Dane Cameron

HTML5 JavaScript und jQuery

Der Crashkurs für Softwareentwickler

dpunkt.verlag

Dane Cameron lebt und arbeitet als Programmierer in Neuseeland und bietet mit seiner Firma Cisdal Software-Lösungen und -Trainings an.

Dane Cameron

HTML5, JavaScript und jQuery

Der Crashkurs für Software-Entwickler

dpunkt.verlag

Lektorat: Boris Karnikowski
Übersetzung: Isolde Kommer, Großerlach, und Christoph Kommer, Dresden
Fachgutachter für die deutsche Ausgabe: Fabian Beimer, Rohrbach
Satz: Tilly Mersin, Großerlach
Herstellung: Frank Heidt
Umschlaggestaltung: Helmut Kraus, www.exclam.de, unter Verwendung eines Motivs von Karuanka Rayker und CoverDesignStudio.com
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 978-3-86490-268-0
PDF 978-3-86491-667-0
ePub 978-3-86491-668-7

1. Auflage 2015

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Inhalt

1 Einleitung

Ich möchte nicht spekulieren, warum Sie dieses Buch lesen – lieber will ich Ihnen erzählen, warum ich es geschrieben habe.

Ich hatte bereits 15 Jahre lang Erfahrung als Java-Softwareentwickler von großen Unternehmensanwendungen gesammelt. Diese Anwendungen waren nicht für Endnutzer bestimmt, sondern erfüllten systeminterne Aufgaben.

Danach kam ich in ein Entwicklerteam, das eine komplexe Webanwendung von Grund auf schrieb. Diese stellte einige Anforderungen, denen ich zuvor noch nie begegnet war:

Anforderungen an Webanwendungen

- Sie musste in der neuesten Version der wichtigsten Browser laufen.
- Sie musste zeitweilig auch ohne Netzwerkverbindung weiterlaufen und daher recht große Datenmengen auf dem Client speichern.
- Sie musste Dateien einlesen, die der Anwender im Offlinebetrieb ausgewählt hatte.
- Sie musste sehr dynamisch sein und komplexe Interaktionen ohne erneutes Laden der Seite erlauben.
- Sie musste mindestens ebenso flüssig wie eine Desktop-Anwendung laufen.
- Sie musste auf bestehenden Standards aufbauen und ohne Browser-Plugins laufen.

Im Lauf der Jahre hatte ich einfache HTML-Seiten und schlichtes JavaScript geschrieben, was mich oft frustrierte – insbesondere JavaScript: Es glich Java (das ich gut kannte), aber vieles passte irgendwie nicht so ganz. Je mehr ich meine Java-Denkweise auf JavaScript übertrug, desto schlimmer schien alles zu werden.

Immerhin erkannte ich, dass ich mir nie Zeit genommen hatte, um JavaScript zu lernen. Ich hatte viele Annahmen darüber getroffen, was JavaScript war und wie es funktionierte – ohne mir aber jemals Zeit zu nehmen, diese Annahmen zu überprüfen.

Bevor ich nun mit meinem neuen Projekt anfing, wollte ich erst von Grund auf den besten Ansatz zur Entwicklung einer Webanwendung mit den verfügbaren Sprachen erlernen. In den letzten Jahren hatte ich selbst genug Webanwendungen genutzt, um das Potenzial der browserbasierten Technologien zu erkennen. Ich wusste aber noch nicht, wie ich mir dieses Potenzial erschließen konnte.

HTML5

Die Plattform, die moderne Browser uns Softwareentwicklern bereitstellen, überraschte mich immer mehr, je mehr ich drüber in Erfahrung brachte. Die lose zu HTML5 zusammengefassten Standards bieten eine große Funktionsvielfalt, von der Datenspeicherung über offline gespeicherte Ressourcen bis hin zur Dateiverwaltung.

JavaScript

Auch JavaScript beeindruckte mich mit wachsendem Kenntnisstand immer tiefer. Auf den ersten Blick gab es vielleicht Ähnlichkeiten mit Java – ich stellte jedoch fest, dass JavaScript in Wirklichkeit viel mehr mit funktionalen Programmiersprachen wie LISP und Scheme gemein hatte. Gerade die Merkmale, die mir anfänglich Schwierigkeiten bereiteten, stellten sich letztlich als besondere Stärken von JavaScript heraus.

jQuery

Schließlich stieß ich auch noch auf die jQuery-Bibliothek. Ein Softwareentwickler kann mit jQuery auch nicht mehr bewerkstelligen als mit nativen JavaScript-APIs. Das Framework bietet jedoch eine so elegante Abstraktionsebene für JavaScript, dass es aus meinem Werkzeugkasten nicht mehr wegzudenken ist. Dank jQuery ließ ich auch meinen letzten Vorbehalt gegenüber browserbasierten Technologien fallen: die umständliche DOM-API.

Dies ist das Buch, das ich zu Beginn meiner Reise gerne gelesen hätte. Es setzt bei Ihnen ein wenig Erfahrung als Softwareentwickler oder Computerprogrammierer voraus und liefert Ihnen die Grundlagen, die Sie verstehen müssen – ohne den Anspruch, jede Sprache bis ins Detail zu behandeln. Haben Sie die Grundlagen einmal verstanden, können Sie Ihr Wissen in diesen Sprachen relativ einfach erweitern.

Zuvor lohnt sich jedoch noch ein Blick auf den Aufstieg von HTML und JavaScript, ganz besonders während der letzten Jahre – und warum es so kam.

1.1 Die Browserkriege

Erster Browserkrieg: Netscape und Microsoft

Der ursprüngliche Browserkrieg tobte Ende der 1990er Jahre zwischen Netscape und Microsoft. Microsoft ging als Sieger daraus hervor und im Ergebnis kam die Browsertechnologie zum Stillstand: Von 2001 bis 2006 änderte sich die Weberfahrung für die Anwender kaum.

Zweiter Browserkrieg: Firefox und Chrome

Der zweite Browserkrieg begann ungefähr 2005, zuerst mit Firefox und dann mit dem Aufkommen von Chrome. Dieser Browserkrieg wurde zur Materialschlacht, bei der alle großen Browseranbieter neue und interessante Funktionen implementierten. Viele von ihnen wurden letztendlich als Teil von HTML5 standardisiert und von anderen Browserherstellern übernommen.

Eine der bedeutendsten Auswirkungen des zweiten Browserkriegs war die drastische Leistungssteigerung von JavaScript.

JavaScript ist eine Interpreter-Sprache. Als solche hinkt ihre Performance einer kompilierten Sprache wie C oder C++ tendenziell immer hinterher. Schließlich muss der zwischengeschaltete Interpreter die auszuführenden Programmbefehle zunächst in Echtzeit in Maschinensprache übersetzen. Im Internet Explorer war die Leistung von JavaScript so erbärmlich, dass sich die Sprache wirklich nur für einfache Aufgaben wie die Überprüfung von Formularen eignete.

JavaScript-Engine »V8«

Besonders Google erkannte in der schlechten JavaScript-Leistung ein bedeutendes Hindernis für die Entwicklung von Webanwendungen. Zur Verbesserung der Leistung brachte Google 2008 die JavaScript-Engine »V8« heraus.

Dadurch wurde die Ausführungsgeschwindigkeit von JavaScript enorm gesteigert, weil der JavaScript-Code nicht bei jeder Ausführung interpretiert, sondern »Just-In-Time« (JIT) in Maschinensprache kompiliert wurde.

Nach dem Erscheinen von V8 bauten alle großen Browserhersteller die JIT-Kompilierung in ihre JavaScript-Engines ein. Es entbrannte ein offensiver Wettbewerb um die schnellste JavaScript-Engine und eine Reihe von Benchmarks zum Geschwindigkeitsvergleich kamen heraus. Microsoft stieg mit dem Internet Explorer 9 in das Rennen ein und konnte die Leistung mit dem Internet Explorer 10 weiter ausbauen.

Auf der aktuellen Entwicklungsstufe von JavaScript gibt es kaum noch Geschwindigkeitsprobleme. Große Webanwendungen laufen häufig ebenso schnell oder sogar schneller als vergleichbare Desktop-Programme. Alle gängigen Browser verfügen über stark optimierte JavaScript-Engines. Die Performance ist so gut, dass JavaScript inzwischen auch außerhalb des Browsers eingesetzt wird: Node.js nutzt JavaScript erfolgreich als serverseitige Technologie (unter Verwendung der V8-JavaScript-Engine).

Node.js

1.2 Der Aufstieg der Webanwendungen

Optimierte Browser-Technologien bildeten die Grundlage zur Entwicklung komplexer Webanwendungen, die komplett im Browser laufen.

In den Anfangstagen des World Wide Web enthielten die Seiten größtenteils statische Inhalte. Bald entwickelten sich die Websites weiter und erlaubten eine dynamische Nutzer-Interaktion. So konnten auch kommerzielle Marktplätze wie Amazon entstehen. Die Technologien zur dynamischen Manipulation von Webseiten wurden als DHTML (Dynamic HTML) bezeichnet. Sie sind die Vorstufe zu den in diesem Buch vorgestellten HTML5-Technologien.

DHTML

Webseiten konnten mit DHTML nach dem Laden in Echtzeit manipuliert werden. Größere Webanwendungen, die auch ohne häufiges Neuladen der Seite auskamen, waren aber immer noch schwer umzusetzen. Das war besonders der mangelnden Performance von JavaScript und den fehlenden Browser-APIs für relativ einfache Aufgaben wie etwa der Datenspeicherung geschuldet.

Gmail

An breiter Front änderte sich das erstmals, als Gmail 2007 einer größeren Öffentlichkeit zugänglich wurde. Gmail beschleunigte mit einem bisher unerreicht großen Onlinespeicher von 1 GB nicht nur den Trend zum Cloud-Computing, sondern es machte auch den Einsatz von AJAX bekannt.

AJAX

Mit Hilfe von AJAX kann eine Webseite mit einem Webserver interagieren, ohne dass sie erneut geladen werden muss. Eine Webanwendung konnte jetzt also aus einer einzelnen Webseite bestehen. Sie aktualisierte sich einfach selbst, während der Nutzer mit ihr interagierte und sie zusätzliche Daten vom Server empfing.

Google hat die Technologien hinter AJAX nicht selbst erfunden (das war in Wirklichkeit Microsoft); und Google hat auch nicht als erstes eine AJAX-basierte Webanwendung entwickelt. Aber erst durch Gmail rückte AJAX ins Rampenlicht und die Programmierer erkannten, was damit möglich war.

In den letzten 5 Jahren ist AJAX als Webtechnologie so dominant geworden, dass man sich kaum noch in die Zeit vor seinem Aufkommen zurückdenken kann.

Jetzt war klar geworden, was sich alles im Browser bewerkstelligen lässt. Und sowohl IT-Abteilungen als auch Endanwender drängten darauf, browserbasierte Webanwendungen für viele Aufgaben einzusetzen, die bisher klassischerweise den Desktopanwendungen vorbehalten waren. Im Browser ausgeführte Webanwendungen haben für die Anwender viele Vorteile:

Vorteile browserbasierter Webanwendungen

- Die Anwendung muss nicht auf jedem Computer installiert werden.
- Neue Software-Versionen können regelmäßig veröffentlicht werden, ohne dass der Anwender selbst Aktualisierungen einspielen muss. Manche Webanwendungen werden sogar täglich aktualisiert.
- Webanwendungen können in einer cloudbasierten Infrastruktur bereitgestellt werden. Die Verfügbarkeit steigt bei gleichzeitig sinkenden Kosten.
- Dieselbe Webanwendung funktioniert auf verschiedenen Geräten, auch auf mobilen Geräten wie Smartphones und Tablets.
- Die Anwender können die Anwendung weltweit und zu jeder Zeit aufrufen.

Im Hinblick auf die für Software-Entwickler bereitstehenden Funktionen und Bibliotheken sind die Webbrowser den Betriebssystemen inzwischen ebenbürtig. Wir können den Browser jetzt also als ausgereifte Plattform für die Softwareentwicklung verstehen – genau wie OS X, Android oder Windows.

1.3 Die vernetzte Welt

Breitband-Netzwerke

Eine weitere wichtige Veränderung der letzten fünf Jahre sind die fast allgegenwärtigen Breitband-Netzwerke. Viele Menschen haben fast immer Zugang zu schnellen Netzwerken, egal ob drahtlos oder kabelgebunden.

Cloud-Computing

Die höhere Netzwerkverfügbarkeit ermöglichte den Aufstieg des Cloud-Computings. Dies ist ein Sammelbegriff für verschiedene Technologien. Im Grund bedeutet es, dass Software-Entwickler keine Gedanken mehr an die Hardware verschwenden müssen – diese wird als Dienstleistung bereitgestellt.

Aus Anwendersicht heißt Cloud-Computing, dass die eigenen Anwendungen und Daten jederzeit und von jedem Ort aus verfügbar sind.

Cloud-Computing ist einer der dynamischsten Trends der IT-Branche und diese Entwicklung wird sich mit einer weiter wachsenden Anzahl netzwerkfähiger Geräte noch ausweiten und beschleunigen. Schätzungen zufolge wird 2020 im Durchschnitt jede Person vier netzwerkfähige Geräte besitzen.

Es ist zwar möglich, cloudbasierte Anwendungen zu schreiben, die nicht in einem Browser laufen – da der Browser jedoch die gemeinsame Konstante aller Geräte ist, ist er die perfekte Plattform für cloudbasierte Anwendungen. Somit läuft eine einzige Version der Anwendung auf allen Plattformen.

1.4 Fazit

Die Schlussfolgerung aus diesem Kapitel lautet, dass HTML5 und JavaScript optimal als treibende Kraft hinter den Anwendungen platziert sind, die von den meisten Anwendern jeden Tag interaktiv eingesetzt werden.

2 Über dieses Buch

2.1 Was Sie brauchen

Für dieses Buch sollten Sie etwas Erfahrung als Softwareentwickler und möglichst auch einige HTML-Grundkenntnisse mitbringen.

Keine Schritt-für-Schritt-Anleitungen

Sie finden hier keine Schritt-für-Schritt-Anleitungen zu den Grundlagen von HTML oder JavaScript. Auch ohne Vorkenntnisse in diesen Sprachen werden Sie beim Schreiben Ihrer eigenen Webanwendung das wesentliche Basiswissen erlangen. Natürlich geschieht das dann nicht mehr unbedingt in der Reihenfolge einer klassischen Einführung.

Nützliche Ressourcen

Wenn Sie zuvor noch nicht mit HTML oder JavaScript gearbeitet haben, könnte sich ein Blick auf die Grundsyntax dieser Sprachen (etwa Schleifen, bedingte Anweisungen usw.) lohnen, bevor Sie weitermachen. Websites wie *https://developer.mozilla.org/de/* oder *https://docs.webplatform.org/* bieten Einführungen in die Grundlagen von HTML und JavaScript.

Benötigte Werkzeuge

Die Übungen in diesem Buch können Sie auf jedem Computer durchführen, der Ihnen Zugriff auf Folgendes gewährt:

- **Einen Texteditor zum Schreiben von Code.** Notepad++ (*http://notepad-plus-plus.org*) ist eine gute Wahl für Windows, für Mac-Rechner empfiehlt sich zum Beispiel Text Wrangler (*http://www.barebones.com/products/textwrangler*). Für Linux-Systeme gibt es Emacs. Sie können natürlich auch eine Integrierte Entwicklungsumgebung (IDE) wie Eclipse nutzen.
- **Chrome- oder Firefox-Webbrowser.** Wenn Sie Firefox wählen, müssen Sie für eine vollständige Sammlung von Entwicklerwerkzeugen außerdem das Firebug-Plugin installieren. Die Beispiele und Screenshots basieren auf Chrome; alles Gezeigte funktioniert aber auch mit Firefox, Internet Explorer 10 oder Safari. Aus meiner Sicht bietet Chrome inzwischen unter allen Browsern die besten Entwicklerwerkzeuge. Daher empfehle ich jedem Neueinsteiger unbedingt diesen Browser.

- **Einen Webserver.** Diesen brauchen Sie erst im späteren Verlauf des Buchs. Dort finden Sie auch ein Kapitel, das die Installation und die Verwendung des Mongoose-Webservers erläutert. Es steht Ihnen frei, auch einen anderen Webserver zu verwenden, beschrieben wird hier aber nur Mongoose.

Beispiele zum Download

Alle Beispiele im Buch finden Sie auf dieser Website:
http://www.dpunkt.de/leseproben/5160/Beispielcode.zip

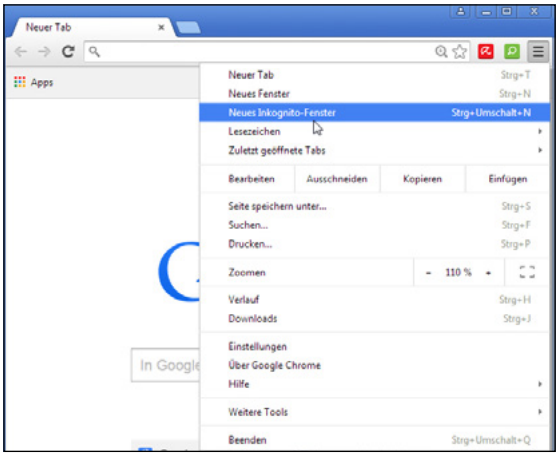
Für jedes Buchkapitel (mit entsprechenden Beispielen) finden Sie eine ZIP-Datei mit den Ressourcen für die Webanwendung, so wie sie am Ende des Kapitels abgedruckt sind.

Wie bereits erwähnt, führt Sie das Buch durch den Erstellungsprozess einer Webanwendung. In der ersten Hälfte können Sie die Webanwendung direkt von Ihrem lokalen Dateisystem in den Browser laden. Ein Webserver ist nicht unbedingt erforderlich.

Alle Webbrowser können HTML-Dateien direkt vom Dateisystem anzeigen: Ziehen Sie die HTML-Datei dazu einfach in Ihren Browser oder verwenden Sie die Menüoption *Datei > Datei öffnen*.

Inkognito-Modus

Leider nehmen die Browser Ressourcen wie JavaScript- und CSS-Dateien häufig in den Cache-Speicher auf, wenn eine Website direkt vom lokalen Dateisystem geladen wird. In Chrome können Sie das umgehen, indem Sie ein neues Fenster im Inkognito-Modus öffnen:



In diesem Modus werden die geladenen Ressourcen nicht in den Browsercache geladen.

Cache deaktivieren

In Chrome können Sie bei geöffneten Entwicklungswerkzeugen den Cache ganz einfach deaktivieren: Klicken Sie dazu einfach auf das Zahnradsymbol und wählen Sie *Disable cache (while DevTools is open)*.

lokaler Webserver

Wenn Ihnen das zu umständlich ist, können Sie Ihre Seiten auch gleich auf einen lokalen Webserver legen. In Kapitel 8 finden Sie eine detaillierte Anleitung zur Installation des Mongoose-Webservers auf Ihrem Computer. Von einem Webserver bezogene Seiten werden teilweise trotzdem noch im Browsercache abgelegt. Es gibt jedoch eine Funktion, mit der Sie die aktuelle Seite neu laden und dabei den Cache überschreiben können. Dann werden die Ressourcen garantiert neu geladen:

Unter OS X drücken Sie dazu die Tastenkombination ⌘+⇧+R, unter Windows Strg+F5.

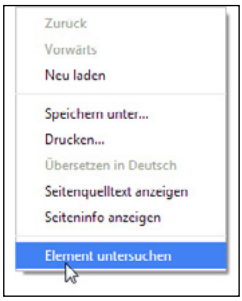
Entwicklerwerkzeuge im Browser

Viele Beispiele in diesem Buch, besonders die aus den vorderen Kapiteln, können direkt in einem JavaScript-Interpreter ausgeführt werden. Alle wichtigen Browser bieten inzwischen Entwicklerwerkzeuge, zu denen auch ein JavaScript-Interpreter gehört.

In Chrome verwenden Sie im geöffneten Browserfenster einfach die folgende Tastenkombination, um zum Interpreter zu gelangen:

- ⌘+⌥+I unter OS X
- F12 oder Strg+⇧+I unter Windows

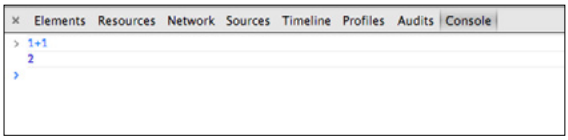
Alternativ können Sie auch irgendwo auf der Webseite einen Rechtsklick ausführen und dann im Kontextmenü *Element untersuchen* auswählen:



Konsole öffnen

Sobald die Entwicklerwerkzeuge angezeigt werden, klicken Sie auf das Register *Console* bzw. *Konsole*.

Zum Beweis, dass es sich um einen echten JavaScript-Interpreter handelt, geben Sie an der Eingabeaufforderung einfach 1+1 ein und drücken Sie die ↵-Taste:



Ein Aspekt der Browser hat in den letzten Jahren beträchtliche Fortschritte gemacht: die mitgelieferten Entwicklerwerkzeuge. Die Hersteller haben erkannt, dass sie einen direkten Nutzen daraus ziehen, wenn die Entwickler ihren Browser verwenden. Darum locken sie die Entwickler mit den angebotenen Werkzeugen inzwischen aktiv an. In diesem Buch erhalten Sie eine Einführung in viele Funktionen der Chrome-Entwicklerwerkzeuge. Es lohnt sich aber, die gebotenen Möglichkeiten selbst auszukundschaften.

2.2 Konventionen

Code wird in diesem Buch in Proportionalschrift dargestellt:

```
Dies ist ein Code
```

Code im JavaScript-Interpreter

Wird Code im JavaScript-Interpreter ausgeführt, steht vor der Eingabe ein >-Zeichen. Wenn Sie die Befehle selbst eintippen, lassen Sie dieses Zeichen weg. Außerdem trennt eine Leerzeile Eingabe und Ausgabe voneinander:

```
> 1 + 1

2
```

Leerzeilen

Werden zwei Befehle gleichzeitig gezeigt, trenne ich sie durch eine weitere Leerzeile zwischen der Ausgabe des ersten Befehls und der Eingabe des zweiten Befehls.

```
> 1 + 1

2

> 2 + 2

4
```

Ist die Ausgabe zum Verständnis des Erklärten unwichtig, habe ich sie manchmal weggelassen.

2.3 Rückmeldungen

Ich möchte sehr gerne Ihre Meinungen (positiv und negativ) zu diesem Buch hören. Bitte schreiben Sie mir eine E-Mail an dane@cisdal.com.

3 Ein kurzer Überblick über Webanwendungen

Dieses Buch soll Ihnen zeigen, wie Sie moderne Webanwendungen schreiben und dabei anstelle von Plugins nur die nativ im Webbrowser bereitgestellten Tools und Technologien verwenden. Es konzentriert sich dabei ausschließlich auf die Sprachen und Bibliotheken, die von den neuesten Versionen der folgenden Webbrowser unterstützt werden:

Unterstützte Browserversionen

- Chrome
- Firefox
- Internet Explorer
- Safari
- Opera

Viele Beispiele in diesem Buch funktionieren in älteren Browsern nicht. Oft gibt es dann auch keine Workarounds. Wenn Sie eine Webanwendung schreiben, lautet die erste Frage ganz klar: »Welche Browser und welche Versionen davon möchte oder muss ich unterstützen?«

Kompromisse

Hier müssen Sie auf jeden Fall Kompromisse eingehen:

Je mehr Browser und Browserversionen Sie unterstützen, desto mehr Anwender können Ihre Webanwendung nutzen. Vergessen Sie nicht: Manche Nutzer, vor allem in Unternehmen, können sich ihren Browser oder die Browser-Version nicht selbst aussuchen.

Je mehr Browser und Browserversionen Sie unterstützen, desto mehr Einschränkungen gibt es hinsichtlich der Verfügbarkeit und Kompatibilität von APIs. Weiter hinten in diesem Buch zeige ich Ihnen die sogenannten »Polyfills«, mit denen Sie die von einem bestimmten Browser nicht unterstützten Funktionen »upgraden« können. Dies funktioniert jedoch nicht immer.

Automatische Aktualisierungen

Alle gängigen Browser bieten mittlerweile automatische Aktualisierungen. Zwar können diese deaktiviert werden, aber dennoch ist es keine bloße Vermutung mehr, dass die meisten Benutzer mit der neuesten Version ihres Lieblingsbrowsers unterwegs sind – zumindest abseits ihres Arbeitsplatzes.

Ausnahme: Internet Explorer

Die wichtigste Ausnahme ist der Internet Explorer. Die Versionen 10 und 11 gibt es für ältere Windows-Versionen nicht. Daher haben viele Nutzer noch ältere Versionen des Internet Explorers. Die meisten Beispiele in diesem Buch funktionieren im Internet Explorer 9, manche auch im Internet Explorer 8. Die Unterstützung des Internet Explorers 6 und 7 ist jedoch ein eher schwieriges Unterfangen.

caniuse.com

Die Website *http://caniuse.com* ist eine unschätzbar wertvolle Ressource zum Funktionsumfang der verschiedenen Browser und Browser-Versionen.

3.1 Was ist eine Webanwendung?

Was ist eine Webanwendung und wie unterscheidet sie sich von einer Website? Das ist eine durchaus berechtigte Frage. Selbst die Einführung zur HTML5-Spezifikation spricht vom »nicht klar umrissenen Bereich der Webanwendungen«.

Eigenschaften der Beispiel-Webanwendung

Es gibt keine endgültige Antwort auf diese Frage; die in diesem Buch entwickelte Webanwendung hat jedoch die folgenden Eigenschaften:

- Für die Benutzeroberfläche wird ein Webbrowser verwendet.
- Der Nutzer kann Aktionen ausführen und Daten manipulieren, ohne die Seite neu laden zu müssen.
- Die Anwendung ist interaktiv und reagiert sofort auf Benutzereingaben.
- Sie speichert für den Benutzer Daten, entweder auf dem Client oder dem Server.
- Wenn die Anwendung auf einen Webserver zugreifen muss, werden asynchrone (A)JAX-Aufrufe verwendet.
- Asynchrone APIs werden gegenüber synchronen APIs bevorzugt. Manche Webanwendungen sind auch dann verfügbar und einsatzfähig, wenn der Benutzer nicht mit dem Internet verbunden ist.

Hinweis

In diesem Buch wird der Unterschied zwischen einer asynchronen und einer synchronen API immer wieder wichtig. Sie werden hierzu zahlreiche Beispiele erhalten. Der wichtigste Unterschied ist:

Eine synchrone API wartet auf eine Rückmeldung. Bis diese eingetroffen ist, werden alle anderen Prozesse der Anwendung blockiert.

Anders die asynchrone API: Hier können weitere Prozesse ausgeführt werden, bis eine Benachrichtigung über die Rückmeldung vorliegt.

Die HTML5-Spezifikation schlägt für Webanwendungen außerdem auch die folgenden Merkmale vor:

Weitere Merkmale von Webanwendungen

- Sie werden gelegentlich genutzt oder aber regelmäßig, jedoch von verschiedenen Orten aus.
- Sie benötigen nur eine geringe CPU-Leistung.

Diese Aussagen sind sicherlich in gewisser Weise richtig. Webanwendungen zur Textverarbeitung sind zum Beispiel deutlich im Vorteil, wenn Dokumente von mehreren Personen an unterschiedlichen Orten bearbeitet werden sollen, selbst wenn sie vielleicht nicht ganz so benutzerfreundlich und leistungsstark sind wie ein richtiges Textverarbeitungsprogramm.

In diesem Kapitel stelle ich Ihnen kurz die Sprachen vor, die wir in diesem Buch zur Entwicklung der Beispiel-Webanwendung verwenden.

3.2 HTML5

Begriffsbestimmung

Der Begriff »HTML5« kann recht verwirrend sein.

HTML5 beinhaltet die Spezifikation einer Markup-Sprache zur Entwicklung von Dokumenten, die im Webbrowser dargestellt werden sollen. Als Auszeichnungssprache erweitert HTML5 frühere Versionen von HTML und XHTML, vereinfacht sie gleichzeitig aber auch.

Als Erweiterung von HTML bietet HTML5 dem Webentwickler eine Reihe von neuen Tags. Viele davon sollen die Beschreibung des Inhalts in HTML verbessern und erleichtern. So gibt es in HTML5 beispielsweise `header`- und `footer`-Elemente. Nach Anwendung dieser Tags sehen die Seiten weder anders aus als zuvor, noch verhalten sie sich anders. Damit sind sie eine der weniger interessanten Neuerungen von HTML5. Wir werden uns in diesem Buch noch mit manchen dieser neuen Tags beschäftigen.

Audio- und Videounterstützung

Weiterhin enthält HTML5 neue Elemente zur Unterstützung von Audio und Video und einen Canvas zur Darstellung von 2D-Formen und Bitmaps. Genau diese Features von HTML5 haben viel Aufmerksamkeit bekommen, vor allem weil HTML5 dadurch zur direkten Konkurrenz von Adobe Flash geworden ist. Apple hat den Einsatz von Adobe Flash auf bestimmten Geräten unterbunden – stattdessen sollen die Websites die entsprechenden HTML5-Funktionen verwenden, da diese standardkonform sind und ohne Plugins auskommen. Diese Multimedia-Funktionalität von HTML5 bleibt in diesem Buch weitgehend außen vor, da sie in der Regel für Web-Anwendungen nicht relevant ist. Man sollte jedoch wissen, dass es sie gibt.

Strenge Regeln kontraproduktiv

Im Rahmen der Vereinfachung insbesondere von XHTML wurde erkannt, dass die strenge Umsetzung der HTML-Standards in früheren Versionen nicht nur unnötig war (nicht regelkonforme Seiten konnten im Browser dennoch angezeigt werden), sondern auch kontraproduktiv (weil es keinen Standard für den Umgang mit aus irgendwelchen Gründen ungültigen Seiten gab, kochte jeder Browseranbieter sein eigenes Süppchen). Die HTML5-Spezifikation liefert den Herstellern jedoch detaillierte Vorgaben, wie ihre Browser ein einheitliches Dokumentobjektmodell aus den vorhandenen Daten erzeugen sollten. In weiten Teilen definiert die HTML-Spezifikation genau diese Regeln – und auch diese würden den Rahmen des Buches sprengen.

Hinweis

Keine Sorge, wenn Sie nicht mit dem Document Object Model (DOM) vertraut sind – es wird noch erläutert. Auch mit XHTML müssen Sie sich nicht auskennen; es ist weitestgehend überholt.

Formularkomponenten

HTML5 verbessert auch die HTML-Formularkomponenten. Es liefert nicht nur neue Feldtypen (zum Beispiel zur Datums- und Farbwahl), sondern auch zusätzliche Attribute für bestehende Eingabefelder. HTML5 ermöglicht zudem die native Validierung von Formularkomponenten.

Standards für APIs

Neben der Markup-Sprache und verschiedenen Formularkomponenten bietet HTML5 eine Reihe von Standards für die APIs, die vom Webbrowser bereitgestellt werden. Diese APIs sind breit gefächert: Das Spektrum reicht von der Offline-Speicherung von Daten und Inhalten über das Lesen von Dateien bis hin zu Hintergrundprozessen, Server-Sent Events und vielem mehr. Genau diese Merkmale von HTML5 machen den Webbrowser erst zu einer echten Plattform für die Anwendungsentwicklung. Für die Beispiel-Webanwendung in diesem Buch verwenden wir viele der neuen APIs.

HTML5 immer noch Arbeitsentwurf

Der HTML5-Standardisierungsprozess ist schon an sich sehr interessant. Viele Standards orientieren sich an bereits bestehenden Browserfunktionen. So wurde beispielsweise die Technologie hinter AJAX (das XMLHttpRequest-Objekt) zunächst als proprietäre Funktion für den Internet Explorer entwickelt. Andere Browser bauten diese Funktion dann nach. Als sie in allen gängigen Browsern unterstützt wurde, wurde sie vom W3C standardisiert. Offiziell gilt HTML5 immer noch als Arbeitsentwurf.

Hinweis

Das World Wide Web Consortium (W3C) ist die Hauptstandardisierungsorganisation für das World Wide Web. Die Webseite finden Sie unter *http://www.w3.org*. Die HTML5-Spezifikation finden Sie unter *http://www.w3.org/TR/html5*.

W3C und WHATWG

Eigentlich wird die HTML5-Spezifikation von zwei verschiedenen Organisationen entwickelt, dem W3C und dem WHATWG. Beide bieten die Standards unter ihren eigenen Lizenzen an. Die Verdienste des WHATWG um HTML5 sind in Wirklichkeit sehr viel höher einzuschätzen als die des W3C. Das W3C wollte zunächst nicht auf HTML5 setzen, sondern XML-basierte Standards weiterentwickeln. Irgendwann erkannte das W3C, dass es aufs falsche Pferd gesetzt hatte und beteiligte sich aktiv an der Entwicklung von HTML5.

Erwähnenswert ist auch, dass die Versionen der Spezifikationen bei W3C und WHATWG nicht komplett übereinstimmen. Vom Standpunkt eines Anwendungsentwicklers aus ist dies weitgehend irrelevant. Unter Programmierern kursiert das Sprichwort: »Code gewinnt immer.« Dies gilt auch für HTML5: Die Spezifikation ist weitgehend uninteressant; es sind die Browser-Implementierungen, die zählen.

Web-SQL-API

Oft wird ein Standard von einem ganz bestimmten Browseranbieter vorangetrieben, was gelegentlich in eine Sackgasse führt. Ein Beispiel ist die Web-SQL-API. In anderen Fällen wird eine Spezifikation vorangetrieben, obwohl sie keine breite Unterstützung findet und daher eine ungewisse Zukunft hat (das gilt beispielsweise für die File-Writer- und die File-System-API). Im Idealfall unterstützen aber die gängigen Browser die API gemäß dem anerkannten Standard.

Lebendiger Standard

Wichtig ist bei HTML5 weiterhin, dass es sich um einen lebendigen Standard handelt. In den folgenden Kapiteln werden Sie sehen, dass im HTML5-Dokumententyp keine Version angegeben wird: Es handelt sich einfach um HTML, das sich gemeinsam mit dem Standard weiterentwickeln wird, und die Browser werden diese Weiterentwicklung übernehmen.

3.3 JavaScript

Native Unterstützung in fast allen Browsern

Die dynamischen und interaktiven Merkmale einer Webanwendung erfordern JavaScript. JavaScript ist die einzige Sprache, die von fast allen vorhandenen Browsern nativ unterstützt wird. Sie tauchte erstmals im Jahr 1995 in einer frühen Version des Browsers Netscape Navigators auf und wurde schnell vom Internet Explorer übernommen.

ECMAScript Version 5

Zunächst möchte ich eine Kleinigkeit hinsichtlich der Terminologie klären: Der Sprachkern von JavaScript wurde als ECMAScript standardisiert. Wenn es in diesem Buch um JavaScript geht, ist damit technisch ECMAScript Version 5 gemeint (die neueste Version – EC6 – befindet sich momentan noch im Entwicklungsstadium).

JavaScript wurde nach der Programmiersprache Java benannt. Dies zielte aber nicht in erster Linie auf eine Ähnlichkeit zwischen den beiden Sprachen ab, sondern man wollte auf den Bekanntheitsgrad von Java aufsetzen. In Wirklichkeit unterscheidet sich JavaScript ganz deutlich von Java, und zwar aus den folgenden Gründen:

Dynamische Typisierung

Anders als Java mit seiner statischen Typisierung unterstützt JavaScript die dynamische Typisierung. Das heißt, dass Sie in JavaScript eine Variable ohne Deklaration ihres Typs verwenden können. Der Typ wird erst zur Laufzeit ermittelt.

First-Class-Funktionen

JavaScript verfügt über First-Class-Funktionen: Es ist möglich, einer Variablen eine Funktion zuzuweisen oder sie als Parameter an eine andere Funktion zu übergeben. Das scheint keine große Sache zu sein – in Wirklichkeit eröffnen sich dem Software-Entwickler damit sehr viele Möglichkeiten und er kann Anwendungen mittels funktionaler Programmierung entwickeln. Diese Techniken werden im Detail in den folgenden Kapiteln erläutert.

Prototyping-Techniken

Zwar gibt es in JavaScript Klassen; die Umsetzung ist jedoch ein wenig verwirrend. Ich empfehle daher, Klassen so weit wie möglich zu vermeiden und Objekte mithilfe von Prototyping-Techniken zu erstellen.

Dieses Buch soll nicht als Handbuch zu allen Funktionen der JavaScript-Sprache dienen. Stattdessen zeigt es die grundlegenden Ansätze, die Sie als Software-Entwickler nutzen können.

Wenn Sie bisher noch keine Zeit hatten, sich mit JavaScript zu beschäftigen, und vor allem wenn Sie bisher nur mit statisch typisierten Sprachen programmiert haben, wird die Eleganz und Flexibilität der JavaScript-Sytax Sie wahrscheinlich beeindrucken.

3.4 jQuery

JavaScript-Bibliothek

jQuery ist eine JavaScript-Bibliothek, die die Entwicklung von JavaScript-Anwendungen im Webbrowser vereinfacht.

Wegen des dokumentenzentrierten Charakters von Webseiten wird JavaScript normalerweise für die Auswahl von Elementen im Dokument (die interne Darstellung eines Dokuments im Browser wird DOM, Document Object Model, genannt), für die Manipulation dieser Elemente oder für die Reaktion auf durch sie ausgelöste Ereignisse verwendet. Diese Merkmale unterstützt JavaScript durch die Document Object Model-API, die auch in der HTML5-Spezifikation enthalten ist. Im Wesentlichen stellt jQuery eine elegante Schnittstelle zur DOM-API dar.

Sizzle

Den Kern von jQuery bildet die Selektor-Engine »Sizzle«. jQuery empfängt Auswahlkriterien auf der Grundlage von CSS-Selektoren und liefert eine Reihe von Elementen aus dem Dokument zurück, die diese Kriterien erfüllen. Auf die so ausgewählten Elemente können Sie dann eine Vielzahl von Funktionen anwenden. So lassen sich verschiedene Operationen durchführen oder Event-Listener hinzufügen.

Obwohl die Möglichkeiten von jQuery die von JavaScript oder der nativen DOM-API keineswegs übersteigen, erfreut es sich aus verschiedenen Gründen enormer Beliebtheit:

Gründe für die Beliebtheit von jQuery

- Sie müssen sich nicht mehr mit den Macken der verschiedenen Browser herumärgern.
- jQuery bietet eine umfangreiche und prägnante Syntax, die von den meisten als eine große Verbesserung gegenüber der Document Object Model-API angesehen wird.
- Sie können sehr einfach benutzerdefinierte Plugins für jQuery schreiben und es so an bestimmte Anforderungen anpassen.
- Es gibt zahlreiche Open-Source-Plugins für jQuery, unter anderem ein beliebtes UI-Toolkit namens jQuery UI.

jQuery hat durchaus Konkurrenz, zum Beispiel von Dojo Toolkit und Prototype. Jedoch hat jQuery auf dem Markt eine kritische Masse erreicht und ist inzwischen beinahe zum Quasistandard für Webanwendungen geworden.

3.5 Cascading Style Sheets

Cascading Style Sheets (CSS) stellen eine Stylesheet-Sprache für HTML dar. Fast alle Gestaltungselemente, die in HTML noch aus der Zeit vor CSS verblieben waren, wurden aus HTML5 gestrichen. Die gesamte Präsentation sollte nun vollständig mit CSS erfolgen.

Trennung von Inhalt und Gestaltung

CSS ermöglicht die Trennung von Inhalt und Gestaltung. Beide können unabhängig voneinander geändert werden. Der Inhalt der Seite soll in HTML erzeugt werden, die Darstellung der Seite in CSS.

Das bedeutet auch, dass ein und derselbe Inhalt für verschiedene Geräte (z. B. Smartphones) wiederverwendet werden kann, indem einfach ein neues Stylesheet verwendet wird.

CSS enthält eine Reihe von Eigenschaften, die beschreiben, wie die Elemente in einem Dokument gestaltet werden sollen, sobald bestimmte Regeln darauf zutreffen, und wie diese Eigenschaften miteinander in Wechselwirkung treten sollen. Die Stile, die auf Elemente angewandt werden können, sind unbeschreiblich vielfältig und wurden in CSS3 noch erheblich erweitert. Die zuletzt genannte Spezifikation läuft größtenteils parallel zu HTML5.

Als Software-Entwickler braucht man normalerweise nicht alle CSS-Funktionen bis ins Detail zu kennen. Wichtig ist es jedoch, die Grundlagen zu verstehen – sonst drohen Frustration und Ärger. Genau wie die übrigen Sprachen in diesem Buch ist CSS nicht so einfach, wie es scheinen mag. Wer sich mit der Entwicklung von Webanwendungen beschäftigt, braucht zwingend solide Grundlagenkenntnisse.

Bei der Entwicklung der Beispiel-Webanwendung in diesem Buch werde ich CSS weitgehend ignorieren. Anhang A enthält jedoch eine ausführliche Einführung – schlagen Sie dort einfach bei Bedarf nach.

4 Die Markup-Sprache HTML5

In diesem Kapitel sehen wir uns die Änderungen an der Markup-Sprache HTML an, die mit den HTML5-Spezifikationen Einzug gehalten haben. Für die beispielhafte Webanwendung nutzen wir viele neue HTML5-Elemente, aber natürlich längst nicht alle. Auch einige neu hinzugekommene Attribute werden eingesetzt.

Wie bereits erwähnt, bezieht sich der Begriff HTML5 gleichzeitig auf eine Markup-Sprache (eine neue Version von HTML) und auf eine Reihe von APIs. Die HTML5-APIs werden in späteren Buchkapiteln besprochen.

4.1 Seitenstruktur

Minimalbeispiel

Oft ist es gut, mit dem einfachsten Beispiel überhaupt anzufangen. In HTML5 sieht das so aus:

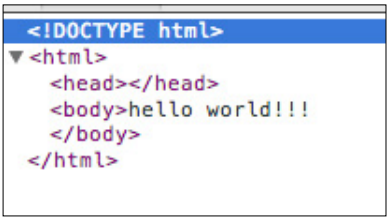
```
<!DOCTYPE html>
hello world!!!
```

Öffnen Sie Ihren Lieblingstexteditor, geben Sie den oben abgedruckten Code ein, speichern Sie das Dokument als `hello.html` und öffnen Sie es in Ihrem bevorzugten HTML5-fähigen Browser.

Dies sieht vielleicht nicht wie eine HTML-Seite aus – es fehlen zum Beispiel die `html`- und `body`-Tags –, aber der Browser weiß den minimalistischen Inhalt trotzdem richtig zu interpretieren.

Chrome-Entwicklerwerkzeuge

Lassen Sie uns das vom Chrome-Browser für diese Seite erstellte Document Object Model untersuchen. Öffnen Sie dazu die Chrome-Entwicklerwerkzeuge und klicken Sie auf das Register *Elements*. Wir sehen jetzt Folgendes:



Standardkonformes Document Object Model

Der Browser hat die HTML-Seite als solche erkannt und daraus ein standardkonformes, internes Document Object Model abgeleitet.

Im Vergleich zu früheren Versionen ist die HTML5-Spezifikation sehr locker. Bisher wurden die HTML-Definitionen nach und nach immer strenger. Am deutlichsten wurde das bei HTML 4.01 Strict, das im Jahr 2000 veröffentlicht wurde, und bei XHTML. Diese Standards legten größten Wert auf strenge Syntaxregeln. So mussten Tags stets geschlossen werden und alle Attributwerte mussten in Anführungszeichen stehen. Dieses Bestreben nach strengen Regeln hatte mehrere Gründe:

Frühere Versionen von HTML und XHTML basierten auf anderen Markup-Sprachen (SGML beziehungsweise XML), woraus sich für HTML Einschränkungen ergaben.

Browser können nach strengen Regeln verfasste Dokumente leichter auswerten. Dadurch verbessert sich die Kompatibilität zu unterschiedlichen Browsern.

Auch andere Werkzeuge können Dokumente besser verarbeiten, die strikten Regeln folgen.

Trotz der immer strenger werdenden HTML- und XHTML-Spezifikationen pochten die Browser niemals wirklich auf deren Einhaltung. Die Browserhersteller gelangten frühzeitig zu der Einsicht, dass es in ihrem besten Interesse war, jegliches Markup so gut wie möglich zu interpretieren – egal, wie sehr es von irgendwelchen Standards abwich. Aus Sicht der Browserhersteller war die Sache ganz einfach: Stellt ein Browser Seiten aus technischen Gründen nicht dar, verliert der Anwender über kurz oder lang die Geduld und steigt auf ein anderes Produkt um, das weniger pingelig agiert.

So hatte sich ein großer Zwiespalt zwischen den Browserherstellern und den Entwicklerteams der technischen Spezifikationen aufgetan. Das ergab keinerlei Sinn, weil die technischen Spezifikationen nur dann von Wert waren, wenn sie auch von den Browserherstellern umgesetzt wurden. Mit einer guten Portion Pragmatismus hat HTML5 diese Situation jetzt völlig auf den Kopf gestellt.

Interpretation ungültigen Markups

Wie bereits angesprochen, beschreibt ein großer Teil der HTML5-Spezifikation, wie die Browser vormals als ungültig abgestempeltes Markup interpretieren sollen. Daher konnte Chrome aus dem »hello

world«-Beispiel ein Document Object Model erzeugen. Und deshalb würde auch Firefox genau dasselbe Document Object Model daraus bauen.

Korrektes Seitenskelett

Das oben gezeigte Beispiel funktioniert zwar, das folgende Seitenskelett ist zur Entwicklung von HTML5-Dokumenten aber doch angemessener:

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

Ich erkläre später die Bedeutung der einzelnen Abschnitte. Zuerst bereichern wir aber unser Dokument noch um einige Inhalte, um unsere Beispiel-Webanwendung auf den Weg zu bringen.

Wir entwickeln in diesem Buch eine Aufgabenliste als Webanwendung. Sie erfüllt folgende Funktionen:

Funktionen der im Buch entwickelten Webanwendung

- Neue Aufgaben anlegen (Dazu gehören auch ein Termin und eine Aufgabenkategorie)
- Aufgaben bearbeiten
- Aufgaben löschen
- Aufgabenliste betrachten
- Überfällige Aufgaben werden gesondert hervorgehoben.
- Aufgaben als erledigt markieren

Diese Webanwendung mag relativ simpel erscheinen. Ihre Komplexität reicht aber aus, um die wichtigen Funktionen jeder Sprache zu zeigen. Zugleich ist sie aber nicht so umfangreich oder kompliziert, dass ich mich im Buch wiederholen müsste oder dass die Aufgabe zu schwierig würde.

Wir schreiben zuerst das Markup, das der Aufgabenliste zugrunde liegt. In diesem Kapitel sind alle Inhalte noch statisch – später ändern wir das und erzeugen mit JavaScript und jQuery dynamische Inhalte.

Beispielprojekt anlegen

Starten Sie das Beispielprojekt, indem Sie irgendwo in Ihrem Dateisystem ein Verzeichnis anlegen und darin eine Datei namens `tasks.html` erzeugen. Diese sollte den folgenden Inhalt haben:

Hinweis

Denken Sie daran, dass Sie alle Beispiele von der Website zum Buch herunterladen können: *http://www.dpunkt.de/leseproben/5160/Beispielcode.zip*. Für jedes Kapitel gibt es eine eigene ZIP-Datei mit allen Beispielen.