingegen unscharf. Der Skalierungsalgorithmus musste in einigen Fällen Näherungswerte zum Hinzufügen neuer, für die vergrößerte Fläche erforderlicher Pixel heranziehen.

Abb. 7–1
Detailansicht einer vergrößerten Grafik, als SVG (links) und als PNG

(rechts); die SVG-Grafik ist viel schärfer und deutlicher

Vektorgrafiken eignen sich nicht für Fotos oder fotorealistische Bilder, dafür aber ausgezeichnet für Illustrationen wie Logos, Symbole und Schaubilder. Umgekehrt sind Pixelbilder weniger gut für Umgebungen geeignet, in denen häufig skaliert wird (etwa auf Mobilgeräten), dafür eignen sie sich aber viel besser für Fotos.

7.2 Scalable Vector Graphics

SVG

Vektorgrafiken im Web werden mit der Auszeichnungssprache *Scalable Vector Graphics (SVG)* erstellt. Diese Sprache fristet schon sehr lange eine Art Schattendasein, doch zwei Faktoren konnten ihr nun neues Leben einhauchen: Ihre Implementation in IE9 und das aufkommende Mobil- und Mehrgeräte-Web. Inzwischen wird SVG von allen großen Browsern unterstützt.

Skalierbarkeit

SVG ist vor allem wegen seiner im letzten Abschnitt angesprochenen Skalierbarkeit so nützlich. Für das responsive Webdesign ist sie ein wahrer Segen. Sie können dasselbe Bild auf jeder Plattform verwenden und vermeiden dabei die in Kapitel 3 angesprochenen Probleme mit adaptiven Bildern. Außerdem können der Aufbau von SVG-Grafiken und die Art ihrer Einbettung in eine Seite die Barrierefreiheit und Durchsuchbarkeit verbessern, wie Sie in den nachfolgenden Abschnitten erkennen werden.

Natürlich eignet sich SVG nicht für alle Anwendungsgebiete – Fotos speichern Sie, wie bereits erwähnt, besser weiterhin als Pixelbilder –, aber es ist wirklich ein sehr nützliches Format für das Multiscreen-Design.

7.2.1 Anatomie eines SVG-Bilds

Lesbare XML-Dateien

Ehe ich Ihnen die Verwendung von SVG in Ihren Seiten erkläre, möchte ich kurz den Code erläutern, der sich dahinter versteckt. Tatsächlich handelt es sich bei SVG nämlich um eine XML-Datei zur Beschreibung des daraus erstellten Bilds; Sie können sie also mit einem Texteditor anzeigen und von Hand verändern – mit Pixelbildern ist das eigentlich undenkbar.

Öffnen Sie eine SVG-Datei in Ihrem bevorzugten Editor oder lassen Sie sich den Quelltext in einem Browserfenster anzeigen und untersuchen Sie den Inhalt. Auch wenn das eigentliche Markup von Datei zu Datei stark variiert, werden Ihnen zumindest die vielen Gemeinsamkeiten auffallen. Zunächst einmal beginnen alle SVG-Dateien mit der XML-Deklaration, welche die von Ihnen verwendete Versionsnummer von XML, die Textcodierungsmethode und das `standalone`-Attribut enthält, das festlegt, ob die Datei andere externe Dateien referenziert oder für sich alleine steht (in den meisten Fällen belassen Sie es einfach bei *no*):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

Wurzelelement svg

Als Nächstes sehen Sie das `svg`-Element – das Wurzelelement – und eine Reihe von Namensräumen mit Informationen über die verwendete Syntax. Das nachfolgende Beispiel stammt aus einer von mir beliebig ausgewählten SVG-Datei. Darin erkenne ich, dass in der Datei das Dublin-Core-Vokabular verwendet wird (auf Seite 40 finden Sie unter »RDFa« weitere Informationen hierzu). Sie steht unter der Creative-Commons-Lizenz, und zur Beschreibung ihrer Inhalte kommen das RDF-, SVG- und XLink-Schema zum Einsatz:

```
<svg xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:cc="http://
web.resource.org/cc/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:svg=»http://www.w3.org/2000/svg" xmlns="http://
www.w3.org/2000/svg" xmlns:xlink=»http://www.w3.org/1999/xlink">
```

Die einfachste Form des `svg`-Elements benötigt nur die Attribute für einen Verweis auf den SVG-Namensraum und die verwendete SVG-

Version:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
```

Unterelemente

Alle Unterelemente des `svg`-Elements zusammen enthalten die zur Erstellung des eigentlichen Bildes erforderlichen Informationen. Der Großteil dieser Information befindet sich in einer Reihe von Elementen, die auf geometrischen Formen zur Erstellung der Zeichnung beruhen: `circle`, `rect`, `ellipse`, `polygon` und `line`. Jedes dieser Formelemente besitzt auch eine Reihe von Attributen zur Beschreibung von Position, Größe und Farbgebung der Form. So zeichnen Sie beispielsweise einen Kreis, dessen Mittelpunkt 50 px vom linken Gitterrand (`cx`) und 75 px vom oberen Gitterrand (`cy`) entfernt ist und der einen Radius von 25 px (`r`), einen schwarzen Rand (`stroke`) von 2 px Breite (`stroke-width`) und eine silberne Hintergrundfarbe (`fill`) besitzt:

```
<circle cx="50" cy="75" r="25" stroke="black" stroke-width="2"
fill="silver"/>
```

Hinweis

Das Element wird mit einem Schrägstrich geschlossen; SVG ist striktes XML, und alle leeren Elemente müssen geschlossen werden.

Zum Vergleich mit dem letzten Code folgt hier ein Rechteck von 100 px Breite (`width`) und 50 px Höhe (`height`); sein Versatz zum linken Gitterrand (`x`) beträgt 25 px und zum oberen Gitterrand (`y`) 50 px. Der Hintergrund ist (`fill`) ist blau:

```
<rect width="100" height="50" x="25" y="50" fill="blue"/>
```

Grafische Editoren: Adobe Illustrator und Inkscape

Ich möchte aus zwei Gründen nicht viel tiefer in den Aufbau von SVG-Bildern eintauchen: Zum einen ist das Thema zu kompliziert für ein einzelnes Buchkapitel und zum anderen denke ich, dass Sie das gar nicht allzu häufig brauchen werden. Sie

werden wahrscheinlich viel eher einen grafischen Editor (wie zum Beispiel Adobe Illustrator oder das Open-Source-Programm Inkscape) zur Bearbeitung von SVG-Dateien nutzen; im Dokumentcode führen Sie dann vielleicht nur noch gelegentlich kleinere Anpassungen durch.

7.2.2 Verknüpfte SVG-Dateien

Sie werden SVG auf Ihren Sites hauptsächlich auf zwei Arten einsetzen. Einmal als direkt in das Markup eingebetteten Code, was ich im nächsten Abschnitt 1.2.3 besprechen werde. Die zweite und einfachere Möglichkeit, die wir zunächst betrachten, besteht darin, eine vorgefertigte SVG-Datei als Bild zu verlinken, so wie Sie es auch mit einer JPG- oder PNG-Datei tun würden. Diese Methode ist hervorragend, wenn Sie ein skalierbares, dekoratives Bild erhalten möchten, allerdings können Sie das SVG-Bild dann nicht als Teil der DOM ansprechen.

.svg-Datei im HTML-Code ...

Zur Verwendung einer verknüpften SVG-Datei muss diese mit der Erweiterung *.svg* (oder *.svgz* für eine komprimierte Datei) abgespeichert werden; dann geben Sie nur noch an der entsprechenden Stelle im Code den Dateipfad zum Bild ein. Im Markup können Sie dazu ein `img`-Element verwenden:

```
<img src="/path/to/foo.svg">
```

... oder in CSS

Oder Sie rufen es per CSS auf, wo immer eine `url()`-Funktion als Argument akzeptiert wird, etwa mit `background-image`:

```
.foo { background-image: url('path/to/foo.svg'); }
```

WARNUNG

Der Nachteil der Verwendung von SVG im `img` -Element oder in CSS ist, dass Sie bestimmte erweiterte Möglichkeiten verlieren, wie etwa

Scripting und Animationen. Wollen Sie diese gerne nutzen, betten Sie den SVG-Code stattdessen ein (mehr dazu im Anschluss).

SVG-Sprites

Icons

Durch ihre Skalierungsmöglichkeiten eignet sich SVG auch besonders gut für Sprites, etwa für Icons. Dabei zeigen Sie nur einen kleinen Ausschnitt eines größeren Bilds an, um die Anzahl der Dateidownloads und damit die Ladezeit der Seite zu verringern. Allerdings braucht das etwas Vorarbeit. Alle gewünschten Icons werden in einer einzigen SVG-Datei übereinandergestapelt. Später werden dann mithilfe von CSS jeweils alle Symbole bis auf eines ausgeblendet (siehe Abb. 7–2).

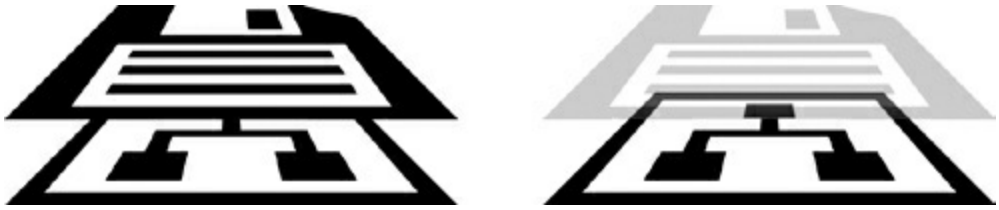


Abb. 7–2
Die Symbole sind in der SVG-Datei übereinan-dergestapelt (links). Mithilfe von CSS werden alle Symbole bis auf eines ausgeblendet (rechts).

Der dazu erforderliche Code sieht ungefähr so aus (und weil es eine Menge zu erklären gibt, habe ich ihn mit Anmerkungen versehen):

```
❶ <svg ...>
   <defs>
❷    <style><![CDATA[
       .icon { display: none; }
❸      .icon:target { display: inline; }
     ]]></style>
   </defs>
❹ <svg viewBox="0 0 30 30">
❺   <g class="icon" id="icon1">...</g>
```

```
  </svg>
  <svg viewBox="0 0 30 30">
   <g class="icon" id="icon2">...</g>
  </svg>
 </svg>
```

Das Wurzelelement ❶ benötigt keine zusätzlichen Namensräume, es kann also so bleiben wie in den vorhergehenden Beispielen. CSS kann direkt in die Datei geschrieben werden ❷. Das ist bei dieser Technik praktisch, weil Sie das Bild auf mehreren Seiten wiederverwenden können; beachten Sie den notwendigen CDATA-Abschnitt, mit dem Sie dem Browser mitteilen, dass hier Text und kein XML geparst werden soll. In den `style`-Regeln geben Sie an, dass die *.icon*-Elemente nicht angezeigt werden sollen bis auf dasjenige mit der `:target`-Pseudoklasse ❸ (ich erkläre bald, warum). Jede Sprite-Ebene im Stapel verfügt über ein `viewBox`-Attribut ❹ mit vier Koordinaten; dieses Attribut legt die Begrenzungen des angezeigten Rechtecks fest. Die ersten beiden Zahlen sind die *x*- und *y*-Koordinaten der oberen linken Ecke, das zweite Zahlenpaar legt Breite und Höhe des darzustellenden Bereichs fest. Schließlich besitzt das `g`-Element (eingesetzt zur Gruppierung von Formen) für jedes Sprite ❺ denselben Klassennamen wie die anderen, allerdings eine jeweils eindeutige `id`.

Nach diesen Vorbereitungen können Sie direkt auf das gewünschte Symbol verlinken. Dazu verwenden Sie in der URL den entsprechenden `id`-Wert, so wie hier gezeigt:

```
.bar { background-image: url('foo.svg#icon1'); }
```

Die Pseudoklasse `:target` wird angewendet, wenn der URL-Hash der Element- `id` entspricht; in diesem Fall ist das URL-Hash *#icon1*. Der `:target`-Selektor wird daher auf das `svg`-Element *#icon1* angewandt. Da die anderen `svg`-Elemente alle nicht angezeigt werden, wird nun *#icon1* gezeigt – egal, wo es sich im Stapel befindet. Ein Beispiel dafür sehen Sie in *svg-icon.html* und als Darstellung in Abb. 7–3.

Abb. 7–3
Für beide Symbole wird dasselbe Quellbild verwendet; die Stapeltechnik verleiht ihnen aber ein unterschiedliches Aussehen.

WARNUNG

Die Stapeltechnik wird in Zukunft eventuell nicht mehr funktionieren, falls sich die Render-Methoden der Browser für SVG verändern. Mehr Details hierzu finden Sie unter dem Link in den Literaturempfehlungen auf Seite 170.

SVG-Sprites mit Fragmenten

Eine alternative Sprite-Technik setzt auf Fragmentidentifizierer (*fragment identifier*). Für diese Methode müssen die Symbole nicht gestapelt werden; im SVG-Sprite-Bogen können alle Symbole einzeln nebeneinander liegen. Ihre Positionen werden dann mit den vier `viewBox`-Koordinaten als Teil der URL in der `svgView`-Funktion aufgerufen. Hier ein Beispiel:

```
.bar { background-image: url('foo.svg#svgView(viewBox(0 0 30
30))'); }
```

Ein Vorteil dieses Ansatzes ist, dass die SVG-Datei keine eigene `id` für jedes Sprite benötigt. Sie können mehrere Sprites oder nur Teile eines Sprites anzeigen. Mit der Stapelmethode ist dies schwerer umzusetzen.

7.2.3 Eingebettetes SVG

Teil des DOM

Wenn Sie die SVG-Grafik verändern wollen, können Sie sie direkt in den HTML-Code einer Seite einbetten. Sie wird dadurch Teil des DOM und kann über JavaScript und CSS angesprochen werden.

embed

Das gelingt auf unterschiedliche Weise, wie etwa durch Anwendung des `embed`-Elements. Dieses Element wurde im Web zum Einbetten von Flash-Dateien verwenden, auch wenn es bis zum Aufkommen von HTML5 nie zum Standard wurde. Sie können Ihrer Seite SVG mit `embed` hinzufügen, indem Sie im `src`-Attribut einen Link zu Ihrer Quelldatei angeben:

```
<embed src="foo.svg"></embed>
```

object

Ebenso können Sie das `object`-Element verwenden; der Pfad zur SVG-Datei ist dabei der Wert des `data`-Attributs:

```
<object data="foo.svg"></object>
```

Mit beiden Ansätzen zum Einbetten von SVG können Sie mithilfe der `getSVGDocument()`-Methode über das DOM auf das Markup zugreifen. Diese liefert ein Objekt zurück, das Sie mit gewöhnlichen DOM-Methoden traversieren können. Der nachfolgende Code zeigt, wie Sie das SVG-Objekt aus einem `embed`-Element erhalten und dann die Anzahl der darin enthaltenen Unterknoten mithilfe des `activeElement`-Attributs mitloggen:

```
var svg = document.querySelector('embed').getSVGDocument();
console.log(svg.activeElement.childNodes.length);
```

Markup direkt in die Seite einfügen

Eine viel einfachere und bessere Methode zum Einbetten von SVG besteht jedoch darin, dass Sie das Markup direkt in die Seite einfügen – in den meisten modernen Browsern ist das möglich. Ein einfaches Beispiel dafür sehen Sie in *svg-embedded.html*. Schauen Sie sich den Code in der Datei und den hier

abgedruckten Code an und vergleichen Sie ihn mit dem Ergebnis, das im Browser dargestellt wird (siehe auch Abb. 7–4)

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
 <rect width="100%" height="100%" fill="#000" />
 <circle cx="150" cy="100" r="80" fill="#FFF" />
 <text x="150" y="125" font-size="60"
    text-anchor="middle">SVG</text>
</svg>
```

Abb. 7–4
Die SVG-Attribute legen ein schwarzes Rechteck und einen weißen Kreis fest (links), das Beispiel zeigt jedoch das Gegenteil (rechts).

Ihnen sollte auffallen, dass der Kreis schwarz dargestellt wird, obwohl sein `fill`-Attribut im Code auf weiß festgelegt wurde. Das liegt daran, dass das eingebettete SVG Teil des DOM wird und ich die Formatierungen mit CSS definiere. In den `style`-Tags im Datei-Header finden Sie diese Regeln:

```
rect, text { fill: #FFF; }
circle { fill: #000; }
```

Da die SVG-Grafik zum DOM gehört, können Sie sie genauso verändern wie jedes andere Element. Einige der Elemente und Attribute, wie etwa die `fill`-Attribute in diesem Beispiel, betreffen die Darstellung und werden daher von CSS angesteuert. Und da SVG zum DOM gehört, brauchen Sie keine speziellen Methoden, um das Element per Skript zu beeinflussen. Sie können dafür also ganz normales DOM-Scripting verwenden:

```
var svg = document.querySelector('svg');
```

```
console.log(svg.childNodes.length);
```

Dieser einfache Zugriff macht die direkte Einbettung ins Markup besonders sinnvoll zum Anzeigen interaktiver Daten, die auf Benutzereingaben reagieren – Elemente können dann nach Bedarf hinzugefügt, entfernt oder verändert werden.

7.2.4 SVG-Filter

defs und filter

SVG bietet eine Reihe grafischer Filter zur Anwendung auf Elemente. Sie können damit Farbwerte anpassen, eine dynamische Weichzeichnung hinzufügen und so weiter. Für diese Struktur legen Sie zunächst ein `defs`-Element direkt nach dem Wurzelelement an. Dieses dient als Container für wiederverwendbare Elemente. Danach kommt ein `filter`-Element als Container für all die einzelnen Filterelemente. Das klingt vielleicht komplizierter, als es tatsächlich ist.

Beispiel: Gaußscher Weichzeichner

Im nachfolgenden Codebeispiel sehen Sie, wie einfach es eigentlich ist. Hier wird ein Gaußscher Weichzeichnereffekt mithilfe des `feGaussianBlur`-Elements angewendet:

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg">
 <defs>
  <filter id="filter1">
   <feGaussianBlur stdDeviation="3" in="SourceGraphic"/>
  </filter>
 </defs>
</svg>
```

Beachten Sie in diesem Code zwei Dinge: zunächst das `id`-Attribut des `filter`-Elements, das ich später zum Aufruf des Filtereffekts verwende, und zweitens die beiden Attribute des `feGaussianBlur`-Elements. Diese sind `stdDeviation`, ein besonderes Attribut dieses Elements, das die Weichzeichnungsstärke angibt, und `in`, ein gemeinsames Attribut aller Filterelemente, das bestimmt, worauf der Filter angewandt werden soll.

Der Standardwert für `in` ist das Schlüsselwort `SourceGraphic`; in einfachen Fällen wie dem vorliegenden könnten Sie ihn demnach sogar weglassen.

Der Filter ist jetzt definiert und kann über seine eindeutige `id` aufgerufen werden. Im nachfolgenden Code wende ich den Filter auf ein Pixelbild an, das über ein `image`-Element und das `xlink:href`-Attribut aufgerufen wird. Dies ist ein Namensraumattribut, ich muss diesen Namensraum also im Wurzelelement aufrufen. Schließlich wende ich den Filter mit dem `filter`-Attribut und eindeutiger `id` als Argument im `url()`-Funktionswert an. Sie sehen das in der Datei *svg-filters.html*; das Ergebnis ist in Abbildung 7-5 dargestellt.

```
<svg version="1.1" xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink">
 <defs>
  <filter id="filter1">
   <feGaussianBlur stdDeviation="3" in="SourceGraphic"/>
  </filter>
 </defs>
 <image xlink:href="foo.jpg" filter="url(#filter1)"/>
</svg>
```

Abb. 7–5
Ein Foto vor (links) und nach (rechts) Anwendung eines SVG-Filters

Filter kombinieren

Sie können mehrere Filtereffekte in einem einzelnen Filter kombinieren, indem Sie sie

als Unterelemente des `filter`-Elements stapeln. Im nächsten Codebeispiel wende ich wie zuvor den Gaußschen Weichzeichner an, allerdings mit einem etwas geringeren `stdDeviation`-Wert und außerdem mit einem zweiten Filter, `feMorphology`. Dieser Filter dient zum Verdünnen oder Verdicken des Quellbilds. In diesem Fall möchte ich es verdünnen. Daher verwende ich den Wert `erode` als `operator`-Attribut, als `radius`-Attribute wähle ich den Wert 2.

Der Code für meinen fertigen Zweistufenfilter, den Sie in der Datei *svg-filters.html* begutachten können (das Ergebnis finden Sie in Abbildung 7-6), sieht so aus:

```
<filter id="filter1">
  <feGaussianBlur stdDeviation="2"/>
  <feMorphology operator="erode" radius="2" />
</filter>
```

Abb. 7–6
Das Originalbild (links) und nach Anwendung des zweistufigen SVG-Filters (rechts)

Filter können Sie mit dem SVG-Element `foreignObject` auf ganze HTML-Blöcke anwenden. Dieses dient als Container für Codeschnipsel, solange es sich dabei um korrekt formatiertes XHTML handelt. Das nächste Beispiel zeigt, wie es funktioniert. Ein paar Dinge sind anzumerken: Zunächst verfügt `foreignObject` über ein `requiredFeatures`-

Attribut. Dieses prüft, ob der Browser das Merkmal des angeforderten Objekttyps unterstützt, in diesem Fall also die Erweiterungen für SVG1.1; außerdem kommt hier derselbe Filter wie in den vorherigen Beispielen zum Einsatz, und drittens hat der Code in dem Element den XHTML-Namensraum.

```
<foreignObject filter="url(#filter1)"
  requiredFeatures="http://www.w3.org/TR/SVG11/
  feature#Extensibility">
  <h1 xmlns="http://www.w3.org/1999/xhtml">SVG</h1>
</foreignObject>
```

Probieren Sie es selbst in der Datei *svg-foreignobject.html* aus. Das Ergebnis sehen Sie in Abb. 7–7.

Abb. 7–7
Mit foreignObject auf HTML-Markup angewandter SVG-Filter

7.2.5 SVG und CSS kommen zusammen

SVG2

Im Moment wird an der zweiten Version von SVG gearbeitet – ihre Bezeichnung lautet schlauerweise SVG2. Ein Ziel dabei ist eine bessere Integration in CSS, weil beide Sprachen bereits viele gemeinsame Merkmale haben. Das neue Modul `Transforms` fasst beispielsweise die beiden äußerst ähnlichen `Transforms`-Module aus CSS und SVG zusammen. Andere neue gemeinsame Merkmale sind etwa erweiterte Bildtechniken wie Masking und Clipping (mit denen Sie Teile eines SVG-Bilds zeigen oder verbergen) sowie Composing und Blending (mit denen Sie Farben und Formen kombinieren).

Filtereffekte

Als Erstes werden wohl aber Filtereffekte

kommen. Diese sind bereits jetzt, während ich dies schreibe, fragmentarisch in einigen Browsern implementiert. Dazu gehören zwei Hauptkomponenten: Erstens werden definierte Filter auf beliebige Seitenelemente angewandt, selbst auf solche außerhalb des SVG-Codeblocks; der native Einsatz des im vorigen Abschnitt definierten Filters in CSS wäre dann beispielsweise so einfach:

```
.foo { filter: url(#filter1); }
```

Funktionsschlüsselwörter

Die zweite Komponente ist der Einsatz von Funktionsschlüsselwörtern – im Wesentlichen sind das Abkürzungen zu vordefinierten SVG-Filtern. Um einem Element einen Gaußschen Weichzeichner hinzuzufügen, könnten Sie beispielsweise die `blur()`-Funktion verwenden, für eine veränderte Sättigung die `saturate()`-Funktion:

```
.foo { filter: blur(3) saturate(0.1); }
```

Wie CSS3 wird auch SVG2 modular aufgebaut sein. Das erlaubt eine gestaffelte Entwicklung und Implementierung; erste Elemente davon könnten Sie also schon recht bald in freier Wildbahn sichten.

7.2.6 Ein Nachteil von SVG

Längere Ladezeit

Bei allen Vorteilen von eingebettetem SVG in Ihren Dokumenten – eine der größten Stärken, der Zugriff über das DOM, ist zugleich auch die größte Schwäche: Die Integration in die Seite hat längere Ladezeiten zur Folge. Die komplexere Seite braucht länger zur Berechnung, und der Speicherbedarf steigt, weil mehr Objekte im Zwischenspeicher gehalten werden.

Daran sollten Sie denken, wenn Sie komplexe SVG-Objekte in Ihre Seite einbinden möchten. Insbesondere Seitenbesucher mit einem leistungsschwächeren tragbaren oder mobilen Gerät könnten darunter zu leiden haben.

7.3 Das canvas-Element

Pixel auf dem Bildschirm

Das `canvas`-Element verdankt seine Namensgebung einer leeren Leinwand, auf die gezeichnet oder gemalt werden kann. Das Zeichnen erfolgt in JavaScript – unter Verwendung einer eigenen API entsteht eine maßgeschneiderte Pixelgrafik. Und tatsächlich ist es wie bei der Leinwand eines Malers – wenn Sie einmal daraufgemalt haben, können Sie Ihr Werk nicht mehr verändern. Es handelt sich nicht um Objekte wie SVG-Elemente, sondern nur um Pixel auf dem Bildschirm; Sie können sie lediglich übermalen.

Einrichten

Die Einrichtung ist äußerst einfach: Fügen Sie Ihrem Code ein `canvas`-Element hinzu – vergeben Sie dabei vielleicht noch eine eindeutige `id`, damit Sie später etwas einfacher darauf zugreifen können. Wichtig sind die Größenattribute und ein alternativer Inhalt für Browser, die kein `canvas` (oder JavaScript) unterstützen:

```
<canvas id="canvas" height="400" width="800">
  <p>Sorry, your browser doesn"t support canvas (or JavaScript)
</p>
</canvas>
```

Weiter mit JavaScript

Alle nachfolgenden Aktionen werden in JavaScript durchgeführt. Im ersten Schritt wählen Sie den `canvas` aus und erstellen einen Kontext. Im Grunde sagen Sie damit nur, dass Sie jetzt darauf zeichnen möchten. Dazu verwenden Sie die Methode `getContext()` und geben den `context` als Argument an. Für einfache, zweidimensionale Formen lautet dieser `2d`. Ich füge noch einige Merkmalüberprüfungen hinzu, um sicherzugehen, dass der Browser die `canvas`-API unterstützt. Mein Code sieht dann schließlich ungefähr so aus:

```
var el = document.getElementById('canvas');
if (el && el.getContext) {
  var context = el.getContext('2d');
  if (context) { ... }
```

```
}
```

Zeichnen

Dann beginnen Sie zu zeichnen. Wie im Fall von SVG werden Zeichnungen als Abfolge von Formen und Linien in einem Koordinatensystem ausgeführt.

fillRect()

Zum Zeichnen eines einfarbig gefüllten Rechtecks verwenden Sie beispielsweise die Methode `fillRect()`, die vier Zahlenwerte als Argumente erlaubt: Die beiden ersten sind die *x*- und *y*-Koordinaten der linken oberen Ecke, darauf folgen Breite und Höhe. Der nachfolgende Code zeichnet ein einfarbiges Quadrat mit 120 Pixeln Seitenlänge, das sich jeweils 20 Pixel von der oberen linken Ecke des `canvas`-Elements entfernt befindet:

```
context.fillRect(20,20,120,120);
```

Sie können die Werte für das Aussehen der gezeichneten Objekte mit einer Reihe von Eigenschaften in der API verändern. Diese werden angewandt, sobald das nächste Mal eine Methode ausgeführt wird, die auf `canvas` zeichnet. Im nachfolgenden Code ändere ich die Füllfarbe und richte dann eine Reihe von Eigenschaften für einen Schlagschatten ein. Sie werden auf das nächste mit `fillRect()` gezeichnete Rechteck und alle weiteren Zeichnungen angewandt, bis ich die Eigenschaften wieder ändere:

```
context.fillStyle = '#ff0000';
context.shadowOffsetX = 3;
context.shadowOffsetY = 3;
context.shadowBlur = 3;
context.shadowColor = 'rgba(0,0,0,0.5)';
context.fillRect(200,80,160,160);
```

arc() und stroke()

Kreise sind etwas komplizierter zu zeichnen. Sie benötigen dazu die `arc()`-Methode, die sechs Argumente entgegennimmt: zunächst die x-und y-Koordinaten des Kreismittelpunkts, als dritten Wert den Radius, als Viertes den Anfangswinkel des Kreisbogens, danach als

Fünftes den Endwinkel – ich verwende Pi mal 2 für einen vollen Kreis – und sechstens schließlich einen booleschen Wert, der angibt, ob der Kreisbogen entgegen dem Uhrzeigersinn gezeichnet wird. Nachdem ich das alles erledigt habe, verwende ich die `stroke()`-Methode zum Zeichnen meiner Kreiskontur:

```
context.arc(360,240,160,0,Math.PI*2,false);
context.stroke();
```

In der Datei *canvas.html* (und in Abbildung 7-8) sehen Sie verschiedene Formen, die ich gezeichnet habe. Genau wie SVG bietet die `canvas`-API aber so viele Möglichkeiten, dass ich sie niemals in einem einzigen Buchkapitel behandeln könnte. Wenn Sie also gerne tiefer einsteigen würden, empfehle ich Ihnen, einige der hervorragenden und kostenfreien Online-Quellen zu besuchen, die ich teilweise auch in den Literaturempfehlungen auf Seite 170 nenne.

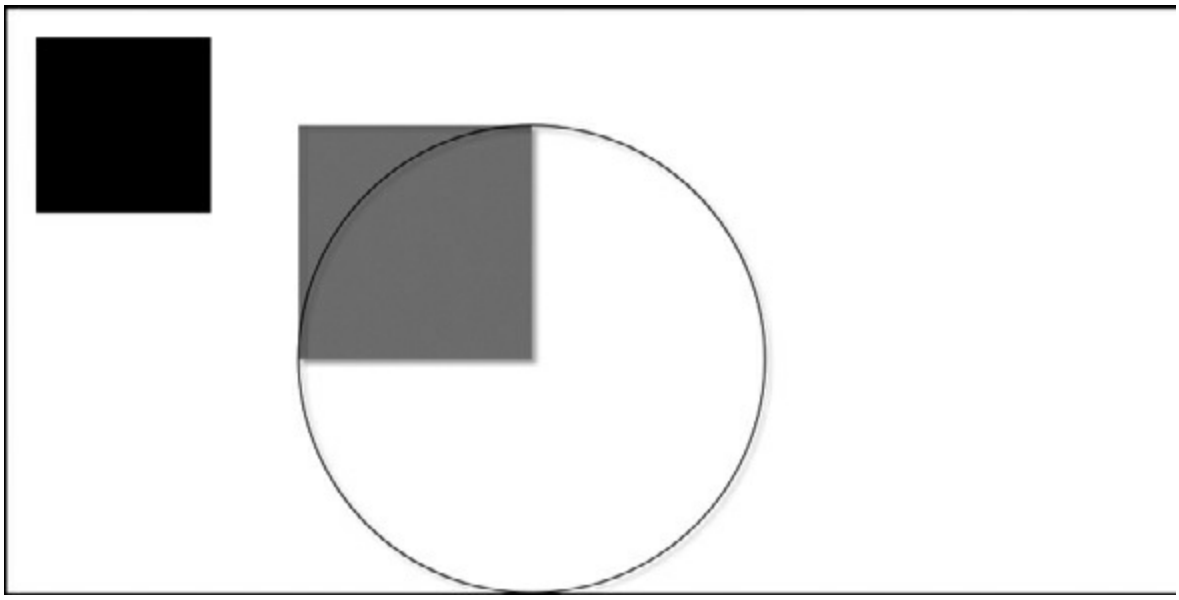


Abb. 7–8
Einfache Formen auf dem Canvas-Element

7.3.1 Bildmanipulation

Das `canvas`-Element ist sehr gut zur Manipulation von Bildern geeignet. Sie können ein Bild in ein `canvas` laden und es anschließend beliebig verändern oder darauf malen. Die Bilddaten holen Sie mit der `getImageData()`-Methode. Sie kennt zur Auswahl des zu manipulierenden Canvas-Bereichs vier Argumente, die Ihnen inzwischen bereits bekannt sind – x, *y*, Breite und Höhe. Es empfiehlt sich, diese Methode innerhalb eines `load`-Ereignisses zu verwenden. Dadurch stellen Sie sicher, dass das Skript nicht ausgeführt wird, bevor das Bild, aus dem Sie die Daten beziehen, vollständig geladen wurde.

Im nachfolgenden Codeschnipsel nutze ich `querySelector()`, um ein Bild auf der Seite zu finden. Nachdem es geladen wurde, zeichne ich es zunächst mittels `drawImage()` auf den Canvas und verwende dann `getImageData()`, um den gesamten Canvas auszuwählen und der Variablen *newImg* zuzuweisen:

```
var img = document.querySelector('img');
img.addEventListener('load', function () {
  context.drawImage(img,0,0);
  var newImg = context.getImageData(150,0,150,225);
}, false);
```

Sobald Sie das Bild haben, können Sie es mit der Eigenschaft `data` bearbeiten. Dabei handelt es sich um ein großes Array, das Informationen zu jedem einzelnen Pixel des Bildbereichs enthält. Jedes Element ist eine Zahl zwischen 0 und 255 im RGBA-Farbmodell: Das erste Element ist der Rotwert des Pixels, das zweite der Grünwert, das dritte der Blauwert und das vierte der Alphawert. Ein vollständig deckendes rotes Pixel hätte die vier Array-Werte [255, 0, 0, 255]. Diese vier Werte wiederholen sich für jedes einzelne Bildpixel.

Pixel einzeln verändern

Sie können also den Farbwert jedes einzelnen Pixels verändern – damit erhalten Sie einen gewaltigen Bearbeitungsspielraum. Im nachfolgenden Code verwende ich eine `for`-Schleife, um alle Pixel durchzugehen und ihre Werte durch Subtraktion des aktuellen Werts von 255 zu invertieren. Anschließend lege ich das veränderte Bild mittels `putImage-Data()` zurück auf dieselben Koordinaten des Canvas. Das

Ergebnis sehen Sie in Abb. 7–9 und auch in der Datei *canvas-image.html*.

```
var img = document.querySelector('img');
img.addEventListener('load', function () {
  context.drawImage(img,0,0);
  var i,
    newImg = context.getImageData(00,0,500,500),
    newLen = newImg.data.length;
  for (i=0;i<newLen;i+=4) {
   newImg.data[i] = 255 - newImg.data[i];
   newImg.data[i+1] = 255 - newImg.data[i+1];
   newImg.data[i+2] = 255 - newImg.data[i+2];
  }
  context.putImageData(newImg,0,0);
}, false);
```

Abb. 7–9
Im rechten Bild wurden die Farben auf dem Canvas invertiert.

Videos live bearbeiten

Eine weitere spannende Sache: Sie können das `canvas`-Element mit dem Live-Videostream aus der von mir in Kapitel 6 beschriebenen `getUserMedia()`-Methode kombinieren und Bilder dadurch unmittelbar bearbeiten. Ich habe zwar keinen Platz, dies weiter zu erörtern, aber Sie sollten sich dazu unbedingt online ein paar tolle Demos ansehen, zum Beispiel »Building a Live Green Screen with

`getUserMedia()` and MediaStreams« von Tim Taubert. Schauen Sie dazu im Abschnitt »Literaturempfehlungen« auf Seite 170 nach.

7.3.2 WebGL

Ihnen ist vielleicht aufgefallen, dass ich beim Anlegen des Kontexts festgelegt habe, dass er zweidimensional (2D) sein soll. Also muss es wohl auch einen 3-D-Kontext geben. So ist es – und mithilfe einer Technologie namens WebGL erhält der 3-D-Canvas Zugriff auf die Grafikkarte des Geräts und ermöglicht hardwarebeschleunigte 3-D-Objekte im Browser. Der Kontextname lautet `webgl`. Testen Sie die Unterstützung ungefähr so:

```
var el = document.getElementById('canvas');
if (el && el.getContext) {
  var context = el.getContext('webgl');
  if (context) { ... }
}
```

Extrem kompliziert

WebGL wird nicht von allen Browsern voll unterstützt, und das könnte auch in Zukunft so bleiben. Die verwendete Sprache ist für einen durchschnittlichen Entwickler (wie zum Beispiel mich) extrem kompliziert, und deshalb übersteigt WebGL den Umfang und die Zielsetzung dieses Buchs deutlich. Wie schon zuvor gesagt: Wenn Sie sich weiter dafür interessieren, werfen Sie einen Blick in die hervorragenden Online-Quellen zu diesem Thema.

7.4 Wann sollten Sie SVG oder Canvas wählen?

Die beiden in diesem Kapitel angesprochenen Bildmethoden sollten Sie nicht als Konkurrenztechnologien, sondern eher als gegenseitige Ergänzung betrachten. Beide erfüllen jeweils eine andere Rolle bei der

Darstellung von Webgrafiken. Womöglich werden Sie am Ende lieber beide in Ihren Websites verwenden, statt sich zwischen ihnen entscheiden zu müssen.

SVG

Die Stärken von SVG liegen besonders in der Skalierbarkeit; dieser Grafiktyp eignet sich daher für jeden Bildschirm, unabhängig von seiner Größe oder Auflösung. Da SVG neue Elemente im DOM erzeugt, ist es sehr hilfreich und nützlich im Zusammenspiel mit JavaScript und lässt sich auch darüber verändern. Auch barrierefreie Zugänge sind deutlich einfacher zu realisieren, indem den angelegten Elementen Alternativtexte zugewiesen werden. SVG ist eine gute Wahl für Logos, Symbole sowie interaktive Schaubilder und Grafiken.

Canvas

Canvas ist pixelbasiert und daher weniger gut zu skalieren. Außerhalb der eigenen API ist keine Manipulation über JavaScript möglich, und es gibt bei derzeitigem Sachstand wenige bis keine Funktionen für den barrierefreien Zugang (obwohl hier bereits an Verbesserungen gearbeitet wird). Canvas eignet sich ideal zur Bildbearbeitung, und da es nicht auf das DOM zugreift (wodurch eine Seite mitunter deutlich ausgebremst wird), lassen sich Elemente damit relativ schnell auf dem Bildschirm umherbewegen. Dies ist für Spieleanwendungen sehr praktisch.

Beide Formate haben Vor- und Nachteile, und für viele Anwendungen ließen sich beide einsetzen. Am besten ist es also, das Problem sorgfältig zu analysieren und beide Lösungen zu testen. Dann sehen Sie, welche Technik besser geeignet ist.

7.5 Zusammenfassung

In diesem Kapitel habe ich die beiden unterschiedlichen Ansätze digitaler Grafik angesprochen – Vektor- und Pixelgrafiken – und wie sie jeweils im Web mithilfe von SVG und `canvas` implementiert werden. Ich bin auf die

grundlegende Syntax für beide eingegangen und habe beleuchtet, für welche Situationen sie jeweils am besten geeignet sind. Sie können über diese Grafikformate selbst noch viel mehr herausfinden. Aus Platzgründen konnte ich auf beide nur ganz knapp eingehen – trotzdem hoffentlich ausführlich genug, um Ihre Neugier zu wecken.

7.6 Literaturempfehlungen

Eine hervorragende *SVG-Einführung* finden Sie auf der Website SVG Basics: http://www.svgbasics.com/, und der *SVG-Primer* des W3C geht noch mehr in die Tiefe: http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html. MDN bietet eine ziemlich vollständige *Elemente- und Attributeliste:* https://developer.mozilla.org/en-US/docs/SVG/.

Die *Technik zum Stapeln von SVG-Symbolen* stammt von Erik Dahlström und wird im Simurai-Blog beschrieben: http://simurai.com/post/20251013889/svg-stacks/. Robert O"Callahan von Mozilla *warnt vor möglichen Veränderungen der Stapeltechnik*: http://robert.ocallahan.org/2012/10/impending-doom-for-svg-stacks-sort-of.html. Ich führe die `svgView()`-Method in meinem Blog *Broken Links* weiter aus: http://www.broken-links.com/2012/08/14/better-svg-sprites-with-fragment-identifiers/.

Die IE-Testdrive-Site bietet ein praktisches *Tool zum Experimentieren mit SVG-Filtereffekten*: http://ie.microsoft.com/testdrive/graphics/hands-on-css3/hands-on_svg-filter-effects.htm.

Dirk Schulze hat eine gute, prägnante *Einführung zu den geplanten Funktionen von SVG2* geschrieben: http://dschulze.com/blog/articles/8/new-features-in-svg2/.

Eine gute Auswahl an *Canvas-Anleitungen* (vom Einsteiger- bis zum Fortgeschrittenenniveau) finden Sie unter http://www.html5canvastutorials.com/, einen hervorragenden *Spickzettel mit den Eigenschaften und Methoden* auf Jacob Seidelins Blog unter http://blog.nihilogic.dk/2009/02/html5-canvas-cheat-sheet.html (letzte Aktualisierung 2009, aber trotzdem noch relevant).

HTML5 Rocks liefert unter http://www.html5rocks.com/en/tutorials/canvas/imagefilters/eine *Anleitung zu fortgeschrittenen Bildbearbeitungseffekten*.

Tim Taubert erklärt in seinem Blog-Beitrag *Building a Live Green Screen with getUserMedia() and MediaStreams* die Grundlagen der Live-Bearbeitung von Videobildern: http://timtaubert.de/blog/2012/10/building-a-live-green-screen-with-getusermedia-and-mediastreams/.

Der Blog Learning WebGL stellt *Anfängerlektionen zur Arbeit in einem dreidimensionalen Kontext* in `canvas` bereit unter http://learning-webgl.com/lessons/, und WebGL.com bietet regelmäßig *Zusammenstellungen von Demos*, *Anleitungen und Entwicklertreffen*: http://www.webgl.com/.

8 Neue Formulare

Die Motivation bei der Entwicklung von Web Applications 1.0 – dem ursprünglichen Vorschlag, der dann zu HTML5 wurde – war die Erweiterung der Leistungsfähigkeit von HTML-Formularen und die Erstellung flexibler Online-Applikationen mit großer Funktionsvielfalt. HTML5 wird diesem Anspruch gerecht: Es bietet eine ganze Palette neuer Elemente, Attribute und Steuerelemente für funktionsreiche, interaktive und informative Formulare in der modernen Webplattform.

Viele JavaScript-Frameworks, wie etwa das beliebte jQuery UI, erweitern die Funktionalität der Formularelemente mit zusätzlichen Bedienelementen und Widgets. Diese reichen von einfachen Verbesserungen wie automatischen Vorschlägen beim Tippen bis hin zu komplexeren Benutzeroberflächenelementen, mit denen Sie etwa ein Datum auswählen können. Ein erklärtes Ziel von HTML5 ist die Standardisierung gängiger Designstrukturen. Daher überrascht es wenig, dass diese häufig genutzten Formularelemente in die Spezifikation mit eingeflossen sind.

Außer für neue Widgets und Interaktionsmöglichkeiten wird JavaScript in Formularen häufig zur Überprüfung von Eingabefehlern eingesetzt und um die korrekte Formatierung der Daten sicherzustellen. So werden unangenehme Überraschungen bereits vor der Übertragung zum Server ausgeschlossen. In HTML5 wurde diese Überprüfung ebenfalls bereits nativ implementiert – mithilfe einiger neuer Attribute und Eingabetypen, die festgelegte Datenmuster erfordern. Dazu kommen in vielen Browsern noch die Anzeige von Benachrichtigungen und Warnmeldungen und eine API, die Entwicklern mehr Flexibilität als beim Einsatz von Skripten bietet.

8.1 Neue Eingabetypen

In HTML 4.01 gab es für Benutzereingaben eine Handvoll Formularelemente, zum Beispiel `textarea`, `checkbox` und `radio`. Am häufigsten kam zweifellos `input` zum Einsatz, meist mit dem `type`-Attributwert `text`. Dieses Arbeitspferd unter den Feldern (ich bezeichne es ab jetzt als *Texteingabe*) wurde immer dann eingesetzt, wenn kein spezielles Bedienelement zur Verfügung stand: als Suchfeld, für Telefonnummern und E-Mail-Adressen … Was immer Sie sich vorstellen können, hat die Texteingabe erledigt. Dazu war nur eine anwender- oder serverseitige Überprüfung erforderlich, um den unterschiedlichen vom Benutzer eingegebenen Daten Bedeutung zu verleihen.

Bei der Arbeit an HTML5 – das ja erklärtermaßen bestehende Nutzungsmuster standardisieren sollte – lag es deshalb nahe, neue Eingabetypen zu definieren und damit die überstrapazierte Texteingabe etwas zu entlasten. Eine Handvoll Eigenschaften hat nun das Erscheinungsbild eines Texteingabefelds, allerdings mit abweichenden `type`-Werten zur Datendefinition.

search

Der erste neue Wert ist `search`. Ganz offensichtlich dient er zur Beschreibung eines Suchfelds. So sieht er aus:

```
<input type="search">
```

In manchen Browsern sieht das Suchfeld etwas anders aus als ein normales Texteingabefeld, hat etwa – passend zum Betriebssystem – abgerundete Ecken oder vielleicht eine Schaltfläche zum Löschen des Feldinhalts. Einige Beispiele sehen Sie in Abb. 8–1. Doch auch wenn Erscheinungsbild und Verhalten eventuell von dem eines Texteingabefelds abweichen, schränkt `search` den Anwender bei der Eingabe hinsichtlich der möglichen Datentypen nicht ein.

Abb. 8–1

Darstellung des Sucheingabefelds in verschiedenen Browsern: Chrome für Ubuntu (links) und Safari für iOS (rechts)

Anders verhält es sich mit den beiden nächsten Werten, `email` und `url`. Was wird der Benutzer damit wohl eingeben können? Richtig! Eine E-Mail-Adresse beziehungsweise eine URL. Die Felder gleichen einem Texteingabefeld; es gibt jedoch einen wichtigen Unterschied: Der Anwender kann hier nur bestimmte Daten in die Felder eingeben. So werden diese Werte eingesetzt:

```
<input type="email">
<input type="url">
```

Der `email`-Typ akzeptiert nur eine korrekt formatierte E-Mail-Adresse – zum Beispiel *peter@broken-links.com*. Der `url`-Typ erfordert eine korrekt formatierte URL inklusive Übertragungsprotokoll, also zum Beispiel *http://broken-links.com*. Gibt der Anwender in eines der Felder einen vom geforderten Muster abweichenden Wert ein, wird das Feld für ungültig erklärt. Ich komme auf dieses Thema etwas später im Abschnitt »Formularüberprüfung auf Anwenderseite« auf Seite 188 zurück.

tel

Eine weitere häufig angeforderte Information ist die Telefonnummer. Der neue Eingabetyp hierfür ist `tel`:

```
<input type="tel">
```

Das tatsächliche Format von Telefonnummern kann stark variieren: Hier sind Zahlen, Buchstaben und Symbole möglich. Dieses Feld hat daher keine Beschränkung, was die einzugebenden Zeichen betrifft.

Prüfmöglichkeiten

Viele Browser stellen die neuen Eingabetypen größtenteils identisch dar – warum sollte man sie dann überhaupt anstelle von `text` verwenden? Nun, ich habe bereits erwähnt, dass sie teilweise eine native Überprüfung bereitstellen. Auf Seite 188 gehe ich darauf im Abschnitt »Formularüberprüfung auf Anwenderseite« noch näher ein. Abgesehen davon bieten sie aber noch einen weiteren großen Vorteil:

Auf vielen Geräten mit Bildschirmtastatur stellen sie dem Anwender ein passendes Tastaturlayout zur Verfügung.

Bildschirmtastaturen

Mit Apples iOS (das vor Erscheinen des iPads noch iPhone OS hieß) kamen auch die benutzerfreundlichen Bildschirmtastatur-Layouts auf, und diese Implementierung ist wohl immer noch das Aushängeschild. Wenn Sie die Beispieldatei *input-types.html* auf einem iOS-Gerät öffnen und nacheinander die Eingabefelder aktivieren, aktualisiert sich jedes Mal die Bildschirmtastatur mit einem passenden Layout (sehen Sie sich Abb. 8–2 an, falls Sie gerade kein iOS-Gerät zur Hand haben): Die `text`-Eingabe zeigt ein normales Tastaturlayout; für die `email`-Eingabe kommen ein @ und ein Punkt hinzu (beides Bestandteile der E-Mail-Adresse). Beim `url`-Eingabefeld sehen Sie einen Punkt, einen Schrägstrich und ein Auswahlfeld für die Top-Level-Domain, und für die `tel`-Eingabe erhalten Sie eine numerische Telefontastatur. All diese Eingabetypen helfen den Nutzern, die korrekten Daten schneller und bequemer einzugeben.

Abb. 8–2
Die Bildschirmtastatur des iPhones, optimiert für die Eingabetypen text, email, url und tel (von links nach rechts)

Den Wert dieser angepassten Layouts können Sie erst dann wirklich erfassen, wenn Sie einmal eine Softwaretastatur ohne dieses Feature benutzen: Das Umschalten zwischen verschiedenen Ansichten für Buchstaben, Groß- und Kleinschreibung, Zahlen und Symbole ist wirklich nervtötend. Helfen Sie Ihren Anwendern also und verwenden Sie diese neuen Typen, wo und wann immer Sie können.

8.2 Neue Attribute

Bevor ich weitere Eingabetypen bespreche, möchte ich kurz abschweifen und auf einige der zahlreichen neuen Attribute eingehen. Diese Attribute versehen HTML-Formulare mit verschiedenen nützlichen und unerlässlichen neuen Eigenschaften und Verhalten. Und wieder basieren sie auf beliebten Skript-Bibliotheken und Lösungen von Entwicklern, die nun offiziell in HTML5 aufgenommen wurden.

8.2.1 autofocus

Fokus bei Seitenaufbau

Das `autofocus`-Attribut ist allen Eingabetypen gemein, und es setzt lediglich den Fokus auf das betreffende Element, sobald die Seite geladen wurde. In einem Texteingabefeld wäre die Eingabemarke dann also bereits im Feld und der Nutzer kann sofort zur Eingabe übergehen. `autofocus` ist ein boolesches Attribut, der Wert zu seiner Aktivierung kann also entweder `autofocus` lauten oder komplett wegelassen werden:

```
<input type="text" autofocus>
```

Werden mehrere Instanzen von `autofocus` auf einer Seite verwendet, wird die erste (in der DOM-Reihenfolge) berücksichtigt.

WARNUNG

Bedenken Sie, dass einige Anwender – vor allem wenn sie auf Hilfstechnologien angewiesen sind – irritiert sein könnten, wenn der Fokus beim Laden der Seite automatisch gesetzt wird. In Bruce Lawsons Artikel »The Accessibility of HTML5 Autofocus« finden Sie einen Diskurs über die beste Vorgehensweise (siehe Abschnitt »Literaturempfehlungen« auf Seite 196.

8.2.2 placeholder

Hinweis für die Eingabe

Das `placeholder`-Attribut können Sie auf jedes textartige Eingabeelement anwenden (zum Beispiel auf die bisher in diesem Kapitel vorgestellten). Als Wert akzeptiert es eine Zeichenkette. Diese Zeichenkette wird im Eingabefeld angezeigt, solange kein Wert vorhanden ist. Verwenden Sie die Zeichenkette, um die Art des angeforderten Werts zu beschreiben. Für ein `email`-Eingabefeld könnte das zum Beispiel folgendermaßen aussehen:

```
<input type="email" placeholder="e.g. foo@bar.com">
```

WARNUNG

Der Platzhaltertext sollte nicht den Eingabewert beschreiben – wie zum Beispiel »E-Mail Arbeit«. Diese Rolle übernimmt das `label`-Element. Denken Sie daran, dass der Platzhaltertext verschwindet, sobald der Nutzer in die Textbox klickt. Er könnte also vergessen, welches Feld er gerade ausfüllt, wenn es keine Beschriftung gibt.

In den meisten Browsern wird der Platzhaltertext heller dargestellt als der Standardtext. Damit wird verdeutlicht, dass es sich nur um Text und nicht um einen Wert handelt. Je nach Farbschema Ihrer Website ist das nicht unbedingt ideal. Leider gibt es keinen standardisierten Selektor zum Ändern der Eigenschaften des Platzhaltertexts. Einige Browser haben jedoch eine eigene Pseudoklasse (oder ein Pseudoelement) implementiert:

```
input:-moz-placeholder {}
input:-ms-input-placeholder {}
input::-webkit-input-placeholder {}
```

8.2.3 autocomplete

Autovervollständigung

Das `autocomplete`-Attribut legt fest, ob der Browser sich an zuvor in ein Feld eingegebene Werte erinnern und diese in Zukunft bei der Eingabe vorschlagen soll. Die Werte sind `on` und `off`; der Standardwert ist `on`. Wenn Sie aber eine Site mit vertraulichen Informationen erstellen, sollten Sie ihn eher auf `off` setzen. Damit erhöhen Sie die Datensicherheit für den Anwender – falls jemand anderes zu einem späteren Zeitpunkt dasselbe Gerät nutzt, werden keine Vorschläge geliefert:

```
<input type="email" autocomplete="off">
```

8.2.4 spellcheck

Rechtschreibprüfung

Viele Browser bieten inzwischen eine eingebaute Rechtschreibprüfung an. Standardmäßig wird sie meist nur auf `textarea`-Elemente angewendet. Sie können die Rechtschreibprüfung – so vorhanden – jedoch mit dem `spellcheck`-Attribut auf beliebige Felder anwenden. Dieses Attribut fällt etwas aus dem Rahmen. Es ist ein boolesches Attribut, das sich aber nicht wie andere Attribute dieses Typs verhält. Es benötigt einen Wert von `true` oder `false`, um die Rechtschreibprüfung ein- oder auszuschalten:

```
<input type="text" spellcheck="true">
```

Standardmäßig kommt ein Wörterbuch in der Sprache des verwendeten Browsers zum Einsatz. Mit dem `lang`-Attribut können Sie aber auch eine Sprache vorgeben. Hat der Anwender das Wörterbuch der geforderten Sprache installiert (im nachfolgenden Beispiel Spanisch), wird dieses für die Rechtschreibprüfung herangezogen:

```
<input type="text" spellcheck lang="es">
```

8.2.5 multiple

Mehrere Einträge pro Feld

Das `multiple`-Attribut eignet sich für Situationen, in denen der Anwender mehrere Einträge pro Feld eingeben oder übermitteln kann. Zusammen mit dem Eingabetyp `file` können Sie damit etwa mehrere Dateien auf dem Gerät des Benutzers auswählen. Bei einer `email`-Eingabe lassen sich wiederum mehrere Adressen aus dem Feld auslesen. Es ist ein boolesches Attribut mit dem Wert true, also genügt der Attributname:

```
<input type="file" multiple>
```

8.2.6 form

Eine Einschränkung bei Formularen in HTML 4.01 war, dass sich alle Formularelemente (inklusive der Senden-Schaltfläche) innerhalb des `form`-Elements selbst befinden mussten. Wurde also kein JavaScript verwendet, mussten alle Elemente nacheinander im Markup stehen, was die Layoutmöglichkeiten für die Seite einschränkte.

Verbindung zwischen Feld und Formular

HTML5 bietet nun als Antwort darauf das `form`-Attribut, das als Wert die `id` eines `form`-Elements akzeptiert. Dadurch entsteht eine Verbindung zwischen dem Feld und dem Formular, egal an welcher Stelle im Markup sich beide befinden. Der Wert des Elements wird dann zusammen mit dem Formular übertragen. In diesem Beispiel wird die Eingabe *#bar* zusammen mit dem Formular *#foo* übertragen.

```
<form id="foo">...</form>
<input type="text" id="bar" form="foo">
```

8.3 Vorschlagslisten

datalist

Wenn Elemente Texteingaben akzeptieren – wie etwa `text`, `url` oder `search` – können Sie

dem Benutzer eine Liste hilfreicher Vorschläge anbieten. Der Browser kann diese anhand der vorhergehenden Benutzereingaben bereitstellen (zur Steuerung dient das zuvor angesprochene `autocomplete`-Attribut). Manchmal ist es aber auch besser, vordefinierte Ausdrücke anzubieten. Letzteres erreichen Sie mithilfe des Elements `datalist`.

Das `datalist`-Element enthält eine Liste mit Vorschlägen, von denen sich jeder in einem `option`-Unterelement befindet (dieses sollte Ihnen bereits aus dem `select`-Element in HTML 4.01 vertraut sein). Als Beispiel zeige ich Ihnen hier eine kurze `datalist` mit wenigen Auswahlmöglichkeiten:

```
<datalist id="apes">
  <option>Chimpanzee</option>
  <option>Gorilla, Eastern</option>
  <option>Gorilla, Western</option>
  <option>Orangutan</option>
</datalist>
```

Das `datalist`-Element wird nicht auf der Seite angezeigt, und es muss sich nicht in der Nähe des Eingabefelds befinden, auf das es sich bezieht. Sie können es vielmehr beliebig im Code platzieren.

Um einen Zusammenhang zwischen dem Eingabefeld, in das der Benutzer seine Daten eingibt, und dem `datalist`-Element mit den Vorschlägen herzustellen, verwendet das Eingabefeld den `id`-Wert des `datalist`-Elements als Wert für sein eigenes `list`-Attribut:

```
<input type="text" list="apes">
```

Wenn der Anwender nun einen Buchstaben (oder eine Buchstabenfolge) eingibt, werden sämtliche Werte der `option`-Elemente aus der `datalist`, die zu dieser Buchstabenfolge passen, in einer Vorschlagsliste unter dem Eingabefeld angezeigt. Hier kann der Benutzer nun einen passenden Eintrag auswählen. In der Datei *input-types-more.html* können Sie es selbst ausprobieren (siehe auch Abb. 8–3).

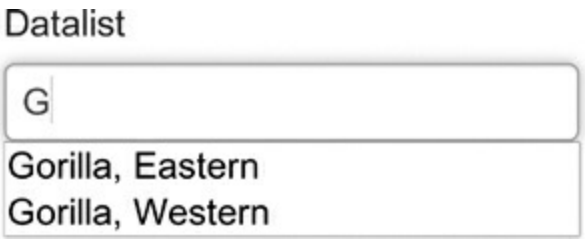


Abb. 8–3
Automatische Vorschläge aus einem datalist-Element

Durch die Verknüpfung von `id-` und `list`-Attributen können bei Bedarf auch mehrere Eingabefelder auf dasselbe `datalist`-Element zurückgreifen.

Wenn der Browser `datalist` nicht unterstützt, wird aus dem Eingabefeld wieder eine einfache Texteingabe. Bitte bedenken Sie dies, wenn Sie bestimmte Eingabewerte vom Anwender benötigen. Achten Sie auch darauf, serverseitig oder in JavaScript alle Eventualitäten zu berücksichtigen.

8.4 Bedienelemente und Widgets

Die bisher vorgestellten neuen Elemente basieren alle auf einem einfachen Textfeld. Einige Formularelemente bieten jedoch auch Bedienelemente mit zusätzlichen Funktionen, zum Beispiel `select` und `checkbox` in HTML 4.01. Wie ich bereits in der Einleitung erwähnt habe, setzen Designer und Entwickler aber auch andere Widgets häufig ein – denken Sie dabei etwa an die Datumsauswahl oder an Schieberegler für Zahlenbereiche. Diese Gestaltungsmuster wurden daher in HTML5 übernommen und standardisiert.

Die Darstellung dieser Steuerelemente und Widgets hängt vom Browser und von der Plattform ab, auf der sie implementiert wurden; die HTML5-Spezifikation besagt nur, dass diese Elemente verwendet werden können – sie gibt nicht das Erscheinungsbild vor. Wenn ein Browser die Steuerelemente nicht nativ unterstützt, sollten sie

stattdessen wie ein normales `text`-Eingabefeld angezeigt werden.

8.4.1 Zahlen

Die bisher besprochenen neuen HTML5-Eingabetypen decken bereits verschiedene Textformate ab. Viele Formulare erfordern aber natürlich vom Benutzer die Eingabe einer Zahl, etwa einer Kreditkartennummer, einer Postleitzahl oder einer Mengenangabe.

number

Das `number`-Eingabefeld dient zur Eingabe von Zahlen. Häufig wird es als Textfeld angezeigt, doch einige Browser fügen auch noch Bedienelemente hinzu. Häufig sind das zwei Pfeile – einer nach oben und einer nach unten –, mit denen der Wert erhöht oder verringert werden kann.

range

Eine ähnliche Eingabe wie `number` ist die `range`-Eingabe. Hier kann der Benutzer einen Wert eingeben, ohne dabei allzu genau auf die exakte Zahl achten zu müssen. Zu diesem Zweck bieten viele Browser für dieses Element einen Schieberegler an.

```
<input type="number">
<input type="range">
```

In Abb. 8–4 können Sie vergleichen, wie `number` und `range` in Chrome für Android dargestellt werden.

Abb. 8–4
Die Typen number und range dargestellt in Chrome für Android

Beide Typen haben einige neue Attribute gemeinsam: `max` und `min` sind Zahlenwerte, die das Maximum und das Minimum der erlaubten Werte festlegen (sind Sie selbst darauf gekommen?), und `step` legt die

Schrittweite beim Erhöhen oder Verringern fest. Der nachfolgende Code zeigt, wie die drei funktionieren könnten; die `number`-Eingabe hat einen Startwert von 50 und lässt sich in Zehnerschritten bis zu einem Maximum von 100 oder einem Minimum von 10 ändern:

```
<input type="number" max="100" min="10" step="10" value="50">
```

Sie können diese Werte auch mit JavaScript und den Methoden `stepUp()` und `stepDown()` verändern. Jede benötigt einen einzelnen Integer-Wert, der den Wert der Eingabe um die entsprechende Schrittzahl verändert; die nachfolgende Syntax verringert beispielsweise den `range`-Eingabewert um drei Schritte:

```
var foo = document.querySelector('input[type=range]');
foo.stepDown(3);
```

Beide Methoden liefern einen Fehler zurück, wenn der Wert durch die angegebene Schrittzahl die `max`- oder `min`-Werte des Elements über- beziehungsweise unterschreitet.

Bei der Arbeit mit Zahlenfeldern können Sie die neu in HTML5 definierte DOM-Eigenschaft `valueAsNumber` nutzen. Diese gleicht der bestehenden `value`-Eigenschaft, liefert den Wert aber als Zahl und nicht als Zeichenkette zurück. Sie brauchen die Typen dann also nicht mehr mit `parseInt()` ineinander umzuwandeln. Der Einsatz von `valueAsNumber` ist sehr simpel:

```
var foo = document.querySelector('input[type=number]');
bar = foo.valueAsNumber;
```

8.4.2 Datum und Zeit

Ein weiteres beliebtes Datenmuster für Formulare ist das Datums- oder Zeitfeld, mit dem der Anwender zum Beispiel sein Geburtsdatum oder Datum und Zeit für eine Lieferung angeben kann. Oft kommt dazu ein in JavaScript erstelltes Auswahlfeld zum Einsatz. Mit diesem häufig verwendeten Widget können Benutzer direkt auf dem Bildschirm ein

Datum auswählen. Bei der Eingabe brauchen sie sich daher nicht um das korrekte Datumsformat zu sorgen.

date

HTML5 bietet eine Reihe neuer Eingabetypen für Daten- und Formularfelder, und viele Browser verfügen über native Datumswahl-Wid-gets, um diese noch weiter zu verbessern. Am häufigsten implementiert wird wahrscheinlich `date`, bei dem der Anwender im Widget ein einzelnes Datum auswählen kann:

```
<input type="date">
```

Die Art der Implementierung unterscheidet sich von Browser zu Browser, und bei Mobil- und Tablet-Geräten finden sich recht deutliche Unterschiede zu Desktop- und Laptop-Browsern. Einige Beispiele dieser Vielfalt sehen Sie in Abb. 8–5.

Abb. 8–5
Das native Datumswahl-Widget in Chrome für Android-Tablet (links), auf dem iPhone (Mitte) und in Chrome für Desktop-PC (rechts)

HTML5 verfügt auch über eine Reihe weiterer Datums- und Zeiteingabemethoden: Für weiter gefasste Daten können Sie mit `month` einen kompletten Monat auswählen und mit `week` eine ganze Woche. Falls Sie eine Zeit ohne zugehöriges Datum abfragen möchten, nutzen Sie `time`, um Stunden und Minuten auszuwählen. Einige Beispiele für verschiedene Zeit- und Datumswahlmöglichkeiten sehen Sie in Abb. 8–6.

Abb. 8–6
Ein Zeitwahl-Widget in Chrome für Android (links) und eine Monatsauswahl für das iPhone (rechts)

Wenn Sie ein Datum und eine Zeit benötigen, erfragen Sie mit der `date-time`-Eingabe beide zugleich. Dieses Feld benötigt einen Wert im Format *YYYY-MM-DD*T*HH*:*MM*Z, wobei das Z als Abkürzung für die UTC-Zeitzone dient. Um die Zeit 14 Uhr am 1. April 2014 zu übermitteln, würden Sie beispielsweise *2014-04-01T14:00Z* verwenden. Wird keine Zeitzone benötigt, können Sie `datetime-local` nutzen. Für beide Typen zeigt das Auswahl-Widget Felder für Datum und Zeit, so wie in Abb. 8–7 dargestellt.

Beispiele für alle datumsbezogenen Eingabetypen finden Sie in *in-put-types-dates.html*. Öffnen Sie die Seite in verschiedenen Browsern und sehen Sie sich die unterschiedlichen Darstellungen an.

Wie bei den `number`-Eingabetypen sind auch hier die Attribute `max` und `min` zulässig. Sie müssen allerdings in einem gültigen Datums- oder Zeitformat vorliegen. Ein komplettes `datetime` benötigt also *YYYY-MM-DD*T*HH*:*MM*Z, während `month` nur mit *YYYY-MM* auskommt. Das `step`-Attribut ist ebenfalls zulässig, aber seine Zeitspanne hängt vom verwendeten Element ab: ein Tag, eine Woche, ein Monat oder eine Zeit in Sekunden.

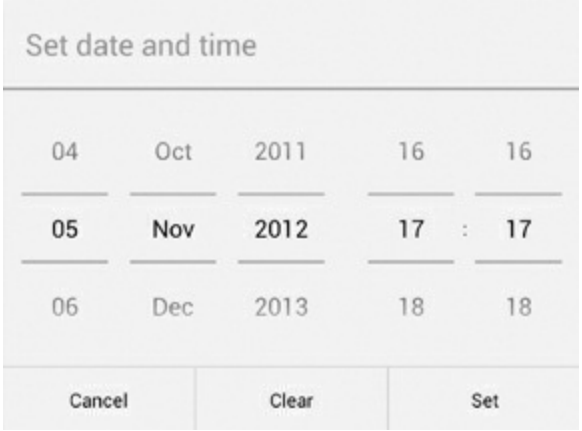


Abb. 8–7
Eine datetime-Auswahl in Chrome für Android

Wenn Sie das alles zusammenfügen, könnten Sie also zum Beispiel Attribute wie nachfolgend gezeigt verwenden. Die `month`-Eingabe erlaubt dem Anwender hier nur die Auswahl von Daten zwischen Januar 2012 und Juni 2016; das `step`-Attribut käme nur zum Tragen, wenn die Methoden `stepDown()` und `stepUp()` eingesetzt würden:

```
<input type="month" max="2016-06" min="2012-01" step="3">
```

WARNUNG

Wenn Sie strenge Vorgaben zu den erforderlichen Daten haben, verlassen Sie sich nicht auf die Attribute `max` und `min`. Manche Browser unterstützen diese nicht. Verwenden Sie immer JavaScript und eine serverseitige Überprüfung. Damit stellen Sie sicher, dass der zulässige Datumsbereich eingehalten wird.

Für Zahlen gibt es die DOM-Eigenschaft `valueAsNumber`, für Daten gibt es `valueAsDate`. Diese Eigenschaft funktioniert genauso, liefert jedoch einen als Datum formatierten Wert zurück; für das Datum *04/01/2014* liefert die `value`-Eigenschaft zum Beispiel *2014-04-01* zurück, die `valueAsDate`-Eigenschaft hingegen *Tue Apr 01 2014 01:00:00 GMT+0100 (BST)* (jedenfalls in meiner Zeitzone).

```
var foo = document.querySelector('input[type=date]');
bar = foo.valueAsDate;
```

8.4.3 Farben

Color

Wenn Sie eine Anwendung entwickeln, die dem Nutzer eigene Anpassungsmöglichkeiten bietet, interessieren Sie sich vielleicht für die `color`-Eingabe in HTML5. Diese zeigt entweder das Standard-Farbwahlfeld des Systems oder, je nach Browser, ein eigenes Widget:

```
<input type="color">
```

Probieren Sie es selbst in *input-types-more.html* aus. Abb. 8–8 zeigt die Implementierung in Chrome für Ubuntu.

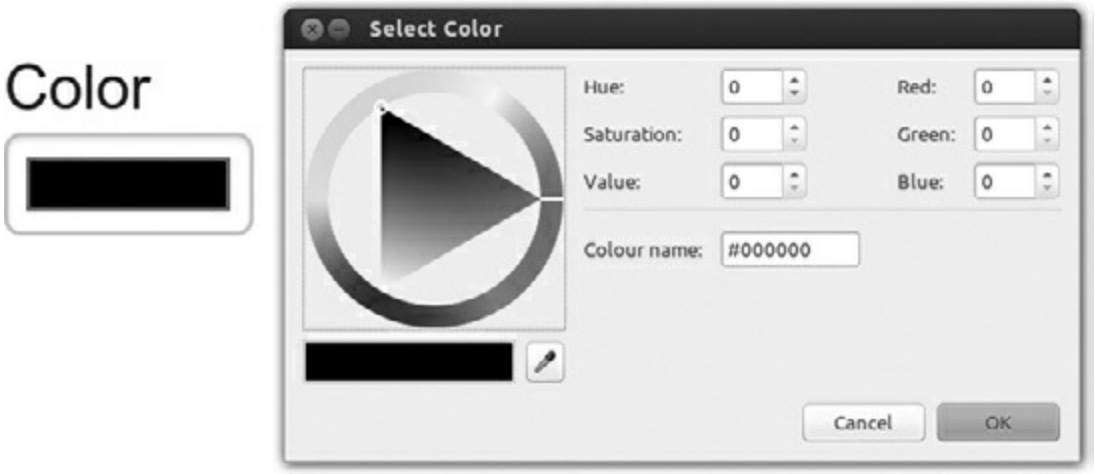


Abb. 8–8
Chrome verwendet für das Color-Element die native Farbwahl des Betriebssystems. So sieht es in Ubuntu aus.

8.5 Informationen anzeigen

Neben der Abfrage von Informationen ist es oft auch hilfreich, dem Anwender Informationen anzuzeigen – zum Beispiel als

Fortschrittsanzeige beim Ausfüllen eines Formulars oder als Ergebnis einer Messung. Eine Reihe neuer Elemente in HTML5 dient genau diesem Zweck.

8.5.1 progress

Fortschrittsbalken

Fortschrittsbalken zeigen Bewegung in Richtung eines vorgegebenen Ziels an. Häufig werden sie in Betriebssystemen und im Web eingesetzt, etwa bei der Installation von Software oder beim Laden von Daten in eine Webanwendung. Mit dem progress-Element verfügen Sie über eine standardisierte Methode zur Integration von Fortschrittsbalken auf Ihren eigenen Seiten. In der einfachsten Form zeigt es einen Fortschritt zwischen 0 und 1 an. Das value-Attribut markiert dabei die aktuelle Position

```
<progress value="0.5">0.5</progress>
```

Auch wenn dieses Element in den meisten Browsern als Balken dargestellt wird – siehe Abb. 8–9 –, ist es trotzdem sinnvoll, den Wert innerhalb des Elements anzugeben. Somit ist auch eine Anzeige in Browsern gewährleistet, die kein grafisches Widget bieten.

Abb. 8–9
Verschiedene Umsetzungen des progress-Elements: Firefox für Android (links) und Opera für Ubuntu (rechts)

Lassen sich die zu verwendenden Werte nicht einfach in einem Bereich von 0 bis 1 ausdrücken, können Sie mit den Attributen max und min einen anderen Bereich angeben:

```
<progress max="20" min="10" value="15">15</progress>
```

Die Aktualisierung des Balkens mithilfe eines Skripts ist einfach. Sie brauchen nur das value-Attribut zu ändern, etwa mit einer Funktion wie der folgenden. Hier wird mit jedem der Funktion *updateProgress* als Argument zur Verfügung gestellten Wert der Fortschrittsbalken entsprechend aktualisiert:

```
var progressBar = document.querySelector('progress'),
   updateProgress = function (newValue) {
     progressBar.value = newValue;
};
```

Den aktuellen Fortschritt zum Ziel erhalten Sie mit der position-Eigenschaft. Diese liefert als Ergebnis den aktuellen Wert geteilt durch den Maximalwert zurück; im letzten Beispiel ist das max-Attribut 20 und der value 15; als position ergibt sich also 0.75:

```
var currentProgress = progressBar.position;
```

8.5.2 meter

Für skalare Messung

Auf den ersten Blick scheint das `meter`-Element ungefähr dasselbe zu sein wie `progress`. Tatsächlich stellen einige Browser beide auch gleich dar. Es gibt jedoch einen semantischen Unterschied; `progress` zeigt die Bewegung auf ein Ziel hin, während `meter` für eine skalare Messung verwendet wird – wie etwa für eine Bewertung oder das Ergebnis einer Umfrage. Im einfachsten Fall zeigt `meter` einen Wert zwischen 0 und 1,0, genau wie `progress`:

```
<meter value="0.5">0.5 of 1</meter>
```

Die Gemeinsamkeiten mit `progress` setzen sich fort, denn auch hier sollten Sie ein Text-Unterelement vorsehen, falls der Browser keine grafische Darstellung beherrscht. Auch die `max`- und `min`-Attribute sind verfügbar, um die Skala zu verändern:

```
<meter max="20" min="10" value="15">15 of 20</meter>
```

low, high und optimum

In einem Punkt unterscheidet sich `meter` jedoch deutlich von `progress` – und zwar in der Anzeige von hohen, niedrigen und dazwischen liegenden Wertebereichen. Dazu dienen drei neue Attribute: `low` legt die obere Grenze des unteren Bereichs fest, `high` legt die untere Grenze des oberen Bereichs fest, und `optimum` legt den Idealwert fest. Wenn diese Attribute vorhanden sind, wird der aktuelle Wert der Anzeige entsprechend angezeigt: Man sieht dann, dass der Wert entweder innerhalb normaler Parameter liegt oder sie über- bzw. unterschreitet.

Das nachfolgende Codebeispiel verdeutlicht dies. Jeder Wert kleiner oder gleich 0,2 wird als niedrig angesehen und in vielen Browsern mit einer Warnfarbe (oft gelb) dargestellt, während ein Wert größer oder gleich 0,8 als hoch angesehen und farblich entsprechend (oft rot) markiert wird. Liegt der Zahlenwert zwischen diesen Begrenzungen, wird er in der Standardfarbe (meist grün) dargestellt:

```
<meter low="0.2" high="0.8" value="0.65">0.65 of 1</meter>
```

Mit der Einführung des `optimum`-Attributs ändert sich dieses Verhalten leicht, denn dadurch lässt sich der Bereich des akzeptablen (oder optimalen) Werts auch aus der Mitte heraus nach oben oder unten verschieben. Für das `meter`-Element im letzten Beispiel gibt es beispielsweise drei Bereiche: niedrig für alle Werte von 0,2 und darunter, hoch für alle Werte von 0,8 und darüber und durchschnittlich für alles größer als 0,2, aber kleiner als 0,8. Wenn Sie nun das `optimum` auf 0,9 setzen, wäre jeder Wert im oberen Bereich optimal und grün eingefärbt, jeder Wert im mittleren Bereich wäre weniger optimal und gelb und alle Werte im unteren Bereich noch weniger optimal und rot eingefärbt. Würde der Optimalwert umgekehrt auf 0,1 gesetzt, kehrten sich diese Optimalbereich entsprechend um.

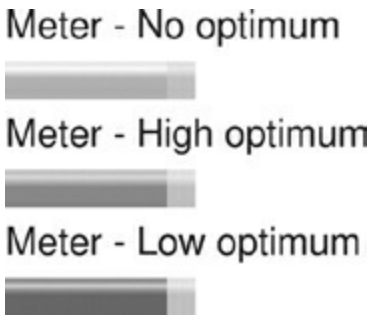


Abb. 8–10
Verschiedene Werte für das Element meter

Falls das alles ein wenig kompliziert klingt, finden Sie im nachfolgenden Markup ein paar verschiedene Beispiele, die Sie auch in der Datei *input-types-meter.html* und in Abb. 8–10 wiederfinden; Ich schlage vor, Sie sehen sich die Datei zu diesem Beispiel an – denn die Farben lassen sich in einem Schwarzweißbuch schlecht wiedergeben! Ich versehe diesen Code mit Anmerkungen, die ich nachfolgend erläutere.

```
❶ <meter low="0.2" high="0.8" value="0.85">0.85 of 1</meter>
❷ <meter low="0.2" high="0.8" optimum="0.9" value="0.85">0.85 of
1
  </meter>
❸ <meter low="0.2" high="0.8" optimum="0.1" value="0.85">0.85 of
1
  </meter>
```

In allen drei Beispielen reicht der untere Bereich von 0 bis 0,2, der mittlere Bereich von 0,21 bis 0,79, der obere Bereich von 0,8 bis 1 und der `meter`-Wert ist 0,85. Die Zahlen entsprechen den Beispielen in Abb. 8–10 in senkrechter Abfolge. In ❶ wurde kein `optimum`-Wert angegeben, der Optimalbereich ist also der mittlere (weder high noch low). Der Wert der Anzeige befindet sich im oberen Bereich, einem Bereich neben dem Optimalbereich, daher ist er gelb eingefärbt. In ❷ beträgt der Optimalwert 0,9. Damit wird der obere Bereich zum Optimalbereich. Der Wert beträgt 0,85, er befindet sich daher im Optimalbereich und wird grün dargestellt. In ❸ beträgt der Optimalwert 0,1, also wird der untere Bereich zum Optimalbereich. Der Wert ist 0,85, liegt also zwei

Bereiche neben dem Optimalbereich und wird deshalb rot dargestellt.

Wie schon gesagt: In Schwarzweiß ist das etwas schwer zu vermitteln, also probieren Sie das Beispiel selbst aus. Wenn Sie es einmal gesehen haben, verstehen Sie dieses Konzept recht einfach und schnell.

8.5.3 output

Ergebnis einer Berechnung oder eine Benutzereingabe

Das `output`-Element zeigt das Ergebnis einer Berechnung oder eine Benutzereingabe an. Es ist besonders praktisch, um das Ergebnis von Interaktionen in anderen Feldern darzustellen. Als rein semantisches Element hat `output` kein Widget und keine Bildschirmpräsenz, solange kein Wert angegeben wird. Im einfachsten Fall kommt es ohne Attribute aus:

```
<output></output>
```

Das `output`-Element wird im Zusammenspiel mit JavaScript noch nützlicher. Seine Kerneigenschaften sind `value`, die seinen Wert erfasst oder festlegt, und `defaultValue` zum Erfassen oder Festlegen des Standardwerts (falls kein Wert angegeben wird). Zur Verdeutlichung der Funktionsweise habe ich ein kurzes Skript verfasst, das das `output`Element aktualisiert, sobald eine `range`-Eingabe verändert wird. Hier ist das Markup:

```
<label for="output">Output</label>
<input type="range" id="range">
<output id="output" for="range"></output>
```

Das folgende Skript wählt zunächst die Elemente aus, mit denen ich interagiere, gibt dann einen Standardwert von 50 für das `output`-Element vor und fügt schließlich einen Event-Listener zur `range`-Eingabe hinzu. Dieser löst aus, sobald sich der Wert verändert. Dieser Listener führt eine anonyme Funktion aus, um den `input`-Wert zu erfassen und den `output`-Wert zu setzen. Hier ist das fertige Skript, das Sie selbst in der Datei *input-types-output.html* ausprobieren können.

```
var range = document.getElementById('range'),
   output = document.getElementById('output');
output.defaultValue = 50;
range.addEventListener('change', function (e) {
  var newValue = e.currentTarget.value;
  output.value = newValue;
}, false);
```

8.6 Formularüberprüfung auf Anwenderseite

HTML5 kann Eingaben prüfen.

Die neuen Eingabetypen sind wirklich hilfreich – das größte Geschenk an die Entwickler macht HTML5 jedoch wohl mit der integrierten clientseitigen Fehlerüberprüfung. Aus Gründen der Sicherheit und Benutzerfreundlichkeit ist es enorm wichtig, die Inhalte eines Formulars vor der Übermittlung auf Fehler zu überprüfen. Bisher gab es dazu keine einfache Möglichkeit, obwohl Hunderte von JavaScript-Bibliotheken geschrieben wurden, um das zu erreichen.

Mit der nativen Formularüberprüfung warnen viele Browser nun automatisch, wenn der eingegebene Wert nicht dem Eingabetyp eines Felds entspricht. Wenn Sie zum Beispiel in Firefox nur Zahlen in ein email-Feld eingeben oder das *http://* zu Beginn einer URL in einem `url`-Eingabefeld weglassen, warnt Sie eine knallrote Umrahmung des Eingabefelds vor den fehlerhaften Werten (siehe Abb. 8–11).

Abb. 8–11
Diese E-Mail-Adresse ist nicht korrekt formatiert. Deshalb wird das Feld in Firefox von einem roten Rahmen umgeben.

Wenn Sie nun versuchen, das Formular abzusenden, erscheint eine

Fehlermeldung wie in Abb. 8–12 auf dem Bildschirm – zumindest, wenn Ihr Browser die anwenderseitige Validierung unterstützt. Jeder Browser hat hier seine eigene Implementierung.

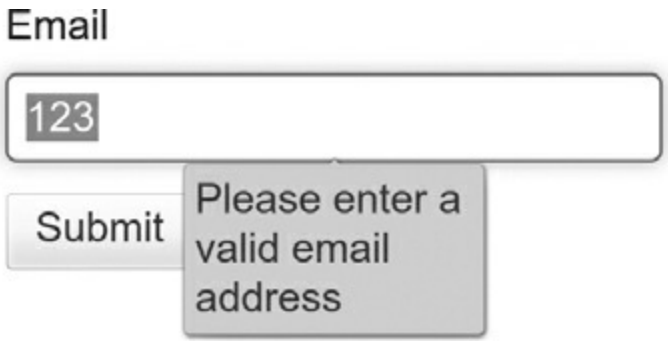


Abb. 8–12
Wenn das Feld nicht validiert werden kann, erscheint eine Warnmeldung (hier in Chrome).

Wenn Sie die Feldinhalte löschen und erneut auf *Absenden* klicken, erscheint kein Fehler. Dieses Feld ist standardmäßig optional – es ist also zulässig, keinen Wert anzugeben. Ein falsch formatierter Wert ist jedoch ungültig. Um das Feld erforderlich zu machen, können Sie das boolesche Attribut `required` hinzufügen:

```
<input type="email" required>
```

Mit diesem Attribut zwingen Sie den Browser, den Wert des Felds zu prüfen; ein leeres Feld oder ein falsch formatierter Wert sind ungültig und liefern einen Fehler zurück. Nur mit einem ordentlich formatierten Wert kann das Formular abgesendet werden.

Der Feldtyp bestimmt das Muster der Werteformats: `email` erfordert eine E-Mail-Adresse, `url` erfordert eine URL mitsamt Protokoll, `date` erfordert Jahr, Monat und Tag und so weiter. Sie können diese Anforderung jedoch mit dem `pattern`-Attribut aushebeln. Der Wert für `pattern` ist ein regulärer Ausdruck (*Regular Expression* oder *Regex*) – das ist eine in vielen Programmiersprachen gebräuchliche Standardmethode zum Abgleich von Zeichenketten. Als einfaches Beispiel möchten Sie vielleicht etwa nur Zahlen in einem `tel`-Eingabefeld zulassen:

```
<input type="tel" pattern="\d*">
```

Wenn Sie versuchen, dieses Feld mit Buchstaben oder nichtnumerischen Zeichen abzusenden, erhalten Sie eine andere Fehlermeldung, in der Sie gebeten werden, das korrekte Muster einzuhalten. Sie können diese Fehlermeldung in einigen Browsern anpassen, indem Sie zusätzlichen Text im `title`-Attribut bereitstellen.

```
<input type="tel" pattern="\d*" title="Numbers only">
```

In diesem Beispiel werden der Bildschirmnachricht die Worte »Numbers only« hinzugefügt (siehe Abb. 8–13).

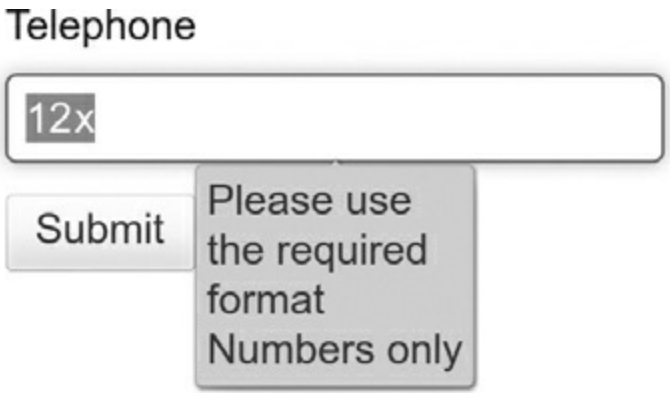


Abb. 8–13
Mit dem title-Attribut können der Warnung weitere Informationen hinzugefügt werden.

Es würde den Umfang und die Zielsetzung dieses Buchs bei Weitem sprengen, näher auf reguläre Ausdrücke einzugehen (ehrlich gesagt: Ich steige selbst kaum durch). Sie finden einige hilfreiche Online-Generatoren für reguläre Ausdrücke, die Ihnen bei Bedarf weiterhelfen können. Einer davon ist in den Literaturempfehlungen auf Seite 196 aufgeführt.

Wenn Sie die Überprüfung eines ganzen Formulars abschalten möchten, können Sie dies mit dem Attribut `novalidate` tun. Dieses Attribut unterdrückt sämtliche Überprüfungsroutinen, unabhängig vom `required`-Status oder der Mustererkennung:

```
<form action="foo" novalidate>...</form>
```

Dasselbe gelingt Ihnen auch auf einer lokaleren Ebene mit dem `formnovalidate`-Attribut für ein `input`- oder `button`-Element.

```
<button type="submit" formnovalidate>Go</button>
```

Diese Option ist hilfreich, wenn Sie die Möglichkeit haben möchten, etwas ohne vorherige Überprüfung abzusenden. In einem Content-Management-System könnte ein Nutzer beispielsweise eine Seite zur späteren Bearbeitung abspeichern, ohne sie zu veröffentlichen. Das ist ein üblicher Vorgang. Die Daten sind möglicherweise noch nicht vollständig und ungültig, der Anwender würde sich mit der Überprüfung also nur herumärgern.

8.7 Die Constraint-Validation-API

Eigene Prüfungen

Native Formularüberprüfung ist toll. Manchmal möchten Sie jedoch vielleicht noch mehr damit anfangen, indem Sie zum Beispiel noch eigene Überprüfungen hinzufügen oder Ihr eigenes Framework für Fehlermeldungen entwickeln. Die Constraint-Validation-API bietet eine Reihe von Objekten, Eigenschaften und Methoden, mit denen Sie das Überprüfungssystem des Browsers flexibel erweitern oder Ihr komplett eigenes System stricken können.

Die erste Eigenschaft ist `willValidate`. Sie liefert `true` oder `false` zurück, falls das Element, das sie aufgerufen hat, validiert wird – sie prüft nicht, ob dessen Wert gültig ist, sondern ob überhaupt eine Überprüfung stattfindet. Vielleicht hilft es, sich den Eigenschaftsnamen eher als *willBeValidated* zu denken. Standardmäßig liefern hier alle Formularelemente `true` zurück, es sei denn, sie wurden explizit von der Überprüfung ausgenommen, etwa mithilfe des `disabled`-Attributs.

Mit `willValidate` könnten Sie Aktionen gezielt nur auf Formularelemente anwenden, die überprüft werden:

```
var inputFields = document.querySelectorAll('input'),
   inputLen = inputFields.length,
   i;
for (i = 0; i < inputLen; i++) {
 if (inputFields[i].willValidate) {
   // Do something
 }
}
```

checkValidity()

Die einfachste Methode, ein Element in einem Formular zu überprüfen, ist die Anwendung der `checkValidity()`-Methode. Sie liefert `true` oder `false` zurück, je nachdem ob das Element mit seinem derzeitigen Wert gültig oder ungültig ist. Diese Methode ist das Herzstück des nachfolgenden Skripts, in dem die Funktion *checkStatus* dazu dient, die `checkValidity()`-Methode auszuführen und die Inhalte eines untergeordneten `p`-Elements je nach Überprüfungsstatus mit einem Häkchen oder einem Kreuz zu aktualisieren.

input

Die Funktion ist mithilfe eines Listeners für das neue `input`-Ereignis an die `email`-Eingabe geknüpft. Dieses Ereignis wird immer dann ausgelöst, wenn der Wert eines `input`- oder `textarea`-Elements aktualisiert wird – ein wenig wie beim `keypress`-Ereignis –, allerdings kann der Anwender hier ganze Textblöcke eingeben, bevor das Ereignis bei einer Eingabepause oder nach Auswahl eines automatischen Textvorschlags ausgelöst wird:

```
var email = document.getElementById('email');
function checkStatus(e) {
  var valid = e.currentTarget.checkValidity(),
  validMsg = e.currentTarget.nextSibling,
  status = (valid) ? ' ✓ ' : ' × ';
  validMsg.textContent = status;
}
email.addEventListener('input', checkStatus, false);
```

Sie können dies in der Beispieldatei *checkvalidity.html* ausprobieren. Diese wendet das Skript auch auf eine `tel`-Eingabe an, die nur Zahlen akzeptiert.

invalid

Neben dem `input`-Ereignis bietet uns HTML5 auch das `invalid`-Ereignis. Dieses wird für ein Element mit ungültigem Wert ausgelöst, wenn das Formular übertragen oder die `checkValidity()`-Methode ausgeführt wird. Es liefert ein Standard-`HTMLEvent`-Objekt zurück. Mit diesem Wissen könnte ich den vorhergehenden Code aktualisieren und zu jedem Feld einen zusätzlichen Event-Listener hinzufügen, der das `invalid`-Ereignis aufzeichnet, sobald es eintritt:

```
inputField[i].addEventListener('invalid', logInvalid, false);
```

Wenn Sie Informationen darüber erhalten möchten, warum ein Wert nicht für gültig erklärt wird, können Sie die `validity`-Eigenschaft verwenden. Diese Eigenschaft liefert ein `validityState`-Objekt mit einer Reihe gültigkeitsbezogener Eigenschaften zurück; ist das Feld zum Beispiel erforderlich und enthält keinen Wert, wird die Eigenschaft `value-Missing` als `true` zurückgeliefert; gibt es hingegen keine Fehler bei der Gültigkeitsprüfung, erhält die `valid`-Eigenschaft den Wert `true`.

Das nachfolgende Skript baut auf dem vorhergehenden auf, es zeigt eine Statusmeldung, wenn die Methode `checkValidity()` den Wert `false` zurückgibt. Mit der `validity`-Eigenschaft wird geprüft, ob einige gängige Status `true` sind, und gegebenenfalls wird eine entsprechende Nachricht ausgegeben. Andernfalls erscheint die Meldung »Unknown error«:

```
var email = document.getElementById('email');
function statusMsg(e) {
 var valid = e.currentTarget.checkValidity(),
    validStatus = e.currentTarget.validity,
    validMsg = e.currentTarget.nextSibling,
    status;
 if (valid) {
  status = ' ';
 } else {
  if (validStatus.patternMismatch) {
   status = 'Pattern mismatch';
  } else if (validStatus.typeMismatch) {
    status = 'Type mismatch';
  }
  ...
```

```
  }
  validMsg.textContent = status;
 }
email.addEventListener('input', statusMsg, false);
```

Sie können das Skript selbst in der Datei *validitystate.html* laufen lassen. Sehen Sie sich das `validityState`-Objekt in der Konsole an und betrachten Sie die verschiedenen `true`- und `false`-Werte. Abb. 8–14 zeigt ein Beispiel in Firebug.

window > **ValidityState**	
customError	false
patternMismatch	false
rangeOverflow	false
rangeUnderflow	false
stepMismatch	false
tooLong	false
typeMismatch	true
valid	false
valueMissing	false
constructor	**ValidityState { }**
* __proto__	**[xpconnect wrapped native prototype] { }**

Abb. 8–14
Das validityState-Objekt als Ausgabe in Firebug, Sie sehen den aktuellen Gültigkeitsstatus des Formularfelds.

Browser mit eingebauten Gültigkeitswarnmeldungen erzeugen ihre eigenen Bildschirmnachrichten. Übertragen Sie zum Beispiel ein leeres Eingabefeld mit `required`-Attribut, erscheint in Firefox eine Aufforderung zum Ausfüllen des Felds. Die `validationMessage`-Eigenschaft enthält diese Nachricht, falls das Feld einen ungültigen Zustand hat; ist der Wert im Feld gültig, enthält die `validationMessage`-Eigenschaft eine leere Zeichenkette.

setCustomValidity()

Diese Botschaft können Sie mit der Methode `setCustomValidity()` verändern, außerdem können Sie damit auch eine eigene Überprüfungsregel definieren. Im nachfolgenden Code habe ich zum

Beispiel zwei `tel`-Eingabetypen, *telHome* und *telWork*, und ich möchte sichergehen, dass die beiden Werte sich voneinander unterscheiden. Dazu füge ich zu *tel-Work* einen `input`-Listener hinzu. Sobald der Benutzer etwas in dieses Feld eingibt, überprüft die Regel, ob der Wert mit dem in *telHome* identisch ist. Trifft dies zu, wird eine benutzerdefinierte Fehlermeldung generiert:

```
var telHome = document.getElementById('tel-home'),
   telWork = document.getElementById('tel-work');
telWork.addEventListener('input', function (e) {
 if (e.currentTarget.value === telHome.value) {
  telWork.setCustomValidity('Must be different');
 } else {
   telWork.setCustomValidity('');
 }
}, false);
```

Sehen Sie sich *customvalidity.html* an, um das Skript in Aktion zu sehen. Das Ergebnis können Sie auch in Abb. 8–15 betrachten.

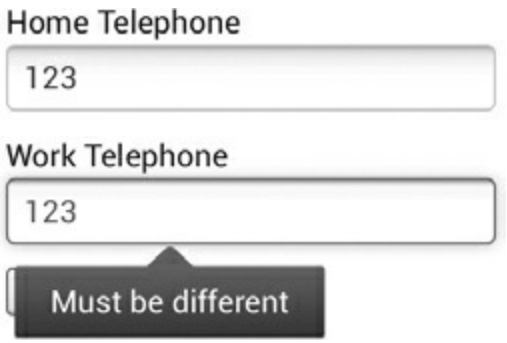


Abb. 8–15
Eine mit setCustomValidity() erstellte benutzerdefinierte Fehlermeldung

8.8 Formulare und CSS

Formularelemente abhängig vom Status formatieren

Eine Reihe neuer CSS-Pseudoklassen, die *UI-Element-Zustände (UI Element States*

Pseudoclasses), weiten die zusätzlichen Formularfunktionen in HTML5 noch mehr aus. Mit diesen Pseudoklassen können Sie Formularelemente aufgrund ihres aktuellen Interaktions- oder Gültigkeitsstatus formatieren. Die Namen der Pseudoklassen sind ziemlich selbsterklärend. Ich denke, ihre Bedeutung wird Ihnen klar sein – nur für den Fall, dass Sie einen harten Tag hatten und nicht mehr klar denken können, werde ich Sie Ihnen kurz erläutern.

Erforderliche Formularfelder (also solche mit auf *true* gesetztem `required`-Attribut) lassen sich mit der `:required`-Pseudoklasse formatieren, das Gegenstück dazu ist `:optional`. Sie könnten erforderlichen Feldern beispielsweise eine dunkle Umrahmung verpassen und optionalen Feldern dementsprechend eine hellere:

```
input:required { border-color: black; }
input:optional { border-color: gray; }
```

Die Selektoren `:valid` und `:invalid` werden auf Elemente entsprechend ihres derzeitigen Gültigkeitsstatus angewandt; `:valid` dient zur Formatierung eines gültigen Formularfelds und `:invalid` zur Formatierung eines ungültigen Formularfelds. Sie könnten daher zum Beispiel den Text jedes `input`-Elements ja nach dessen Status in einer anderen Farbe formatieren: Grün für gültige und Rot für ungültige Felder:

```
input:valid { color: green; }
input:invalid { color: red; }
```

Hinweis

Die letzte Spezifikation der CSS-Selektoren schlägt eine `:user-error`-Pseudoklasse vor, die `:invalid` ähnelt, jedoch nur angewandt wird, wenn vom Benutzer eingegebene Daten nicht dem geforderten Muster in einem Feld entsprechen. Zum aktuellen Zeitpunkt wurde diese Pseudoklasse noch in keinem Browser implementiert.

Formularelemente mit dem `disabled`-Attribut können Sie mit der

Pseudoklasse `:disabled` formatieren, umgekehrt dient `:enabled` zur Formatierung von Feldern mit `enabled`-Attribut. Ebenso lässt sich ein Formularelement mit auf *true* gesetztem `readonly`-Attribut mithilfe von `:read-only` formatieren, das Gegenstück dazu ist `:read-write`.

Die Eingabetypen `radio` und `checkbox` verfügen über eigene Pseudoklassen: `:indeterminate` gilt, wenn der Benutzer die Eingabe weder annoch abgewählt hat, und `:checked`, wenn der Benutzer die Eingabe angewählt hat. Seltsamerweise gibt es keine Entsprechung für das Abwählen der Eingabe, hier müssen Sie den `not()`-Selektor verwenden:

```
input[type="checkbox"]:not(:checked) { ... }
```

Für `number`- und `range`-Eingaben können Sie die Pseudoklassen `:in-range` und `:out-of-range` nutzen, die angewandt werden, wenn ein Wert innerhalb oder außerhalb des akzeptablen Bereichs liegt – entweder 0 bis 1 oder ein mithilfe von `max` und `min` selbst definierter Bereich. Kommen Bildschirmbedienelemente zum Einsatz, kann der Anwender diese Grenzen nicht überschreiten, der Selektor `:out-of-range` wird daher wohl eher selten zum Einsatz kommen.

8.9 Zusammenfassung

Verglichen mit den bescheidenen Anfängen sind Formulare in HTML5 nun extrem flexibel und leistungsfähig. Viele Funktionen, die die Entwickler über Jahre hinweg per JavaScript hinzufügten, werden nun nativ unterstützt. Alleine die neuen Eingabetypen sind eine willkommene Erweiterung, besonders für die Anwender von Bildschirmtastaturen, und die neuen Attribute erlauben eine bessere Kontrolle über die Formularelemente.

Das i-Tüpfelchen ist jedoch die eingebaute Fehlerüberprüfung – sie ersetzt viele verschiedene JS-Bibliotheken, die zur Lösung des Problems der anwenderseitigen Validierung geschrieben wurden. Das ist äußerst

praktisch; doch erst die Constraint-Validation-API gibt Entwicklern volle Handlungsfreiheit und ermöglicht browser- und plattformübergreifende, benutzerdefinierte Überprüfung auf vollkommen standardisierte Weise.

Mit den zusätzlichen Formatierungsmöglichkeiten durch die neuen CSS-Pseudoklassen sind HTML5-Formulare ein Paradebeispiel für die Leistungsfähigkeit der neuen Webplattform.

8.10 Literaturempfehlungen

Das stets hilfreiche MDN liefert einen präzisen und vollständigen *Leitfaden zu den neuen Eingabetypen*, zu finden ist er unter https://developer.mozilla.org/en-US/docs/HTML/Element/Input/. PPK bietet detaillierte *Tabellen zur Desktop-Unterstützung* unter http://www.quirksmode.org/html5/inputs.html. Das Ganze gibt es auch noch für Mobilgeräte unter http://www.quirksmode.org/html5/inputs_mobile.html.

Ryan Seddon hat ein *Polyfill für HTML5-Formularfunktionen in Browsern ohne native Unterstützung* geschrieben: Zu finden ist es unter https://github.com/ryanseddon/H5F/.

Bruce Lawsons Diskussion zu Autofokus und Barrierefreiheit, *The Accessibility of HTML 5 Autofocus* finden Sie auf seinem Blog unter http://www.brucelawson.co.uk/2009/the-accessibility-of-html-5-autofocus/.

HTML5 Rocks hat einen guten *Überblick über die Constraint-Validation-API*, zu finden unter http://www.html5rocks.com/en/tutorials/forms/constraintvalidation/

Rubular ist eine hilfreiche *Anwendung zum Testen regulärer Ausdrücke*. Sie zielt zwar auf Ruby ab, funktioniert aber ebenso gut für JavaScript

und HTML5-Formulare: http://rubular.com/.

Und Ihr bescheidener Autor hat bei HTML5 Doctor eine *Einführung in CSS3-Pseudoklassen* geschrieben: http://html5doctor.com/css3-pseudo-classes-and-html5-forms/.

9 Multimedia

Audio- und Videoinhalte erfreuen sich im Web auf Seiten wie YouTube, Dailymotion und SoundCloud seit Jahren großer Beliebtheit. Trotzdem waren sie immer Inhalte zweiter Klasse, und für ihre Funktion waren Plugins von Drittanbietern erforderlich (insbesondere Flash).

Für Websitebetreiber und Browserhersteller ist es natürlich keine gute Langzeitstrategie, sich von einem Plugin abhängig zu machen – nicht zuletzt deshalb, weil die Nutzer selbst für die Aktualisierung des Plugins sorgen müssen. Sicherlich ist es am besten, die Fähigkeit zur Audio- und Medienwiedergabe zusammen mit dem Browser zu aktualisieren, zumal die meisten Browser heute über einen regelmäßigen Aktualisierungszyklus verfügen.

Plugins können auch Stabilitätsprobleme verursachen, wie bekanntlich Steve Jobs anmerkte, als er erläuterte, warum iOS das Flash-Format nicht unterstützt:

> Wir wissen auch aus eigener Erfahrung, dass Flash der Hauptgrund für Systemabstürze bei Macs ist. Wir haben mit Adobe gemeinsam an der Lösung dieser Probleme gearbeitet, aber sie bestehen jetzt schon seit einigen Jahren. Wir wollen die Zuverlässigkeit und Sicherheit unserer iPhones, iPods und iPads nicht durch die Flash-Unterstützung einschränken.

Aus diesen Gründen ist die native Audio- und Videounterstützung im Browser so wichtig. Genau darauf zielt einer der großen Vorstöße von HTML5 in Form einiger heute bereits weithin in Browsern implementierten Medienelemente ab. Diese steuern die Wiedergabe von Audio und Video aus unterschiedlichen Quellen. Neben einer robusten HTML-Implementation bieten sie außerdem eine umfassende API und eine Reihe von Ereignissen, mit deren Hilfe Entwickler alle Funktionen exakt steuern können. Als Teil der Webplattform ist

ihr Zusammenspiel mit anderen Inhalten gut abgestimmt. Die Möglichkeiten reichen viel weiter als mit isolierten Plugins, und die neuen Medienelemente werden damit zu echten Webinhalten erster Klasse.

Nach der Untersuchung der Medienelemente werfe ich einen kurzen Blick auf die künftige Entwicklung von Multimedia im Web – vom Abmischen von Audio und Effekt-APIs bis hin zur Zukunft (oder jedenfalls der möglichen Zukunft) der Sprach-, Video- und Datenkommunikation in Echtzeit – dem WebRTC-Projekt.

9.1 Die Medienelemente

audio und video

Zum Abspielen von Audio- oder Videodateien benötigen Sie das `audio`-beziehungsweise das `video`-Element. Beide sind gleich aufgebaut und haben dieselben Attribute und Unterelemente. Ihr Hauptunterschied liegt in der Art der wiedergegebenen Medien. Im einfachsten Fall benötigt jedes Element nur ein einzelnes Attribut, `src`, das den Pfad zur abzuspielenden Mediendatei angibt:

```
<audio src="foo.oga"></audio>
<video src="foo.ogv"></video>
```

Was Sie zu sehen bekommen, hängt vom Element ab: Bei `video` sehen Sie das erste Bild der Videodatei in ihrer natürlichen Auflösung; bei `audio` sehen Sie gar nichts. Abspielen können Sie die Datei in keinem Fall, weil dazu die Bedienelemente auf dem Bildschirm fehlen. Diese fügen Sie mithilfe des `controls`-Attributs hinzu:

```
<audio src="foo.oga" controls></audio>
<video src="foo.ogv" controls></video>
```

Wie in Abb. 9–1 dargestellt, werden jetzt Bedienelemente für jedes der Medienelemente angezeigt. Sie können sich das auch selbst in der Datei *media-elements.html* ansehen. Ich schlage vor, Sie betrachten diese Beispieldatei in Chrome, Firefox oder Opera – die Gründe dafür werden

Sie in diesem Kapitel noch erkennen.

Mit diesen Attributen wird eine Mediendatei erst auf Anfrage des Benutzers und nur einmal abgespielt; bei Bedarf können Sie auch mit dem `autoplay`-Attribut festlegen, dass die Datei sofort nach dem Laden abgespielt wird. Mit dem `loop`-Attribut läuft sie außerdem in einer Dauerschleife (bis der Anwender auf Pause klickt bzw. tippt oder die Seite verlässt). Beides sind boolesche Attribute, die demnach keinen Wert erfordern:

```
<audio src="foo.oga" autoplay loop></audio>
```

Abb. 9–1
Native Bedienelemente für Video (oben) und Audio (unten) in Chrome für Ubuntu

WARNUNG

Automatisch abgespielte Sounds wirken sich nachteilig auf die Barrierefreiheit aus. Lesen Sie zu diesem Thema den auf Seite 214 unter »Literaturempfehlungen« aufgeführten Ratschlag »Audio Control advice« aus den WACG.

Auf einigen Mobilgeräten werden Mediendateien nicht eingebettet auf der Seite abgespielt; stattdessen öffnen sie sich im Medienplayer des Geräts. In solchen Fällen wird das `autoplay`-Attribut ignoriert.

muted

Bevor Sie eine Datei automatisch oder als Schleife abspielen, sollten Sie daran denken, dass der Ton einige Nutzer stören oder verwirren könnte. Überlegen Sie sich dann, ob Sie den Ton nicht lieber standardmäßig stumm schalten möchten. Dazu verwenden Sie das boolesche Attribut `muted`:

```
<video src="foo.ogv" autoplay muted></video>
```

preload

Normalerweise wird nicht die gesamte Mediendatei zusammen mit der Seite geladen. Das geschieht erst beim Starten der Wiedergabe. Stattdessen werden Datei-Metadaten wie Länge, Dateigröße und so weiter in den Speicher geladen. (Eine wichtige Ausnahme bildet dabei das `autoplay`-Attribut; wird es angewendet, muss die ganze Datei heruntergeladen werden). Dieses Verhalten ändern Sie bei Bedarf mithilfe des `preload`-Attributs mit seinen drei möglichen Werten: `metadata` ist der Standardwert, dessen Verhalten soeben beschrieben wurde, `auto` weist den Browser an, die Datei herunterzuladen, weil der Benutzer sie vermutlich abspielen wird, und `none` bedeutet, dass die Datei wahrscheinlich nicht verwendet wird und daher gar nichts geladen werden soll.

```
<video src="foo.ogv" preload="auto"></video>
```

Hinweise, keine Befehle!

Beachten Sie, dass dies nur Hinweise und keine Befehle sind; der Browser trifft am Ende selbst die Entscheidung darüber, ob Mediendateien heruntergeladen werden. Dazu wertet er Variablen wie die aktuelle Netzwerkverbindung und die zugrunde liegende Gerätehardware aus. Mobile Browser ignorieren den `auto`-Wert wahrscheinlich, denn der Download eines womöglich riesigen Videos verbietet sich bei einem volumenbeschränkten Internetzugang des

Nutzers von selbst. In vielen Fällen wird nur der Wert `metadata` akzeptiert.

9.1.1 Zusätzliche Attribute für das video-Element

height und width

Die Elemente `audio` und `video` haben viele Attribute gemeinsam. Letzteres fällt im Seitenlayout natürlich viel stärker auf und verfügt daher über einige zusätzliche Attribute. Videodateien werden standardmäßig in der Auflösung auf der Seite angezeigt, in der sie encodiert wurden. Das können Sie mit den Attributen `height` und `width` verändern:

```
<video src="foo.ogv" height="360" width="240"></video>
```

Beachten Sie, dass diese Attribute die Größe des `video`-Elements angeben, nicht die Größe des Videos selbst. Wenn die von Ihnen angegebenen Werte also vom Seitenverhältnis der Videodatei abweichen, wird das Video nicht gestreckt, um den Leerraum auszufüllen; es behält seine Größe vielmehr bei und wird in der Mitte des Elements zentriert. Sie können das in der Datei *video-dimensions.html* (und in Abb. 9–2) sehen. Möchten Sie dieses Verhalten gerne ändern, verwenden Sie dazu die Eigenschaften `object-fit` und `object-position` aus Kapitel 3.

Abb. 9–2
Die Videodatei behält ihr Seitenverhältnis bei, wenn die Größe des video-Elements verändert wird.

poster

Das `video`-Element ist standardmäßig leer oder zeigt eine plattformabhängige Grafik an, bis das erste Bild des Videoclips geladen wurde. Danach wird das erste Videobild als Standbild dargestellt. Wenn Sie gerne ein anderes Standbild oder ein komplett anderes Bild anzeigen möchten, geben Sie dies mit dem `poster`-Attribut an:

```
<video src="foo.ogv" poster="foo.png"></video>
```

9.1.2 Mehrere Quelldateien

Ogg Theora und Ogg Vorbis

In den bisher gezeigten Beispielen verwende ich die Dateinamenerweiterung *.ogv* für Videodateien und *.oga* für Audiodateien. Beide Erweiterungen gehören zum *Ogg*-Format. Ogg ist ein Containerformat, das demnach die unterschiedlichsten Video- und Audioformate enthalten kann; der gängigste Codec für Ogg-Video ist *Theora*, sein Audio-Pendant nennt sich *Vorbis*.

Sowohl Theora als auch Vorbis sind freie Formate – bei ihrer Verwendung fallen also keine Lizenzgebühren an. Damit scheinen sie prädestiniert für den Einsatz im Web. Und tatsächlich werden sie in einigen Sites, zum Beispiel Wikimedia, zur Codierung der Medieninhalte genutzt. Sie werden allerdings von nicht vielen Browsern unterstützt. Die Gründe dafür erkläre ich in »Krieg der Formate« auf Seite 203. Was können Sie also tun?

Sowohl für `audio`- als auch für `video`-Elemente können Sie mehrere Quelldateien angeben. Sie lassen das `src`-Attribute weg und verwenden stattdessen untergeordnete `source`-Elemente. Im nachfolgenden Beispiel gebe ich drei verschiedene Quelldateien für ein `video`-Element an: Das erste ist eine *.ogv*-Datei, das zweite eine *.webm*-Datei und das dritte eine

.mp4-Datei. Der Browser spielt das erste von ihm unterstützte Videoformat ab.

```
<video>
  <source src="foo.ogv" type="video/ogg"></source>
  <source src="foo.webm" type="video/webm"></source>
  <source src="foo.mp4" type="video/mp4"></source>
</video>
```

Das Beispiel *media-elements-sources.html* demonstriert diese Funktion; die Datei scheint sich nicht von *media-elements.html* zu unterscheiden – anders als in diesem ersten Beispiel sollte es nun jedoch egal sein, in welchem Browser Sie die Datei öffnen. Solange der Browser HTML5-Video unterstützt, müsste eine der bereitgestellten Quelldateien abgespielt werden.

type

Beachten Sie, dass ich das `type`-Attribut verwendet habe, das den MIME-Typ der verschiedenen Formate angibt und dem Browser mitteilt, welche Datei er abspielen soll, ohne dass der Browser dazu selbst auf die Datei zugreifen muss. Dieses Attribut ist verzichtbar, aber sein Einsatz hat sich bewährt (auch wenn einige ältere Browser Probleme mit dem `type`-Attribut hatten). Wie immer sollten Sie selbst testen, ob es zu irgendwelchen Problemen kommt.

media

Das `source`-Element besitzt ein `media`-Attribut, das eine Medienabfrage als Wert akzeptiert (siehe Kapitel 3). Die angegebene Datei wird also nur verwendet, wenn die Abfrage `true` zurückgibt – das ist praktisch, wenn Sie verschiedene Dateien für unterschiedliche Geräte bereitstellen. Sehr wahrscheinlich möchten Sie keine 1920×1080-Videodatei für einen 480×320-Bildschirm bereitstellen oder umgekehrt. Im nachfolgenden Codebeispiel wird eine andere Quelldatei, *foo-hd.ogv*, verwendet, sobald der Bildschirm mindestens 1920 Pixel breit ist und ein Seitenverhältnis von 16/9 besitzt:

```
<video>
  <source src="foo-hd.ogv" media="(device-aspect-ratio: 16/9) and
          (min-device-width: 1920px)"></source>
```

```
  <source src="foo.ogv"></source>
</video>
```

Der wahre Nutzen dieses Attributs bietet Anlass zu Kontroversen; manche halten es für eine bessere Lösung, die Videoqualität in Abhängigkeit von Geräteauflösung und Netzwerkstatus mithilfe von JavaScript anzupassen – dafür wurde auch bereits eine Spezifikation vorgeschlagen. Für die nahe Zukunft stellt das `media`-Attribut allerdings den besten Ansatz für die adaptive Videowiedergabe dar.

9.1.3 Fallbacks

Für den Fall, dass der Browser eines Anwenders keine Medienelemente unterstützt, sollten Sie eine Alternativlösung bereitstellen. Ansonsten bekommt der Nutzer nur einen leeren Seitenbereich zu sehen. Alles innerhalb des `video`-Elements wird dargestellt, falls das Video nicht verfügbar ist. Die Alternativlösung kann also so schlicht wie eine einzelne Textzeile sein, die das Problem erläutert. Oder vielleicht zeigen Sie stattdessen ein Standbild aus dem Video an.

```
<video src="foo.ogv">
  <img src="foo.png">
</video>
```

Krieg der Formate

Wie ich bereits einige Male erwähnt habe, gibt es kein einziges Audio- oder Videoformat, das Unterstützung in allen Browsern erfährt. Sie müssen also mehrere Quelldateien verwenden, um jedem gerecht zu werden. Die Gründe hierfür liegen im Patent- und Lizenzrecht.

Ich möchte nicht zu sehr ins Detail gehen, weil das ein wenig von unserem Thema abschweift. Kurz gesagt: Von den wichtigsten Browserherstellern legen Apple und Microsoft ihren Schwerpunkt auf das MP4-Format. Dieses Format ist patentiert und lizenziert, und es stellt schon beinahe einen Branchenstandard dar. MP4 unterstützt als

Container zahlreiche Formate – die wichtigsten darunter sind H.264 für Video und AAC für Audio.

Firefox und Opera können oder wollen keine kommerziellen Formate für ihre Browser lizenzieren, also haben sie die Ogg-Formate implementiert. Google implementierte sowohl Ogg als auch MP4, hat dann aber zusätzlich noch ein eigenes freies Format namens WebM herausgebracht, das auch von Opera und Firefox mit aufgenommen wurde. Apple und Microsoft beharren darauf, dass WebM ihre Patente verletze, Ogg nicht gut genug sei und MP4 die bessere Hardware-Performance biete. Wir befinden uns nun also in einer Art Pattsituation.

Grob gesagt sieht es nun so aus: Safari (für Desktops und alle iOS-Geräte), Android, Chrome, Internet Explorer und viele Smart-TVs unterstützen kommerziell lizenziertes MP3-Audio und MP4-Video. Chrome, Firefox und Opera unterstützen Ogg-Video. Diese drei sowie aktuellere Android-Geräte unterstützen Ogg-Audio und WebM, und auch die neuesten Smart-TVs von Samsung unterstützen WebM. Das Ganze ist eigentlich ein Albtraum.

Die Audio-Situation könnte sich mit einem neuen Format namens Opus klären. Dieses freie Format verletzt offenbar keine Patente, und es wird als Teil des WebRTC-Projekts eingeführt (das später in diesem Kapitel noch angesprochen wird). Für Video ist allerdings keine Lösung in Sicht. Wenn Sie also natives Video auf Ihrer Site verwenden, müssen Sie es mindestens zweifach codieren und mehrere Quelldateien bereitstellen – oder Sie denken über Flash als Alternativlösung nach.

Flash

Wenn die Medien unbedingt für jedermann zugänglich sein müssen, können Sie auch ein Flash-Objekt als Fallback verwenden; falls der Browser keine native Video- und Audiounterstützung bietet, kommt stattdessen das Plugin zum Einsatz. Und Sie können auch eine Alternative zur Alternative im `object`-Element platzieren, um alle Eventualitäten abzudecken.

```
<video src="foo.ogv" height="240" width="360" poster="foo.png">
 <object width="640" height="360"
         type="application/x-shockwave-flash" data="foo.swf">
  <param name="movie" value="foo.swf">
  <param name="flashvars"
    value="controlbar=over&amp;image=foo.png&amp;file=foo.mp4">
  <img src="foo.png" width="360" height="240">
 </object>
</video>
```

Dieser Code ist recht lang, deckt aber scheinbar alle Eventualitäten ab. Entwickelt wurde er von Kroc Camen, dessen Artikel »Video for Everybody!« ich Ihnen zur Lektüre empfehle. Dort finden Sie alle Anmerkungen und Vorbehalte, die ich hier nicht aufführen konnte (siehe »Literaturempfehlungen« auf Seite 214.

9.1.4 Untertitel und Beschriftungen

track

Sie können zur Mediendatei noch zusätzliche Textspuren laufen lassen; etwa Untertitel für Videodateien oder für beide Medientypen die zugehörigen Metadaten. Sie können beliebig viele Spuren verwenden – etwa für verschiedensprachige Untertitel oder eine Bildunterschriftspur und eine Metadaten-Spur –, jede Textspur wird mit einem untergeordneten `track`-Element hinzugefügt.

Dieses Element verfügt über einige Schlüsselattribute: `kind` teilt dem Browser den Zweck der Spur mit. Mögliche Werte sind etwa *subtitles*, *captions* und *descriptions*; `src` beschreibt den Pfad zur Datei; `srclang` gibt die Sprache der Spur an; `label` ist die Beschreibung der Spur, die dem Benutzer angezeigt wird, falls mehrere Spuren zur Auswahl stehen (und die Benutzeroberfläche des Browsers dies unterstützt). Der boolesche Wert `default` gibt bei mehreren Spuren die standardmäßig verwendete Spur an. Dies stellt sicher, dass eine Spur gezeigt wird, auch wenn nur die erste Instanz erkannt wird.

Das nachfolgende Beispiel zeigt, wie Sie zwei verschiedensprachige

Untertitel – Englisch und brasilianisches Portugiesisch – im `video`-element bereitstellen könnten:

```
<video src="foo.ogv">
  <track kind="subtitles" src="foo.en.vtt" srclang="en"
         label="English" default>
  <track kind="subtitles" src="foo.pt.vtt" srclang="pt-br"
         label="Brazilian Portuguese">
</video>
```

Ein Beispiel für Untertitel finden Sie in der Datei *video-subtitles.html* und in Abb. 9–3. Wenn die native Video-Benutzeroberfläche Ihres Browsers dies unterstützt, versuchen Sie, zwischen den Sprachen hin-und herzuschalten.

Abb. 9–3
Mit dem track-Element eingeblendete Untertitel

WebVTT

Je nach Browser werden unterschiedliche Dateitypen für Untertitel und Beschriftungen akzeptiert. Als Standard bildet sich *WebVTT* heraus – ein einfaches Format, das Untertitel von einer Start- bis zu einer Endzeit einblendet. Hier ein Beispiel:

```
WEBVTT
```

```
1
00:00:00.500 --> 00:00:04.000
Hallo, liebe Leser

2
00:00:06.000 --> 00:00:10.000
Danke, dass Sie dieses Buch lesen
```

Der erste Untertitel erscheint nach einer halben Sekunde auf dem Bildschirm und wird bis zur vierten Sekunde angezeigt; der zweite Untertitel erscheint in der sechsten Sekunde und bleibt bis zur zehnten Sekunde. Es gibt zwar zusätzliche Einstellungsmöglichkeiten, im Kern ist WebVTT jedoch sehr schlicht gehalten.

9.1.5 Codierung

FFmpeg

Beim Codieren (*encoding*) von Videos für das Web müssen Sie verschiedene Variablen berücksichtigen. Meiner Erfahrung nach lassen sich zum Beispiel manche für iOS codierten Videos nicht auf Android abspielen und umgekehrt. Wenn Sie sich in der Kommandozeile wohlfühlen, ist FFmpeg die beste Wahl: ein freier und extrem vielseitiger Open-Source-Encoder. Sie müssen den Pfad zur Quelldatei und eine Reihe (optionaler) Codierungsparameter übergeben und schließlich auch den Pfad zur Zieldatei. Hier sehen Sie ein einfaches Beispiel zur Umwandlung von AVI in MP4:

```
ffmpeg -i foo.avi foo.mp4
```

Miro Video Converter und HandBrake

Wenn Sie eine grafische Benutzeroberfläche bevorzugen, haben Sie eine breite Auswahl an Videokonvertern. Ich nutze zwei einfach bedienbare und kostenfreie Programme. Miro Video Converter gibt es für Mac oder Windows und ist sehr praktisch für schnelle Umwandlungen. Sie ziehen nur eine Datei ins Programmfenster, wählen einen Ausgabemodus aus einer Liste, die auch MP4, Ogg Theora und WebM enthält, und das Tool gibt Ihnen die

konvertierte Datei aus. Wenn Sie mehr Einfluss auf das Umwandlungsergebnis nehmen möchten, ist HandBrake wahrscheinlich das einfachste und mächtigste Werkzeug. Allerdings können Sie damit nur MP4 ausgeben und müssen daher möglicherweise doch noch auf Miro umsteigen. Bei beiden Werkzeugen handelt es sich um grafische Oberflächen für FFmpeg.

9.2 Media Fragments

Ausschnitte aus einer Mediendatei

Manchmal möchten Sie vielleicht nur einen Ausschnitt einer Mediendatei abspielen, ohne den Clip von Hand zu schneiden. JavaScript bietet Ihnen diese Möglichkeit (siehe nächster Abschnitt). Viele Browser unterstützen aber auch die Media Fragments URI. Diese Information wird an die URL der Mediendatei angehängt und dann werden einige Parameter dafür festgelegt.

Eine Zeitspanne auswählen

Zur Einstellung einer Zeitspanne wie in diesem Codebeispiel gehen Sie etwa auf diese Weise vor:

```
<audio src="foo.oga#t=4,8"></audio>
```

#t ist die Abkürzung für einen Zeitbereich. Die beiden kommagetrennten Werte danach sind Start- und Endzeit in Sekunden. In diesem Fall würde der Clip zwischen den Zeitmarken für vier und acht Sekunden abgespielt werden. Sie können eine der Zahlen weglassen; lassen Sie die erste weg, bedeutet das »vom Anfang bis hierhin abspielen«, lassen Sie die zweite Zahl weg, bedeutet es »von hier bis zum Ende abspielen«. In keinem Fall dürfen Sie aber das Komma weglassen. Um von der Zeitmarke 7,5 Sekunden bis zum Ende abzuspielen, verwenden Sie zum Beispiel den folgenden Code:

```
<video src="foo.oga#t=7.5,"></video>
```

Das Media-Fragments-URI-Modul enthält neben dem Zeitbereich noch

viele weitere Optionen, nur dieses wird jedoch bisher auf breiter Basis in den Browsern unterstützt.

9.3 Die Media-API

Interaktion

Ein Hauptkritikpunkt an Plugins zum Abspielen von Audio oder Video ist, dass es sich dabei um »dumme« Seitenelemente handelt; sie ermöglichen keine Interaktion, außer das Plugin stellt explizit Möglichkeiten dafür bereit. Die Medienelemente in HTML5 bieten jedoch den Vorteil einer umfangreichen API, über die Sie Informationen zur Mediendatei abrufen können und die Interaktionen erleichtert. Der Zugriff auf die API erfolgt über eine Reihe von Schnittstellen: Die Medienelemente besitzen in der `HTMLMediaElement`-Schnittstelle gemeinsame Eigenschaften und Methoden; für Audio gibt es zudem die einzigartige Schnittstelle `HTMLAudioElement` und für Video `HTMLVideoElement`.

play() und pause()

Die beiden zuerst vorgestellten Methoden dienen wichtigen Grundfunktionen: `play()` und `pause()` werden einfach auf das Medienelement angewandt. Falls Sie selbst entwickelte Steuerelemente auf der Seite verwenden, benötigen Sie am Ende wohl ein Skript wie das folgende, um sie zu kontrollieren:

```
var pause = document.getElementById('pause'),
   play = document.getElementById('play'),
   video = document.getElementById('video");
pause.addEventListener('click', function () {
 video.pause();
}, false;
play.addEventListener('click', function () {
  video.play();
}, false;
```

Getrennte Schaltflächen erscheinen etwas verschwenderisch, effizienter ist es, sie zu einer Play/Pause-Taste zusammenzufassen. Dies gelingt mit

einem zusätzlichen `if...else`-Ausdruck im Code, in dem Sie mit der booleschen `paused`-Eigenschaft überprüfen, ob sich das Video im Pausenzustand befindet. Als Beispiel sehen Sie sich die Datei *media-play-pause.html* und Abb. 9–4 an.

```
var playPause = document.getElementById('play-pause'),
   video = document.getElementById('video');
playPause.addEventListener('click', function () {
 if (video.paused) {
  video.play();
 } else {
  video.pause();
 }
}, false);
```

Abb. 9–4
Ein einfacher Play/ Pause-Umschalter, der über die Media-Elements-API realisiert wurde

Die Attribute von Medienelementen werden in der API gespiegelt, Sie können sie also dynamisch aktualisieren: `preload` kann das `preload`-Attribut auslesen oder einstellen, und es akzeptiert dieselben Werte; `autoplay`, `controls`, `loop` und `muted` setzen die entsprechenden booleschen Attribute, und `src` aktualisiert den Pfad auf eine andere Mediendatei.

Wenn Sie den Pfad zu einer neuen Datei angeben, müssen Sie diese mit der `load()`-Methode in den Puffer laden:

```
video.src = 'bar.oga'
; video.load();
```

currentSrc, currentTime

Weitere Informationen über die Mediendatei lassen sich mithilfe anderer Eigenschaften abrufen. Die Eigenschaft `currentSrc` liefert die URL der aktuell abgespielten Datei zurück; falls Sie mehrere `source`-Elemente haben, erfahren Sie so, auf welches Element der Browser aktuell zugreift (abhängig von den Formaten und Medienabfragen). Die Eigenschaft `currentTime` liefert einen Zeitwert das aktuellen Wiedergabepunkts in Sekunden; Sie können mit dieser Eigenschaft auch eine Zeit vorgeben – das ist praktisch, wenn Sie eine eigene Steuerung mit Positionsbalken umsetzen möchten:

```
video.currentTime = 4.5;
```

Wenn der Nutzer den Positionsbalken verwendet und die Medienwiedergabe an anderer Stelle fortgesetzt werden soll, liefert die Eigenschaft `seeking` den Wert `true` zurück. Die Medienlänge erhalten Sie mit der `duration`-Eigenschaft und die Zeit, an der die Wiedergabe beginnt, mit `initialTime`; beide liefern ebenfalls einen Sekundenwert zurück.

Beispiel

Im nachfolgenden Code sehen Sie ein Anwendungsbeispiel für diese Eigenschaften. Darin habe ich das `video`-Element und ein `progress`-Element zur Messung des Wiedergabefortschritts im Medium definiert. Ich möchte gerne den Wert des Fortschrittsbalkens so verändern, dass der Wiedergabefortschritt des Videos angezeigt wird; dieses Verhältnis ergibt sich aus dem Quotienten der Werte `currentTime` und `duration`.

Damit dies funktioniert, nutze ich das neue `timeupdate`-Ereignis. Ich gehe darauf noch detaillierter im Abschnitt »Medienereignisse« auf Seite 212 ein. Vorerst müssen Sie nur wissen, dass das Ereignis ausgelöst wird,

sobald sich die aktuelle Abspielzeit der Mediendatei verändert (wie es etwa beim Abspielen geschieht). Wird das Ereignis ausgelöst, aktualisiert sich der Fortschrittsbalken und zeigt die aktuelle Wiedergabeposition an. Sie können dies selbst in der Datei *media-progress.html* ausprobieren, die in Abb. 9–5 dargestellt ist.

```
var progress = document.querySelector('progress'),
   video = document.querySelector ('video');
video.addEventListener('timeupdate', function () {
 progress.value = video.currentTime / video.duration;;
}, false);
```

Abb. 9–5
Ein Fortschrittsbalken verwendet die Timing-Eigenschaften der Media-Elements-API.

volume

Sie können die Lautstärke mit der `volume`-Eigenschaft abrufen oder einstellen, ihr Wert bewegt sich zwischen 0 und 1. Diese Eigenschaft eignet sich in Kombination mit einem `range`-Element zum Erzeugen eines eigenen Bedienelements (siehe nachfolgendes Codebeispiel). Ändert das `range`-Element seinen Wert, wird das `change`-Ereignis ausgelöst und dadurch die `volume`-Eigenschaft auf den aktuellen

Wert geteilt durch 100 aktualisiert (um ihn an den Lautstärkebereich anzupassen). Probieren Sie es selbst in der Datei *media-volume.html* (siehe Abb. 9–6):

```
var video = document.querySelector('video'),
   volume = document.getElementById('range');
volume.addEventListener('change', function (e) {
   video.volume = e.currentTarget.value / 100;
}, false);
```

Abb. 9–6
Steuerung der Videolautstärke mit dem range-Element. Nun haben Sie eine komplette einfache Videosteuerung aus HTML-Elementen und der Media-Elements-API angelegt.

Die letzten drei Codebeispiele zeigen, wie einfach Sie eine eigene Mediensteuerung aus HTML5-Formularen and UI-Elementen erstellen können. Mit etwas Zusatzarbeit können Sie alle grundlegenden Mediensteuerungen umsetzen und sogar noch weiter gehen und komplett maßgeschneiderte Benutzerschnittstellen in Ihre Websites integrieren.

9.3.1 Ladestatus

Mediendateien sind oft sehr groß und werden nicht in einem Durchgang Ein paar Dinge sollten Sie wissen – den aktuellen Ladestatus und ob die Datei abspielbereit ist.

networkState

Den Ladezustand ermitteln Sie mit der `networkState`-Eigenschaft, die vier Zustände annehmen kann: 0 bedeutet, dass noch keine Daten vorliegen, 1 bedeutet, dass sich das Netzwerk im Leerlauf befindet, 2 bedeutet, dass das Medium geladen wird und 3 heißt, das Medium wurde fertig geladen. Mit dieser Eigenschaft könnten Sie auf dem Bildschirm eine Ladeanzeige für das Medium erstellen:

```
if (video.networkState === 2) { ... }
```

readyState

Noch hilfreicher ist womöglich die `readyState`-Eigenschaft. Wie die vorhergehende Eigenschaft besitzt auch sie Statuswerte; diese geben allerdings an, ob das Medium abspielbereit ist – ob die Metadaten oder die gesamte Datei geladen wurden und so weiter. Diese fünf Zustände gibt es:

- `0` gibt an, dass keine Medieninformation vorliegt.
- `1` gibt an, dass die Metadaten des Mediums geladen wurden.
- `2` gibt an, dass Daten zum aktuellen Einzelbild an der Abspielposition vorliegen.
- `3` gibt an, dass Daten zum aktuellen Einzelbild und wenigstens dem nachfolgenden Einzelbild vorliegen.
- `4` gibt an, dass ausreichend Daten und Übertragungsbandbreite vorhanden sind, um die Mediendatei bis zum Ende abzuspielen.

Manchmal wollen Sie eine Funktion vielleicht nur ausführen, wenn Metadaten geladen wurden, etwa um die Länge eines Videos zu ermitteln. Dazu müssen Sie prüfen, ob der Wert für `readyState` zumindest 1 beträgt:

```
if (video.readyState > 0) { ... }
```

Hierfür sind permanente Abfragen erforderlich (vielleicht mit Hilfe von `setInterval()`). Diese Lösung ist daher nicht immer optimal. Besser ist es, den Browser dies mit Hilfe eines Ereignisses melden zu lassen. – ich beschreibe dies weiter unten im Abschnitt 9.4, »Medienereignisse«.

9.3.2 Zusätzliche Eigenschaften für Audio und Video

HTMLAudioElement

Neben den gemeinsamen Eigenschaften und Methoden aller Medientypen verfügen Audio und Video auch noch über eine eigene Schnittstelle. Die `HTMLAudioElement`-Schnittstelle bietet nur eine einzige weitere Methode: `audio()`. Dies ist ein Konstruktor zum Anlegen eines neuen `audio`Elements. Bei Bedarf können Sie noch eine Quell-URL als Argument verwenden:

```
var audio = new Audio('foo.oga');
```

HTMLVideoElement

Die Schnittstelle `HTMLVideoElement` bietet eine Reihe von Zusatzeigenschaften, die sich auf das Erscheinungsbild des Videos auswirken. Mit der `poster`-Eigenschaft lesen oder setzen Sie das `poster`-Attribut. Die verbleibenden vier Attribute betreffen die Abmessungen: `height` und `width` dienen zur Dimensionierung des Elements, während `videoHeight` und `videoWidth` die Abmessungen des im Elements angezeigten Videos beschreiben.

9.4 Medienereignisse

Media Events

Beim Laden, Abspielen und Interagieren mit Medienedateien werden JavaScript-Ereignisse ausgelöst. Diese bieten reichliche Möglichkeiten zum Anhängen von Ereignisbehandlungsroutinen. Beginnen wir mit Ereignissen für die unterschiedlichen Statuswerte der

`readyState`-Eigenschaft: Statuswert 1 entspricht `loadedmetadata`, Statuswert 2 `loadeddata`, Statuswert 3 `canplay` und Statuswert 4 `canplaythrough`.

Der nachfolgende Code schreibt zum Beispiel eine Nachricht in die Konsole, wenn ein Video den Statuswert 4 erreicht hat – wenn also genug Daten vorhanden sind, um das Medium bis zum Ende abzuspielen:

```
video.addEventListener('canplaythrough', function () {
  console.log('Can play through.');
}, false);
```

Für den Wiedergabestatus wird `playing` ausgelöst, wenn das Medium erstmals abgespielt wird und `pause`, wenn die Wiedergabe angehalten wird. Wird die Wiedergabe nach einer Pause fortgesetzt, wird das Ereignis `play` ausgelöst, und am Dateiende wird `ended` ausgelöst. Wird die Wiedergabe aus irgendeinem Grund unterbrochen – wenn etwa das Medium abgespielt wird, der Benutzer es aber manuell erneut von Anfang an starten lässt – wird das Ereignis `abort` ausgelöst.

Das `volumechange`-Ereignis tritt ein, wenn sich der Wert der `volume`-Eigenschaft ändert oder das `muted`-Attribut umgeschaltet wird, `seeking` tritt bei Verwendung des Positionsbalkens ein und `seeked`, wenn die neue Position gefunden wurde. Bei Aktualisierung der `currentTime`-Eigenschaft wird das Ereignis `timeupdate` ausgelöst.

9.5 Erweiterte Medieninteraktion

Allein die beiden bisher in diesem Kapitel angesprochenen Medienelemente (und die dazugehörigen APIs) bieten viel mehr Möglichkeiten für Medien als die Entwickler in der Vergangenheit jemals hatten – und doch ist das erst der Anfang. Es gibt bereits Pläne für noch viel feinere Kontrollmöglichkeiten und zusätzliche Erweiterungen zur nativen Medienwiedergabe im Browser. An der Spitze stehen dabei erweiterte Audiofähigkeiten und Peer-to-Peer-Datenverbindungen.

9.5.1 Web-Audio-API

Klänge erzeugen und bearbeiten

Für alle, die über die einfache Wiedergabe von Audiodateien im Web hinausgehen wollen, zielt ein aufkommender Standard namens Web-Audio-API auf die komplexe Behandlung und Synthetisierung von Klängen in Webanwendungen. Die Web-Audio-API basiert auf dem Konzept von Audio Routes, einem gängigen Werkzeug in der Tontechnik (aber für mich viel zu hoch!).

Wie das `canvas`-Element verwendet die Web-Audio-API einen Kontext, der mit der `AudioContext()`-Method konstruiert wird:

```
var context = new AudioContext();
```

Es würde den Horizont dieses Buchs (und leider auch meinen eigenen) weit überschreiben, weiter auf dieses Thema einzugehen. Bei Interesse an der erweiterten Audiointeraktion empfehle ich Ihnen, den Artikel »Web Audio API – Getting Started« auf der CreativeJS-Site zu lesen (siehe »Literaturempfehlungen« auf Seite 214).

9.5.2 WebRTC

Webgeräte sollen in Echtzeit kommunizieren.

In Kapitel 6 hatten wir uns die `getUserMedia()`-Methode angesehen. Ich erwähnte dort, dass sie Teil des größer angelegten WebRTC-Projekts ist. WebRTC ist ein spannender Vorschlag, der darauf abzielt, dass alle mit dem Web verbundenen Geräte untereinander in Echtzeit kommunizieren können – mit Audio, Video und Daten, und das alles über ein einzelnes Standardprotokoll.

Momentan erlauben dies viele Tools, doch sie alle sind auf Plugins oder Zusatzprogramme angewiesen. Und nur sehr wenige verwenden tatsächlich dasselbe Protokoll und können sich auch miteinander unterhalten. So entsteht eine Reihe »ummauerter Räume«, zwischen

denen keine Datenübertragung möglich ist. WebRTC möchte diese Mauern niederreißen.

WebRTC verfügt über drei Schlüssel-APIs: *MediaStream* erlaubt den Zugriff auf Datenströme, etwa von einer Kamera oder einem Mikrofon (mittels `getUserMedia()`), *PeerConnection* ermöglicht die Sprachoder Videokommunikation zwischen Geräten und *DataChannel* dient der allgemeinen Datenübertragung. Während ich dies schreibe, bieten Chrome und Firefox bereits eine experimentelle Unterstützung für alle drei, und Opera unterstützt MediaStream über `getUserMedia()`.

Wie so oft in der Medienlandschaft des Web blickt aber selbst die derzeitige WebRTC-Spezifikation in eine ungewisse Zukunft. Von Microsoft wurde auf Grundlage der Erfahrungen mit der übernommenen Skype-Plattform eine alternative Spezifikation, CU-RTC-Web, vorgeschlagen. Es ist wahrscheinlich vergebliche Liebesmühe, eine der beiden Spezifikationen näher zu beleuchten. Darum belasse ich es hierbei: Echtzeitkommunikation im Web wird kommen, auch wenn die genaue Form bisher noch nicht bekannt ist.

9.6 Zusammenfassung

In diesem Kapitel haben Sie Möglichkeiten kennengelernt, Medien auf Ihrer Seite mit `audio`- und `video`-Elementen darzustellen. Dazu kommen zahlreiche Attribute zur Steuerung der Mediendarstellung. Sie haben auch die Media-API kennengelernt, die tiefgreifende Interaktionen mit den Medienelementen erlaubt, und dazu kommt noch die große Bandbreite an Ereignissen, die bei Medieninteraktionen ausgelöst werden.

Ich habe auch die zukünftigen Audio-APIs kurz angesprochen – ein Thema, das ich gerne genauer beleuchtet hätte, wovon mich das unsichere Umfeld aber abhält. Schließlich habe ich das WebRTC-Projekt angesprochen, eine spannende Sammlung von APIs, die uns ganz erstaunliche neue Möglichkeiten zum Austausch von Video, Audio und

Daten zwischen Geräten und ohne Drittanbieter-Plugins bereitstellen möchte. Dieses Projekt ist definitiv ein eigenes Buch wert.

9.7 Literaturempfehlungen

Den gesamten Text von Steve Jobs *Thoughts on Flash* finden Sie unter http://www.apple.com/hotnews/thoughts-on-flash/.

Hinweise zur Barrierefreiheit von Audio und Video im Zusammenhang mit automatischer Wiedergabe erhalten Sie auf der Audio-Control-Seite der WCAG unter http://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-dis-audio.html.

MDN liefert unter https://developer.mozilla.org/en-US/docs/Media_formats_supported_by_the_audio_and_video_eleme *Tabellen zum aktuellen Implementations-Status von Medienformaten in verschiedenen Browsern*, und Kroc Camens Technik und Anmerkungen zur Implementation für verschiedene Browser *Video for Everybody!* gibt es auf *http://camendesign.co.uk/code/video_for_everybody/*.

Die beste *Einführung zu WebVTT und das track-Element* finden Sie auf Dev.Opera unter http://dev.opera.com/articles/view/an-introduction-to-webvtt-and-track/, und auf Anne van Kesterens Website wartet ein nützliches *WebVTT-Validierungswerkzeug:* http://quuz.org/webvtt/.

Ich habe in diesem Kapitel drei Codierungswerkzeuge angesprochen: Das *Kommandozeilen-Tool FFmpeg* gibt es auf http://ffmpeg.org/, den *Miro Video Converter* unter http://www.mirovideoconverter.com/ und *HandBrake* unter http://handbrake.fr/.

Die *Media-Fragments-Spezifikation des W3C* finden Sie unter

http://www.w3.org/TR/media-frags/.

MDN bietet die besten *Dokumentationen für die Media-API und Ereignisse*, die ich gefunden habe: https://developer.mozilla.org/en-US/docs/DOM/HTMLMediaElement/beziehungsweise https://developer.mozilla.org/en-US/docs/DOM/Media_events/.

CreativeJS liefert unter http://creativejs.com/resources/web-audio-api-getting-started/ einen tollen *Einführungsartikel zur Web-Audio-API*.

Über WebRTC werden Sie in Zukunft sicherlich noch viel mehr hören, Sam Dutton hat bereits eine gute Einführung dazu auf *HTML5 Rocks* geschrieben: http://www.html5rocks.com/en/tutorials/webrtc/basics/. Microsofts *Einführung zum CU-RTC-Web-Vorschlag* finden Sie im Interoperability-Blog unter http://blogs.msdn.com/b/interoperability/archive/2012/07/28/custon ubiquitous-real-time-communication-over-the-web-cu-rtc-web.aspx.

10 Web-Apps

Bisher ging es in diesem Buch hauptsächlich um Webtechnologien, die überall eingesetzt werden können – von offenen Websites bis hin zu gerätespezifischen Webanwendungen. Im vorliegenden Kapitel mache ich einen kleinen Schlenker und spreche über die zusätzlichen Schritte zur Erstellung von Web-Apps – wie Sie also Ihre Site anpassen können, um sie über einen App-Store verfügbar zu machen oder um sogar selbst dafür zu sorgen, dass sie auf einem Gerät installiert werden kann.

Ehe wir die technischen Hintergründe der App-Entwicklung besprechen, sollten wir zunächst die wichtigsten Haupttypen definieren:

- Native Apps *kommen ohne Webtechnologien aus. Sie werden speziell für bestimmte Plattformen wie OS X, iOS, Windows und Android programmiert.*
- Web-Apps *nutzen Sprachen der Webplattform. Zu dieser Kategorie zählen gehostete Websites oder in einem komprimierten Container enthaltene Websites, die dann auf einem Gerät installiert werden.*
- Hybrid-Apps *verwenden ebenfalls Webtechnologien, sie werden aber in einem nativen Container verpackt (oder »gewrappt«).*

In diesem Kapitel gehe ich besonders auf die beiden letzten App-Typen ein, also auf die Web- und Hybrid-Apps. Sie erfahren auch, wie sie diese in einigen der zahlreichen App-Stores anbieten können. Doch ist selbst das nicht so einfach, wie es klingt: Es gibt viele rivalisierende Plattformen und Stores, und alle verwenden unterschiedliche Entwicklungsplattformen und Einreichungsprozeduren. Es gibt keine Standardmethode zur Definition einer App (obwohl dazu einige Vorschläge existieren).

Glücklicherweise haben viele neue App-Stores erkannt, dass sie durch den Einsatz von Webtechnologien Entwickler anziehen können: Es bedeutet eine große Zeit- und Geldersparnis, wenn nicht für jede einzelne Plattform eine

eigene App geschrieben werden muss. Viele Anbieter wie Microsoft (Windows 8), Samsung (Smart TV) und Opera (Opera TV Store) betreiben ihre Apps mithilfe von Technologien der Webplattform. Eine einmal geschriebene App läuft bisher trotzdem noch nicht überall: Sie müssen immer noch die speziellen APIs jedes einzelnen Geräts sowie unterschiedlich ausgebaute Browserunterstützungen berücksichtigen. Trotzdem kann die Entwicklung einer Web-App deutlich effizienter sein, als jeweils native Apps für Android (in Java) oder für iOS (in Objective-C) zu schreiben.

Neben den verschiedenen Methoden zur App-Erstellung betrachte ich auch Möglichkeiten, Ressourcen offline zur Verfügung zu stellen und sie auf dem Gerät des Anwenders zu speichern. Für den Fall, dass einmal keine Internetverbindung vorhanden ist, was selbst im heutigen Drahtloszeitalter noch recht häufig vorkommt, ist das praktisch. Sie erreichen dies mithilfe des Application Cache (gemeinhin unter der Bezeichnung AppCache *bekannt). Dieser ist zwar recht leistungsfähig, hat aber dennoch seine Tücken.*

10.1 Web-Apps

Zunächst möchte ich die Web-Apps betrachten, also Apps, die komplett auf Technologien der Webplattform basieren. Trotz ihrer Bezeichnung müssen sie aber nicht notwendigerweise im offenen Internet zu finden sein; sie lassen sich auch verpacken und über Online-Stores zum Download anbieten. Es gibt natürlich viele App-Portale, aber ich betrachte insbesondere die beiden mit den geringsten Einstiegshürden: den Chrome Web Store, den es schon seit ein paar Jahren gibt, und den neueren Firefox Marketplace. Beide werden von Browserherstellern betrieben; der große Unterschied zwischen ihnen ist jedoch, dass der Chrome Web Store nur im Chrome-Browser funktioniert, während der Firefox Marketplace einen offenen Standard anstrebt, den jeder Browser verwenden kann.

10.1.1 Hosted und Packaged Apps

Ehe wir fortfahren und uns die verschiedenen Anforderungen der App-Stores ansehen, wollen wir erst einige Minuten Zeit investieren und zwei verschiedene Untertypen von Web-Apps definieren: *Hosted Apps* und *Packaged Apps*. Bei den meisten App-Stores können Sie zwar beide Varianten einreichen, doch hat jeder App-Typ bestimmte Vor- oder Nachteile (meist geht es dabei um Einschränkungen oder Berechtigungen für den Zugriff auf Kernfunktionen des Geräts). Wenn Sie diese im Vorfeld kennen, entscheiden Sie sich leichter für den richtigen Ansatz zur Entwicklung Ihrer App.

Hosted Apps

Dateien auf einem externen Server

Bei der gehosteten App liegen alle Dateien auf einem externen Server, auf den meist über eine öffentliche URL zugegriffen werden kann. Eine solche Hosted App ist im Prinzip wie eine Website mit ein paar zusätzlichen Metadaten zur Indexierung durch App-Stores. Gehostete Web-Apps werden nicht auf dem Gerät installiert; dort existieren nur Verknüpfungen, die einen Browser oder einen eingebetteten Web-Viewer starten, sobald der Benutzer die App aufruft.

Wenn Sie eine Web-App selbst hosten, können Sie ohne erneute Genehmigung des Stores Aktualisierungen durchführen (aus Entwicklersicht durchaus ein Vorteil). Als Kehrseite ergibt sich jedoch ein erhöhtes Sicherheitsrisiko (bösartige Entwickler könnten schädlichen Code einschmuggeln). Gehostete Apps erhalten daher allgemein keinen Zugriff auf geschützte Geräte-APIs.

Packaged Apps

Dateien auf dem Gerät

Im Gegensatz zu gehosteten Apps enthält eine Packaged App alle zu ihrer Ausführung erforderlichen Ressourcen in einer einzigen (meist gezippten) Archivdatei. Die enthaltenen Dateien werden auf dem

Gerät installiert – die App kann sich zwar mit externen Webdiensten verbinden, ist aber nicht ständig auf diese Verbindung angewiesen. Wenn Sie Ihre App auf diese Weise verpacken, kann der Store-Betreiber ihre Inhalte begutachten und autorisieren. Falls die App für sicher befunden wird, können erweiterte Zugriffsrechte gewährt werden. In der Praxis bedeutet das für den Entwickler Zugriff auf geschützte Geräte-APIs, wie etwa jene für Adressbuch oder Textnachrichtendienste.

Packaged Apps lassen sich schneller aufrufen und schließen als gehostete Apps, weil sich alle erforderlichen Dateien auf dem Gerät befinden. Sie sind also weniger von der Netzwerkleistung abhängig. Der Entwickler muss jedoch dafür sorgen, dass die Daten lokal gespeichert und im Online-Betrieb mit dem Server synchronisiert werden können. Der Entwicklungsaufwand erhöht sich dadurch. Sie können dazu auf Local Storage zurückgreifen (siehe Kapitel 6), weitergehende Möglichkeiten bei der Offline-Verfügbarkeit erfordern aber möglicherweise den Einsatz von AppCache, worauf ich im Abschnitt »Application Cache« auf Seite 227 eingehe.

10.1.2 Manifest-Dateien

Egal, ob eine App online oder als Installer-Paket vorliegt – sobald sie in einem App-Store angeboten werden soll, muss sie einige Daten enthalten, die sie als zu diesem bestimmten Store gehörige App kennzeichnen. Zur Formatierung oder Präsentation dieser Daten gibt es bestimmte Vorgaben. Meist genügt es, einige verbindliche Datenfelder auszufüllen und ein Symbol zur Anzeige auf dem Zielgerät beizufügen. Beides gehört in die sogenannte *Manifest-Datei*.

Die Manifest-Datei ist eine einfache XML- oder JSON-Textdatei, die sich irgendwo im Verzeichnis mit Ihren App-Ressourcen befindet. Sie enthält Schlüsselinformationen über die App, etwa ihren Namen und eine Beschreibung. Die geforderten Informationen variieren allerdings von Store zu Store etwas, wie ich in den folgenden Abschnitten näher ausführe.

Der Chrome Web Store

JSON-Manifest

Zur Veröffentlichung im Chrome Web Store muss die Manifest-Datei im JSON-Format vorliegen, den Namen *manifest.json* tragen und im Verzeichnis *myapp* im Wurzelverzeichnis der App gespeichert werden. Der nachfolgende Code mit Anmerkungen zeigt ein kurzes, minimalistisches Beispiel von *manifest.json* für den Chrome Web Store.

```
   {
❶  "name": "App Name",
❷  "description": "Short description",
❸  "version": "0.1",
❹  "manifest_version": 2,
❺  "icons":{
      "128": "icon_128.png"
    },
❻  "app": {
     "launch": {
      "web_url": "http://foo.example.com/"
    }
   }
}
```

Der erste Eintrag ❶ im Manifest ist der Name Ihrer App. Als Nächstes ❷ folgt die Beschreibung, die maximal 132 Zeichen lang sein sollte und die kein HTML enthalten darf. In der `version`-Eigenschaft ❸ geben Sie die Versionsnummer Ihrer App an, was hilfreich für Bug-Reports ist und um Anwender über Aktualisierungen zu informieren. Die Manifest-Version ❹ teilt dem Chrome Web Store mit, dass Sie das aktuellste Manifest-Format verwenden (Version 1 wird seit Januar 2014 nicht mehr unterstützt).

Das `icons`-Objekt ❺ enthält eine Liste von Bilddateien zur Darstellung in unterschiedlichen Auflösungen je nach Anzeigeort; hier wird für Auflösungen von 128 x 128 Pixeln die Datei *icon_128.png* angezeigt. Das `app`-Objekt ❻ enthält schließlich das Unterfeld `web_url`. Hier wird die URL der Seite definiert, die beim Starten Ihrer App

angezeigt werden soll.

Der Firefox Marketplace

Open-Web-App-Manifest

Im Firefox Marketplace wird ein sogenanntes Open-Web-App-Manifest benötigt. Wie der Name vermuten lässt, hofft Mozilla darauf, dass dieses Format sich als allgemeiner Standard für Web-Apps durchsetzen wird, wodurch die Übertragung von Apps in unterschiedliche App-Stores zukünftig erleichtert werden könnte. Dieses Manifest heißt in der Regel *manifest.webapp* und wird mit der MIME-Typbezeichnung *application/x-web-app-manifest+json* versehen. Eine vollständige Anleitung finden Sie in den Folien aus Robert Nymans Vortrag »WebAPIs and Apps«, der im Abschnitt »Literaturempfehlungen« auf Seite 231 aufgeführt wird.

Das Manifest für Open Web App liegt im JSON-Format vor und ähnelt vom Aufbau jenem des Chrome Web Store. Im folgenden Listing sehen Sie ein Minimalbeispiel mit Anmerkungen.

```
  {
❶ "name": "App Name",
❷ "description": "Short description",
❸ "version": "0.1",
❹ "launch_path": "http://foo.example.com/",
❺ "icons": {
    "128": "icon_128.png"
  }
}
```

Die Felder `name` ❶ und `version` ❸ und das `icons`-Objekt ❺ sind genauso wie zuvor für den Chrome Web Store beschrieben. Das `description`-Feld ❷ ist ebenfalls gleich, allerdings sind hier bis zu 1024 Zeichen zulässig. Der größte Unterschied ist das Feld `launch_path` ❹, das dasselbe Verhalten aufweist wie `app > launch > web_url` im Chrome-Manifest. Eine Manifest-Version ist nicht verbindlich vorgeschrieben, Sie könnten jedoch als Erweiterung eine hinzufügen, wenn Sie die App an den

Chrome Web Store übermitteln (der dies erfordert).

10.1.3 W3C-Widgets

Widgets-Manifest

Das W3C hat seinen eigenen Standard für Packaged Apps entwickelt, die hier unter der Bezeichnung *Widgets* geführt werden. Widgets funktionieren wie andere Packaged-Apps, außer dass die Manifest-Datei mit der Bezeichnung *config.xml* im XML-Format vorliegt und alle Dateien mit der Dateiendung *.wgt* gezippt werden.

Das im XML-Format vorliegende Manifest unterscheidet sich naturgemäß ein wenig von den vorhergehenden, aber die meisten Informationen sollten Ihnen trotzdem vertraut vorkommen:

```
<widget version="0.1">
  <name>App Name</widgetname>
  <description>Short description</description>
  <content src="http://foo.example.com"/>
  <icon src="icon_128.png"/>
</widget>
```

Widgets gibt es schon eine ganze Weile, und sie gelten als recht stabil, allerdings mangelt es etwas an Implementierungen im modernen Geräteumfeld. Der Opera-Browser nutzt ein Widget als Basis zur Spezifikation seiner Erweiterungen. Der größte Widget-Anwender ist allerdings derzeit wahrscheinlich das PhoneGap-Projekt, das ich nachfolgend erläutere.

10.2 Hybrid-Apps

Wenn Sie Ihre Web-Apps in den großen gerätebezogenen App-Stores anbieten möchten – etwa im Apple App-Store, bei Google Play oder im Windows Store –, dann müssen Sie eine Hybrid-App erstellen. Diese Apps gleichen den Packaged Apps insofern, dass alle Ressourcen in einer

einzelnen Archivdatei gebündelt werden. Sie gehen aber noch einen Schritt weiter und hüllen die Dateien in eine native Shell – den sogenannten Wrapper. Dadurch wird sichergestellt, dass sich die App ins Betriebssystem integrieren lässt, was Sicherheit, höhere Leistung und ebenso den Zugriff auf geschützte Geräte-APIs bedeutet. Es gibt eine Reihe von Softwarelösungen zur Erstellung von Hybrid-Apps, doch eine der am häufigsten genutzten – und sicherlich die am einfachsten zu erlernende – ist PhoneGap.

10.2.1 PhoneGap

Auch bekannt als Apache Cordova

Obwohl PhoneGap dem kommerziellen Unternehmen Adobe gehört, handelt es sich dennoch um eine kostenfreie Open-Source-Software. Unter Verwendung von Technologien der Webplattform lassen sich damit halb native mobile Anwendungen erstellen. PhoneGap ist eine Distribution der Software *Apache Cordova*, die PhoneGap hieß, bevor sie an Adobe verkauft wurde und die Dinge kompliziert wurden. Die Namen werden häufig synonym verwendet. Ich bleibe jetzt der Übersicht halber durchgängig bei der Bezeichnung PhoneGap.

PhoneGap läuft auf verschiedenen Plattformen, darunter auch – und das ist entscheidend – iOS, Android und Windows Phone. Als nativer Wrapper für Webplattform-Code erlaubt es zudem den Zugriff auf Geräte-APIs, die nicht immer über den Browser verfügbar sind. Hierzu dient eine eigene API. Sie verwendet, wann immer möglich, die Standard-API des jeweiligen Geräts, fungiert als Vermittler zwischen unterschiedlichen API-Implementationen auf Geräteseite und stellt bei Bedarf auch anderweitig neue Methoden und Objekte zur Verfügung.

Um PhoneGap einzurichten, müssen Sie das SDK für jedes Ihrer Zielgeräte herunterladen und sich in manchen Fällen (insbesondere für iOS), ein Entwicklerzertifikat besorgen. Wenn Sie all das erledigt haben, schließen Sie die Installation für jede Umgebung ab (vollständige

Anweisungen finden Sie auf der PhoneGap-Website, die ich in den Literaturempfehlungen auf Seite 231 verlinkt habe) und beginnen ein neues Projekt. Dabei wird eine Verzeichnisstruktur mit einigen wichtigen Dateien angelegt, darunter auch die erforderlichen für den Zugriff auf die API. Mittlerweile ist PhoneGap auf node.js portiert worden, was das Erstellen von Projekten für die jeweiligen Zielplattformen stark vereinfacht.

Berechtigungen erteilen

Nachdem Sie die Einrichtung abgeschlossen haben, können Sie alle zum Ausführen der App benötigten Dateien zum Projekt hinzufügen und dann sofort die PhoneGap-API nutzen. Einige in der API enthaltenen Eigenschaften erfordern es, Berechtigungsanfragen in eine Manifest-Datei einzufügen – PhoneGap nutzt (wie bereits zuvor erwähnt) die XML-basierte Widgets-Spezifikation für Anfragen. Um beispielsweise Zugriff auf die Notification-API anzufordern, binden Sie das Notification-Plugin per Kommandozeile wie folgt ein

```
cordova plugin add org.apache.cordova.dialogs
```

und ergänzen das XML der jeweiligen Zielplattform wie in der Dokumentation[1] beschrieben.

Die PhoneGap-API

navigator

Die PhoneGap-API besteht größtenteils aus einer Reihe von Eigenschaften und Methoden des `navigator`-Objekts. Einige dieser Methoden werden bereits von den Browsern bereitgestellt. Ist dies der Fall, wie etwa für das in Kapitel 6 beschriebene `geolocation`-Objekt, wählt die Phone-Gap-API zunächst die native Implementierung. Im Zweifelsfall dient eine eigene Implementierung als Alternativlösung für Systeme, auf denen die Eigenschaft nicht verfügbar ist.

contacts

Andere Eigenschaften benötigen höhere

Berechtigungen und sind daher auf einigen Plattformen exklusiv durch PhoneGap verfügbar. Denken Sie beispielsweise an das `contacts`-Objekt, das den Zugriff auf die Kontaktliste des Benutzers ermöglicht – so etwas wollten Sie sicherlich nicht ohne ausdrückliche Genehmigung ins Web stellen.

Das `contacts`-Objekt hat zwei Funktionen: `create()` und `find()`. Um Elemente aus der Kontaktliste auszuwählen, verwenden Sie Letzteres mit zwei verpflichtenden Argumenten: `contactFields` (eine Liste der zurückzuliefernden Felder) und `contactSuccess` (eine Funktion, die bei einer erfolgreichen Anfrage ausgeführt wird).

Im nachfolgenden Code stelle ich eine Anfrage zum Erhalt des Namens und Geburtstags aller Kontakte im Adressbuch. Hat die Anfrage Erfolg, wird eine anonyme Funktion ausgeführt. Diese Funktion durchläuft alle Ergebnisse (im von mir definierten *contacts*-Objekt) und fügt sie einer Zeichenkette namens *contactDOB* hinzu, die ich hier zum Füllen einer Liste verwende.

```
navigator.contacts.find(['displayName','birthday'], function
(results) {
  var i, contactDOB;
  for (i=0; i < results.length; i++) {
    contactDOB += '<li>' + results[i].displayName + '(' +
                   results[i].birthday + ')</li>';
  }
 });
```

Kommende Browser und webbasierte Betriebssysteme (wie etwa Firefox OS) werden in Zukunft vermutlich direkten Zugriff auf einige dieser Methoden bieten. Der Nutzen von PhoneGap wird wahrscheinlich nur noch darin bestehen, auf älteren Geräten Polyfills für einzelne Features anzubieten. Den Entwicklern von PhoneGap ist dies bewusst, wie sie in einem Blogbeitrag namens »Beliefs, Goals and Philosophy« eindrucksvoll darstellen:

> *The ultimate purpose of PhoneGap is to cease to exist.*[2]

PhoneGap-Ereignisse

Neben seiner Device-Properties-API bietet PhoneGap auch eine Reihe äußerst nützlicher Ereignisse, die Veränderungen des Gerätestatus registrieren, zum Beispiel Netzwerkstatus und -durchsatz oder Batteriefüllstand. Diese Ereignisse sind entscheidend für besonders anwenderfreundliche Apps. Daten werden also zum Beispiel gespeichert, bevor die Batterie zur Neige geht oder wenn die Netzwerkverbindung abbricht.

deviceready

Das wichtigste Event ist `deviceready`. Es wird ausgelöst, wenn PhoneGap vollständig geladen wurde und ausgeführt werden kann. Das `deviceready`-Ereignis ist von der Funktion her mit dem `DOMContentLoaded`-Ereignis aus Kapitel 5 identisch – Sie sollten es ebenfalls in jedem Skript verwenden, um sicherzugehen, dass alle erforderlichen Bibliotheken geladen und initialisiert wurden, bevor der Rest des Skripts abläuft. Alle Funktionen, die Zugriff auf die PhoneGap-API erfordern, sollten daher in der Rückruffunktion untergebracht werden:

```
document.addEventListener('deviceready', function () {
  // All PhoneGap-related functions
}, false);
```

pause, *resume*

Weitere Ereignisse sind `pause` und `resume`. Ersteres wird ausgelöst, wenn die aktuelle App geschlossen und in einen Hintergrundprozess verlagert wird, das zuletzt Genannte, wenn sie wieder aktiviert wird. Sie könnten zum Beispiel Daten speichern, wenn die App in den Hintergrundprozess verlagert wird, damit die Anwender ihre Arbeit auf keinen Fall verlieren:

```
document.addEventListener('pause', function () {
  // Backup data
}, false);
```

Einige Ereignisse treten ein, wenn die Netzwerkverbindung der App getrennt oder hergestellt wird (`offline` und `online`); wenn Lautstärke-

oder Anruftasten gedrückt werden (`volumedownbutton`, `volumeupbutton`, `start-callbutton` und `endcallbutton`) und wenn sich der Batteriezustand ändert (`batterystatus` erkennt Änderungen im Zustand, während `batterylow` und `batterycritical` beim Erreichen eines niedrigen oder kritischen Ladestands – nach geräteseitiger Interpretation – ausgelöst werden).

10.2.2 Titanium

Ein weiterer beliebter Ansatz zur Veröffentlichung von Hybrid-Apps führt über Appcelerator Titanium. Statt eines Wrappers für Web-Apps im Stil von PhoneGap ist Titanium ein SDK, mit dem Sie Anwendungen in JavaScript entwickeln und dabei native UI-Elemente der Zielgeräteplattform nutzen können. Das hat den Vorteil, dass Ihre App später eher wie eine native App aussieht und sich auch so »anfühlt«. Dazu müssen Sie bei der Entwicklung allerdings vom Standardansatz der offenen Webplattform abweichen.

Aus diesem Grund bespreche ich Titanium in diesem Buch nicht weiter. Es kann aber sicher nicht schaden, schon einmal davon gehört zu haben.

10.3 TV-Apps

Bisher habe ich in diesem Kapitel hauptsächlich über die Entwicklung für Mobilgeräte gesprochen, ein neuer und florierender Bereich ist jedoch auch die Anwendungsentwicklung für Smart-TVs mit Internetanbindung. Die großen Mitspieler in diesem Feld sind Fernsehgerätehersteller wie Samsung, Sony und Panasonic; Anbieter von Set-Top-Boxen wie Roku und Boxee sowie der Browserhersteller Opera. Apps, die man auf Smart-TVs installieren kann, sind insgesamt eine gute Idee. Schließlich ist es einfacher, mit der Fernbedienung ein paar Icons durchzuklicken, als eine lange URL einzugeben.

Frühe Smart-TV-Modelle hatten proprietäre, zueinander inkompatible Entwicklungsplattformen, aber im Jahr 2012 kam es zu einer branchenweiten Bewegung in Richtung Webplattform-Technologien, wodurch sich die Entwicklung von TV-Apps vereinfachte. Jeder Smart-TV-Anbieter unterhält sein eigenes Entwicklerprogramm nebst Dokumentation; einige davon sind kostenfrei. Wenn Sie gerne in diesem Bereich tätig werden möchten, bietet das Entwicklerportal von Opera einige schöne Artikel und Tutorials zur optimalen Vorgehensweise (siehe Abschnitt »Literaturempfehlungen« auf Seite 231).

10.4 Webinos

EU-Projekt

Wenn Sie mit Ihren Projekten auf echte Cross-Device-Kommunikation abzielen, sehen Sie sich einmal das Webinos-Framework an. Diese browserbasierte Open-Source-Anwendungsplattform wird von der Europäischen Union zusammen mit großen Namen wie BMW, Samsung, Sony und dem W3C gefördert. Sie soll die Kommunikation zwischen Geräten vereinfachen, von Fahrzeugelektronik über Mobilgeräte bis hin zu Ihrem Fernseher.

Das browsergestützte Webinos setzt ebenfalls auf Technologien der Webplattform; es basiert sogar auf Node.js und nutzt HTML5-WebSockets zur Kommunikation. Die erste Veröffentlichung erfolgte 2012, und die Entwicklung dauert noch an, aber das Potenzial ist faszinierend.

10.5 Application Cache

AppCache

Die Anwender haben gewisse Erwartungen an webbasierte Apps. Dazu gehört insbesondere, dass sie auch offline funktionieren sollten oder

zumindest bei einem Verbindungsabbruch die Daten speichern. Eine Möglichkeit, offline Ressourcen bereitzustellen, ist die lokale Speicherung der Daten mit der in »Web Storage« auf Seite 142 besprochenen API. Ein besserer Ansatz könnte jedoch die Verwendung des Application Cache (oder AppCache) sein. Diese Schnittstelle listet Dateien auf, die vom User Agent heruntergeladen und im Zwischenspeicher gehalten werden sollen, damit sie auch bei unterbrochener Netzwerkverbindung verfügbar sind.

Der erste Schritt zur Erstellung eines AppCache führt über das Anlegen einer Manifest-Datei. Das ist eine Textdatei mit der Endung *.appcache*, die mit dem MIME-Typ *text/cache-manifest* gekennzeichnet werden muss. Verlinken Sie dieses Manifest mit dem `manifest`-Attribut des `html`-Elements auf jeder Seite, die auch offline verfügbar sein soll:

```
<html manifest="foo.appcache">
```

Einige Browser warnen den Benutzer davor, dass Ihre App Daten offline speichern will, und fragen dafür um Erlaubnis. Abb. 10–1 zeigt, wie das in Firefox für Android aussieht.

Beim Einsatz des Application Cache auf Ihrer Site müssen Sie einige Fallstricke beachten. Diese werden in Jake Archibalds bei *A List Apart* veröffentlichten Artikel »Application Cache is a Douchebag« (siehe Abschnitt »Literaturempfehlungen« auf Seite 231 ausführlich beschrieben. Ich werde in den nachfolgenden Abschnitten nur auf einige davon kurz eingehen.

Abb. 10–1

Manche Browser, etwa Firefox, fordern vom Anwender eine Erlaubnis an, bevor der Application Cache Dateien speichern kann.

10.5.1 Inhalte der AppCache-Datei

Die Datei *.appcache* beginnt mit den Worten `CACHE MANIFEST`. Darauf folgt eine Liste aller Dateien, die zwischengespeichert werden sollen. Jede Datei steht in einer neuen Zeile und Sie können einzeilige Kommentare einfügen, indem Sie eine Zeile mit einem Raute-Zeichen (#) beginnen.

Das nachfolgende Listing zeigt eine Manifest-Datei, die drei Dateien im Cache speichert. Der Kommentar mit Version und Datum ist nicht erforderlich, wird aber später noch nützlich werden:

```
CACHE MANIFEST
# Version 0.1, 2013-04-01
index.html
foo.css
foo.js
```

master entries und explicit entries

Sie brauchen die Seiten, mit denen die *.appcache*-Datei verknüpft ist, nicht explizit aufzuführen, da jede Seite mit `manifest`-Attribut im `html`-Element automatisch zwischengespeichert wird. Diese automatisch im Cache gespeicherten Dateien werden *master entries* genannt, im Manifest aufgeführte Dateien heißen hingegen *explicit entries*. Falls einige Dateien Online-Zugriff benötigen (etwa um mittels JavaScript auf eine Datenbank zuzugreifen), können Sie eine Art »Whitelist« von Dateien anfertigen, die immer über das Netzwerk geladen werden sollen. Dazu listen Sie sie nach dem `NETWORK:`-Header auf. Diese Dateien sind als *network entries* bekannt.

Im folgenden Beispiel werden die im */dynamic*-Verzeichnis enthaltenen Dateien über das Netzwerk statt aus dem Cache geladen:

```
NETWORK:
/dynamic
```

Sie können auch Alternativdateien hinzufügen, falls das Laden einer Ressource zum Beispiel wegen Netzwerkausfall oder aus anderen Gründen scheitert. Das erledigen Sie unter dem `FALLBACK:`-Header: Jede neue Zeile enthält eine Datei oder ein Verzeichnis, gefolgt von einem Leerzeichen und dann der alternativen Ressource.

Im folgenden Beispiel wird *fallback.html* aus dem Cache angezeigt, falls eine der Dateien im */templates*-Verzeichnis nicht in die Seite geladen werden kann:

```
FALLBACK:
/templates/ fallback.html
```

fallback entries

Diese Dateien heißen *fallback entries*, und zusammen mit den drei vorhergehenden Einträgen vervollständigen Sie die Kategorieliste der zwischengespeicherten Dateien.

10.5.2 Die Caching-Abfolge

Beim Laden Ihrer Seite sucht der Browser zunächst nach der Manifest-Datei. Falls diese existiert und bisher noch nicht geladen wurde, lädt der Browser zuerst ganz normal die Seitenelemente. Dann holt er sich von jeder Datei, die im Manifest aufgeführt ist, bisher aber noch nicht geladen wurde, eine Kopie. Der Browser legt alle aufgeführten Einträge im Cache ab, damit sie beim nächsten Besuch von dort aus geladen werden.

Wenn eine Manifest-Datei existiert und die Seite bereits zuvor einmal geladen wurde, lädt der Browser zuerst alle im Cache befindlichen Dateien, danach folgen gegebenenfalls die übrigen zur Anzeige der Seite erforderlichen Dateien. Als Nächstes überprüft er, ob die Manifest-Datei eventuell aktualisiert wurde, und lädt in diesem Fall eine weitere Kopie aller darin aufgeführten Dateien, die dann im Cache gespeichert und erst beim nächsten Laden der Seite verwendet werden; sie werden dem Benutzer nicht sofort dargestellt.

Eine wichtige Besonderheit ist, dass der Browser nur dann nach neuen Dateiversionen sucht, wenn sich die Manifest-Datei selbst verändert. Hier ist die Versionsnummer oder ein Zeitstempel sehr nützlich: Verändern Sie eines davon (oder beides), erkennt der Browser, dass das Manifest aktualisiert wurde, und lädt dann auch die aktualisierten Dateien.

10.5.3 Die AppCache-API

window.applicationCache

Wenn Sie per JavaScript auf den Cache zugreifen müssen, können Sie das `window.applicationCache`-Objekt einsetzen. Dieses Objekt enthält einige Eigenschaften und Methoden, mit denen Sie erzwingen können, dass nach dem erstmaligen Laden der Seite aktualisierte Dateien nachgeladen werden. Auch andere Interaktionen mit dem Manifest sind möglich.

Cache-Status

Die `status`-Eigenschaft liefert den aktuellen Cache-Status zurück. Dazu sind jedem Status ein numerischer Wert und eine benannte Konstante zugeordnet:

- `0` (UNCACHED) bedeutet, dass es keinen Cache gibt.
- `1` (IDLE) bedeutet, dass der Cache gerade nicht aktualisiert wird.
- `2` (CHECKING) bedeutet, dass der Browser die Manifest-Datei auf Aktualisierungen hin überprüft.
- `3` (DOWNLOADING) bedeutet, dass neue Ressourcen hinzugefügt werden.
- `4` (UPDATEREADY) bedeutet, dass ein neuer Cache verfügbar ist.
- `5` (OBSOLETE) bedeutet, dass der derzeitige Cache nicht mehr aktuell ist.

Mit der `update()`-Methode können Sie eine Überprüfung der Manifest-Datei erzwingen, bei der nach Aktualisierungen gesucht wird. Die `update()`-Methode weist der `status`-Eigenschaft während der Überprüfung des Manifests den Wert 2 zu; nach der Prüfung beträgt der

Wert 3, falls aktualisierte Ressourcen gefunden wurden, und schließlich 4, nachdem die aktualisierten Dateien im Cache zur Verfügung stehen.

swapCache()

Wenn der `status`-Wert 4 beträgt, können Sie die aktualisierten Dateien mithilfe der `swapCache()`-Methode laden. Denken Sie an diesem Punkt daran, dass der Browser die derzeit im Cache befindlichen Dateien geladen hat, um die Ladezeit der Seite für den Benutzer zu verringern. Alle im Cache aktualisierten Dateien werden dem Benutzer erst beim erneuten Laden der Seite angezeigt. Um das zu vermeiden, können Sie ein AppCache-Ereignis einsetzen, das zu verschiedenen Zeiten im Cache-Zyklus ausgelöst wird, insbesondere dann, wenn sich die `status`-Eigenschaft verändert. Ändert sich der `status`-Wert beispielsweise in 2, wird das `checking`-Ereignis ausgelöst, ändert er sich in 3, wird das `downloading`-Ereignis ausgelöst.

Der nachfolgende Code zeigt einen Ansatz zum Neuladen der Seite mit aktualisierten Ressourcen. Hier wird das `updateready`-Ereignis ausgelöst, nachdem ein neuer Cache heruntergeladen wurde, woraufhin eine Funktion ausgeführt wird, die erneut prüft, ob die `status`-Eigenschaft den Wert to UPDATEREADY (4) aufweist. Ist das der Fall, tauscht `updateready` den Cache aus und fragt den Anwender, ob die Seite neu geladen und die neuesten Dateiversionen dargestellt werden sollen.

```
var myCache = window.applicationCache;
myCache.addEventListener('updateready', function(e) {
  if (myCache.status === myCache.UPDATEREADY) {
    myCache.swapCache();
    if (confirm('Load new version?')) {
     window.location.reload();
  }
 }
}, false);
```

Durch die Nachfrage beim Benutzer vor dem erneuten Laden der Seite wird sichergestellt, dass dieser keinen Datenverlust erleidet oder durch ein erzwungenes Neuladen in seiner Handlung unterbrochen wird. Viele beliebte Web-Apps folgen diesem Ansatz, darunter auch die Google-Tools.

10.6 Zusammenfassung

In diesem Kapitel haben wir einen (notgedrungen) kurzen Blick darauf geworfen, wie man Websites in ein zur Veröffentlichung in App-Stores geeignetes App-Format verpackt. Ich habe den Unterschied zwischen gehosteten Apps und Packaged Apps angesprochen, ebenso wie die zur Übermittlung an zwei App-Stores erforderlichen Manifest-Dateien und den Manifest-Standard des W3C. Ich bin auch auf Hybrid-Apps eingegangen, die Webtechnologien in einem nativen Wrapper verpacken, und ich habe erklärt, wie Sie die Open-Source-Software PhoneGap verwenden, um einen weitergehenden Gerätezugriff zu erhalten, als es mit den meisten Browsern möglich ist. Schließlich ging ich mit AppCache auf eine Möglichkeit ein, Daten für den Offline-Zugriff auf einem Gerät zu speichern. Wir haben uns die Arbeitsweise der API und den Zugriff über JavaScript angesehen und uns auch kurz mit ihren Einschränkungen befasst.

10.7 Literaturempfehlungen

Der *Chrome Web Store* hat eine detaillierte Dokumentation zur Vorbereitung der einzureichenden Apps und natürlich auch zur Manifest-Datei: https://developers.google.com/chrome/web-store/docs/get_ started_simple/.

Näheres zur Manifest-Datei für Firefox Marketplace finden Sie unter https://developer.mozilla.org/docs/Apps/Manifest/. Auch ein nützliches *Tool zur Überprüfung von Manifesten* wird angeboten: https://marketplace.firefox.com/developers/validator/. Robert Nymans Vortrag *Web APIs and Apps* liefert eine hervorragende Einführung zur Entwicklung von Apps für den Firefox Marketplace: http://www.slideshare.net/robnyman/web-apis-apps-mozilla-london/.

Die letzte Version der *Widgets-Spezifikation des W3C* liegt unter http://w3.org/TR/widgets/, und eine *hilfreiche Einführung* finden Sie auf Peter-Paul Kochs Site – auch wenn sie bereits vor einigen Jahren geschrieben wurde und sich die Spezifikation seither in einigen kleinen Details verändert hat: http://quirksmode.org/blog/archives/2009/04/introduction_to.html.

Das *PhoneGap-Projekt* finden Sie unter http://phonegap.com/, und die *API* ist unter http://docs.phonegap.com/ dokumentiert. Den PhoneGap-Artikel *Beliefs, Goals, and Philosophy* können Sie unter http://phonegap.com/2012/05/09/phonegap-beliefs-goals-and-philosophy/nachlesen.

Mehr zum Titanium-Projekt finden Sie unter http://appcelerator.com/platform/.

Jede Smart-TV-Plattform hat ihr eigenes Entwicklerforum, ein gutes *Beispiel für den Einstieg* bietet die Samsung-Site unter http://www.samsungdforum.com/.

Der Opera TV Store fasst immer mehr Fuß; mehr darüber erfahren Sie unter http://business.opera.com/partners/tv/store/. Die Dev.Opera-Site bietet einige tolle *Referenzartikel zur Gestaltung und Entwicklung für Fernsehgeräte*; diese können sie unter http://dev.opera.com/tv/nachlesen.

Das *Webinos-Projekt*, das eine einheitliche Geräte-API anstrebt, befindet sich auf http://www.webinos.org/.

Das Mozilla Developer Center verfügt unter https://developer.mozilla.org/en-US/docs/HTML/Using_the_application_cache/ über eine hervorragende *AppCache-Dokumentation*, und Mark Christian und Peter Lubbers haben eine nützliche *Seite mit Fakten über App-Cache* ins Leben gerufen: http://appcachefacts.info/. Alle Pferdefüße von

AppCache und Tipps und Techniken zur Verwendung sind in Jake Archibalds Artikel *Application Cache is a Douchebag* bei *A List Apart*: http://www.alistapart.com/articles/applicationcache-is-a-douchebag/aufgeführt.

http://docs.phonegap.com/en/3.2.0/cordova_notification_notification.m

Frei übersetzt: Der eigentliche Zweck von PhoneGap besteht darin, sich selbst überflüssig zu machen.

11 Die Zukunft

Bisher habe ich in diesem Buch nur solche Webfunktionen beschrieben, die wenigstens in einigen Browsern relativ stabil sind oder in naher Zukunft stabil sein sollten. Jetzt sind wir aber im letzten Kapitel angelangt, und ich kann endlich einige der eher experimentellen Funktionen am Horizont ansprechen.

Überall sind Veränderungen geplant: Eine neue Revision von JavaScript mit dem Codenamen Harmony, die ursprünglich im Lauf des Jahres 2013 herauskommen sollte, wird nun für 2014 erwartet und sollte dann in den nächsten Jahren Eingang in die Browser finden; dem W3C wurden viele neue APIs vorgeschlagen, darunter auch eine zum Aufspüren von Netzwerkgeräten mittels Universal Plug and Play (UPnP) und eine zur Messung des Umgebungslichts; die Arbeit am Entwurf der HTML5.1-Spezifikation macht gute Fortschritte und viele CSS-Module streben bereits das Level 4 an. Ich könnte endlose Veränderungen aufzählen – aber ich konzentriere mich auf solche, die unsere Arbeitsweise meiner Ansicht nach am stärksten beeinflussen und gute Chancen auf eine Implementierung haben werden.

11.1 Webkomponenten

Web Components

Es ist kaum übertrieben zu sagen, dass die Web-Components-Spezifikation die radikalsten Veränderungen an HTML seit seiner Entstehung vor über 20 Jahren vorschlagen. Selbst das so stark gehypte HTML5 ist in Wahrheit nur ein kleines Versionsupdate, das nichts wirklich Neues mitbringt.

Web Components ist ein Sammelbegriff für eine Reihe von Ergänzungen zu HTML und dem DOM, die die Erstellung multifunktionaler Benutzeroberflächen für Webanwendungen

unterstützen – eine Art wiederverwendbare Widget-Spezifikation. Während ich dies schreibe, existieren vier Hauptkomponenten: *Templates*, *Decorator*, *Custom Elements* und das *Shadow DOM*. Ich beschreibe ihre Funktion nachfolgend im Einzelnen – zuerst möchte ich aber zusammenfassen, was sie gemeinsam leisten können.

Eines der Hauptprobleme beim Entwickeln von Anwendungskomponenten in HTML besteht heute darin, dass ihre Bestandteile (sprich: Elemente) Teil des DOM sind. Aus diesem Grund sind sie offen für Konflikte mit CSS oder JavaScript. Das könnten Vererbungskonflikte sein, wie auf übergeordnete Elemente angewandte Regeln, die in Komponentenelemente kaskadieren, oder – umgekehrt – auf Komponentenelemente angewandte Regeln, die auf andere Elemente im DOM kaskadieren.

Ein weiteres Problem sind Namenskonflikte, wenn dieselbe Klasse oder ID unwissentlich auf verschiedenen Seiten einer Site eingesetzt wird. Für ein Element bewusst deklarierte Regeln werden so unbeabsichtigt auch auf andere Elemente angewandt. Dieses Problem findet sich häufig auf großen Sites, denen ein klares Schema bei der Namensgebung fehlt. Durch Konflikte in JavaScript kann es sich sogar noch verschlimmern, wenn Selektoren unerwünschtes Funktionsverhalten auf ein Element anwenden.

Am besten lassen sich solche Konflikte durch Abtrennung der Komponente vom Rest des DOM vermeiden; jegliche Abhängigkeit wird damit vermieden. Diese in der objektorientierten Programmierung grundlegende Technik ist als *Datenkapselung* bekannt.

Datenkapselung im HTML-DOM

Mit Web Components wird versucht, die Datenkapselung im HTML-DOM umzusetzen; Sie können Elemente erstellen, die nur in der Darstellung einer Seite und nicht im DOM selbst erscheinen. Web Components werden einen Weg zur Erstellung von Widgets bereitstellen, die Sie wiederholt in vielen verschiedenen Seiten einer Site hinweg einsetzen können, ohne dass Sie sich dabei um Konflikte mit bestehendem CSS- und JavaScript-Code

sorgen müssen – schließlich besitzt das Widget sein eigenes, paralleles DOM.

Die Web-Components-Spezifikation ist derzeit immer noch im Entwurfsstadium. Ich werde die Konzepte deshalb nicht im Detail beleuchten, wohl aber auf Grundlagen eingehen, weil sie so wichtig werden könnten.

11.1.1 Templates

Templates bieten vielleicht den besten Einstieg zum Verständnis von Web Components. Die Idee, bei der Entwicklung wiederverwendbare Codeblöcke (oder *templates*) einzusetzen, ist in der Webentwicklung seit einiger Zeit fest verwurzelt – auch wenn wir bisher nie eine native Implementation in HTML gesehen haben. Zum Einsatz von Templates waren bisher immer serverseitige Sprachen oder JavaScript (wie etwa die Mustache-Bibliothek aus Kapitel 5) erforderlich.

Stellen Sie sich ein Web-Components-Template als eine Art inaktiven DOM-Bereich vor. Das Wichtige daran ist, dass die Inhalte vom Browser geparst, aber nicht dargestellt werden. Bilder und andere externe Elemente werden also nicht geladen und enthaltene Skripte werden nicht ausgeführt. Im Vergleich zu per CSS ausgeblendeten Elementen, die aber dennoch geladen werden, kann dies einen gewaltigen Leistungsschub ergeben.

template

Ein Template wird mit dem `template`-Element deklariert, und jegliche Unterelemente ergeben den Inhalt des Templates. Der nachfolgende Codeblock zeigt ein Template-Element mit der `id` *#foo*, das zwei Unterelemente besitzt (ein `h2`- und ein `p`-Element). Außerhalb des Templates befindet sich ein `div`-Element mit der `id` *#bar*, das ein `h1`-Element enthält.

```
<template id="foo">
  <h2>Gorilla Beringei</h2>
  <p>A species of the genus Gorilla...</p>
```

```
</template>
<div id="bar">
  <h1>Eastern Gorilla</h1>
</div>
```

Würden Sie diese Seite mit den Entwicklerwerkzeugen Ihres Browsers betrachten, könnten Sie das `template`-Element ohne Inhalte sehen, da die Inhalte dieses Elements für das DOM im Grunde unsichtbar sind.

content

Per Skript greifen Sie auf das Template über das `content`-Objekt zu, welches die untergeordneten Elemente des Templates als HTML-Fragment zurückliefert. Im nächsten Codeabschnitt wird das Template zum Beispiel der Variablen *tpl* zugewiesen und das `content`-Objekt in der Konsole ausgegeben:

```
var tpl = document.getElementById('foo');
console.log(tpl.content);
```

Sobald Sie das Fragment haben, können Sie es nach Belieben manipulieren. Der nachfolgende Code legt mittels `cloneNode()` einen Klon des Inhalts an und fügt diesen mittels `appendChild()` in *#bar* ein:

```
var bar = document.getElementById('bar'
  clone = tpl.content.cloneNode(true);
bar.appendChild(clone);
```

An dieser Stelle würden Sie, wenn Sie das DOM inspizieren, das folgende Markup finden:

```
<template id="foo"></template>
<div id="bar">
  <h1>Eastern Gorilla</h1>
</div>
```

Die Seite würde jedoch dargestellt, als hätten Sie dieses Markup verwendet:

```
<div id="bar">
  <h1>Eastern Gorilla</h1>
  <h2>Gorilla Beringei</h2>
  <p>A species of the genus Gorilla...</p>
```

```
</div>
```

Sie können es sich selbst in der Beispieldatei *templates.html* ansehen. Die Ausgabe ist in Abb. 11–1 dargestellt (im Anhang A finden Sie Informationen zur aktuellen Browserunterstützung). Beachten Sie, dass ich in meinen Entwicklerwerkzeugen die Option *Show Shadow DOM* aktivieren musste, damit die Inhalte des `template`-Elements im DOM dargestellt wurden; wäre diese Option ausgeschaltet, würde das Element leer erscheinen.

Abb. 11–1 *Der Code-Inspektor zeigt die Inhalte des Template-Elements, das außerhalb des regulären DOM existiert.*

Code, der innerhalb des `template`-Elements funktionslos war, tritt in Aktion, sobald dieses in ein anderes DOM-Element eingefügt wird. Dann werden sämtliche externen Ressourcen geladen, Skripte geparst und so weiter.

11.1.2 Decorator

Benutzerdefiniertes Markup

Decorator erweitern den Nutzen der Templates durch die Möglichkeit, benutzerdefiniertes Markup durch CSS hinzuzufügen. Decorator verwenden das `decorator`-Element, dem eine eindeutige `id` zugewiesen werden muss. Innerhalb des `decorator`-Elements finden Sie ein `template`-Element mit etwas benutzerdefiniertem Markup und das `content`-Element. In diesem wird das Element, auf das die Regel angewendet wird, dargestellt. Unklar? Ich habe auch etwas Zeit gebraucht, um das zu verstehen.

Lassen Sie uns das in Häppchen unterteilen. Der nachfolgende Code zeigt ein Beispiel für einen Decorator. Ich habe ihm (zur Abwechslung) die eindeutige `id` *#foo* zugewiesen. Darin befindet sich ein Template mit einem `div`-Tag, in dem sich das `content`- und ein `h2`-Element befinden.

```
<decorator id="foo">
  <template>
   <div>
    <content></content>
    <h2>A great ape!</h2>
   </div>
  </template>
</decorator>
```

Stellen Sie sich nun vor, im Hauptdokument hätte ich ein `h1`-Element mit der `id` *#bar*, so wie im nachfolgenden Code:

```
<h1 id="bar">Gorilla</h1>
```

Ich wende den Decorator mit CSS und der neuen `decorator`-Eigenschaft an, die als Wert eine `url()`-Funktion mit der `id` des Decorators besitzt.

```
h1#bar { decorator: url(#foo); }
```

Anschließend wird das im Template *#foo* enthaltene Markup zum Markup des Elements *#bar* hinzugefügt, wobei *#bar* selbst das `content`-Element von *#foo* ersetzt. Dies geschieht allerdings erst bei der Darstellung, und

das DOM wird dadurch nicht verändert. Bei einer Inspektion enthält das DOM nur das Element *#bar*, und doch wird das Element so dargestellt, als würde folgendes Markup verwendet:

```
<div>
  <h1 id="bar">Gorilla</h1>
  <h2>A great ape!</h2>
</div>
```

Templates und Decorator können aber noch mehr – um Ihnen das zu zeigen, muss ich zunächst einen kleinen Exkurs einfügen und auf *Scoped Styles* eingehen.

Scoped Styles

Vererbungs- und Namenskonflikte

Eine der größten Stärken von CSS ist die *Vererbung* – durch die Regelkaskade können Werte auf mehrere unterschiedliche Elemente angewandt werden. Diese Stärke kann allerdings auch zum Nachteil werden, wenn Sie auf großen Sites mit vielen Stylesheets arbeiten. Vererbungs- und Namenskonflikte, die ich bereits zu Beginn dieses Abschnitts erwähnt hatte, sind hier nicht ungewöhnlich.

scoped

Das Attribut *scoped* bietet eine Möglichkeit, diese Konflikte zu vermeiden. Werden im Dokument `style`-Elemente mit dem Attribut `scoped` verwendet, vererben sich alle darin enthaltenen Regeln nur auf die entsprechenden Unterelemente des betreffenden Elements – ihre Reichweite ist also begrenzt, und anderswo im Dokument werden sie nicht angewendet.

Im nachfolgenden Code sehen Sie dies in Aktion: Ein `style`-Tag mit dem Attribut `scope` wird innerhalb eines `div`-Elements angewendet, und die auf das `h1`-Element angewendeten Regeln gelten nur für das `h1`-Tag innerhalb dieses Elements (mit der `id` *#foo*) und nicht für dasjenige außerhalb des `div`-Tags (mit der `id` *#bar*). Durch das Attribut `scoped` gilt die

Regel nur für die Unterelemente des `div`-Elements.

```
<div>
  <style scoped>
   h1 {
    background-color: #333;
    color: #FFF;
   }
  </style>
  <h1 id="foo">Scoped</h1>
</div>
<h1 id="bar">Not Scoped</h1>
```

Sehen Sie sich die Beispieldatei *scoped-style.html* an. Das `h1`-Tag mit der `id` *#bar* steht im DOM nach demjenigen mit der `id` *#foo*, Sie würden also erwarten, dass die Regeln im `style`-Element für beide gelten. Tatsächlich bewirkt das Attribut `scoped` aber, dass die Regeln nur innerhalb des übergeordneten `div`-Elements Anwendung finden. Das Ergebnis sehen Sie in Abb. 11–2 und in *scoped-style.html*.

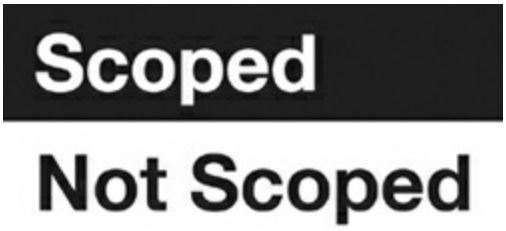


Abb. 11–2 *Die Regeln des ersten h1-Elements gelten nicht für das zweite h1-Element, weil die Regeln nur auf das übergeordnete div-Element angewandt werden.*

Scoped Styles und Templates

Diese Möglichkeit, den Geltungsbereich von Stilen zu beschränken, eignet sich ideal zur Datenkapselung. Besonders gut lässt sie sich mit Templates und Decorators verbinden. Ein typisches Beispiel wäre etwa der erste Codeblock im Abschnitt »Decorator« auf Seite 237: Hier könnte ich einen Satz Regeln definieren, der nur dann auf das ursprüngliche `h1`-Element angewandt wird, wenn der Decorator mit

einem durch das `scoped`-Attribut in seiner Reichweite begrenzte `style`-Tag innerhalb des `template`-Elements angewendet wird:

```
<decorator id="foo">
  <template>
    <div>
     <style scoped>
     h1 { color: red; }
     </style>
     <content></content>
     <h2>A great ape!</h2>
   </div>
 </template>
</decorator>
```

In diesem Fall wird das `h1`-Element nur dann rot eingefärbt, wenn der Decorator angewandt wird. Besser noch: Diese Farbe wird wegen ihres beschränkten Geltungsbereichs auf keines der nachfolgenden `h1`-Elemente im Dokument angewandt – ein perfektes Beispiel für Datenkapselung.

11.1.3 Custom Elements

Decorators sind nützlich, um ein Element mit etwas zusätzlichem Markup anders zu präsentieren – wenn Sie aber tiefer gehende Änderungen anstreben, verwenden Sie ein *Custom Element*. Der wichtigste Unterschied zwischen Custom Elements und Decorators ist, dass Letztere vorübergehender Natur sind; sie lassen sich durch Änderung eines Attributs oder Selektors anwenden oder entfernen. Custom Elements hingegen sind beständig; sie werden beim Parsen des DOM angewandt und lassen sich nur durch Skripte ändern oder entfernen.

Erweiterte Templates, die ein Standardelement verbessern oder ersetzen

Ein Custom Element ist wie ein erweitertes Template, das ein Standardelement verbessert oder ersetzt. Ein solches benutzerdefiniertes HTML-Element erstellen Sie mit dem `element`-Element (dieser Absatz stellt einen

Rekordversuch bei der Verwendung des Worts »Element« dar), welches über einige neue Attribute verfügt, die ich in Kürze erläutern werde. Innerhalb dieses Elements können Sie ein `template`-Element mit neuem Markup hinzufügen, ebenso wie mit dem Attribut `scoped` versehene Styles oder gar ein Skript.

Klingt das etwas verwirrend? Der nachfolgende Codeschnipsel zeigt ein einfaches Beispiel: ein `element` mit einem `template` darin, welches wiederum ein `div`-Tag enthält, in dem sich seinerseits das von mit auf Seite 237 in »Decorator« vorgestellte `content`-Element befindet. Das `element` hat zwei Attribute: `extends`, das als Wert den Namen des zu erweiternden Elements übernimmt (in diesem Fall handelt es sich um ein `button`-Element), und `name`, ein benutzerdefinierter und eindeutiger ID-Wert (der mit *x-* beginnen muss, um Konflikte zu bestehenden Elementen zu vermeiden).

```
<element extends="button" name="x-foobutton">
 <template>
  <div id="foo">
   <content></content>
  </div>
 </template>
</element>
```

Nach der Definition des Custom Element können Sie es mit dem `is`-Attribut auf ein bestehendes Element anwenden. Das `is`-Attribut wird auf das zu erweiternde Element angewandt und es akzeptiert als Wert den eindeutigen Bezeichner aus dem für das Custom Element definierten `name`-Attribut (*x-foobutton*). Tatsächlich ist es nicht so kompliziert, wie es vielleicht klingen mag:

```
<button is="x-foobutton">Go</button>
```

Das Ergebnis ist dasselbe wie bei einem Decorator: Das Markup des Custom Element ergänzt das Markup des Elements, auf das es angewandt wurde, allerdings nur in der fertigen Darstellung. Beim Blick ins DOM zeigt sich zwar nur das `button`-Element, es wird jedoch so dargestellt:

```
<div id="foo">
```

```
  <button>Go</button>
</div>
```

Das Beispiel ist simpel; es verdeutlicht jedoch, wie Sie mit den Erweiterungsmöglichkeiten dieser Technik auf einfache Weise komplett maßgeschneiderte Widgets programmieren könnten, die sich über viele Dokumente hinweg wiederverwenden ließen. Im Ergebnis ließen sich viele unserer sperrigen Widgets (wie Karusselle, Akkordeons und Datumswähler) auf bestehende Elemente anwenden, ohne das DOM mit unnötigem Code zu füllen – mit den zusätzlichen Vorteilen der Datenkapselung zur Vermeidung von Konflikten.

Ich habe bereits erwähnt, dass der Hauptunterschied zwischen einem Custom Element und einem Decorator in der Beständigkeit des Markups liegt. Ein Vorteil davon ist, dass Skripte in ein Custom Element gepackt werden können und dann immer präsent sind (darauf können Sie sich bei den kurzlebigeren Decorators nicht verlassen). Das alles bedeutet, dass Sie sogar für jedes Custom Element eine imperative API definieren könnten, um damit die Interaktivität auf eine völlig neue Ebene zu heben.

11.1.4 Das Shadow DOM

Der letzte Bestandteil der Web-Components-Spezifikation ist das *Shadow DOM*. Das ist nicht nur ein cooler Name für einen Superschurken, sondern auch eine Möglichkeit, mittels Skript die im bereits beschriebenen parallelen DOM befindlichen Elemente zu erstellen und darauf zuzugreifen. Decorator verwenden CSS zur Manipulation von Elementen, Custom Elements greifen dafür auf HTML zurück und das Shadow DOM nutzt zu diesem Zweck Skripte.

Eine neue, abgekapselte Knotenstruktur

Das Shadow DOM beschreibt die Fähigkeit eines Browsers, eine neue, völlig abgekapselte Knotenstruktur im bestehenden DOM zu erzeugen. Dazu erzeugt der Browser eine *Shadow Root* in einem Element. Sie kann wie eine normale

Knotenstruktur traversiert und manipuliert werden. (Ein Shadow Tree erscheint nicht im DOM, wird aber dargestellt.)

Jetzt ein Beispiel. Der nachfolgende Codeabschnitt enthält einfaches Markup: ein `div`-Tag namens *#foo*, in dem ein einzelnes `h1`-Element enthalten ist. Dies ist der Basiscode im DOM, in den ich nun eine Shadow Root einfügen werde.

```
<div id="foo">
 <h1>Hello, world!</h1>
</div>
```

Jetzt füge ich eine neue Shadow Root in das `div`-Element ein und hänge dann ein neues Element an die neue Wurzel an. Nachfolgend erkläre ich diesen Code Schritt für Schritt.

```
   var foo = document.getElementById('foo'),
❶    newRoot = foo.createShadowRoot(),
❷    newH2 = document.createElement('h2');
   newH2.textContent = 'Hello, shadow world!';
❸ newRoot.appendChild(newH2);
```

Zunächst wurde eine neue Shadow Root in *#foo* erstellt ❶. Dazu dient die Methode `createShadowRoot()`. In den beiden nachfolgenden Zeilen ❷ lege ich ein neues `h2`-Element mit dem Textinhalt *Hello, shadow world!* an. Und schließlich ❸ hänge ich das neue `h2`-Element der neuen Shadow Root an.

Wenn dieser Code ausgeführt wird, sehen die Anwender ein `h2`-Element mit dem Text *Hello, shadow world!*. Würden sie jedoch das DOM betrachten, könnten sie den Originalinhalt *Hello, world!* sehen. Das `h1`-Element wurde komplett vom neuen Shadow-Baum ersetzt. Auf das DOM hat dies keinen Einfluss.

Abb. 11–3 zeigt, wie dies in den Chrome-Entwicklertools dargestellt wird. Die Inhalte der Shadow Root werden in einer neuen Knotenstruktur unter dem Bezeichner *#shadow-root* angezeigt.

Abb. 11–3 *Die Shadow-Root ist im DOM-Baum deutlich gekennzeichnet.*

Wenn Sie den Inhalt des Elements, in dem Sie eine neue Wurzel angelegt haben, nicht ersetzen möchten, können Sie wieder auf das `content`-Element zurückgreifen (auch das habe ich bereits im Abschnitt »Decorator« auf Seite 237 vorgestellt), um die Originalelemente einzufügen. Ich veranschauliche das im folgenden Code, mit dem ich das `content`-Element erstelle und es dann an die neue Shadow Root anhänge. Im Ergebnis sieht der Anwender zuerst das neue Schatten-`h2` und danach folgt das ursprüngliche `h1`-Element – obwohl im DOM nur das `h1`-Element zu sehen ist.

```
var content = document.createElement(content);
newRoot.appendChild(content);
```

Sie können auch Templates zusammen mit Shadow Trees verwenden. Hier sehen Sie beispielsweise, wie Sie ein HTML-Fragment aus dem template *#foo* an das Shadow DOM anhängen.

```
var foo = document.getElementById('foo');
newRoot.appendChild(foo.content);
```

Das Shadow DOM geht sogar noch weiter als dieses einfache Beispiel,

und es ist ein sehr flexibles und leistungsfähiges Werkzeug. Ich kann ihm hier jetzt nicht mehr Zeit widmen, aber im Abschnitt »Literaturempfehlungen« auf Seite 255 finden Sie einige Links zu detaillierteren Artikeln.

11.1.5 Die Puzzleteile zusammenfügen

Ich habe nur an der Oberfläche der Web-Components-Spezifikation gekratzt, aber ich hoffe, Sie konnten durch diese Einblicke schon Feuer fangen. Mit Web Components stehen vollständig wiederverwendbare Codebestandteile zur Verbesserung bestehender Elemente vor der Tür. Diese Komponenten werden vollständig vom Rest des Dokuments abgekapselt sein; der Browser stellt sie dar, über das DOM sind sie jedoch nicht ansprechbar. Ein paralleles Schatten-DOM erlaubt jedoch vollständigen Zugriff auf die Elemente innerhalb jeder einzelnen Komponente und entsprechende Veränderungsmöglichkeiten.

Wenn die Web-Components-Spezifikation implementiert ist, wird sie unsere Herangehensweise bei der Entwicklung von Anwendungen und Websites vollkommen umkrempeln. Und wenn Sie das nicht spannend finden, sind Sie vielleicht in der falschen Branche unterwegs!

11.2 Die Zukunft von CSS

CSS3 hat das Web in verschiedener Hinsicht revolutioniert – etwa mit der Einführung von dynamischen Elementen wie Übergängen und Animationen und durch echte Lösungen von Layoutproblemen im Web. Diese Funktionen sind aber nur die Spitze des Eisbergs zukünftiger Möglichkeiten. Viele große Technologieunternehmen sehen ihre Zukunft im Web, und ihr Engagement bei der Gestaltung zukünftiger Webstandards bringt einen enormen Innovationsschub mit sich.

Adobes umfassende Wendung in Richtung offener Standards (eine angenehme Überraschung!) macht das Unternehmen beispielsweise zu

einem bedeutenden Mitspieler bei der Browserentwicklung, und seine Erfahrung im Grafik- und Publishing-Bereich fließt mit in CSS ein. Als erste Früchte dieser Arbeit ermöglichen CSS-*Regions* und -*Exclusions* zukünftig eine grundlegend andere Herangehensweise an das Seitenlayout, und dynamische Websites können es endlich mit dem Desktop Publishing aufnehmen.

Die größte Triebkraft bei der Entwicklung von CSS hat jedoch die Webentwickler-Gemeinde. Von Entwicklern erstellte JavaScript-Bibliotheken wie jQuery und Modernizr haben einen direkten Einfluss auf die Sprache, wie Sie bereits am Beispiel von `querySelector()` und `querySelectorAll()` (in Kapitel 5) sehen konnten. Und dieser Einfluss wird mit der Einführung der *Feature Queries* weiterhin deutlich zu spüren sein.

Zudem gewöhnen sich Frontend-Entwickler durch die wachsende Beliebtheit von CSS-Präprozessoren wie Sass und LESS zunehmend an den Einsatz von Programmierprinzipien wie Variablen und Funktionen. Der Bedarf, diese in der Sprache zu verankern, manifestiert sich durch *kaskadierende Variablen*.

11.2.1 Regions

Flexibler Spaltensatz

In Kapitel 4 habe ich den CSS-Spaltensatz angesprochen, mit dem sich Inline-Inhalte in Spalten unterteilen lassen, sodass sie von der ersten in die zweite und in die dritte Spalte fließen usw. Stellen Sie sich vor, die Spalten würden nicht direkt nebeneinander liegen – die erste befindet sich links auf der Seite, die zweite liegt rechts und die dritte am unteren Seitenrand –, aber der Inhalt durchfließt trotzdem nacheinander die Spalten. Darum geht es im Kern beim neuen CSS-Konzept *Regions*.

Regions funktionieren so: Ein Element wird zur Quelle erklärt und der Inhalt dieser Quelle fließt in ein weiteres Element oder in eine Reihe weiterer Elemente, die sogenannte *Region Chain*. In der Praxis bedeutet dies, dass Ihr Inhalt nacheinander mehrere Elemente durchfließen kann,

und das unabhängig von deren Reihenfolge im DOM.

Hier ist ein kleines Beispiel, das mit drei `div`-Elementen anfängt: einmal *#foo* und zweimal *.bar*. Das erste, *#foo*, erhält einen Inhalt, die beiden anderen bleiben leer:

```
<div id="foo">
 <p>...</p>
</div>
<div class="bar"></div>
<div class="bar"></div>
```

Im nächsten Schritt holen Sie den Inhalt aus *#foo* und kopieren ihn in eine benannte Textkette, einen sogenannten *Named Flow*, als würden Sie ihn etwa in einer Variablen ablegen (oder in der Zwischenablage Ihres Computers). In CSS-Regions erzeugen Sie diesen Named Flow durch Deklaration der `flow-into`-Eigenschaft für das Quellelement (*#foo*). Als Wert für die Variable können Sie einen beliebigen eindeutigen Namen wählen, ich nenne meine also *myFlow*. Nachdem ich mein Flow (oder meine Zwischenablage, um bei der Analogie zu bleiben) benannt habe, kann ich es in andere Elemente hineinfließen lassen, die unter dem Titelnamen »*Regions*" geführt werden:

```
#foo { flow-into: myFlow; }
```

WARNUNG

Im Internet Explorer 10 muss das Quellelement ein `iframe` sein, und der Inhalt im `body` des verlinkten Dokuments wird zum Flow-Inhalt.

Wenn diese Eigenschaft angewendet wird, werden das Quelleelement und seine untergeordneten Elemente nicht mehr auf dem Bildschirm ausgegeben, obwohl sie im DOM nach wie vor sichtbar und zugreifbar sind.

Als Nächstes muss ich eine Region Chain definieren, also die Bereiche, die der Inhalt durchfließen soll. Für jedes Element in der Region Chain sollte die `flow-from`-Eigenschaft deklariert werden. Als

Wert dient dabei zuvor definierte Named Flow. Der nachfolgende Code zeigt, wie der Inhalt des Flows *myFlow* in alle mit *.bar* benannten Regionen hineinfließt.

```
.bar { flow-from: myFlow; }
```

Der Inhalt in *myFlow* fließt in DOM-Reihenfolge durch die Region Chain. Zunächst fließt er in die erste Instanz von *.bar*, sämtlicher Übersatz fließt in die zweite Instanz von *.bar* und so weiter, bis der Inhalt erschöpft ist. Probieren Sie es in der Datei *regions.html* aus (siehe auch Abb. 11–4).

Wie ich bereits zu Anfang dieses Abschnitts erwähnt habe, funktionieren CSS-Regions wie ein mehrspaltiges Layout, wobei die Spalten aber nicht unmittelbar nebeneinander liegen müssen. Dieses neue Element bietet faszinierende Möglichkeiten zur Erstellung dynamischer und interessanter Seitenlayouts, die von der jahrelangen Erfahrung im Bereich gedruckter Medien inspiriert sind.

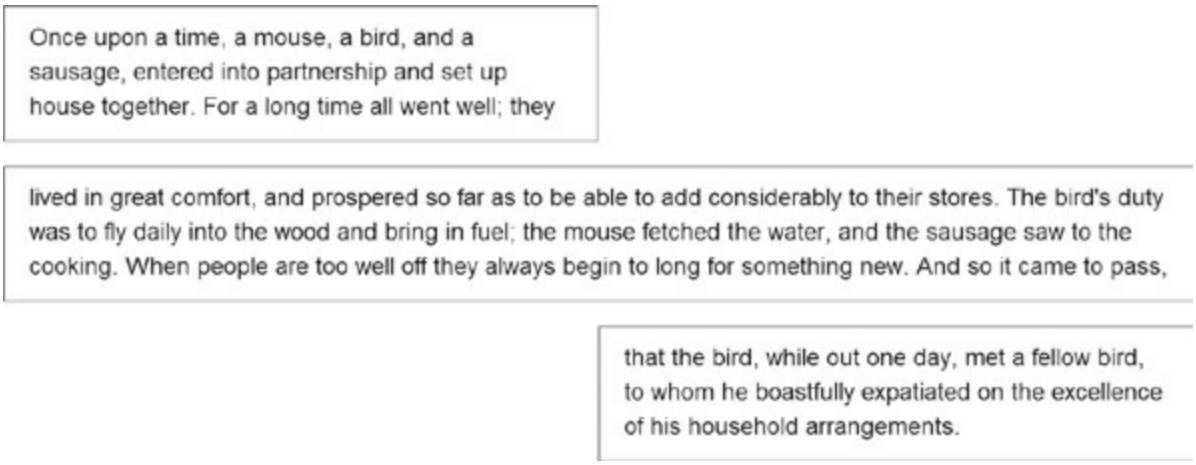


Abb. 11–4
Mit CSS-Regions können Sie Inhalt durch mehrere Elemente fließen lassen, die nicht direkt beisammen liegen müssen.

11.2.2 Exclusions

Textfluss

CSS Exclusions können Sie sich als eine Art positionierter Float-Elemente vorstellen – ein früheres Konzept hat sie tatsächlich als genau das beschrieben. In CSS2.1 finden Sie Float-Elemente nur auf der linken Seite und der übrige Inhalt umfließt sie dabei rechts oder umgekehrt. Mit CSS Exclusions soll es jedoch möglich werden, ein Element von Inhalt umfließen zu lassen, egal an welcher Stelle der Seite es sich befindet.

Sehen Sie sich zur Veranschaulichung folgendes Markup an, das das Container-Element *#foo*, etwas Textinhalt in einem `p`-Tag und das untergeordnete Element *#bar* enthält.

```
<div id="foo">
 <p>...</p>
 <div id="bar"></div>
</div>
```

Ich möchte *#bar* absolut über dem Inhalt in *#foo* platzieren. Hierzu benötige ich etwa diese Style-Regeln:

```
#foo { position: relative; }
#bar {
 left: 20px;
 position: absolute;
}
```

Hier liegt *#bar* auf einer über den Textinhalt gelegten Ebene. Wie in Abb. 11–5 zu sehen ist, verdeckt es alles dahinter Liegende. Ich möchte aber, dass *#bar* Teil derselben Ebene wird und dass es vom Text umflossen wird. *#bar* soll also ein *Exclusion Element* werden.

Genau das kann ich durch Anwendung der `wrap-flow`-Eigenschaft erreichen. Jeder untergeordnete Inhalt umfließt dadurch das neue Exclusion-Element entsprechend des Werts der Eigenschaft.

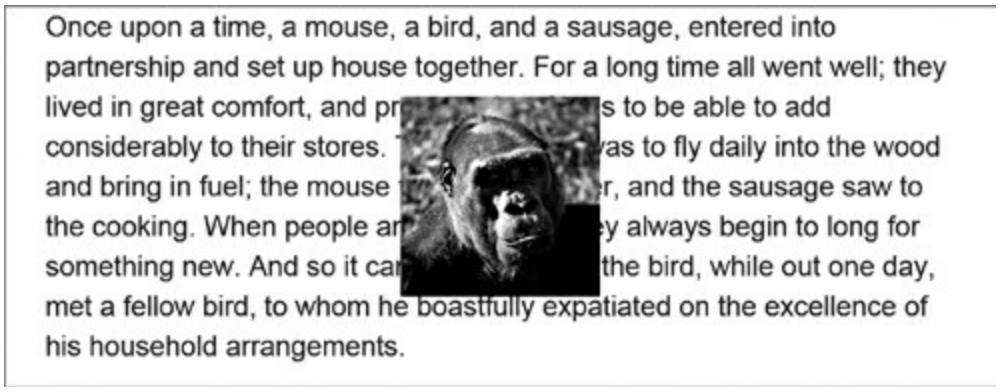


Abb. 11–5
Ein absolut platziertes Element beeinflusst den Textfluss des darunter liegenden Inhalts nicht.[1]

Das nachfolgende Beispiel verwendet den Wert `both`, um das Exclusion-Element beidseitig von Text umfließen zu lassen:

```
#bar { wrap-flow: both; }
```

Den Unterschied erkennen Sie in Abb. 11–6. Die Elemente sind genau wie vorher platziert; weil *#bar* nun aber zum Exclusion-Element deklariert wurde, wird es vom Inhalt in *#foo* beidseitig umflossen.

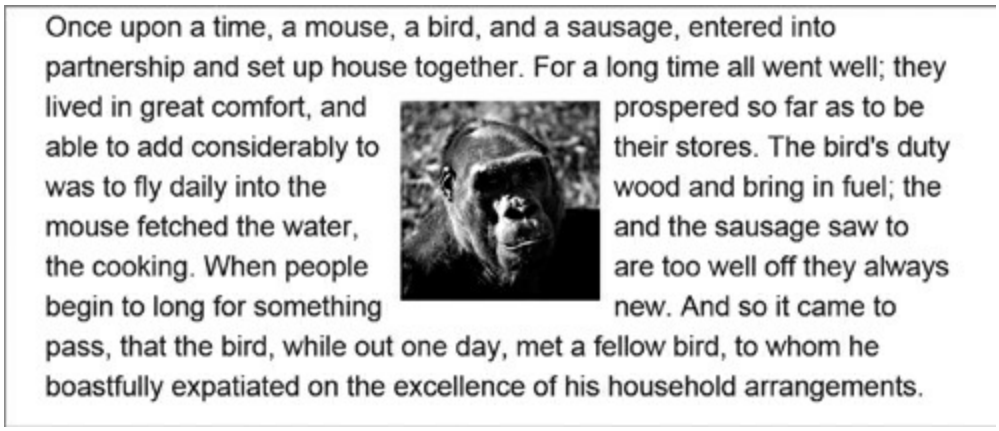


Abb. 11–6
Wenn Sie die wrap-flow-Eigenschaft anwenden, wird das Element im Umflussverhalten des Dokuments berück-sichtigt und der Inhalt umfließt es auf beiden Seiten.

Weitere mögliche Werte für `wrap-flow` sind etwa:

- `start`, um den Inhalt (in von links nach rechts geschriebenen Sprachen) links vom Exclusion-Element umfließen zu lassen, aber nicht rechts davon
- `end` für das Gegenteil
- `maximum` und `minimum`, um den Inhalt jeweils nur auf der Seite mit dem meisten (oder wenigsten) Platz zwischen dem Inhalt und seinem Container-Element umfließen zu lassen
- `clear`, um Inhalt nur über und unter dem Exclusion-Element umfließen zu lassen

Abb. 11–7 zeigt ein paar unterschiedliche Werte im Einsatz. Das erste Beispiel hat den Wert `start`, also umfließt der Inhalt das Exclusion-Element links. Das nächste verwendet den Wert `end`, der Inhalt umfließt das Element also auf der entgegengesetzten Seite. Und im letzten Beispiel sorgt der Wert `clear` dafür, dass das Element seitlich überhaupt nicht umflossen wird.

Damit Sie den Umfluss von Inline-Elementen um das Exclusion-Element besser steuern können, gibt es die `wrap-through`-Eigenschaft. Diese akzeptiert zwei verschiedene Werte: `flow` und `none`. Der zuerst genannte ist der Standardwert; er sorgt dafür, dass Inline-Inhalte das Exclusion-Element umfließen. Der zuletzt genannte erlaubt dies nicht. Das ist hilfreich, wenn Sie den Inhaltsumfluss auf Einzelelementbasis steuern möchten.

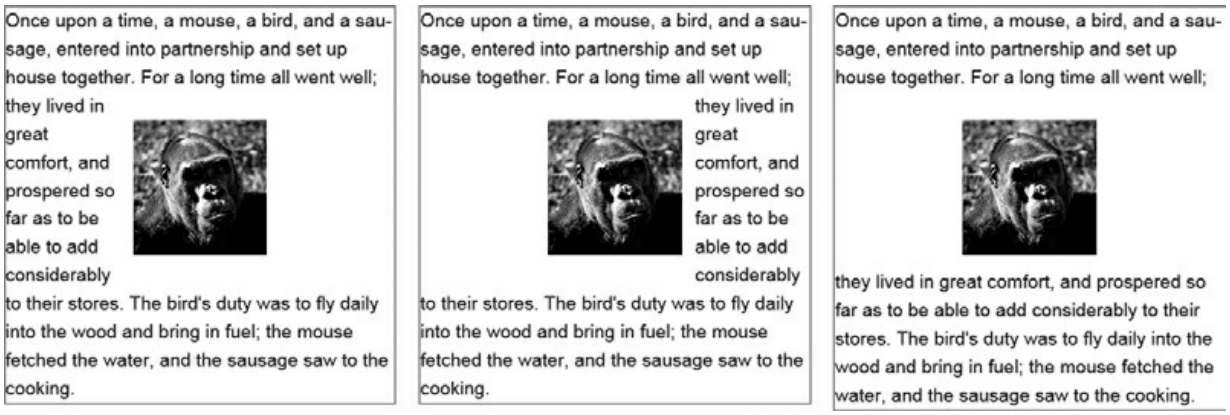


Abb. 11–7 *Je nachdem, welche Werte auf wrap-flow angewendet werden, umfließt der Inhalt das Exclusion-Element auf verschiedenen Seiten.*

Exclusions und Grids

Für mich ist das Spannendste an CSS Exclusions ihr Zusammenspiel mit dem Modul Grid Layout, das ich in Kapitel 4 vorgestellt habe. Jedes Grid-Element lässt sich in ein Exclusion-Element verwandeln, was die Möglichkeiten von Rasterlayouts enorm erweitert. Als einfaches Beispiel gehen Sie von einem Raster aus drei Spalten und zwei Zeilen aus:

```
E {
  display: grid;
  grid-columns: 1fr 1fr 1fr;
  grid-rows: 100px 1fr;
}
```

Auf diesem Raster platziere ich zwei Elemente mit überlappenden Zellen (sie überlappen in Reihe zwei, Spalte zwei):

```
F, G { grid-column-span: 2; }
F {
  grid-column: 2;
  grid-row-span: 2;
}
G { grid-row: 2; }
```

Nach den üblichen Platzierungsregeln würde Element `G` über Element `F` gestapelt, da es im DOM erst später auftaucht. Indem wir aber Element `G` zum Exclusion-Element erklären, umfließt es der Inline-Inhalt von Element `F`:

```
G {
  grid-row: 2;
  wrap-flow: both;
}
```

Wie Sie in der Beispieldatei *grid-exclusion.html* (und in Abb. 11–8) sehen, nimmt `F` nun beim Umfließen von Element `G` etwa die Form eines

umgekehrten L ein. Dieses Layoutmuster findet sich recht häufig in gedruckten Medien, und daher ist es wirklich eine Steigerung, dass Sie es nun auch im Webdesign einsetzen können. Mit Gestaltungsparadigmen, die beim Buch- und Magazinsatz schon seit Hunderten von Jahren selbstverständlich angewendet werden, läuten CSS Exclusions ein komplett neues Zeitalter für das Layout von Webseiten ein.

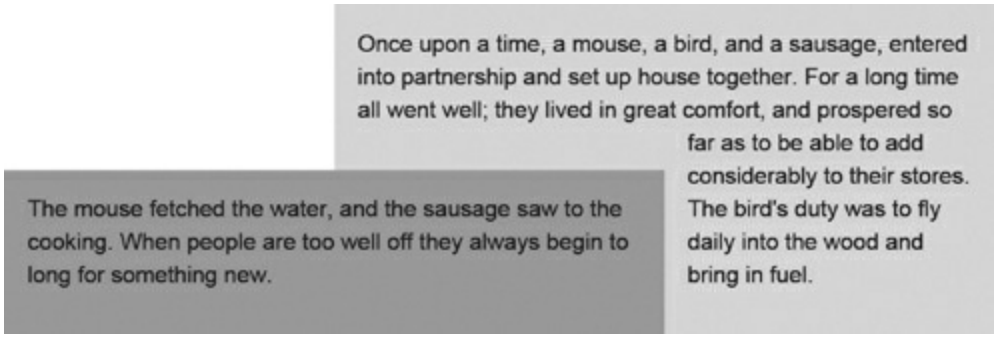


Abb. 11–8
CSS Exclusions arbeiten nahtlos mit CSS-Rastern zusammen. So werden Layoutmuster ermöglicht, die zuvor nicht umsetzbar waren.

Shaped Exclusions

CSS Shapes

Die in den Beispielen dieses Abschnitts verwendeten CSS Exclusions verwendeten bisher normale Block-Elemente; sie erscheinen kastenförmig. In Zukunft sollen sie aber nicht auf rechteckige Exclusions beschränkt bleiben, denn zwei neue Eigenschaften sollen Ihnen das Zeichnen geometrischer Exclusion-Formen ermöglichen. Diese Funktionalität wird inzwischen in einer separaten Spezifikation behandelt, nämlich als »CSS Shapes« (*http://www.w3.org/TR/css-shapes-1/*).

Die Eigenschaften `shape-inside` und `shape-outside` werden als Wert entweder eine geometrische Grundform wie einen Kreis oder eine Ellipse akzeptieren oder auch eine Reihe von Koordinaten, aus denen eine völlig freie Polygonform entsteht. Inhalte könnten die Form dann entweder von außen oder von innen umfließen (oder beides). Damit

bieten sich Möglichkeiten für differenzierte Layouts, wie sie im Druck schon lange üblich sind und die sich nun mit der Dynamisierung der Webinhalte immer weiter verbessern.

11.2.3 Layouts: Noch weiter in die Zukunft geblickt

Während ich dies schreibe, befindet sich eine Reihe neuer Regeln und Eigenschaften in verschiedenen Implementierungsstadien – von kaum vorhanden bis hin zu reinen Vorschlägen. Ich hoffe, dass sie alle übernommen und umgesetzt werden, weil sie Lösungen für unterschiedliche Probleme bieten. Die CSS-Spezifikation befindet sich ständig im Fluss, und nichts kann als selbstverständlich angesehen werden; diese Regeln und Eigenschaften könnten umgesetzt werden, sie könnten teilweise und mit abweichender Syntax oder auch überhaupt nicht implementiert werden.

Es lohnt sich meiner Ansicht nach trotzdem, einen Blick darauf zu werfen. Dafür sprechen zwei Gründe: Zunächst einmal können Sie daran erkennen, wie über Lösungen zu den Problemen des Weblayouts nachgedacht wird – und außerdem werden Sie sie im Fall ihrer Implementierung vielleicht verwenden müssen.

Box Alignment

Mit dem Box-Alignment-Modul wird eine über viele Module hinweg einheitliche Syntax zur Ausrichtung von Elementen innerhalb ihres übergeordneten Elements angestrebt. Als Inspiration für Box Alignment dient die Flexbox-Syntax: `justify-`Eigenschaften werden dabei für die Ausrichtung an der Inline- und Hauptachse verwendet (inline/main axis), `align-`Eigenschaften für die Stapelausrichtung/querende Achse (stacking/cross-axis). Um beispielsweise ein Element entlang seiner Hauptachse auszurichten, würden Sie die `justify-self`-Eigenschaft verwenden; zur Ausrichtung der untergeordneten Elemente eines Elements entlang der Querachse käme dann `align-content` zum Einsatz.

Line Grid

Neben dem wohlbekannten Raster aus Spalten und Zeilen verwenden Typografen häufig auch einen sogenannten *vertikalen Rhythmus*. Das ist ein zweites Raster, das sich aus den Textzeilen und Überschriften auf einer Seite ergibt. Wenn Sie einen vertikalen Rhythmus verwenden, versuchen Sie die senkrechten Abmessungen und Ausrichtungen von Objekten im Sinne einer besseren Lesbarkeit harmonisch zum Text zu gestalten.

Das Modul Line Grid erzeugt auf der Grundlage von `font-size` und `line-height` von Textelementen einen vertikalen Rhythmus. An diesem virtuellen Linienraster können Sie dann Ihre Objekte besser ausrichten. Block-Elemente lassen sich an festen Gitterpunkten einrasten, und somit lassen sich die von der Browser-Engine vorgegebenen Standardpositionen übergehen.

Paged Media

Scrolling ist heute de facto der Standard im Umgang mit umfangreichen Bildschirminhalten. Mit Geräten wie etwa TV-Fernbedienungen und E-Book-Readern mit E-Paper-Displays gestaltet sich das aber nicht immer einfach. Ein besserer Ansatz für solche Geräte könnte stattdessen eine Einzelseitendarstellung sein.

Mit den im Modul Paged Media vorgeschlagenen Funktionen können Sie das leicht umsetzen. Das Modul führt die Eigenschaft `overflow-style` ein. Ein Wert von `paged-x` oder `paged-y` legt automatisch eine horizontale (oder vertikale) Seitenfolge an, während `-controls` (wie etwa `paged-x-controls`) vom Browser erstellte Bildschirmbedienelemente beifügt, falls die entsprechende Nutzerschnittstelle sie benötigt.

Ein anderer Vorschlag sind Seitenvorlagen, die dieses Konzept noch erweitern. Auf allen Seiten fest verankerte Inhaltsbereiche ermöglichen eine einheitliche Leseerfahrung und reichhaltige interaktive Layouts im Magazinstil.

11.2.4 Feature Queries

In Kapitel 5 habe ich über die JavaScript-Bibliothek Modernizr gesprochen. Mit ihr lassen sich bestimmte Fähigkeiten im Browser des Anwenders aufspüren. Ich erwähnte auch kurz die native Adaption in CSS durch die `@supports`-Regel. Auf diese möchte ich nun etwas näher eingehen, denn sie ist extrem nützlich und findet schnellen Einzug in die Browser.

@supports

Die `@supports`-Regel verhält sich wie eine Medienabfrage: Sie erstellen eine logische Abfrage, und falls diese »true« zurückmeldet, werden die Regeln innerhalb der nachfolgenden Klammern angewendet. Anstelle von Medienfunktionen werden hier allerdings CSS-Eigenschaft-Wert-Paare, sogenannte *Feature Queries*, abgefragt. Um beispielsweise zu prüfen, ob der Browser eines Anwenders die `column-count`-Eigenschaft unterstützt und Sie dementsprechende Stile anbieten können, ließe sich eine Abfrage wie die folgende konstruieren:

```
@supports (column-count: 1) { ... }
```

Wie bei den Media Queries können Sie mithilfe logischer Operatoren auch komplexere Abfragen erstellen. Die nachfolgende verwendet den `and`-Operator und stellt damit Browsern mit Unterstützung für die beiden Eigenschaften `column-count` und `box-sizing` Stile zur Verfügung:

```
@supports (column-count: 1) and (box-sizing: border-box) { ... }
```

Mit dem `or`-Operator können Sie auch Abfragen erstellen, die bestimmte Funktionen aufdecken. Das ist extrem nützlich, wenn Sie herstellerspezifische Eigenschaften mit entsprechendem Präfix berücksichtigen. Im nächsten Beispiel werden sowohl die `hyphens`- als auch die `-moz-hy-phens`-Eigenschaften abgeprüft. Sobald eine davon unterstützt wird, kommt die Regel zur Anwendung:

```
@supports (-moz-hyphens: auto) or (hyphens: auto) { ... }
```

Mit dem `not`-Operator können Sie Stile für Browser bereitstellen, die

eine bestimmte Eigenschaft nicht unterstützen. (Im Gegensatz zu den anderen Operatoren muss dieser in runden Klammern stehen.)

```
@supports (not (-webkit-hyphens: auto)) { ... }
```

Feature Queries bieten eine API, die ebenso leicht wie die `@supports`-Regel einzusetzen ist. Sie können beispielsweise mithilfe der `CSS.supports()`-Methode auf eine Einzelfunktion testen, indem Sie ein Eigenschaft-Wert-Paar als zwei Einzelargumente übergeben. Hier wird der `flex`-Wert für die `display`-Eigenschaft abgeprüft:

```
var supports = CSS.supports('display','flex');
```

Und Sie können auch komplette Abfragen als Einzelargument übergeben, wenn Sie diese als Zeichenkette in Hochkommas setzen:

```
var supports = CSS.supports('(column-count: 1) and (display:
flex)');
```

Das Modernizr-Projekt hat bereits begonnen, dies in seine Bibliothek zu implementieren. Falls eine native Unterstützung für `CSS.supports()` gegeben ist, greift das Skript darauf zu, falls nicht, verwendet es die eingebauten Testroutinen von Modernizr.

11.2.5 Cascading Variables

Der Nutzen von Variablen hat sich über die Jahre hinweg in nahezu jeder Programmiersprache erwiesen, trotz regelmäßiger Bitten aus der Community wurden sie jedoch niemals in CSS implementiert. Mit rasch zunehmender Beliebtheit der CSS-Präprozessoren lernt aber eine ganze Coder-Generation Variablen in ihren Stylesheets zu lieben, und die Rufe nach einer nativen Implementierung in die Sprache lassen sich nicht länger ignorieren.

Im aktuellen Vorschlag haben CSS-Variablen eine beschränkte Reichweite. Eine echte Variable lässt jede Art von Wert zu und kann überall im Code eingesetzt werden – etwa um einer Variablen einen Selektor zuzuweisen. Den vorgeschlagenen CSS-Variablen können nur

gültige CSS-Werte zugewiesen werden, und sie lassen sich nur als Wert einer Eigenschaft einsetzen. Zur Abgrenzung werden sie daher als *Cascading Variables* bezeichnet.

Jede Cascading Variable wird mit einer *Custom Property* deklariert. Das ist ein mit `var-` beginnender, benutzerdefinierter Eigenschaftsname, dem ein Wert zugewiesen wird. Hier wird der benutzerdefinierten Eigenschaft *var-foo* der Farbwert *#F00* zugewiesen:

```
:root { var-foo: #F00; }
```

Beachten Sie, dass ich zur Deklaration dieser Eigenschaft den `:root`-Selektor verwendet habe. (Ich erkläre gleich noch, warum.)

Um den Wert der benutzerdefinierten Eigenschaft zu verwenden, rufen Sie ihn mit der `var()`-Funktion auf. Der benutzerdefinierte Name (der Teil nach `var-`) steht dabei in runden Klammern. Der Wert der benutzerdefinierten Eigenschaft wird zugleich der aufrufenden Eigenschaft als Wert übermittelt. Im nachfolgenden Listing wird im `h1`-Element zweimal die *varfoo*-Eigenschaft mit der `var(foo)`-Funktion aufgerufen: einmal für die `border-bottom`-Eigenschaft und einmal für `color`. Auf jede Eigenschaft wird dementsprechend der Farbwert *#F00* angewendet.

```
h1 {
  border-bottom: 1px solid var(foo);
  color: var(foo);
}
```

Cascading Variables haben eine begrenzte *Reichweite*, sie gelten also nur für das Element, für das sie deklariert wurden, sowie für mögliche Unterelemente. Ich habe im Beispiel für diesen Absatz den `:root`-Selektor zur Deklaration einer benutzerdefinierten Eigenschaft verwendet – die Variable ist daher *global gültig*: Sie kann auf jedes Seitenelement angewendet werden. Hätte ich einen anderen Selektor verwendet, wäre der Wert der in der benutzerdefinierten Eigenschaft deklarierten Variablen nur in den Unterelementen der entsprechenden Elemente bzw. des entsprechenden Elements gültig.

Im nachfolgenden Codebeispiel wird etwa die benutzerdefinierte Eigenschaft *var-foo* mit dem Wert *#F00* für das `:root`-Element deklariert, doch ich schreibe darunter noch einen anderen Wert und Selektor. In diesem Fall wäre der Variablenwert eigentlich für das ganze Dokument *#F00*, durch die zweite Zeile ergibt sich nun aber für das *.bar*-Element und seine Unterelemente der Variablenwert *#00F*.

```
:root { var-foo: #F00; }
.bar { var-foo: #00F; }
```

Hinweis

Auf längere Sicht werden auch die beliebtesten Präprozessor-`Mixins` in CSS implementiert werden. Ein Mixin gleicht einer erweiterten Variablen, mit deren Hilfe sich Codeabschnitte über mehrere Selektoren hinweg wiederverwenden lassen. Es wurde sogar schon über die Implementierung vollständiger Variablen gesprochen, die dann Eigenschaftsnamen und Selektoren ersetzen könnten.

11.3 Zusammenfassung

In diesem Kapitel haben wir uns einige der etwas experimentelleren Funktionen der Webplattform angesehen. Diese befinden sich noch in der Testphase und können sich daher noch verändern. Sie sind aber so leistungsfähig und potenziell wichtig für die Zukunft der Plattform, dass ich das Buch nicht zu Ende schreiben konnte, ohne sie zu erwähnen.

Als Erstes waren das die Web Components, die größte Veränderung an HTML seit seiner Erfindung. Web Components bestehen aus einer Reihe von Funktionen. Sie erzeugen ein paralleles DOM zur mehrfachen Nutzung von Codeabschnitten, mit deren Hilfe die Standardelemente in HTML erweitert und verbessert werden. Dies geschieht bei vollständiger Datenkapselung, um Konflikte mit anderen CSS-Regeln und JavaScript-Funktionen auszuschließen.

Als Nächstes befassten wir uns mit der Zukunft von CSS, einer Sprache, die dank der Einflussnahme großer Technologiefirmen ebenfalls großen Veränderungen unterworfen ist. CSS Regions und Exclusions werden möglicherweise die benötigten Werkzeuge für dynamische, individuelle Layouts bereitstellen, die allen Umsetzungsmöglichkeiten in Printmedien das Wasser reichen (oder sie sogar übertreffen?) könnten.

Schließlich behandelte ich neue CSS-Funktionen, die auf Grundlage von Innovationen aus der Webentwickler-Community entstehen. Dazu gehören Feature Queries, die CSS eine native Feature-Detektion im Stil von Modernizr bescheren, und Cascading Variables, mit denen die besten Präprozessor-Funktionen allmählich Einzug in die Sprache selbst halten.

11.4 Literaturempfehlungen

Web Components sind in diesem Moment noch ziemlich neu, es gibt dazu also noch nicht so viele Ressourcen. Als Erstes sollten Sie sich dazu die *entwicklerfreundliche Einführung der Verfasser der Spezifikation* ansehen und danach dann vielleicht die *Präsentation von Eric Bidelman*. Beide Quellen vermitteln Ihnen gut die Grundkonzepte. Zu finden sind sie unter http://dvcs.w3.org/hg/webcomponents/raw-file/tip/explainer/index.html und http://html5-demos.appspot.com/static/webcomponents/index.html (eventuell benötigen Sie hier Google Chrome für die korrekte Anzeige).

Das Shadow DOM ist bisher der am besten implementierte Bestandteil der Web Components und verfügt als solcher über eine umfangreichere Online-Dokumentation. Sowohl *Sitepoint* als auch *HTML5 Rocks* liefern gut verständliche Erklärungen zum Thema, diese finden Sie unter http://www.sitepoint.com/the-basics-of-the-

shadow-dom/ und
http://www..html5rocks.com/en/tutorials/webcomponents/shadowdc

Falls Ihr Browser keine Custom Elements unterstützt, sehen Sie sich *X-Tags* an. Diese von Mozilla entwickelte, experimentelle Bibliothek bildet das Verhalten von Custom Elements nach und bietet dabei ein großes Register vorgefertigter Komponenten: http://x-tags.org/.

Im Internet Explorer 10 wurden CSS Regions als Erstes implementiert, die entsprechende *Dokumentation* eignet sich daher zum Einstieg in die Grundlagen. Siehe http://msdn.microsoft.com/en-us/library/ie/hh673537%28v=vs.85%29.aspx/.

CSS Exclusions finden sich ebenfalls in IE10, sehen Sie also auch hier zuerst in der *Microsoft-Dokumentation* nach. Anschließend schauen Sie sich ein paar *Demos von Adobe* an. Siehe http://msdn.microsoft.com/en-us/library/ie/hh673558%28v=vs.85%29.aspx/ und http://adobe.github.com/web-platform/samples/css-exclusions/.

Im *MDN* findet sich die beste Dokumentation von Feature Queries, obwohl die API derzeit noch undokumentiert ist. Siehe https://developer.mozilla.org/en-US/docs/CSS/@supports/.

Während ich dies schreibe, ist zu Cascading Variables nur etwas im *Spezifikationsentwurf* unter http://dev.w3.org/csswg/css-variables/ zu finden.

Falls der *Vorschlag für Box Alignment* immer noch aktuell ist, während Sie dies lesen, können Sie seinen Fortschritt hier verfolgen: http://dev.w3.org/csswg/css3-align/. Den Vorschlag zu *Line Grid* finden Sie unter http://dev.w3.org/csswg/css-line-grid/.

Håkon Wium Lie und Chris Mills haben eine sehr schöne Einführung zum Seitenkonzept mit CSS geschrieben. Lesen Sie dazu ihren

Artikel *Opera Reader: Paging the Web*: http://people.opera.com/howcome/2011/reader/index.html. Mehr zu Seitenvorlagen finden Sie auf dem *Adobe Web Platform-Blog* unter http://blogs.adobe.com/webplatform/2012/05/31/pagination-templates-in-css/.

Dieses Gorillafoto stammt von Chris Willis und ist unter http://www.fotopedia.com/items/flickr-3059127796/ zu sehen. Es steht unter einer Creative-Commons-Lizenz.

Anhänge

A Browserunterstützung

Nach dem Erscheinen der englischen Originalausgabe dieses Buchs[1] hat sich bei den Browsern viel getan. Das Chromium-Projekt, auf dem der Browser Chrome basiert, verwendet nicht mehr WebKit, sondern seinen eigenen Fork namens Blink. Gerade als ich das englische Manuskript fertigstellt hatte, gab Opera bekannt, seine eigene Engine fallen zu lassen, und arbeitet nun ebenfalls auf Grundlage von Chromium bzw. Blink. Außerdem erschienen wichtige neue Versionen des Internet Explorers und Safari (11 und 7).

Die in den Browsern implementierten Features verändern sich ständig. Ich kann deshalb bestenfalls eine Momentaufnahme liefern. Wenn Sie über den Einsatz eines in diesem Buch beschriebenen Features nachdenken, dann schauen Sie immer erst auf den folgenden Seiten nach dem aktuellsten Informationsstand:

- HTML5 Please, *http://html5please.com/*
- The CSS3 Test, *http://css3test.com/*
- The HTML5 Test, *http://html5test.com/*
- When can I Use..., *http://caniuse.com/*

Als ich Mitte 2012 mit dem Schreiben dieses Buchs anfing, musste ich mich für bestimmte Features entscheiden, die ich behandeln wollte. Darunter befanden sich neben den bereits gut implementierten auch solche, die meiner Einschätzung nach gute Chancen auf Implementierung nach Drucklegung des Buchs (oder kurz darauf) haben würden. Während ich nun, Ende 2013, diese Zeilen schreibe, scheint es mir, dass die allgemeine Implementierung einiger der im Internet Explorer 10 enthaltenen Features länger auf sich warten lässt als erwartet – das gilt etwa für Grid Layout, Regions und Exclusions –, ansonsten geht es aber überall gut voran.

Die wichtigsten Browser

Es gibt viel zu viele Browser, als dass ich für jeden einen vernünftigen Überblick der unterstützten Funktionen erstellen könnte. Stattdessen halte ich mich in diesem Anhang – wie schon im ganzen Buch – an die wichtigsten modernen Desktop-Browser: Chrome, Firefox, Internet Explorer 10+ und Safari sowie ihre mobilen Pendants.

Als die englische Originalausgabe dieses Buchs in den Druck ging, gab Opera bekannt, seine eigene Presto-Rendering-Engine auszumustern und in zukünftigen Browserversionen stattdessen Chromium einzusetzen, also jenen Entwicklungszweig von WebKit, auf dem auch Chrome basiert. Das heißt aber nicht, dass Presto kurzfristig von der Bildfläche verschwinden wird – die Engine wurde bereits in zahlreiche Geräte eingebettet, die kaum Updates erfahren werden, etwa in Fernsehern und Spielekonsolen. Langfristig sollten zwar dieselben Funktionen wie in Chrome unterstützt werden; wegen der breiten Nutzung früherer Versionen behandle ich Opera hier gesondert.

Wenn ich von Mobil-Browsern spreche, meine ich in der Regel sowohl Smartphone- als auch Tablet-Varianten. Meist handelt es sich dabei um die Mobilversion von Safari und dem Android-Browser (beide basieren zwar auf WebKit, aber sie unterscheiden sich doch in etlichen Punkten). Firefox, Internet Explorer und Opera verwenden auf unterschiedlichen Plattformen dieselbe Rendering-Engine (beachten Sie aber die vorhergehenden Absätze über Opera). Also erwähne ich die Mobilversionen dieser Browser nur dann, wenn es Unterschiede gibt (was nur selten der Fall ist).

Wenn ich von Android spreche, meine ich den Standardbrowser, der in den meisten Android-Versionen enthalten ist. Neuere Versionen werden wahrscheinlich mit der neuen Mobilversion von Chrome ausgeliefert werden, die, ebenso wie Firefox und Opera, als mehr oder weniger identisch zur entsprechenden Desktop-Version angesehen werden kann.

Wie bereits erwähnt, gibt es keinen echten Ersatz für vollwertige

Tests auf den entsprechenden Geräten. Falls möglich, sollten Sie sich eine Gerätebibliothek anlegen oder einem Open Device Lab in Ihrer Region beitreten. Ist das gar keine Option, sprechen Sie mit anderen Entwicklern über deren Erfahrungen.

Experimentelle Features aktivieren

Viele Browser, insbesondere Chrome und Firefox, gehen inzwischen viel vorsichtiger bei der Implementierung experimenteller Funktionen vor. Zuvor wurden Features implementiert, mit Hersteller-Präfix versehen und allen Anwendern verfügbar gemacht. Jetzt müssen Sie gewisse Funktionen erst ausdrücklich mit einem entsprechenden Konfigurations-Flag aktivieren.

In Firefox geben Sie dazu `about:config` in die Adressleiste ein. Es erscheint eine Warnmeldung, die Sie vor den Konsequenzen bei Änderungen der erweiterten Browsereinstellungen warnt. Wenn Sie sich davon nicht abschrecken lassen, können Sie das gewünschte Feature hier finden und einschalten. Nach einem Neustart des Browsers steht Ihnen die Funktion dann zur Verfügung.

In Chrome ist der Ablauf nahezu identisch, außer dass Sie `chrome://flags` eingeben und keine Warnmeldung erscheint. Die Funktionen werden meist durch Anklicken eines Links mit der Beschriftung *Aktivieren* freigeschaltet.

Kapitel 1: Die Webplattform

Jeder moderne Desktop-Browser wird mit einer Reihe von Entwicklerwerkzeugen ausgeliefert, zu denen auch eine Konsole gehört (nur beim Internet Explorer bis Version 7 fehlt diese). Auf Mobilgeräten und Tablets ist die Sache etwas komplizierter: Die meisten Browser bringen standardmäßig keine Entwicklerwerkzeuge mit, sie lassen sich

zum Debuggen aber mit ihren Desktop-Pendants verbinden. Dieser Vorgang ist auf Seite 25 im Abschnitt »Testen, testen und nochmals testen« beschrieben.

Kapitel 2: Struktur und Semantik

Die neueren Strukturelemente von HTML5 werden in Internet Explorer 9 und höher und in allen anderen modernen Browsern dargestellt. Einige dieser Elemente werden immer noch diskutiert, während ich dies schreibe. Manche Browser bieten erstmals Unterstützung für ein `main`-Element (obwohl sich der für die Spezifikation verantwortliche Ian Hickson dagegen ausgesprochen hat). Andere Elemente werden möglicherweise wieder verworfen.[2]

Die attributbasierten Erweiterungen zu Barrierefreiheit und Semantik, WAI-ARIA, Mikroformate, RDFa und Mikrodaten können in jedem Browser verwendet werden. Die Microdata-API ist in Firefox und Opera implementiert, wurde aber in Chrome wieder entfernt, weil den Entwicklern die Nachfrage zu gering schien.

Datenattribute werden ebenfalls von allen Browsern unterstützt, obwohl die auf `dataset` basierende API in Internet Explorer oder Android bis Version 2.3 nicht enthalten ist. Die jQuery-Methode funktioniert auf allen Browsern.

Kapitel 3: Mediengerechtes CSS

Während ich dies schreibe, gibt es Media Queries ab Internet Explorer 9 sowie in allen anderen großen Browsern neueren Datums. Media-Features hinsichtlich der Gerätedimensionen sind am weitesten verbreitet. Das Media-Feature `resolution` findet sich in Internet Explorer 10+, Firefox und Opera, aber noch nicht in Safari (obwohl es Ende 2012 in den WebKit-Core implementiert wurde).

Das Attribut `devicePixelRatio` findet sich in IE11 und allen anderen modernen Browsern, inklusive ihrer mobilen Varianten. Die Einheit `dppx` ist in Chrome und Firefox implementiert. Die `@viewport`-Regel wurde in alten Opera-Versionen, Internet Explorer 10+ und WebKit implementiert, jeweils mit entsprechendem Hersteller-Präfix. Die `matchMedia`-API findet sich in Internet Explorer 10+ und allen anderen modernen Browsern, aber nicht in Android bis Version 2.3.

Die CSS-Eigenschaft `box-sizing` ist in allen Browsern enthalten, in Firefox und Android-Versionen bis 3.0 benötigt sie aber ein Hersteller-Präfix. Firefox unterstützt als einziges den `padding-box`-Wert. Die Funktion `calc()`findet sich in IE9 und darüber sowie in modernen Browsern; bis zur Version 6.0 von Safari (Mac OS und iOS) erfordert sie das Präfix `-webkit-`. In Android und Opera fehlt die Funktion.

Die relativen Längeneinheiten – `vh`, `vw` usw. – finden sich in IE9 (mit einigen Bugs) und IE10+, Firefox und den meisten WebKit-Browsern außer Android. In Opera fehlen sie jedoch. Die Einheit `rem` wurde in IE9 und höher und in allen anderen großen Browsern umgesetzt.

Die Eigenschaften `object-fit` und `object-position` sind nur in Opera implementiert und in der Spezifikation als »gefährdet« (*at risk*) gekennzeichnet. Besonders weil Opera nun auf Chromium basiert, hatte ich in dieser Hinsicht in der Originalversion des Buchs aufs falsche Pferd gesetzt.

Kapitel 4: Neue CSS-Konzepte

Die Eigenschaften für mehrspaltiges Layout wurden in IE10+ und allen übrigen modernen Browsern implementiert. Hersteller-Präfixe sind in allen Browsern außer IE und alten Opera-Versionen erforderlich. In Firefox fehlt zudem die Unterstützung für die `column-span`-Eigenschaft. Nur alte Opera-Versionen und IE10+ unterstützen die `break-before`- und `break-after`-Eigenschaften.

Flexbox wird in allen großen Browsern unterstützt; Safari (iOS und

Mac OS) erfordern allerdings das Präfix `-webkit-`, und Safari-Browser der Version 6 und älter verwenden eine Hybrid-Syntax aus der aktuellen und einer älteren Version: Die `justify-content`-Eigenschaft ist nicht implementiert, stattdessen gibt es die ältere `box-pack`-Eigenschaft. In Safari 7 wurde dies behoben.

IE10 verwendet ebenfalls eine veraltete Syntax mit dem Präfix `-ms-`. Ich empfehle Ihnen für ausführliche Informationen den »Internet Explorer 10 Guide for Developers« zur Lektüre: *http://msdn.microsoft.com/library/ie/hh673531%28v=vs.85%29.aspx/*. Im Internet Explorer wurde das Präfix entfernt.

Firefox benötigt keine Präfixe, unterstützt aber nur einzeiliges Flexbox, die `flex-wrap`-Eigenschaft und das Kürzel `flex-flow` werden also ignoriert.

Die Syntax des Grid-Layout-Moduls hat sich seit dem Erscheinen des Buchs erneut geändert. Wie in Kapitel 4 beschrieben, unterstützt IE10+ momentan eine ältere Syntaxversion mit dem `-ms-`-Präfix. Apple kündigte an, dass die Implementierung in Safari 7 sich einer aktuelleren Syntax bedienen würde; aber dies scheint im endgültigen Release nicht der Fall zu sein. Die `grid-template`-Eigenschaft wurde derzeit noch in keinem Browser implementiert.

Kapitel 5: Modernes JavaScript

Das `async`-Attribut findet sich in IE10 und den meisten anderen Browsern außer Android bis Version 2.3 und alten Opera-Versionen. Für das `defer`-Attribut gilt dasselbe, jedoch wird es auch mindestens bis zurück zur IE-Version 8 unterstützt.

Die `addEventListener()`-Methode wurde in IE9 und höher und in allen anderen großen Browsern umgesetzt, ebenso wie das `DOMContentLoaded`-Ereignis.

Trotz einiger Patentunsicherheiten finden wir Touch Events in Chrome, Firefox, Opera, Safari für iOS und Android. IE10+ bietet

Unterstützung für Pointer Events mit Hersteller-Präfix, wie etwa `MSPointer-Down`.

Die Methoden `querySelector()` und `querySelectorAll()` wurden vollständig in alle modernen Browser ab IE8 implementiert. Die `getElementsByClassName()`-Methode ist fast ebenso verbreitet, es fehlt lediglich die Unterstützung in IE8. Das `classList`-Objekt wurde ab IE10 und in den meisten anderen Browsern außer Android-Version 2.3 und darunter umgesetzt.

Kapitel 6: Geräte-APIs

Die Geolocation-API ist in IE9 und allen anderen großen Browsern vertreten. Die Device-Orientation-API findet sich in mobilen WebKit-Browsern sowie in den Mobilversionen von Chrome und Firefox. Denken Sie aber daran, dass diese APIs von bestimmten Hardwarefunktionen des Telefons abhängig sind; nur weil die Device-Orientation-API in einem Browser implementiert ist, muss das Gerät nicht notwendigerweise mit einem Beschleunigungssensor ausgestattet sein.

Die Full-Screen-API ist in IE11 und allen anderen modernen Browsern implementiert, jeweils mit entsprechenden Hersteller-Präfixen. Die Implementierungen Firefox unterscheiden sich geringfügig voneinander; statt eines Erklärungsversuchs verweise ich auf den MDN-Artikel »Using Fullscreen Mode« unter *https://developer.mozilla.org/docs/DOM/Using_fullscreen_mode/*. Firefox, WebKit und Blink sowie IE11 unterstützen die Pseudoklasse `:full-screen`wiederum mit entsprechenden Hersteller-Präfixen.

Die Vibration-, Battery-Status- und Network-Information-APIs sind nur in der mobilen Firefox-Version verfügbar. In Chrome 30 wurde Vibration implementiert, ist jedoch standardmäßig abgeschaltet. Die übrigen wurden im Laufe des Jahres 2012 in WebKit übernommen, doch konnte ich noch keine funktionierenden Implementierungen sehen.

Die `getUserMedia()`-Methode ist im alten Opera-Browser zu finden, mit Hersteller-Präfix ebenso in Firefox und Chrome (`mozGetUserMedia`, `webkitGetUserMedia`).

Web Storage gibt es in IE8 und höher sowie in allen anderen wichtigen Browsern. Die Drag-and-Drop-API wird in IE8 und IE9 zum Teil und in IE10 und anderen großen Desktop-Browsern vollständig unterstützt. Naturgemäß gibt es dafür keine Unterstützung in Mobilbrowsern.

Die File-API ist in Firefox, Chrome, Safari (iOS und Mac OS) sowie in Opera vollständig implementiert. Teilweise Unterstützung gewähren IE10 und Android. Die FileReader-API wird ab IE10 und von allen anderen Desktop-Browsern vollständig bereitgestellt sowie von WebKitMobilbrowsern inklusive Android ab Version 3.0.

Kapitel 7: Bilder und Grafiken

Eine gewisse SVG-Unterstützung bieten IE9 und höher, Android ab Version 3.0 und alle übrigen modernen Browser. Bei den SVG-Filtern ist die Abdeckung etwas schlechter: In Android fehlen sie ganz, im Internet Explorer ab Version 9 gibt es sie nur mit großen Einschränkungen) nur in Firefox lassen sich SVG-Filter auf HTML-Elemente zuverlässig anwenden. Die neue CSS-Funktion `filter()` ist derzeit in Chrome und Safari mit dem `-webkit`-Präfix enthalten. Der Einsatz von Fragment Identifiers in SVG ist nur in IE10 und höher sowie Firefox möglich.

Das `canvas`-Element wird in IE9 und höher und in allen anderen wichtigen Browsern unterstützt. Firefox, Chrome, Safari (Desktop) und Opera verfügen alle über eine WebGL-Implementierung, auch wenn diese in einigen Fällen (insbesondere in Safari und Chrome für Android) standardmäßig deaktiviert ist.

Kapitel 8: Neue Formulare

Die Unterstützung für die unterschiedlichen neuen Formularelemente, besonders jene mit Bildschirmbedienelementen, ist unter den Browsern sehr unterschiedlich ausgeprägt und ändert sich ständig. Statt hier den Versuch einer Darstellung zu unternehmen, verweise ich lieber auf den HTML5-Test, wo Sie die umfangreichste und aktuellste Dokumentation des Sachstands finden. Die neuen `input`-Typen funktionieren in der Regel zuverlässig, weil Browser alternativ auf den `text`-Typ zurückgreifen, falls ein anderer Wert nicht erkannt werden sollte.

Die Form-Validation-API wurde in IE10+ und allen anderen großen Browsern umgesetzt. Safari unterstützt die API, liefert aber keine Fehlermeldungen auf dem Bildschirm.

Kapitel 9: Multimedia

Die `video`- und `audio`-Elemente finden sich mitsamt der zugehörigen APIs in IE9 und höher sowie in allen anderen wichtigen Browsern. Allerdings müssen Sie die Einschränkungen hinsichtlich der unterstützten Dateitypen beachten, die ich in Kapitel 9 erläutert habe. Das `track`-Element wird in den Desktopversionen von IE10+, Safari 6+ und höher, Chrome und Opera unterstützt; Chrome für Android bietet als einziger Mobilbrowser ebenfalls Unterstützung. Media Fragments sind in Firefox und WebKit-Browsern implementiert.

Eine experimentelle Implementierung der Web-Audio-API mithilfe des `webkitAudioContext()`-Konstruktors findet sich in Chrome und Safari (iOS und Desktop) und soll auch in Firefox aufgenommen werden. Von WebRTC wird derzeit nur die in Kapitel 6 erwähnte `getUserMedia()`-Methode unterstützt.

Kapitel 10: Webanwendungen

IE10 und alle anderen großen Browser unterstützen AppCache.

Kapitel 11: Die Zukunft

Chrome bietet am meisten Unterstützung der neuen Web-Components-Funktionen; implementiert wird hier der Shadow DOM (mithilfe des mit Hersteller-Präfix versehenen `webkitShadowRoot()`-Konstruktors, der jedoch explizit freigeschaltet werden muss). Das `template`-Element ist in Chrome und Firefox implementiert. Dasselbe gilt für Scoped Styles und die Registrierung benutzerdefinierter Elemente mit `document.register()`.

CSS Regions sind in IE10 und höher, Safari 7 und Chrome vorhanden. In allen drei Fällen benötigen die Eigenschaften Hersteller-Präfixe, und IE10 und höher lässt nur den Inhalt eines `iframe` als Quelle zu.

Exclusions gibt es derzeit nur in IE10 und höher mit dem `-ms-`-Präfix.

Die Feature-Queries-Regel `@supports` sowie die DOM-Methode `CSS.supports()` wird von Chrome und Firefox unterstützt.

Cascading Variables sind nur in Chrome implementiert und müssen explizit freigeschaltet werden.

Anmerkung des Verlages: Dieses Kapitel wird vom Autor aktualisiert und online zur Verfügung gestellt (http://modernwebbook.com/). Für die vorliegende deutsche Übersetzung konnten wir den Stand vom November 2013 verwenden. Es lohnt sich natürlich, gelegentlich auf der Website des Autors oder auf der dpunkt-Buchwebseite (http://www.dpunkt.de/gasston) nach weiteren Updates zu schauen.

Dies gilt etwa für hgroup, das bereits nicht mehr dazugehört.

B Literaturempfehlungen

Dieser Anhang fasst noch einmal alle in den »Literaturempfehlungen« der einzelnen Kapitel aufgeführten Links zusammen, sodass Sie sie bequem nachschlagen können. Eine stets aktuelle und gegebenenfalls um neue und interessante Links erweiterte Version dieser Liste finden Sie auf der Website zum englischen Buch (http://modernwebbook.com).

Einleitung

Die in der Einleitung genannten Statistiken stammen aus vielen verschiedenen Quellen, vor allem *The Mobile Industry in Numbers* von VisionMobile auf http://www.visionmobile.com/blog/2012/10/infographic-the-mobile-industry-in-numbers/ sowie »The Internet of Things« von Cisco auf http://blogs.cisco.com/news/the-internet-of-things-infographic/.

Eine gute Einführung in das Internet of Things finden Sie im *The Next Web*-Artikel *Why 2013 will be the year of the Internet of Things*: http://thenextweb.com/insider/2012/12/09/the-future-of-the-internet-of-things/.

David Storey schrieb einen hervorragenden Beitrag über das nicht auf Smartphones ausgerichtete mobile Web: *See your site like the rest of the world does. On the Nokia X2-01*. Sie finden ihn auf http://generatedcontent.org/post/31441135779/mobileweb-row/.

Der beste Artikel, den ich über die Webentwicklung für mobile Geräte gelesen habe und aus dem ich in diesem Kapitel zitiere, ist *The 10 Principles of Mobile Interface Design* von Jonathan Stark:

http://www.netmagazine.com/features/10-principles-mobile-interface-design/. Jason Grigsbys hervorragender Artikel *Responsive Design for Apps* ist ein guter Einstieg in die Gestaltung für unterschiedliche Bildschirmabmessungen und -darstellungen: http://blog.cloudfour.com/responsive-design-for-apps-part-1/. Der im *UX Magazine* erschienene Artikel *Five Lessons from a Year of Tablet UX Research* von Brennan Browne enthält herausragende Erkenntnisse über die Nutzung von Tablets: http://uxmag.com/articles/five-lessons-from-a-year-of-tablet-ux-research/.

Ein guter Einstieg für Anna Debenhams Untersuchungen über Spielekonsolen-Browser ist ihr Artikel *Testing Websites in Game Console Browsers* auf *A List Apart*: http://www.alistapart.com/articles/testing-websites-in-game-console-browsers/.

Noch einmal Jason Grigsby: Er hielt den wunderbaren Vortrag *The Immobile Web* über die Entwicklung für TV-Geräte. Das Video finden Sie auf http://vimeo.com/44444464/, die zugehörigen Präsentationsfolien auf http://www.slideshare.net/grigs/the-immobile-web/.

Die komplette Untersuchung zur Multi-Device-Nutzung finden Sie im Google-Blog-Beitrag *Navigating the new multi-screen world* auf http://googlemobileads.blogspot.co.uk/2012/08/navigating-new-multi-screen-world.html.

Es ist immer gut, zukunftsweisende Websites zu gestalten. Mehr auf http://futurefriend.ly/.

Kapitel 1

Die *Liste der Technologien, die die Webplattform ausmachen*, finden Sie auf http://platform.html5.org/. Bruce Lawson schlug den Begriff

NEWT in seinem Blog vor: http://www.brucelawson.co.uk/2010/meet-newt-new-exciting-web-technologies/.

Die *HTML5-Spezifikation des W3C* finden Sie auf http://www.w3.org/TR/html5/ und die *Live-Spezifikation des WHATWG* auf http://whatwg.org/html. Es gibt dort auch eine nützliche *Edition for Web Developers*, in der gewisse obskure Sprachelemente weggelassen werden und die deshalb besser lesbar ist: http://developers.whatwg.org/.

Die *vollständige Formulierung des HTML5-Standards* gibt es auf http://html5boilerplate.com/. Denken Sie daran: Verwenden Sie nur die von Ihnen benötigten Teile; Sie müssen nicht alles wörtlich umsetzen.

Mehr über den Implementierungsgrad verschiedener Funktionen bieten Alexis Deverias Site *Can I Use* auf http://caniuse.com/, die Community-Site *HTML5 Please* auf http://html5please.com/ sowie *The HTML5 Test* auf http://html5test.com/.

Die Website *LabUp!* ist eine Ressource für Open Device Labs: http://lab-up.org/. Der Testleiter der BBC, David Blooman, beschäftigte sich in dem langen und ausführlichen Artikel *Testing for Dummies* damit, wie ein großes Unternehmen Multi-Device-Tests durchführt: http://mobiletestingfordummies.tumblr.com/post/20056227958/testi

Patrick Meenans Präsentationsfolien für seinen Vortrag *Taming the Mobile Beast* enthalten eine Fülle von Links und Informationen über Tests auf mobilen Geräten: http://www.slideshare.net/patrick-meenan/velocity-2012-taming-the-mobile-beast/. Und Anna Debenhams Artikel für A List Apart *Testing Websites in Game Console Browsers* handelt von ... na ja, der Titel ist ziemlich selbsterklärend: http://www.alistapart.com/articles/testing-websites-in-game-con-sole-browsers/.

Opera bietet eine *detaillierte Anleitung zum Remote-Debugging*:

http://www.opera.com/dragonfly/documentation/remote/. *weinre* können Sie sich von http://people.apache.org/~pmuellr/weinre/docs/latest/ herunterladen. Weitere *Informationen über Adobe Edge Inspect* erhalten Sie auf http://html.adobe.com/edge/inspect/.

Kapitel 2

HTML5 Doctor ist die beste Quelle für Informationen über die meisten HTML5-Themen. Sie finden beispielsweise im folgenden *Artikel von Mike Robinson* die meiner Meinung nach beste Definition der neuen Gliederungsalgorithmen: http://html5doctor.com/outlines/. Das *in Abb. 2–1 gezeigte Schaubild* können Sie von http://html5doctor.com/resources/#flowchart/herunterladen.

Lesen Sie auch *Derek Johnsons Artikel* im Smashing Magazine: http://coding.smashingmagazine.com/2011/08/16/html5-and-the-document-outlining-algorithm/.

Für tiefere Einblick in das Problem der strukturierenden Elemente in HTML5 (Sectioning Elements) empfehle ich Luke Stevens" Buch *The Truth About HTML5*. Sie finden es auf http://www.truthabouthtml5.com/. Wenn Sie die *vollständige HTML5-Spezifikation* studieren möchten, um selbst zu entscheiden, rate ich Ihnen zur Entwicklerversion auf http://developers.whatwg.org/sections.html.

Die *vollständige WAI-ARIA-Spezifikation* finden Sie auf http://www.w3.org/TR/wai-aria/. Für Informationen über die Barrierefreiheit von HTML5 lohnt es sich, den *Paciello Group Blog* zu lesen. Wichtig ist in diesem Zusammenhang der Beitrag http://www.paciellogroup.com/blog/2010/10/using-wai-aria-landmark-roles/.

Divya Manians Artikel über Semantik erschien im Smashing Magazine auf

http://coding.smashingmagazine.com/2011/11/11/our-pointless-pursuit-of-semantic-value/. Wenn Sie mehr über Aboutness und die Bedeutung der Semantik lesen möchten, empfehle ich unbedingt das Buch *Ambient Findability: What We Find Changes Who We Become* von Peter Morville (O"Reilly, 2005). Die Website http://webdatacommons.org/ liefert *Informationen und Statistiken über Sites, die strukturierte Daten nutzen.*

Lesen Sie *alles über Mikroformate* auf http://microformats.org/. Eine Überarbeitung der Syntax, Microformats 2.0, begann im Jahr 2010 und ist immer noch in Entwicklung begriffen; erfahren Sie mehr darüber auf http://microformats.org/wiki/microformats-2.

Möchten Sie mehr über das RDFa-Format lesen, hat das W3C eine exzellente *Einführung* veröffentlicht: http://www.w3.org/TR/xhtmlrdfa-primer/. Die beste *Ressource über Mikrodaten* stammt ebenfalls vom HTML5 Doctor: http://html5doctor.com/microdata/. Sind Sie ganz masochistisch veranlagt und möchten Sie die *Spezifikation im Ganzen* lesen, finden Sie diese auf http://www.w3.org/TR/microdata/.

Weitere *Informationen über Schema.org* erhalten Sie auf http://schema.org/, und die *Google-Dokumentation über Rich Snippets* befindet sich auf http://support.google.com/webmasters/bin/answer.py?hl=en&answer=99170. Das *Test-Tool* finden Sie auf http://www.google.com/webmasters/tools/richsnippets/.

John Resig schrieb in seinem Blog http://ejohn.org/blog/html-5-data-attributes/ eine einprägsame *Einführung zu den Datenattributen*, und die `data()`-Methode wird vollständig auf der jQuery-Website auf http://api.jquery.com/data/ dokumentiert.

Kapitel 3

Die erste Anlaufstelle, wenn Sie mehr über Media Queries erfahren möchten, sollte Zoe Mickley Gillenwaters Post *Essential Considerations for Crafting Quality Media Queries* sein: http://zomigi.com/blog/essential-considerations-for-crafting-quality-media-queries/.

Die maßgebliche Autorität für Mobilgeräte ist Peter-Paul Koch. Wenn Sie mehr über virtuelle und physische Pixel erfahren möchten, empfehle ich Ihnen für den Einstieg seinen Artikel *A Pixel Is Not a Pixel Is Not a Pixel*: http://www.quirksmode.org/blog/archives/2010/04/a_pixel_is_not.ht Wikipedia bietet eine *Auflistung der gängigen Geräteauflösungen und Pixeldichten*: http://en.wikipedia.org/wiki/List_of_displays_by_pixel_density.

Patrick Lauke schrieb einen Artikel über benutzerkontrollierte DPR: *devicePixelRatio in Opera Mobile*. Sie finden ihn auf http://my.opera.com/ODIN/blog/2012/07/05/devicepixelratio-in-opera-mo-bile. Matt Wilcox"s Artikel *The Responsive Design Process* enthält ein gutes Glossar mit Schlüsselbegriffen sowie zahlreiche praktische Design-Hinweise: http://mattwilcox.net/archive/entry/id/1078/.

Mehr darüber, wie unterschiedliche Browser Dezimalstellen runden, erfahren Sie in John Albin Wilkins" Beitrag *Responsive Design"s Dirty Little Secret*: http://www.palantir.net/blog/responsive-design-s-dirty-little-secret/.

Paul Irishs Blogbeitrag *box-sizing: border-box FTW* legt seine Gründe dar, warum diese Eigenschaft global gesetzt werden sollte: http://paulirish.com/2012/box-sizing-border-box-ftw/.

Luke Wroblewskis Buch *Mobile First* ist bei A Book Apart erschienen: http://www.abookapart.com/products/mobile-first/.

Mehr über Inhalts-Breakpoints erfahren Sie in dem Artikel des australischen Webdesign-Studios Jordesign

(http://www.jordesign.com/blog/responsive-breakpoints-from-the-content-out/) und dem *Artikel des Entwicklers Thierry Koblentz* (http://coding.smashing-magazine.com/2012/03/22/device-agnostic-approach-to-responsive-web-design/).

Die *Geschichte des momentan wichtigsten Vorschlags für responsive Bilder und Informationen zu seinem aktuellen Status* finden Sie auf der Website der Responsive Images Community Group: http://www.w3.org/community/respimg/.

Matt Wilcox' *Tool für adaptive Bilder* finden Sie auf http://adaptive-images.com/.

Kapitel 4

Das stets zuverlässige MozDev enthält eine klare Einführung zu mehrspaltigen Layouts, *Using CSS multi-column layouts:* https://developer.mozilla.org/en-US/docs/CSS/Using_CSS_multi-column_layouts/.

Die Flexbox-Syntax hat sich so oft geändert, dass es quasi keine aktuelle Online-Ressource mehr gibt! Auch wenn sich Stephen Hays Artikel *Learn You a Flexbox for Great Good!* auf http://www.the-hay-stack.com/2012/01/04/learn-you-a-flexbox/ auf eine veraltete Syntax bezieht, sollten Sie ihn lesen, denn Stephens Kenntnisse des CSS-Layouts sind unvergleichlich.

Die beste Erklärung des Grid-Layout-Moduls, zumindest hinsichtlich der Implementierung im Internet Explorer 10, finden Sie im *»Internet Explorer 10 Guide for Developers«*: http://msdn.microsoft.com/en-us/library/ie/hh673533%28v=vs.85%29.aspx/.

Kapitel 5

Die *Illustration in Abb. 5–1* stammt aus Peter Beverloos Blog: http://peter.sh/experiments/asynchronous-and-deferred-javascript-execution-explained/.

Christian Heilmann hat eine fundierte *Einführung in JavaScript-Ereignisse* für das Smashing Magazine geschrieben: http://coding.smashingmagazine.com/2012/08/17/javascript-events-responding-user/. Die *PointerEvents-Bibliothek* finden Sie bei GitHub unter https://github.com/toolkitchen/PointerEvents/.

Die jQuery-Website http://jquery.com/ liefert *Hinweise für den Einstieg*, die ausgezeichnete Dokumentation finden Sie unter http://docs.jquery.com/Main_Page. Die statistischen Erhebungen zur Verwendung von jQuery entstammen dem Blogbeitrag *jQuery Now Runs on Every Second Website* unter http://w3techs.com/blog/entry/jquery_now_runs_on_every_second_v

Alle *Bibliotheken für Mobilgeräte* sind vollständig dokumentiert: jQuery Mobile unter http://jquerymobile.com/, Zepto.js unter http://zeptojs.com/und jQTouch unter http://jqtouch.com/.

YepNope.js finden Sie unter http://yepnopejs.com/ und eine gute *Einführung* unter http://net.tutsplus.com/tutorials/javascript-ajax/easy-script-loading-with-yepnope-js/.

Die Modernizr-Website http://modernizr.com/ bietet eine *vollständige Dokumentation* und ein *konfigurierbares Build-Tool*. Den Leitfaden *The All-In-One Entirely-Not-Alphabetical No-Bullshit Guide to HTML5 Fallbacks* (ich habe den Titel nicht erfunden) finden Sie unter https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills/.

Christopher Coenraets hat ein hervorragendes *Einstiegstutorial für Mustache* geschrieben, allerdings hat sich die Syntax geringfügig geändert: http://coenraets.org/blog/2011/12/tutorial-html-templates-with-mustache-js/. Die *vollständige Dokumentation zu Mustache.js* befindet sich hier: https://github.com/janl/mustache.js/.

Es gibt zahlreiche *Entwicklertools zum Experimentieren und Debuggen*, ganz hervorragend sind sowohl http://jsbin.com/ als auch http://jsfiddle.net/.

Die *aktuellste Version des Grid-Layout-Moduls* finden Sie unter http://dev.w3.org/csswg/css-grid.

Lesen Sie *Mark Boultons offenen Brief* zur Grid-Layout-Terminologie auf http://www.markboulton.co.uk/journal/comments/open-letter-to-w3c-css-working-group-re-css-grids.

Kapitel 6

Dive Into HTML5 liefert unter http://diveintohtml5.info/geolocation.html eine ausführliche *Erklärung der Geolocation-API*, während der MozDev-Artikel *Orientation and Motion Data Explained* einen guten Überblick zur dreidimensionalen Ausrichtung und Bewegung gibt: https://developer.mozilla.org/en-US/docs/DOM/Orientation_and_motion_data_explained/.

Die Fullscreen-API wird im Sitepoint-Artikel *How to Use the HTML5 Full-Screen API* von Craig Buckler erklärt, allerdings wurde die API geringfügig überarbeitet, während ich dies schrieb; einige Objektnamen oder Eigenschaften könnten sich also geändert haben. Den Artikel finden Sie unter http://www.sitepoint.com/html5-full-screen-api/.

Die *Battery-Status-API* wird von David Walsh unter http://davidwalsh.name/battery-api/ gut erklärt, und eine *Diskussion der früheren und neu aktualisierten Network-Information-API* findet sich hier: http://nostrongbeliefs.com/a-quick-look-network-information-api/.

HTML5 Rocks bietet im Artikel *Capturing Audio & Video in HTML5* die beste Erklärung von `getUserMedia()`: http://www.html5rocks.com/en/tutorials/getusermedia/intro/. Die

vollständige *Zielsetzung des WebRTC-Projekts* ist unter http://www.webrtc.org/ aufgelistet.

(Wieder einmal) MozDev liefert eine prägnante *Einführung in die Web-Storage-API:* https://developer.mozilla.org/en-US/docs/DOM/Storage/.

Der am besten verständliche *Leitfaden zur Drag-and-Drop-API*, den ich finden konnte, stammt von den HTML5 Doctors unter http://html5doctor.com/native-drag-and-drop/, während das fünfteilige *Working with Files in JavaScript* von Nicholas Zakas eine exzellente Ressource zur File-API darstellt: http://www.nczonline.net/blog/2012/05/08/working-with-files-in-javascript-part-1/.

Die *APIs, aus denen das Firefox-OS-Projekt besteht*, sind unter https://wiki.mozilla.org/WebAPI/ aufgeführt, und die Folien der Präsentation *WebAPIs and Apps* von Robert Nyman liefern einen tollen Überblick darüber: http://www.slideshare.net/robnyman/webapis-apps-mozilla-london/. *Are We Mobile Yet?* zeigt aktuell und auf einen Blick, wie es um die Implementierung von APIs bestellt ist: http://arewemobileyet.com/.

Kapitel 7

Eine hervorragende *SVG-Einführung* finden Sie auf der Website SVG Basics: http://www.svgbasics.com/, und der *SVG-Primer* des W3C geht noch mehr in die Tiefe: http://www.w3.org/Graphics/SVG/IG/resources/svgprimer.html. MDN bietet eine ziemlich vollständige *Elemente- und Attributeliste:* https://developer.mozilla.org/en-US/docs/SVG/.

Die *Technik zum Stapeln von SVG-Symbolen* stammt von Erik Dahlström und wird im Simurai-Blog beschrieben: http://simurai.com/post/20251013889/svg-stacks/. Robert

O"Callahan von Mozilla *warnt vor möglichen Veränderungen der Stapeltechnik*: http://robert.ocallahan.org/2012/10/impending-doom-for-svg-stacks-sort-of.html. Ich führe die `svgView()`-Method in meinem Blog *Broken Links* weiter aus: http://www.broken-links.com/2012/08/14/better-svg-sprites-with-fragment-identifiers/.

Die IE-Testdrive-Site bietet ein praktisches *Tool zum Experimentieren mit SVG-Filtereffekten*: http://ie.microsoft.com/testdrive/graphics/hands-on-css3/hands-on_svg-filter-effects.htm.

Dirk Schulze hat eine gute, prägnante *Einführung zu den geplanten Funktionen von SVG2* geschrieben: http://dschulze.com/blog/articles/8/new-features-in-svg2/.

Eine gute Auswahl an *Canvas-Anleitungen* (vom Einsteiger- bis zum Fortgeschrittenenniveau) finden Sie unter http://www.html5canvastutorials.com/, einen hervorragenden *Spickzettel mit den Eigenschaften und Methoden* auf Jacob Seidelins Blog unter http://blog.nihilogic.dk/2009/02/html5-canvas-cheat-sheet.html (letzte Aktualisierung 2009, aber trotzdem noch relevant).

HTML5 Rocks liefert unter http://www.html5rocks.com/en/tutorials/canvas/imagefilters/ eine *Anleitung zu fortgeschrittenen Bildbearbeitungseffekten*.

Tim Taubert erklärt in seinem Blog-Beitrag *Building a Live Green Screen with getUserMedia() and MediaStreams* die Grundlagen der Live-Bearbeitung von Videobildern: http://timtaubert.de/blog/2012/10/building-a-live-green-screen-with-getusermedia-and-mediastreams/.

Der Blog Learning WebGL stellt *Anfängerlektionen zur Arbeit in einem dreidimensionalen Kontext* in `canvas` bereit unter http://learning-webgl.com/lessons/, und WebGL.com bietet regelmäßig *Zusammenstellungen von Demos*, *Anleitungen und Entwicklertreffen*: http://www.webgl.com/.

Kapitel 8

Das stets hilfreiche MDN liefert einen präzisen und vollständigen *Leitfaden zu den neuen Eingabetypen*, zu finden ist er unter https://developer.mozilla.org/en-US/docs/HTML/Element/Input/. PPK bietet detaillierte *Tabellen zur Desktop-Unterstützung* unter http://www.quirksmode.org/html5/inputs.html. Das Ganze gibt es auch noch für Mobilgeräte unter http://www.quirksmode.org/html5/inputs_mobile.html.

Ryan Seddon hat ein *Polyfill für HTML5-Formularfunktionen in Browsern ohne native Unterstützung* geschrieben: Zu finden ist es unter https://github.com/ryanseddon/H5F/.

Bruce Lawsons Diskussion zu Autofokus und Barrierefreiheit, *The Accessibility of HTML 5 Autofocus* finden Sie auf seinem Blog unter http://www.brucelawson.co.uk/2009/the-accessibility-of-html-5-autofocus/.

HTML5 Rocks hat einen guten *Überblick über die Constraint-Validation-API*, zu finden unter http://www.html5rocks.com/en/tutorials/forms/constraintvalidation/

Rubular ist eine hilfreiche *Anwendung zum Testen regulärer Ausdrücke*. Sie zielt zwar auf Ruby ab, funktioniert aber ebenso gut für JavaScript und HTML5-Formulare: http://rubular.com/.

Und Ihr bescheidener Autor hat bei HTML5 Doctor eine *Einführung in CSS3-Pseudoklassen* geschrieben: http://html5doctor.com/css3-pseudo-classes-and-html5-forms/.

Kapitel 9

Den gesamten Text von Steve Jobs *Thoughts on Flash* finden Sie unter http://www.apple.com/hotnews/thoughts-on-flash/.

Hinweise zur Barrierefreiheit von Audio und Video im Zusammenhang mit automatischer Wiedergabe erhalten Sie auf der Audio-Control-Seite der WCAG unter http://www.w3.org/TR/UNDERSTANDING-WCAG20/visual-audio-contrast-dis-audio.html.

MDN liefert unter https://developer.mozilla.org/en-US/docs/Media_formats_supported_by_the_audio_and_video_eleme *Tabellen zum aktuellen Implementations-Status von Medienformaten in verschiedenen Browsern*, und Kroc Camens Technik und Anmerkungen zur Implementation für verschiedene Browser *Video for Everybody!* gibt es auf *http://camendesign.co.uk/code/video_for_everybody/*.

Die beste *Einführung zu WebVTT und das track-Element* finden Sie auf Dev.Opera unter http://dev.opera.com/articles/view/an-introduction-to-webvtt-and-track/, und auf Anne van Kesterens Website wartet ein nützliches *WebVTT-Validierungswerkzeug:* http://quuz.org/webvtt/.

Ich habe in diesem Kapitel drei Codierungswerkzeuge angesprochen: Das *Kommandozeilen-Tool FFmpeg* gibt es auf http://ffmpeg.org/, den *Miro Video Converter* unter http://www.mirovideoconverter.com/ und *HandBrake* unter http://handbrake.fr/.

Die *Media-Fragments-Spezifikation des W3C* finden Sie unter http://www.w3.org/TR/media-frags/.

MDN bietet die besten *Dokumentationen für die Media-API und Ereignisse*, die ich gefunden habe: https://developer.mozilla.org/en-US/docs/DOM/HTMLMediaElement/ beziehungsweise https://developer.mozilla.org/en-US/docs/DOM/Media_events/.

CreativeJS liefert unter http://creativejs.com/resources/web-audio-api-getting-started/ einen tollen *Einführungsartikel zur Web-Audio-API.*

Über WebRTC werden Sie in Zukunft sicherlich noch viel mehr hören, Sam Dutton hat bereits eine gute Einführung dazu auf *HTML5 Rocks* geschrieben:

http://www.html5rocks.com/en/tutorials/webrtc/basics/. Microsofts *Einführung zum CU-RTC-Web-Vorschlag* finden Sie im Interoperability-Blog unter http://blogs.msdn.com/b/interoperability/archive/2012/07/28/custon ubiquitous-real-time-communication-over-the-web-cu-rtc-web.aspx.

Kapitel 10

Der *Chrome Web Store* hat eine detaillierte Dokumentation zur Vorbereitung der einzureichenden Apps und natürlich auch zur Manifest-Datei: https://developers.google.com/chrome/web-store/docs/get_started_simple/.

Näheres zur Manifest-Datei für Firefox Marketplace finden Sie unter https://developer.mozilla.org/docs/Apps/Manifest/. Auch ein nützliches *Tool zur Überprüfung von Manifesten* wird angeboten: https://marketplace.firefox.com/developers/validator/. Robert Nymans Vortrag *Web APIs and Apps* liefert eine hervorragende Einführung zur Entwicklung von Apps für den Firefox Marketplace: http://www.slideshare.net/robnyman/web-apis-apps-mozilla-london/.

Die letzte Version der *Widgets-Spezifikation des W3C* liegt unter http://w3.org/TR/widgets/, und eine *hilfreiche Einführung* finden Sie auf Peter-Paul Kochs Site – auch wenn sie bereits vor einigen Jahren geschrieben wurde und sich die Spezifikation seither in einigen kleinen Details verändert hat: http://quirksmode.org/blog/archives/2009/04/introduction_to.html.

Das *PhoneGap-Projekt* finden Sie unter http://phonegap.com/, und die *API* ist unter http://docs.phonegap.com/ dokumentiert. Den PhoneGap-Artikel *Beliefs, Goals, and Philosophy* können Sie unter http://phonegap.com/2012/05/09/phonegap-beliefs-goals-and-philosophy/ nachlesen.

Mehr zum Titanium-Projekt finden Sie unter http://appcelerator.com/platform/.

Jede Smart-TV-Plattform hat ihr eigenes Entwicklerforum, ein gutes *Beispiel für den Einstieg* bietet die Samsung-Site unter http://www.samsungdforum.com/.

Der Opera TV Store fasst immer mehr Fuß; mehr darüber erfahren Sie unter http://business.opera.com/partners/tv/store/. Die Dev.Opera-Site bietet einige tolle *Referenzartikel zur Gestaltung und Entwick-lung für Fernsehgeräte*; diese können sie unter http://dev.opera.com/tv/ nachlesen.

Das *Webinos-Projekt*, das eine einheitliche Geräte-API anstrebt, befindet sich auf http://www.webinos.org/.

Das Mozilla Developer Center verfügt unter https://developer.mozilla.org/en-US/docs/HTML/Using_the_application_cache/ über eine hervorragende *AppCache-Dokumentation*, und Mark Christian und Peter Lubbers haben eine nützliche *Seite mit Fakten über App-Cache* ins Leben gerufen: http://appcachefacts.info/. Alle Pferdefüße von AppCache und Tipps und Techniken zur Verwendung sind in Jake Archibalds Artikel *Application Cache is a Douchebag* bei *A List Apart*: http://www.alistapart.com/articles/applicationcache-is-a-douchebag/ aufgeführt.

Kapitel 11

Web Components sind in diesem Moment noch ziemlich neu, es gibt dazu also noch nicht so viele Ressourcen. Als Erstes sollten Sie sich dazu die *entwicklerfreundliche Einführung der Verfasser der Spezifikation* ansehen und danach dann vielleicht die *Präsentation von Eric Bidelman*. Beide Quellen vermitteln Ihnen gut die Grundkonzepte. Zu finden sind sie unter http://dvcs.w3.org/hg/webcomponents/raw-

file/tip/explainer/index.html und http://html5-demos.appspot.com/static/webcomponents/index.html (eventuell benötigen Sie hier Google Chrome für die korrekte Anzeige).

Das Shadow DOM ist bisher der am besten implementierte Bestandteil der Web Components und verfügt als solcher über eine umfangreichere Online-Dokumentation. Sowohl *Sitepoint* als auch *HTML5 Rocks* liefern gut verständliche Erklärungen zum Thema, diese finden Sie unter http://www.sitepoint.com/the-basics-of-the-shadow-dom/ und http://www..html5rocks.com/en/tutorials/webcomponents/shadowdo

Falls Ihr Browser keine Custom Elements unterstützt, sehen Sie sich *X-Tags* an. Diese von Mozilla entwickelte, experimentelle Bibliothek bildet das Verhalten von Custom Elements nach und bietet dabei ein großes Register vorgefertigter Komponenten: http://x-tags.org/.

Im Internet Explorer 10 wurden CSS Regions als Erstes implementiert, die entsprechende *Dokumentation* eignet sich daher zum Einstieg in die Grundlagen. Siehe http://msdn.microsoft.com/en-us/library/ie/hh673537%28v=vs.85%29.aspx/.

CSS Exclusions finden sich ebenfalls in IE10, sehen Sie also auch hier zuerst in der *Microsoft-Dokumentation* nach. Anschließend schauen Sie sich ein paar *Demos von Adobe* an. Siehe http://msdn.microsoft.com/en-us/library/ie/hh673558%28v=vs.85%29.aspx/ und http://adobe.github.com/web-platform/samples/css-exclusions/.

Im *MDN* findet sich die beste Dokumentation von Feature Queries, obwohl die API derzeit noch undokumentiert ist. Siehe https://developer.mozilla.org/en-US/docs/CSS/@supports/.

Während ich dies schreibe, ist zu Cascading Variables nur etwas im *Spezifikationsentwurf* unter http://dev.w3.org/csswg/css-variables/ zu finden.

Falls der *Vorschlag für Box Alignment* immer noch aktuell ist, während Sie

dies lesen, können Sie seinen Fortschritt hier verfolgen: http://dev.w3.org/csswg/css3-align/. Den Vorschlag zu *Line Grid* finden Sie unter http://dev.w3.org/csswg/css-line-grid/.

Håkon Wium Lie und Chris Mills haben eine sehr schöne Einführung zum Seitenkonzept mit CSS geschrieben. Lesen Sie dazu ihren Artikel *Opera Reader: Paging the Web*: http://people.opera.com/howcome/2011/reader/index.html. Mehr zu Seitenvorlagen finden Sie auf dem *Adobe Web Platform-Blog* unter http://blogs.adobe.com/ webplatform/2012/05/31/pagination-templates-in-css/.

Index

3-D-Canvas 169
-controls 251
-ms--Präfix 266
-webkit-device-pixel-ratio 60
-webkit-Präfix 263, 264, 265
:disabled-Pseudoklasse 195
:indeterminate 195
:invalid-Selektor 194
:out-of-range 195
:read-only 195
:read-write 195
:target 159
:valid-Selektor 194
@import-Regel 52
@supports-Regel 251
@viewport-Regel 63, 262

A

Abmessungen des Viewports 53
Aboutness 38
Accessible Rich Internet
    Applications 35
Achsen 134
Adaptives Webdesign 67

addEventListener()-Methode 112, 112, 263
add()-Methode 119
Adobe Edge Inspect 27
Akkuladezustand 139
align-items-Eigenschaft 93
Almost Standards Mode 19
Android 4
Anonyme Funktionen 112
Apache Cordova 222
APIs
    AppCache 229
    Battery Status 264
    canvas 165
    Constraint Validation 191
    Datenattribute 46
    Drag and Drop 145, 264
    File 148, 265
    FileReader 265
    Full-Screen 264
    Geolocation 264
    matchMedia 262
    Media 207
    Mikrodaten 42
    Network
        Information 140, 264
    Notification 223
    Orientation 134
    PhoneGap 224
    Vibration 137, 264

Web Audio 213, 266
Web Storage 142
AppCache 227, 266
API 229
Dateiinhalt 228
Ereignis 230
Appcelerator Titanium 225
Apple App-Store 222
application-Role 36
Apps 217
Hosted 219
hybride 15, 217
native 217
Packaged 219
Web- 217
App-Stores 217
Archibald, Jake 227
article-Element 30
aside-Element 30
Aspect Ratio 55
async 110
Asynchrone Ausführung von Skripten 110
Atkins, Tab 105
Attribute, boolesche 21
Attributselektor 37
Audens, W. H. 38
audio 198
Fallbacks 202
Quelldateien 201

type-Attribut 201
audio-Element 266
Audioformate 203
audio()-Methode 211
Auflösungsabhängige
Darstellung 59
Ausrichtung 103
Ausrichtungseigenschaften 92
autocomplete-Attribut 177
autofocus-Attribut 176
autoplay-Attribut 199
Autovervollständigung in Formularen 177

B

banner-Role 37
Barrierefreiheit 35, 262
Batteriefüllstand 139
battery-Objekt 139
Battery-Status-API 264
Bedienelemente (Formulare) 180
Benannte Funktionen 112
Benutzerinteraktion 64
Berechnung in Formularen 187
Beschleunigungssensor 264
Beschriftungen 204
Beverloo, Peter 130
Bewegung im Raum 134
Bibliotheken
für Mobilgeräte 130

JavaScript 119
jQTouch 122
YepNope 122
Zepto.js 122
Bidelman, Eric 255
Bilder 153
responsive 78
skalieren 78
Bildmanipulation 167
Bildschirmauflösung 59
Bildschirmdiagonale 7
Bildschirmgrößen 8
Blackberry 4
Blooman, David 28
Boolesche Attribute 21
border-bottom-Eigenschaft 253
Boulton, Mark 105, 108
Box Alignment 250
box-pack-Eigenschaft 263
box-sizing-Eigenschaft 69, 262
break-after-Eigenschaft 87, 263
break-before-Eigenschaft 87, 263
break-inside 87
Breakpoints 67
für den Inhalt 73
Breite 53
Browne, Brennan 13
Browser 15
Darstellungsmodi 19

die wichtigsten 260
Browserunterstützung 24
Bruchteileinheit 99

C

Cache-Status 229
calc()-Funktion 70
Camen, Kroc 204, 214
canplay 212
canplaythrough 212
Canvas versus SVG 169
canvas-API 165
canvas-Element 165, 265
und Videos 168
zur Bildmanipulation 167
captions 204
Cascading Variables 252, 266
chargingTime-Attribut 139
Christian, Mark 232
Chrome Web Store 220
Cisco 12
classList-Objekt 118, 119, 264
clear 247
clear()-Methode 144
clearWatch()-Methode 133
Clientseitiges Template-System 125
Codierung von Videos 206
Coenraets, Christoph 130
color-Eigenschaft 253

color-Element 184
color-Geräteeigenschaft 65
column-count-Eigenschaft 84
column-fill-Eigenschaft 85
column-gap-Eigenschaft 86
column-rule-Eigenschaft 86
column-span-Eigenschaft 263
column-width 84
complementary-Role 37
Constraint-Validation-API 191
contactFields-Argument 224
contacts-Objekt 224
contactSuccess-Argument 224
contains()-Methode 119
Content Breakpoints 73
contentinfo-Role 37
coords-Unterobjekt 132
CSS
    DOM-Methode 266
    Exclusions 246
    Flexbox 88
    Formulare 194
    Klassen manipulieren 119
    mediengerechtes 51
    Mehrspalten-Layout 84
    neue Konzepte 83
    Rasterlayout-Syntax 98
    SVG 157, 164
CSS3 und Media Queries 22

CSS-Frameworks 24
CSS-Selektoren, JavaScript 117
currentSrc-Eigenschaft 208
currentTarget 113
currentTime-Eigenschaft 208
Custom Elements 233, 239
Custom Property 253

D

Dahlström, Erik 171
Darstellungsmodi 19
DataChannel 213
datalist-Element 179
data()-Methode 47
dataset 262
Dateien 148
Datenattribute 46, 48
    API 46
    und jQuery 47
Datenkapselung im HTML-DOM 234
datetime-Auswahl 183
Datum 181
Datumswahl-Widget 182
Debenham, Anna 7, 13, 28
Debuggen 128
Decorator 233, 237
defer 110
descriptions 204
Desktop-Browser 3

Desktop-Geräte 2
Deveria, Alexis 28
Device Adaptation 64
device-height-Feature 55
deviceorientation-Ereignis 135
Device Pixel Ratio 58
devicePixelRatio-Attribut 262
deviceready-Ereignis 225
Device Storage 150
device-width-Feature 55
dischargingTime-Atrribut 139
display-Eigenschaft 98
Doctype 20
document.register() 266
DOM, Shadow 241
DOMContentLoaded-Ereignis 114, 263
dots per pixel 58
downloading-Ereignis 230
DPI 58
DPPX 58, 61
DPR
    ansprechen 60
    Vergleich 59
Drag-and-Drop-API 145, 148, 264
Drehung um die Achsen 135

E

E-Book-Reader 7
Edge Inspect 27

Eingabe-Events 115
Eingabemechanismus 64
Eingabetypen 174
Eingebettetes SVG 160
Elektronisches Papier 65
Elementreihenfolge ändern 90
embed-Element 160
Emulatoren 26
Encoding 206
endcallbutton 225
Entwicklerkonsole 16
Entwicklertools im Browser 16
Event-Bubbling 111
Event-Handler hinzufügen 111
Event-Listener entfernen 113
event-Objekt 112
Exclusions 246, 266
exitFullScreen()-Methode 136
explicit entries 228
Explicit Sectioning (Explizite Gliederung) 32

F

Fallbacks 202, 228
    Flash 204
Farben 184
Farbwert von Pixeln ändern 168
Feature Queries 251
    API 252
Features, experimentelle aktivieren 261

Fernbedienung 6
Fernseher 6
FFmpeg 206
File-API 148, 265
FileReader-API 265
fill-Attribut 161
fillRect() 166
Filter 161
Filtereffekte 164
filter-Element 162
filter()-Funktion 265
find()-Methode 118
findAll()-Methode 118
Firefox 4
    experimentelle Features aktivieren 261
    Präfixe 263
Firefox Marketplace 221
Firefox OS 4, 150
Flash 75
    als Multimedia-Fallback 204
Flash-Format 197
Flexbox 88
    align-items 93
    deklarieren 88
    Elementreihenfolge ändern 90
    flex-Eigenschaft 94
    Haupt- und Querachse 91
    justify-content 92
flex-direction-Eigenschaft 90

flex-Eigesnchaft 94
flex-flow-Kürzel 263
flex-order-Eigenschaft 90
flex-wrap-Eigenschaft 95, 263
Float-Elemente 246
flow-Wert 95, 247
Fokus bei Seitenaufbau 176
font-size 250
footer-Element 31
foreignObject-Element 163
form-Attribut 178
Formen zeichnen 167
form-Role 37
Formulare 173
    Bedienelemente 180
    Berechnung 187
    CSS 194
    Datum 181
    Farben 184
    Fortschrittsbalken 185
    Informationen 184
    Pseudoklassen 194
    Vorschlagslisten 179
    Widgets 180
    Zahleneingabe 180
    Zeit 181
Formulareinträge, mehrere pro Feld 178
Formularüberprüfung 188
Form-Validation-API 265

Fortschrittsbalken 209
in Formularen 185
Fragmente, SVG-Sprites 159
Fragment Identifiers 265
Framework 24
JavaScript 120
Full-Screen-API 264
fullScreenEnabled-Attribut 136
Funktionen, anonyme 112
Funktionen, benannte 112
Funktionsschlüsselwörter 164

G

Gaußscher Weichzeichnereffekt 161
Geolocation-API 264
geolocation-Objekt 132
Geolokalisierung 132
Geräte, mobile 3
Geräteabmessung 53
Geräteanpassung 61, 64
Geräte-APIs 131
Geräteemulatoren 26
Gerätelandschaft 2
Geräteparks 26
Geräteposition 134
Geschwindigkeit 10
getCurrentPosition()-Methode 132
getElementById() 117
getElementsByClassName() 264

getElementsByTagName() 117
getSVGDocument() 160
getUserMedia() 141, 213, 264, 266
GIF 154
Gillenwater, Zoe Mickley 57, 80
Gliederung, implizite und explizite 32
Google Play 222
GPS-Gerät 133
Grafiken 153
Graustufen-Displays 65
grid areas 97
grid-column-align-Eigenschaft 103
grid-column-Eigenschaft 101
grid-columns-Eigenschaft 99
Grid-Elemente (grid items) 97
Grid Layout 97, 248
    Syntax 104, 263
    Terminologie 105
grid lines 97
grid-row-align-Eigenschaft 103
grid-row-Eigenschaft 101
grid-rows-Eigenschaft 99
grid-template-Eigenschaft 106
grid templates 105
grid tracks 97
Grigsby, Jason 13

H

H.264 203

HandBrake 206
Harmony 233
Hauptachse 91
Hay, Stephen 107
hCard-Mikroformat 39
header-Element 31
height-Eigenschaft 53
Heilmann, Christian 130
Hickson, Ian 18, 261
high-Attribut 186
Höhe 53
Hosted Apps 218
hover-Zustand 64
HTML, Überschriftelemente 31
HTML5 18
    Definition 17
    Formulare 174
    Gerüst 19
    neue Elemente 30
    responsive Bilöder 78
    Standardverfahren 20
    strukturierende Elemente 30
HTML5 Boilerplate 20
HTML5 Doctor 33, 49, 50, 152
HTML5 Please 24, 25, 28, 259
HTML5 Rocks 151, 171, 196, 215, 255
HTMLAudioElement-Schnittstelle 211
HTML-DOM, Datenkapselung 234
HTMLMediaElement-Schnittstelle 207

HTMLVideoElement 211
Hybrid-Apps 150, 217, 222

I

Icons 158
icons-Objekt 221
iframe 54
Implementierungen, experimentelle 23
Implicit Sectioning (implizite Gliederung) 32
input 265
Interaktion mit Klassen 118
Internet Explorer 4
invalid-Ereignis 192
iPad 5
iPhone 4
Irish, Paul 81
Item-Element 41
itemscope-Attribut 41

J

JavaScript 109
    asynchron ausführen 110
    Bibliotheken 119
    CSS-Selektoren 117
    Framework 120
    Neuerungen 109
JAWS 35
Jobs, Steve 197, 214

Jordesign 81
JPG 154
jQTouch 122
jQuery 120, 262
    und Datenattribute 47
jQuery-data()-Methode 48
jQuery Mobile 122
JS Bin 128
JS Fiddle 128
JSON-Format 220
JSON-Manifest 220
justify-content-Eigenschaft 92, 263

K

Kameras 141
kind-Attribut 204
Kindle 7
Klassen
    Interaktion 118
    manipulieren 119
Klassennamen suchen 118
Klischees 10
Koblentz, Thierry 75, 81
Kobo 7
Koch, Peter Paul 80, 232
Konsole 16
Kontext 9
    Klischees 10
Kreise zeichnen 166

L

LabUp! 28
Ladestatus 139, 210
Landmark Roles 36
Längenangaben 53
    dynamisch berechnen 70
    mischen 68
    relative 262
    relativ zum Viewport 71
Laptop 2
Lauke, Patrick 81
Lautstärke 210
Lawson, Bruce 17, 27, 176
Layout 22
    mehrspaltiges 263
Layout-Viewport 61
level-Attribut 139
Lie, Håkon Wium 256
Line Grid 250
line-height 250
loadeddata 212
loadedmetadata 212
low-Attribut 186
Lubbers, Peter 232

M

main-Role 37
Manian, Divya 38

Manifest-Dateien 220
Markup, semantisches 37
master entries 228
matchMedia() 66
matchMedia-API 262
Maus 64
max-Präfix 54, 56
max-width 54
Media-API 207
media-Attribut 202
Media-Eigenschaften 64
Media Features 51
Media Fragments 206, 266
Media Queries 22, 51
    Bildschirmauflösung 59
    Internet Explorer 262
    kombinieren 56
    mit JavaScript 66
    negieren 56
MediaQueryList-Objekt 66
MediaStream 213
Media-Types-Syntax 52
Medienabfrage 202
Mediendateien, Ladestatus 210
    Mobilgeräte 199
Medienelemente 198
Medienereignisse 212
Mediengerechtes CSS 51
Medieninteraktion, erweiterte 212

Mediensteuerung 210
Medientyp 52
Meenan, Patrick 28
Mehrspaltiges Layout 84, 263
meta-Element 20
meter-Element 186
Mickley Gillenwater, Zoe 57
Mikrodaten 39, 41
    API 42
Mikrofone 141
Mikroformate 39
Milestones 129
Mills, Chris 256
min-Präfix 54, 56
min-width 54
Miro Video Converter 206
Mobilbrowser 61
Mobile First 73
Mobilgeräte 3
Modernizr 123
monochrome-Geräteeigenschaft 65
Morville, Peter 50
MozDev 107, 151, 152
mozGetUserMedia 264
MP4-Format 203
MSPointerDown 264
Multi-column Layout 84
Multimedia 198, 266
multiple-Attribut 178

Multiscreen-Welt 8
Mustache 125
muted-Attribut 199

N

Namenskonflikte 234, 238
Native Apps 217
nav-Element 30
navigation-Role 37
navigator.battery-Objekt 139
navigator-Objekt 138, 141, 224
NetFront 7
Network-Information-API 140, 264
networkState-Eigenschaft 210
NEWT 17
Nintendo 1, 7
none-Wert 247
Nook 7
Notification-API 223
not-Operator 252
Nyman, Robert 152, 221, 231

O

object-Element 160
object-fit-Eigenschaft 76, 200, 262
object-position-Eigenschaft 77, 200, 262
O"Callahan, Robert 171
Ogg Theora 201

Ogg Vorbis 201
Open Device Labs 26
Open-Web-App-Manifest 221
Opera 4
    Chromium, 262
    Mini 4
    Presto 3, 260
Opera TV Store 218, 232
optimum-Attribut 186
orientation 56
Orientation-API 134
Orientierung im Raum 134
Ortswechsel 133
Outline-Algorithmus 35
output-Element 187
overflow-style-Eigenschaft 251

P

Paciello Group Blog 50
Packaged Apps 218
    Standard 222
padding-box-Wert 262
Paged Media 250
paged-x 251
paged-y 251
pause-Ereignis 225
pause()-Funktion 207
PeerConnection 213
Phablet 74

PhoneGap 150, 222
    API 224
    Ereignisse 225
Physische Pixelanzahl 58
picture-Element 79
Pixel 57
Pixelanzahl, physische 58
Pixeldichte 58
Pixelgrafiken 154
placeholder-Attribut 176
Platzhaltertext in Formularen 177
play()-Funktion 207
Plugins 75, 197
    Kritikpunkte 207
PNG 154
Pointer Events 115, 116, 264
pointer-Feature 64
Polyfills 127
Position des Anwenders 133
Positionsbalken 209
poster-Attribut 200, 211
PPI 58
Präfixe
    Firefox 263
    herstellerspezifische 23
Präprozessoren 24
preload 199
Presto 3, 260
progress-Element 185

property-Attribut 41
Pseudoklassen für Formulare 194
px-Einheit 54

Q

Quelldateien für
    Mediendateien 201
Querachse 91
querySelector() 117, 167
querySelectorAll() 117, 264
querySelector()-Methode 264
Quirks-Modus 19

R

Rasterbereiche 97, 105
Rasterelemente 97
    erzeugen 101
Rasterlayout 97
    Ausrichtung 103
    Bruchteileinheit 99
    Hauptbegriffe 98
    Stapelfolge 103
    Terminologie 105
Rasterlinien 97
    wiederholen 100
Raster-Tracks 97
RDFa 39, 40
    Core 40

Lite 40, 41
readyState-Eigenschaft 211, 212
Rechteck zeichnen 166
Rechtschreibprüfung in Formularen 177
Region Chain 245
Regions 244
Rel-Tag 39
removeEventListener()-Methode 113
removeItem()-Methode 144
requestFullScreen()-Attribut 136
requiredFeatures-Attribut 163
resolution 60, 262
Resource Description Format in Attributes 40
Responsives Webdesign 67
Objektersetzung 75
resume-Ereignis 225
Retina-Display 62
Rhythmus, vertikaler 250
Rich Snippets 45
Robinson, Mike 49
role-Attribut 36
root em (rem) 71
Rubular 196

S

Safari für iPhone 62
Scalable Vector Graphics 155
Schema.org 44
Schulze, Dirk 171

scoped-Attribut 238
Scoped Styles 238, 266
script-Feature 65
SDKs 26
search-Role 37
section-Element 30
Sectioning Content 30, 33
Sectioning Roots 33
Seidelin, Jacob 171
Seitenverhältnis 55
Semantik 29
Semantisches Markup, Bedeutung 37
Sequenzielle Nutzung 8
setInterval() 211
Shadow DOM 234, 241, 266
Shadow-Root 242
Shaped Exclusions 249
Shims 127
Simulatoren 26
Skalare Messung 186
Smart-TV 226
Snapshots 129
Software-Development-Kits 26
Souder, Stve 52
source-Element 202
Spalten
    Abstand 85, 86
    Breite 85
    Linien 86

-satz 244
-spanne 86
spellcheck-Attribut 177
Spielekonsolen 7
Sprites 158
Standardmodus im Browser 19
Stapelfolge 103
Stark, Jonathan 5, 13
startcallbutton 225
status-Eigenschaft 229, 230
Stevens, Luke 34, 50
Storey, David 12
Streaming 142
Structural Elements 30
Struktur 29
Strukturierende Elemente 30
subtitles 204
supports() 266
SVG 155
Aufbau 157
Browserunterstützung 265
einbetten 160
Element 156
Fragment Identifiers 265
Funktionsschlüsselwörter 164
Nachteil 165
Skalierbarkeit 169
und CSS 164
versus Canvas 169

SVG2 164
SVG-Dateien, verknüpfte 157
SVG-Filter 161, 265
SVG-Sprites 158
    Fragmente 159
svgView-Funktion 159
swapCache()-Methode 230

T

Tabellenkalkulationsprogramm 98
Tablets 5
Taubert, Tim 169, 171
template-Element 235, 266
Templates 126, 233, 234
Template-System 125
Testen 25, 128
Textfluss 246
Theora 201
timeupdate-Ereignis 209, 212
Titanium 225
toggle()-Methode 119
touchend-Event 138
Touch Events 115, 263
track-Element 204, 266
Traversierung 121
    des DOM-Baums 118
TV-Apps 226
type-Attribut 201

U

Überschriftelemente 31
Ubuntu Phone 8
Umbrüche 86
Untertitel 204
update()-Methode 230
updateready-Ereignis 230
User Agents 15
UX Magazine 13

V

valid-Eigenschaft 192
var()-Funktion 253
Vektorgrafiken 154
Verbindungsinformation 140
Verbindungsqualität 140
Vererbung 238
Vererbungskonflikte 234, 238
version-Eigenschaft 220
Vertikaler Rhythmus 250
vh-Eigenschaft 71, 262
vibrate()-Methode 138
Vibration 137, 138
Vibration-API 137, 264
Video 198
    Beschriftungen 204
    Codierung 206
    Fallbacks 202

poster-Attribut 201
Quelldateien 201
type-Attribut 201
Untertitel 204
video-Element 266
Attribute 200
Videoformate 203
Videokonverter 206
viewBox-Attribut 158
Viewport 53
neuer 54
relative Längenangaben 71
viewport-Meta-Tag 62
Visitenkarten-Syntax 39
Vokabulare 44
Vollbildanzeige 137
volumechange-Ereignis 212
volumedownbutton 225
volume-Eigenschaft 210
volumeupbutton 225
Vorbis 201
Vorschlagslisten 179
vw-Eigenschaft 71, 262

W

W3C 18
W3C-Widgets 222
WAI-ARIA 35
Walsh, David 151

Wartungsfreundlichkeit 38
watchPosition()-Methode 133
Webanwendungen 15
Web-Apps 217
Web-Audio-API 213, 266
Web Components 48, 233
    Datenkapselung 234
    Templates 234
Webdesign, adaptives und responsives 67
Webfrontends, moderne 1
WebGL 169, 265
Webinos 226
WebKit 60
webkitAudioContext()-Konstruktor 266
WebKit-Engine 4
webkitGetUserMedia 264
webkitShadowRoot()-Konstruktor 266
Webkomponenten 48, 233
WebM 203
Webplattform 17
WebRTC 213
WebSMS 150
Web Storage 264
Web-Storage-API 142
WebTelephony 150
Web-TV 6
WebVTT 205
Weichzeichnereffekt 161
WHATWG 18

Widgets 222
    Formulare 180
    Manifest 222
width-Eigenschaft 53
Wiedergabestatus 212
Wilcox, Matt 80, 81
window.applicationCache-Objekt 229
Windows Phone 4
Windows Store 222
Wrap 95
wrap-flow 247, 248
wrap-flow-Eigenschaft 247
Wrapper 222
Wrapper, native 150
wrap-through-Eigenschaft 247
Wroblewski, Luke 73

X

x-, y- und z-Achse 134

Y

y-Achse 134
YepNope 122

Z

z-Achse 134
Zahleneingabe in Formulare 180

Zeichnen
    Kreise 166
    Rechtecke 166
    verschiedene Formen 167
Zeit 181
Zeitwahl-Widget 182
Zepto.js 122
Zoomstufe 63
Zwischendinger 7