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. Wäh-

1. 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.

rend 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]

2. Dies gilt etwa für `hgroup`, das bereits nicht mehr dazugehört.

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 Un-

terstützung für Pointer Events mit Hersteller-Präfix, wie etwa `MSPointerDown`.

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 WebKit-Mobilbrowsern 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.