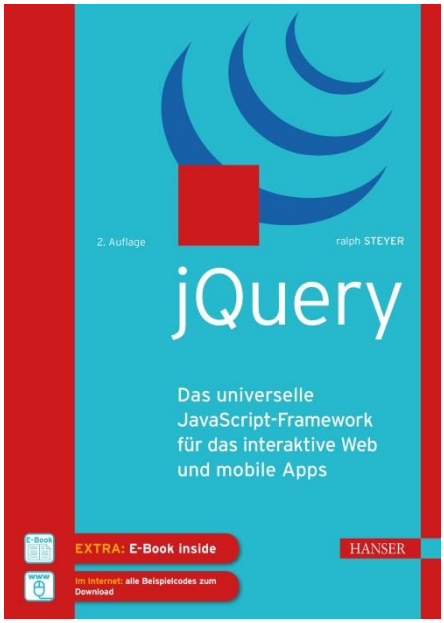
HANSER



Leseprobe

zu

jQuery

Das universelle JavaScript-Framework für das interaktive Web und mobile Apps

von Ralph Steyer

ISBN (Buch): 978-3-446-45558-0
ISBN (E-Book): 978-3-446-45651-8

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/
sowie im Buchhandel

Inhalt

Vorwort ..... **XV**

1 Einleitung **1**

1.1 Das Umfeld 1

1.2 Frameworks und Toolkits 2

1.3 Was behandeln wir in diesem Buch? 3

1.3.1 Das Kern-Framework jQuery 4

1.3.2 Plugins, UI, Mobile & Co 4

1.3.3 Wen sehe ich als Zielgruppe für das Buch? 5

1.4 Schreibkonventionen 6

1.4.1 Listings 6

1.5 Was benötigen Sie beziehungsweise was nutzt Ihnen? 6

1.5.1 Hardware und Betriebssystem 6

1.5.2 jQuery, jQuery Mobile, jQuery UI & mehr 7

1.6 Die Browser 14

1.6.1 Verschiedene Betriebssysteme und virtuelle Maschinen zum Testen 15

1.7 Die Entwicklungsumgebungen und nützliche Helferlein 16

1.7.1 Editoren – oft bereits mit gewisser Hilfestellung 17

1.7.2 Integrierte Entwicklungsumgebungen 17

1.7.3 Kleine Helferlein 22

1.7.4 Node.js und Git 23

1.7.5 Integrierte Entwicklungstools in Browser und Browser-Add-ons 24

1.7.6 Der Webserver zum realistischen Testen 25

1.8 Zusammenfassung 26

2 Sprung ins kalte Wasser **27**

2.1 Zugriff auf Elemente und Schutz des DOM 27

2.2 Veränderung der Webseite mit DHTML 33

2.3 Animiertes Verkleinern und Vergrößern eines Elements 36

2.4 Attribute dynamisch verändern 41

2.5 Zusammenfassung 43

3 Grundlagenwissen ... 45
3.1 Das WWW, das Web 2.0 und das Client-Server-Prinzip im Internet ... 45
3.1.1 Programmierung im WWW ... 46
3.1.2 Das Web 2.0 und Ajax ... 46
3.2 JavaScript und das Verhältnis zu jQuery ... 47
3.2.1 Die allgemeine Einbindung von JavaScript in Webseiten ... 48
3.2.2 JSON ... 50
3.3 DOM und Objekte ... 51
3.3.1 DOM und der Zugriff auf Elemente einer Webseite ... 52
3.4 Style Sheets und DHTML ... 53
3.4.1 CSS – die Standardsprache des Webs ... 53
3.4.2 Die konkrete Syntax von CSS-Deklarationen ... 54
3.4.3 Selektoren ... 55
3.5 Zusammenfassung ... 55

4 Wie jQuery grundsätzlich arbeitet ... 57
4.1 Grundsätzliches zum Zugriff auf Elemente der Webseite ... 58
4.1.1 Ein Beispiel für unterschiedliche Knoten ... 59
4.2 Der jQuery-Namensraum und das jQuery-Objekt ... 62
4.3 Spezielle Datentypen und Strukturen in Query ... 63
4.3.1 Methoden ... 63
4.3.2 Optionen ... 64
4.3.3 jqXHR ... 64
4.4 Die Funktion jQuery() und der Alias $() ... 64
4.4.1 Der Kontext ... 66
4.4.2 Verketten von Methoden und die jQuery-Warteschlange ... 67
4.4.3 Funktionsaufrufe nacheinander ausführen – die jQuery-Warteschlange ... 67
4.4.4 jQuery.then() und jQuery.when() ... 68
4.5 Funktionen nach Aufbau des DOM ausführen ... 69
4.5.1 Callback oder anonyme Funktion als Parameter für jQuery() ... 69
4.5.2 Das ready-Event ... 71
4.5.3 document.ready() in eine externe JavaScript-Datei auslagern ... 72
4.5.4 Mit jQuery.holdReady() das ready-Event beeinflussen ... 73
4.6 Ein Element mit jQuery() erstellen und in die Webseite einfügen ... 73
4.6.1 Ein trickreiches kleines Problem – können Sie es lösen? ... 75
4.6.2 Optionen zum Initialisieren von Attributen ... 79
4.7 Direkter Zugriff auf DOM-Elemente mit get( ) ... 81
4.8 Gemeinsame Verwendung von jQuery und anderen Frameworks ... 81
4.8.1 Die Funktion jQuery.noConflict() ... 82
4.8.2 Einen alternativen Alias verwenden ... 83
4.9 Datenspeicherung im DOM ... 84

4.10 Mehr zum Kontext und jQuery-Utilities ... 90
4.11 Zusammenfassung ... 92

5 Umgang mit Selektoren und Filtern ... 93
5.1 Klassische DOM-Zugriffsmethoden ... 93
5.1.1 Objektfelder ... 94
5.1.2 Zugriff über einen Namen ... 94
5.1.3 Verwandtschaftsbeziehungen ... 94
5.1.4 Elementnamen, IDs und Klassennamen ... 94
5.1.5 DOM-Zugriffsmöglichkeiten versus jQuery ... 95
5.2 Was versteht man unter Selektoren? ... 95
5.2.1 Was ist ein Selektor? ... 95
5.2.2 Was sind Filter? ... 96
5.2.3 CSS3, SQL und XPath als Grundlagen und Hintergrund ... 96
5.3 Basisselektoren und hierarchische Selektoren ... 97
5.3.1 Beispiele zur Verdeutlichung ... 99
5.3.2 Potenzielle Fallen ... 110
5.4 Filterausdrücke ... 110
5.4.1 Basisfilter ... 111
5.4.2 Inhaltsfilter ... 118
5.4.3 Sichtbarkeitsfilter ... 121
5.4.4 Kindfilter ... 125
5.4.5 Attributfilter ... 128
5.4.6 Filter für Formularelemente und Formularfilter ... 133
5.4.7 Formularfilter zur Auswahl aufgrund des Zustands ... 137
5.5 Filtermethoden ... 139
5.5.1 Praktische Beispiele von Filtermethoden ... 140
5.6 Zusammenfassung ... 149

6 Zugriff auf die Elemente einer Webseite ... 151
6.1 Inhalte von Knoten abfragen und verändern – html() und text() ... 151
6.1.1 Ein Beispiel zu html() und text() ... 152
6.2 Inhalt von Formularfeldern – val() ... 156
6.2.1 Ein Beispiel für den Zugriff auf Formularfelder ... 156
6.3 Zugriff auf Attribute und Eigenschaften mit attr() und prop() ... 158
6.3.1 Zugriff auf Attribute ... 158
6.3.2 Zugriff auf Eigenschaften ... 158
6.3.3 Unterschied zwischen Attributen und Eigenschaften ... 159
6.3.4 Das Beispiel zum Zugriff auf Eigenschaften und Attribute ... 160
6.4 Einfügen von Knoten in eine Webseite ... 166
6.4.1 append() und prepend() ... 166
6.4.2 appendTo() und prependTo() ... 171

6.5 Knoten nachher oder vorher einfügen ... 176
6.5.1 after() und before() ... 177
6.5.2 insertAfter() und insertBefore() ... 177
6.6 Ummanteln ... 177
6.6.1 Einzeln mit wrap() ummanteln ... 177
6.6.2 Alles ummanteln mit wrapAll() ... 180
6.6.3 Innere Bereiche ummanteln mit wrapInner() ... 181
6.6.4 Den Mantel ablegen – unwrap() ... 182
6.7 Ersetzen mit replaceWith() und replaceAll() ... 182
6.7.1 Ersetzen mit replaceWith() ... 182
6.7.2 Alles ersetzen mit replaceAll() ... 186
6.8 Überflüssige Knoten entfernen ... 188
6.8.1 Die Methoden empty() und remove() ... 188
6.8.2 Die Alternative zu remove() – detach() ... 194
6.8.3 Löschen von Attributen ... 195
6.9 Vervielfachen mit clone() ... 195
6.10 Suchen & Finden ... 195
6.10.1 Von Kindern und Eltern ... 196
6.10.2 Von Geschwistern ... 199
6.10.3 Nachfolger mit has() suchen ... 201
6.11 Finden mit find() und contents() ... 201
6.12 Mit each() und map() über Arrays und Objekte iterieren ... 203
6.12.1 jQuery.each() und jQuery.map() ... 204
6.12.2 Die Methoden each() und map() ... 209
6.13 Die Methode add() ... 210
6.14 Die Methoden end() und andSelf() ... 212
6.15 Zusammenfassung ... 213

7 Layout und Formatieren mit Style Sheets unter jQuery ... 215
7.1 Hintergrundinformationen ... 215
7.1.1 CSS in jQuery – Vermischung von Layout und Funktionalität? ... 216
7.2 Die Methode css() ... 217
7.2.1 Abfragen von Stileigenschaften ... 217
7.2.2 Setzen von Eigenschaften ... 217
7.3 Klassen von Elementen verändern ... 218
7.3.1 Klassen hinzufügen – addClass() ... 218
7.3.2 Klassen wegnehmen – removeClass() ... 227
7.3.3 Klassen umschalten mit toggleClass() ... 227
7.3.4 Test auf eine Klasse – hasClass() ... 228
7.4 Methoden zur Positionierung ... 228
7.4.1 Bestimmen der Position mit position() ... 229
7.4.2 Position relativ zum Dokument – offset() ... 234
7.4.3 Methoden zum Scrollen ... 238

7.5 Höhe und Breite ... 241
7.5.1 height() und width() ... 241
7.6 Innere und äußere Maße ... 245
7.7 jQuery.cssHooks ... 248
7.8 Zusammenfassung ... 250

8 Ein Praxisbeispiel – eine Datumskomponente ... 251
8.1 Das Ziel ... 251
8.1.1 Die Basiswebseite ... 252
8.2 Die CSS-Datei – Templates ... 253
8.3 Die JavaScript-Datei ... 254
8.4 Zusammenfassung ... 259

9 Ereignisbehandlung unter jQuery ... 261
9.1 Grundlagen zu Ereignissen, Eventhandlern, Triggern und Datenbindung ... 262
9.1.1 Ereignisse ... 262
9.1.2 Allgemeines zu Eventhandlern ... 262
9.1.3 HTML-Eventhandler ... 263
9.1.4 JavaScript/DOM-Eventhandler ... 263
9.1.5 Das Ereignisobjekt ... 264
9.1.6 Blubbern und die Bubble-Phase ... 265
9.1.7 Datenbindung ... 267
9.1.8 Trigger ... 267
9.1.9 Delegation ... 268
9.1.10 Versprechen (Promises) ... 268
9.2 Das Ereignisobjekt in jQuery ... 269
9.2.1 Der Konstruktor von jQuery.Event ... 269
9.2.2 Die Eigenschaften des Ereignisobjekts jQuery.Event ... 270
9.2.3 Die Methoden eines Objekts vom Typ jQuery.Event ... 274
9.2.4 Die Ausführung des bereitstehenden Ereignisses anhalten ... 277
9.3 Ich habe fertig – $(document).ready() ... 278
9.4 Event-Helper ... 278
9.5 Erweiterte Methoden für das Eventhandling ... 283
9.5.1 Datenbindung ... 283
9.5.2 Triggern ... 288
9.6 Live Events ... 292
9.6.1 Die veraltete Methode live() ... 292
9.6.2 Die veraltete Methode delegate() und die delegate-Variante von on() 293
9.6.3 Die Methoden die() und undelegate() ... 297
9.7 jQuery.proxy() ... 297
9.8 Weiterentwicklung der Datumskomponente ... 300
9.9 Zusammenfassung ... 302

10 Effekte und Animationen ... 303
10.1 Grundsätzliche Anwendung ... 303
10.1.1 Speed is all you need ... 303
10.1.2 Die Angabe eines Callback ... 305
10.1.3 Verkettung ... 305
10.1.4 Warteschlangen ... 306
10.1.5 Beendigung mit stop(), finish() und jQuery.fx.off ... 306
10.1.6 Endlosanimationen ... 307
10.1.7 Art der Animationen – Easing ... 307
10.2 Konkrete Effekte mit Standardmethoden ... 309
10.2.1 Anzeigen und Wegblenden – die Methoden show() und hide() ... 309
10.2.2 Gleiteffekte – slideDown(), slideUp() und slideToggle() ... 309
10.2.3 Transparenzeffekte – fadeIn(), fadeOut() und fadeTo() sowie toggle() ... 312
10.3 Individuelle Animationen mit animate() ... 316
10.4 Zusammenfassung ... 324

11 Asynchrone Programmierung ... 325
11.1 Ajax und XMLHttpRequest (XHR) – Grundlagen ... 326
11.1.1 Ein XMLHttpRequest-Objekt manuell erzeugen ... 326
11.1.2 Die Methoden eines XHR-Objekts ... 327
11.1.3 Die Eigenschaften eines XHR-Objekts ... 328
11.1.4 Das Datenformat bei einer Ajax-Kommunikation ... 328
11.1.5 Exemplarischer Ablauf einer Ajax-Anfrage ... 329
11.2 Spezialitäten bei der Ajax-Unterstützung in jQuery ... 330
11.2.1 JSONP und Remote Requests ... 330
11.2.2 Das jqXHR-Objekt ... 331
11.2.3 Grundsätzliches zu Methoden in jQuery für Ajax-Anfragen ... 331
11.2.4 Vermeidung von Caching ... 333
11.3 Anwendung der Standardmethoden für Ajax ... 333
11.3.1 $.get() und $.post() ... 333
11.3.2 JSON-Daten anfordern und verarbeiten – getJSON() und parseJSON() ... 342
11.4 Ein Skript per Ajax nachladen – jQuery.getScript() ... 344
11.5 Die allgemeine Variante zum Laden von Daten – load() ... 346
11.5.1 Filter angeben ... 347
11.6 Serialisieren von Daten ... 351
11.6.1 Die Methode serialize() ... 351
11.6.2 Die Methode serializeArray() ... 353
11.6.3 Die allgemeine Version zum Serialisieren – $.param() ... 353
11.7 Vorgabewerte für Ajax – jQuery.ajaxSetup() ... 354

11.8 Ajax Events und Ajax-Eventhandler ... 354
11.8.1 Lokale Ereignisse ... 354
11.8.2 Globale Ereignisse ... 356
11.9 Die vollständige Kontrolle ... 357
11.9.1 jQuery.ajax() ... 357
11.9.2 Erweiterte Techniken für $.ajax() ... 365
11.10 Web Worker ... 367
11.10.1 Was ist ein Web Worker? ... 367
11.10.2 Erzeugen von Web Workern ... 368
11.10.3 Die Kommunikation mit einem Web Worker ... 369
11.10.4 Einen Worker mit terminate() beenden ... 369
11.10.5 Ein Beispiel zu einem klassischen Web Worker ... 370
11.11 Deferred Object und Promises ... 372
11.11.1 Das Umfeld – Promises ... 373
11.11.2 Die speziellen jQuery-Methoden zum Umgang mit einem Deferred Object ... 376
11.11.3 Ein konkretes Beispiel mit Deferred Objects ... 378
11.12 Ajax mit Deferred Objects ... 379
11.12.1 Generische Ajax-Methoden mit then() verketten ... 380
11.12.2 Das Laden von Skripts mit Deferred Objects ... 380
11.12.3 JSONP und Deferred Objects ... 381
11.12.4 Mehrere Ajax-Anfragen ausführen und synchronisieren ... 382
11.13 Das Callbacks Object ... 383
11.14 Zusammenfassung ... 387

12 jQuery UI ... 389
12.1 Was versteht man unter jQuery UI? ... 390
12.1.1 Komponenten zur Unterstützung der Interaktion ... 390
12.1.2 Widgets ... 390
12.1.3 Erweiterte Effekte ... 391
12.1.4 Das Themen-Framework samt ThemeRoller ... 391
12.2 Der Einstieg in jQuery UI ... 392
12.3 Wie nutzt man jQuery UI grundsätzlich? ... 393
12.3.1 Download und der ThemeRoller ... 393
12.3.2 Die Bereitstellung und Einbindung lokaler Ressourcen ... 397
12.3.3 Einbindung über ein CDN ... 398
12.4 Verwenden der Komponenten in jQuery UI ... 399
12.4.1 Die Defaulteinstellung ... 400
12.4.2 Einige grundsätzliche Regeln zu Komponenten und Widgets ... 403
12.4.3 Eigenschaften/Optionen von Komponenten ... 403
12.4.4 Methoden von Komponenten ... 406
12.4.5 Events bei Komponenten und Widgets ... 410

12.5 Ein Überblick über die Komponenten und Widgets ... 410
12.5.1 Die Interaktionskomponenten ... 410
12.5.2 Die Widgets ... 411
12.6 Effekte ... 423
12.6.1 Die Methode effekt() ... 423
12.6.2 Farbanimationen mit animate() ... 423
12.7 Zusammenfassung ... 424

13 jQuery Mobile ... 425
13.1 Grundlagen ... 426
13.1.1 Die Plattformen ... 427
13.1.2 Widgets und Features ... 429
13.1.3 Download und Bereitstellung ... 429
13.1.4 Themes und der ThemeRoller ... 431
13.2 Das Rollensystem und data-role ... 432
13.3 Der grundsätzliche Aufbau einer mobilen Seite ... 432
13.3.1 Ein erstes vollständiges Beispiel ... 433
13.4 Verknüpfen von Seiten ... 434
13.4.1 Externe Links mit Hijax ... 435
13.4.2 Interne Links und das spezielle Verständnis einer Seite ... 435
13.4.3 Zurück in der Historie ... 436
13.5 Die Übergänge ... 438
13.6 Dialoge ... 439
13.7 Schaltflächen ... 440
13.7.1 Schaltflächen mit Icons ... 440
13.7.2 Blockelement oder Inline-Element ... 441
13.7.3 Gruppierung ... 442
13.8 Toolbars und Navigationsbars ... 443
13.9 Listen ... 445
13.10 Formularelemente ... 446
13.10.1 Feldcontainer ... 447
13.10.2 Die verschiedenen Formularelemente ... 447
13.10.3 Deaktivieren von Elementen ... 448
13.10.4 Plugin-Methoden bei Formularelementen ... 448
13.10.5 Abschicken der Formulardaten ... 448
13.11 Spezielle Ereignisse ... 448
13.11.1 Berührungsereignisse ... 449
13.11.2 Änderung der Orientierung ... 449
13.11.3 Verschiebeereignisse ... 449
13.11.4 Seitenereignisse ... 450
13.11.5 Ein Beispiel mit Reaktionen auf Events ... 450
13.12 Kollabierte und expandierte Inhalte ... 452
13.13 Zusammenfassung ... 454

14 Plugins ... 455
14.1 Die Plugin-Seiten von jQuery ... 455
14.2 Ein vorhandenes Plugin suchen und verwenden ... 458
14.3 Eigene Plugins erstellen ... 465
14.3.1 Warum eigene Plugins erstellen? ... 465
14.3.2 Erstellen eines ersten Plugins ... 466
14.3.3 Die wesentlichen Regeln zur Erstellung eines einfachen Plugins ... 468
14.3.4 Regeln zur Erstellung komplexerer Plugins ... 469
14.3.5 Ein Beispiel für ein Plugin mit Optionen ... 470
14.3.6 Ein weiteres Beispiel für ein Plugin mit Optionen ... 472
14.3.7 Ein Plugin veröffentlichen ... 473
14.4 Zusammenfassung ... 477

15 Das Habitat rund um jQuery ... 479
15.1 Sizzle ... 479
15.2 QUnit ... 482
15.2.1 xUnit-Testing ... 482
15.3 Bootstrap ... 488
15.3.1 Responsive Design ... 488
15.3.2 Bootstrap zur Umsetzung von RWD ... 489
15.3.3 Herunterladen von Bootstrap ... 490
15.3.4 Eine Basis-Vorlage ... 490
15.3.5 Ein Kontaktbeispiel ... 491
15.4 Zusammenfassung ... 493

16 Anhang ... 495
16.1 Grundlagen zu JavaScript ... 495
16.1.1 Case-Sensitivität ... 495
16.1.2 Variablen, Literale und Datentypen ... 495
16.1.3 Funktionen und Methoden ... 497
16.1.4 Objekte in JavaScript ... 499
16.2 Die Webseite zum Buch ... 501

Index ... 503

Vorwort

Das World Wide Web ist schon viele Jahre unglaublich erfolgreich und hat sich gerade die letzten Jahre bezüglich seiner zukünftigen technischen Basis als auch der Form der Darstellung von Inhalten sowie der Interaktion mit dem Anwender und der Dynamik stark weiterentwickelt. Das Stichwort *Dynamik* deutet schon an, dass statische, passive Webseiten überholt sind und immer seltener werden. Bereits die Gegenwart und insbesondere die Zukunft gehören sogenannten Rich Internet Applications (RIAs). Gerade aber diese interaktiven Seiten und Applikationen im Web sind ohne geeignete Frameworks kaum noch effektiv zu erstellen und zu warten, nicht zuletzt auch deswegen, weil viele optische sowie funktionale Features wie animierte Inhaltsaufbereitung oder komfortable Benutzereingabemöglichkeiten mittlerweile verbreitet sind und vom verwöhnten Anwender ebenfalls erwartet werden. Dementsprechend wird jedoch der Aufwand zur Erstellung von solchen Webangeboten immer größer.

Nun war es einige Jahre nicht wirklich deutlich zu erkennen, in welche Richtung sich das World Wide Web zur Umsetzung solcher anspruchsvoller Applikationen wirklich entwickelt. Es gab längere Zeit verschiedene technische Ansätze, die als verschiedene Optionen für die Zukunft des Webs offen waren. Aber wenn Sie aktuell die populären und halbwegs modern gemachten interaktiven Angebote im World Wide Web betrachten, werden Sie eigentlich nur noch konservativ programmierte Applikationen auf Basis von Ajax (Asynchronous JavaScript and XML) sowie klassisches DHTML (Dynamic HTML) vorfinden. Ein paar Seiten setzen vielleicht noch auf das veraltete, proprietäre Flash, aber die verschwinden mehr und mehr. Daneben gab es aber über eine geraume Zeit Versuche, neuere proprietäre Techniken wie JavaFX, Silverlight oder AIR/Flex einzusetzen und teils sogar vollkommen auf HTML, CSS und JavaScript zu verzichten. Aber die Aktivitäten der Hersteller in Hinsicht auf die Weiterentwicklung mit proprietären Ansätzen ist mittlerweile fast vollkommen zum Erliegen gekommen.

Dementsprechend setzen aktuell für interaktive anspruchsvolle Webapplikationen die meisten Firmen, Organisationen sowie auch Privatanwender weiter ganz konservativ auf dynamisches HTML und Ajax, zumal sich mit HTML 5 und CSS 3 offene Standards etablieren, die zudem auch von den Anbietern der proprietären Techniken ganz offiziell unterstützt werden. Und nicht zuletzt setzen die großen – in der Hinsicht unabhängigen – Hersteller wie Google oder Apple ebenso explizit auf HTML 5 und CSS 3 für die Zukunft.

Ihnen sollte nun etwas aufstoßen, dass ich im Zusammenhang mit Ajax und HTML5/CCS3 von **konservativ** spreche. Es ist noch nicht ganz so lange her, da war Ajax das Buzzword schlechthin im World Wide Web. Ajax ist die programmiertechnische Basis dessen, was um das Jahr 2005/2006 als Web 2.0 in aller Munde war. Immerhin hat erst Ajax es möglich gemacht, bei Bedarf nur die Inhalte einer Webseite auszutauschen, die tatsächlich neu vom Webserver angefordert werden müssen. Die bereits geladene Webseite bleibt bei einer Datennachforderung per Ajax im Browser vorhanden und mittels DHTML wird gezielt an einer bestimmten Stelle in der Webseite ein Austausch bestehenden Inhalts durch die neu nachgeladene Information vorgenommen. Dabei kann die nachgeladene Information entweder aus Klartext (inklusive HTML-Fragmenten) oder aus strukturiertem XML oder JSON (JavaScript Object Notation) bestehen. Die Vorteile dieser Vorgehensweise sind bei stark interaktiven Applikationen mit häufigem Serverkontakt sofort offensichtlich und mittlerweile voll etabliert.

Dennoch bedeutet die Verwendung von Ajax respektive DHTML keinen Einsatz von modernen Webtechniken, denn die Grundlagen dieser damit zusammengefassten Technologien gibt es alle bereits seit 1997, was meine Bezeichnung als **konservativ** verdeutlicht. Und dass sich Ajax erst fast zehn Jahre später wie eine Explosion über das World Wide Web verbreitet hat, zeigt aus meiner Sicht ganz deutlich, dass das Internet und das World Wide Web in der Entwicklung recht träge, konservativ und langsam sind. Diese von mir provokant formulierte These soll nun das Internet und das World Wide Web nicht diskreditieren! Es ist nur so, dass sich im Internet Technologien nur sehr langsam durchsetzen können, weil sich alle Beteiligten an diesem komplexen, sehr sensiblen Gebilde World Wide Web auf eine gemeinsame Basis einigen müssen. Und das dauert eben! In der Regel viele Jahre. Und auf Dauer setzt sich scheinbar im Web nur das durch, was gut und einfach ist.

Wenn Sie nun aber eine moderne Webapplikation auf Basis von Ajax und DHTML erstellen wollen, ist eine Programmierung von Hand wie erwähnt sehr aufwendig und fehlerträchtig. Zwar ist das grundsätzliche Erstellen von DHTML- bzw. Ajax-Applikationen nicht sonderlich schwierig, wenn man die Grundlagentechniken HTML bzw. XHTML, CSS und JavaScript beherrscht. Das Zusammenspiel dieser – einzeln gesehen – in der Tat recht einfachen Webtechnologien im Client kann jedoch äußerst diffizil sein, was nicht zuletzt ein Resultat der Browserkriege des letzten Jahrtausends ist. Dazu kommen im Fall von Ajax oft noch der permanente Austausch von Daten zwischen Client und Webserver sowie die sehr feingliedrige Verteilung von Geschäftslogik zwischen Client und Server hinzu.

Zudem erzwingt die eingeschränkte Leistungsfähigkeit von JavaScript oft eine nicht ganz triviale Programmierung von Strukturen, die in leistungsfähigeren (insbesondere objektorientierten) Programmiertechniken auf dem Silbertablett serviert werden. So gesehen ist die Erstellung einer interaktiven Applikation für das Web heutzutage durchaus mit der Komplexität der Erstellung einer Desktop-Applikation bzw. einer verteilten Netzwerkapplikation zu vergleichen, wenn sie sich an den aktuellen Ansprüchen der Konkurrenz messen will. Das erkenne ich auch daran, dass in meinen Schulungen zu JavaScript, Ajax oder CSS mehr und mehr Programmierer sitzen, die aus mächtigen Sprachen wie Java oder C# kommen (früher war das Vorwissen im Programmierumfeld eher geringer). Für mich ist das ein deutliches Zeichen, dass die Ansprüche an eine moderne RIA steigen und von reinen Designern nicht mehr erfüllt werden können. Anders ausgedrückt – mit modernen RIAs ist das Web endgültig den Kinderschuhen entwachsen.

Nicht zuletzt bringt die Erstellung von modernen Webseiten und insbesondere Ajax-RIAs ein hohes Maß an Tests und Anpassung an verschiedene Webbrowser und Plattformen mit sich. An den unterschiedlichsten Stellen warten tückische Fallstricke. Natürlich ist die manuelle Erstellung von komplexen DHTML-Aktionen wie Drag & Drop oder animierten Menüs nicht jedermanns Sache. Ihnen sind sicher ebenfalls die oft extrem diffizilen Abhängigkeiten von den verschiedenen Browsern, Browserversionen und Betriebssystemplattformen bekannt. Diese Probleme nehmen zwar in modernen Browsern ab, sind aber immer noch vorhanden und zudem nutzen gerade viele Firmen auch noch alte Browser.

Sogenannte Frameworks und Toolkits für Ajax bzw. JavaScript versprechen nun für viele Aufgabenstellungen und Probleme im Umfeld von modernen Webseiten Abhilfe. Sie stellen vielfach vor allem JavaScript-Funktionsbibliotheken mit getesteten und hochfunktionellen Lösungen sowie ausgereifte Style Sheets bereit, damit Sie nicht jedes Mal das Rad neu erfinden und dessen einwandfreie Funktionalität umfangreich testen müssen. Dazu gibt es gelegentlich auch spezielle Tools und Programme, die eine Arbeit mit diesen Bibliotheken unterstützen oder gar erst möglich machen. Auch bringen einige mächtige HTML-Editoren mittlerweile sogar eigene Frameworks mit.

Wir werden uns in diesem Buch nun – wie der Titel unzweifelhaft aussagt – jQuery widmen und schauen, wie Sie dieses geniale Framework einsetzen können, um Ihre Webapplikationen zu verbessern bzw. die Erstellung zu vereinfachen oder bestimmte Features gar erst möglich zu machen. Wenn Sie die Möglichkeiten von jQuery nicht schon kennen, lassen Sie sich positiv überraschen, wie einfach Ihnen dieses mächtige Werkzeug Webseiten ermöglicht, die alle moderne Effekte und Features enthalten.

Zu diesem Einstieg in jQuery wünsche ich Ihnen viel Spaß und viel Erfolg. Doch vorher möchte ich ein paar abschließende Bemerkungen zu meiner Person machen. Meinen Namen werden Sie auf dem Buchumschlag oder am Ende des Vorworts gelesen haben – Ralph Steyer. Ich habe in Frankfurt/Main an der Goethe-Universität Mathematik studiert (Diplom) und danach anfangs einen recht typischen Werdegang für Mathematiker genommen – ich bin erst einmal bei einer großen Versicherung gelandet, aber schon da mit EDV-Schwerpunkt. Zunächst arbeitete ich einige Jahre als Programmierer mit Turbo Pascal und später mit C und C++. Nach vier Jahren wechselte ich in die fachliche Konzeption für eine Großrechnerdatenbank unter MVS. Die Erfahrung war für meinen Schritt in die Selbstständigkeit sehr motivationsfördernd, denn mir wurde klar, dass ich das nicht auf Dauer machen wollte. Seit 1996 verdiene ich daher meinen Lebensunterhalt als Freelancer, wobei ich fliegend zwischen der Arbeit als Fachautor, Fachjournalist, EDV-Dozent, Consultant und Programmierer wechsele. Daneben referiere ich gelegentlich auf Webkongressen, unterrichte an verschiedenen Akademien und Fachhochschulen, übersetze gelegentlich Fachbücher oder nehme Videotrainings auf. Das macht aus meiner Sicht einen guten Mix aus, bewahrt vor beruflicher Langeweile und hält mich sowohl in der Praxis als auch am Puls der Entwicklung. Insbesondere habe ich das Vergnügen und gleichzeitig die Last, mich permanent über neue Entwicklungen auf dem Laufenden zu halten, denn die Halbwertszeit von Computerwissen ist ziemlich kurz. Dementsprechend ist mein Job zwar anstrengend, aber vor allem immer wieder spannend. Doch nun ab in die Welt von jQuery!

Ralph Steyer

Frühjahr 2018

2 Sprung ins kalte Wasser

In diesem Kapitel werden wir ohne weitere Vorbereitungen Kontakt zu jQuery schaffen und erste Beispiele mit jQuery erstellen. Wir springen also direkt ins kalte Wasser. Sie sollen bereits in dieser frühen Phase Ihres Einstiegs in dieses faszinierende Thema ein Gefühl für das bekommen, was man mit jQuery anstellen kann und was Ihnen dieses Framework bringt. Dabei wird bewusst in Kauf genommen, dass zu diesem Zeitpunkt bei den Quelltexten Fragen offen bleiben. Diese Fragen werden aber im Laufe der folgenden Kapitel geklärt. Die Erläuterungen zu den Listings werden auch in dieser Phase nur oberflächlich[1] sein, um nicht vom Stock zum Stöckchen zu geraten. Wir wollen möglichst schnell zur Praxis mit jQuery kommen und erst einmal spielen. Und das bedeutet Beispiele erstellen.

2.1 Zugriff auf Elemente und Schutz des DOM

Wenn Sie sich bereits etwas mit der Programmierung im WWW auskennen, wissen Sie, dass man auf die Bestandteile einer Webseite per JavaScript oder einer anderen Skriptsprache im Browser über ein Objektmodell mit Namen **DOM** (Document Object Model) zugreifen kann. Es gibt für so einen Zugriff verschiedene Standardtechniken, die aber alle ihre spezifischen Schwächen haben. Insbesondere müssen Sie beim Zugriff auf ein einziges Element der Webseite (oder eine Gruppe) in der Regel ziemlich viele Zeichen eingeben. Das ist mühselig und fehleranfällig. Die meisten Frameworks stellen deshalb eine Notation zur Verfügung, über die so ein Zugriff mit einer verkürzten, vereinheitlichten Schreibweise erfolgen kann. Und zudem kompensieren die dahinterliegenden Mechanismen der Frameworks diverse Schwächen der Standardzugriffsverfahren, indem sie vor allen Dingen browserabhängige Besonderheiten kompensieren sowie diverse fehlende Funktionalitäten des reinen DOM-Konzepts ergänzen. Besonders wichtig – diese Kompensation ist in der Regel auf allen offiziell unterstützten Browsern getestet und funktioniert deshalb sehr zuverlässig.

[1] Aber keinesfalls unwichtig.

Das folgende Beispiel zeigt weiterhin eine andere wichtige Funktionalität von jQuery – den Schutz des DOM. Was es damit auf sich hat, wird natürlich noch viel genauer erläutert. Nur soweit vorab – beim Laden (Parsen) der Webseite verarbeiten verschiedene Browser die Webseite unterschiedlich und es kann beim Zugriff auf die Elemente der Webseite zu einer Vielzahl von Problemen kommen. Das gilt vor allen Dingen dann, wenn man in einem Skript **zu früh** auf Elemente einer Webseite zugreifen will – also bevor der Browser den DOM korrekt aufgebaut hat. Hier bietet jQuery ein zuverlässiges Verfahren, um diesem Problem Herr zu werden. Und was Ihnen das Beispiel quasi nebenbei noch zeigt ist, wie Sie per jQuery standardisiert auf Inhalte von Elementen mit Text zugreifen und auf Ereignisse reagieren können. Doch genug der Vorbemerkung – hier ist unser erstes Listing (*kap2_1.html*):

Listing 2.1 Das erste jQuery-Beispiel

```
<!DOCTYPE html>
<html lang="de" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>Schutz des DOM</title>
    <link href="lib/css/kap2_1.css" rel="stylesheet" type="text/css" />
    <script src="https://code.jquery.com/jquery-3.2.1.min.js"
            integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
            crossorigin="anonymous"></script>
  <script type="text/javascript">
    $(document).ready(function () {
      $("#a").click(function () {
        $("#ausgabe").html("Muss das sein?");
      });
      $("#b").click(function () {
        $("#ausgabe").html("Ein nettes Spiel.");
      });
      $("#c").click(function () {
        $("#ausgabe").html("Ein seltsames Spiel. " +
          "Der einzig gewinnbringende Zug ist " + "nicht zu spielen!");
      });
    });
  </script>
</head>
<body>
  <h1>Willkommen bei WOPR</h1><h3>Wie wäre es mit einem kleinen Spiel?</h3>
  <button id="a">Tic Tac Toe</button><button id="b">Schach</button>
  <button id="c">Weltweiter Thermonuklearer Krieg</button>
  <div id="ausgabe"></div>
</body>
</html>
```

Erstellen Sie die HTML-Datei einfach in einem eigenen Arbeitsverzeichnis und speichern Sie sie unter dem genannten Namen.

Hinweis

Beachten Sie, dass jQuery in dem Listing (als auch folgenden) von einem **CDN** geladen wird, aber das ist für die Funktionalität irrelevant. Sie können die JavaScript-Datei von jQuery auch von Ihrem Webserver oder lokal referenzieren. Aber gerade für die ersten Beispiele ist die Referenz auf ein CDN einfacher, da Sie das Framework nicht selbst bereitstellen müssen. Die Attribute `integrity` `crossorigin` werden dabei für die Überprüfung der Unterressourcenintegrität (**SRI** – Subresource Integrity) verwendet. Dadurch können Browser sicherstellen, dass auf Servern von Drittanbietern gehostete Ressourcen nicht manipuliert wurden.

Die Verwendung von SRI wird allgemein empfohlen, wenn Bibliotheken aus einer Drittanbieterquelle geladen werden. Sollte es Probleme geben, wenn man beispielsweise eine Webseite aus einer IDE wie Visual Studio und deren internen Webserver oder Ihren lokalen Webserver lädt, verzichten Sie zur Entwicklungszeit einfach auf die beiden Attribute und notieren für die Referenz auf die jQuery-Bibliothek einfach `<script src="https://code.jquery.com/jquery-3.2.1.min.js"></script>`. Das funktioniert einwandfrei. Wenn Sie Webseiten veröffentlichen und ein CDN nutzen, sollten Sie aber auf jeden Fall die Attribute verwenden.

In der Praxis fasst man alle eigenen Ressourcen eines Projekts meist innerhalb eines eigenen Verzeichnisses zusammen. Für ein Webprojekt ist es das Sinnvollste, diese Verzeichnisse im freigegebenen Ordner Ihres Webservers anzulegen. Im Fall von Apache/XAMPP wäre das in der Regel das Verzeichnis *htdocs*. Das hat den Vorteil, dass Sie – wenn der Webserver läuft – zum Test unmittelbar über HTTP und einen richtigen Webaufruf gehen können und nicht nur die Datei über das FILE-Protokoll in den Browser laden müssen (also das klassische Öffnen als Datei oder das einfache Reinziehen der Datei in den Browser). Letzteres ist ja nicht praxisorientiert, da später die Seiten natürlich auch vom Besucher über einen Webserver angefordert werden.

Wenn Sie mit einer IDE wie Eclipse oder dem Visual Studio arbeiten, kann man meist direkt aus der IDE eine Webseite über einen integrierten Webserver ausführen und in einen Browser Ihrer Wahl (der natürlich installiert sein muss) laden. In Visual Studio können Sie die Ausführung über die Funktionstaste Strg + F5 aufrufen.

Im Head der Webseite sehen Sie einen Verweis auf eine CSS-Datei *kap2_1.css*, die wir der Vollständigkeit hier kurz angeben wollen (sie spielt aber im Grunde keine Rolle):

Listing 2.2 Die referenzierte CSS-Datei

```
body {
  background:black; color:white; font-size:20px;
}
#ausgabe {
  background:white; color:red; font-size:20px; padding:10px; margin:10px;
  border-width:1pt; border-style:solid; width:350px; min-height:75px;
}
```

Praxistipp

Wenn Sie in Visual Studio in einer Projektmappe mehrere Projekte zusammenfassen, müssen Sie immer das richtige Startprojekt festlegen, wenn Sie aus der IDE eine Webseite über den integrierten Webserver von Visual Studio in einen Browser laden wollen. Sonst kann es zu Fehlern kommen. Das Startprojekt können Sie im Projektmappen-Explorer mit einem Klick mit der rechten Maustaste auf die passende Projektmappe und das Kontextmenü festlegen.

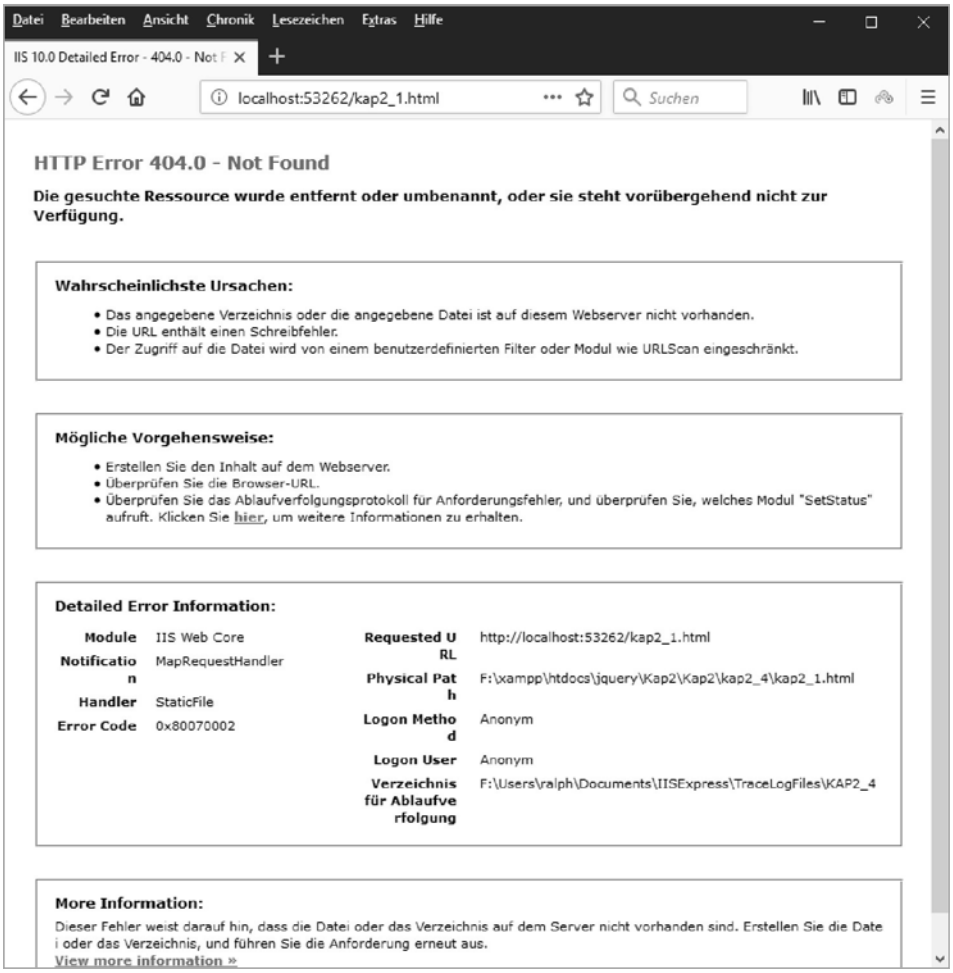


Bild 2.1 Bei einer Meldung der Art findet der Webserver die HTML-Datei nicht oder darf darauf nicht zugreifen.

Praxistipp

Die CSS-Datei befindet sich in einem Unterverzeichnis *lib* des Projektverzeichnisses, in dem die Webseite gespeichert wurde. Dieses Verzeichnis enthält noch ein weiteres Unterverzeichnis *css*, worin konkret die Datei gespeichert ist. Wenn wir mit externen JavaScript-Dateien arbeiten, werden diese dann in einem Unterverzeichnis *js* des Unterverzeichnisses *lib* des Projektverzeichnisses gespeichert. Diese Strukturierung hat sich in der Praxis so oder ähnlich auf breiter Front durchgesetzt. Das bedeutet, dass sich auch die jQuery-Bibliothek genau da befinden wird, wenn Sie statt eines CDN eine heruntergeladene Version verwenden, die Sie dann selbst bereitstellen. Aber selbstverständlich können Sie auch eine ganz andere Pfadstruktur wählen. Nur sollten Sie grundsätzlich strukturieren und das konsequent durchziehen.

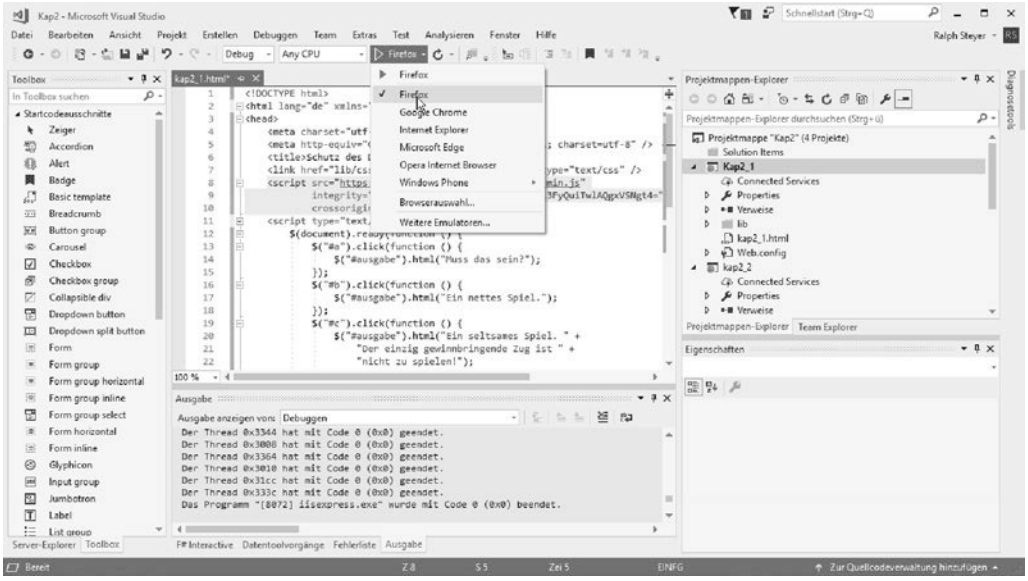


Bild 2.2 Die Projektstruktur – hier im Projektmappen-Explorer von Visual Studio gut zu sehen

In den Zeilen hinter der Referenz auf die externe CSS-Datei sehen Sie die Referenz auf eine externe JavaScript-Datei – die jQuery-Bibliothek, die in dem konkreten Fall wie gesagt von einem CDN geladen wird. In den folgenden Zeilen steht ein gewöhnlicher JavaScript-Container. In diesem wird mit `$(document)` die Webseite angesprochen. Die Funktion `$()` steht in jQuery für eine Kurzschreibweise, um ein Element der Webseite zu referenzieren. Sie ist **der** (!) zentrale Dreh- und Angelpunkt des gesamten Frameworks und Sie finden diese verkürzten Zugriffsschreibweisen auch in den folgenden Zeilen immer wieder. Nur wird dort als Parameter eine sogenannte ID eines Elements verwendet.

Hinweis

Beachten Sie, dass ein Element (im Sinne von jQuery) als Parameter von `$()` nicht in Hochkommata eingeschlossen wird, eine ID (oder ein anderer Selektor) hingegen schon.

Widmen wir uns kurz der Methode `ready()`. Diese stellt sicher, dass die enthaltenen Aufrufe erst dann ausgeführt werden, wenn die Webseite vollständig geladen und der DOM korrekt aufgebaut wurde. Wie schon angedeutet und ohne richtig in die Tiefe zu gehen – das ist bereits ein Feature, dessen Wert man hoch einschätzen muss.

Hinweis

Für die Leser mit entsprechenden Vorkenntnissen ein kleiner Vorgriff – die Methode `ready()` ist eine Alternative für den Eventhandler `onload`. Dieser Eventhandler galt jedoch lange Zeit als unzuverlässig, weil er fehlerhaft in verschiedenen älteren Browser implementiert war.

Im Inneren der `ready()`-Methode werden drei Ereignisbehandlungsroutinen notiert, die jeweils die Reaktion bei einem Klick auf die angegebenen Elemente spezifizieren. In unserem Beispiel sind das drei Schaltflächen, die jeweils mit einer eindeutigen ID gekennzeichnet sind.

Hinweis

Die Methode `click()` kapselt naheliegender Weise den Funktionsaufruf des Eventhandlers `onclick`.

Die Zuordnung zur richtigen Funktion erfolgt über die ID und das Auslösen der Funktion innerhalb der Methode `click()`. Beachten Sie, dass wir hier eine sogenannte **anonyme Funktion** (also ohne Bezeichner) verwenden.

Interessant wird es, wenn ein Anwender nun auf eine Schaltfläche klickt. Dabei wird in einem Bereich der Webseite eine spezifische Textausgabe angezeigt. Dazu verwenden wir wieder `$()` und eine ID für die Selektion des Bereichs (ein `div`-Block) und die Methode `html()` für den Zugriff auf den Inhalt.

Hinweis

Die Methode `html()` ist in jQuery die Alternative zur Verwendung von `innerHTML`. Das Interessante dabei ist, dass `innerHTML` schon in der Praxis seit vielen Jahren verwendet, aber erst mit HTML5 offiziell standardisiert wird.

Bild 2.3 Die Webseite mit den drei Buttons – der Anwender hat die dritte Schaltfläche angeklickt.

Hinweis

Wir werden in allen folgenden Beispielen darauf verzichten, den gesamten Head der Webseite abzudrucken. Die Notation würde nur Platz im Buch verschwenden, da sie immer (fast) gleich ist. Nur bei interessanten Änderungen wird der Code abgedruckt.

2.2 Veränderung der Webseite mit DHTML

Grundsätzlich kann man mit Style Sheets die Optik einer Webseite viel besser und effektiver gestalten als mit reinem HTML. Insbesondere kann man damit das Layout von der Struktur der Seite abtrennen. Diese Aussagen sollten – so richtig sie auch sind – für Sie kalter Kaffee sein. Wenn Sie nun die Style Sheets einer Seite dynamisch mit JavaScript verändern, reden wir von DHTML. Aber auch Animationseffekte wie das Ein- und Ausblenden von Teilen einer Webseite über andere JavaScript-Techniken gehören dazu. Lassen Sie uns in diesem und dem folgenden Beispiel ansehen, wie Sie animierte Webseitenänderungen mit jQuery schnell, einfach und bequem und dennoch zuverlässig in den unterschiedlichen Browsern bewerkstelligen können. In diesem Beispiel wechseln wir im Wesentlichen dynamisch die CSS-Klasse eines Elements.

Zuerst betrachten wir eine kleine CSS-Datei, die in der folgenden Webseite eingebunden werden soll und im *lib/css*-Verzeichnis gespeichert sein sollte (*kap2_2.css*):

Listing 2.3 Die neue CSS-Datei

```
body {
  background: black; color: white; font-size: 20px;
}
div {
```

```
  background: white; color: red; font-size: 20px; padding: 10px; margin: 10px;
  border-width: 1pt; border-style: solid; width: 350px; min-height: 75px;
}
.mKlasse {
  background: red; color: yellow; font-size: 24px;
}
```

In der CSS-Datei passiert nicht sonderlich viel. Es werden ein paar Eigenschaften für die gesamte Webseite und alle Elemente vom Typ *div* festgelegt. Von hauptsächlichem Interesse ist die Klasse *.mKlasse*. Diese soll beim Laden der folgenden Webseite noch nicht verwendet werden, sondern erst dynamisch bei einer Anwenderaktion zugewiesen werden. Und dazu kommen JavaScript und jQuery zum Einsatz (*kap2_2.html*):

Listing 2.4 Verändern der verwendeten CSS-Klasse

```
...
  <script type="text/javascript">
    $(document).ready(function () {
      $("#a").click(function () {
        $("#c").addClass("mKlasse");
      });
      $("#b").click(function () {
         $("#c").removeClass("mKlasse");
      });
    });
  </script>
</head>
<body>
  <h1>Veränderung von Style Sheets mit jQuery</h1>
  <button id="a">CSS-Klasse hinzufügen</button>
  <button id="b">CSS-Klasse entfernen</button><hr />
  <div id="c">Niemand ist weiter von der Wahrheit entfernt als derjenige,
    der alle Antworten weiß.</div><hr />
  <div id="div1">
    Vom Wahrsagen lässt sich wohl leben, aber nicht vom Wahrheit sagen.
  </div>
</body>
</html>
```

In dem Beispiel sehen Sie unterhalb einer Überschrift zwei Schaltflächen und zwei Texte jeweils innerhalb eines DIV-Bereichs, der mit einer Trennlinie abgegrenzt wird. Das ist pures HTML. Im Head finden Sie wieder die Verknüpfung mit der CSS-Datei (nicht abgedruckt).

Bild 2.4 Die Seite nach dem Laden

Das Aussehen des Textes unterhalb der Schaltflächen beziehungsweise den ersten DIV-Container wollen wir jedoch mit jQuery nun manipulieren. Dazu hat der DIV-Container eine ID. Der Text darunter ist als Vergleichstext gedacht.

Das Beispiel verwendet zum Zugriff auf Elemente der Webseite die im ersten Beispiel schon besprochenen jQuery-Mechanismen. Auch für die Reaktion auf den jeweiligen Klick auf eine Schaltfläche verwenden wir wieder die Methode `click()`. Soweit nichts Neues also.

Nun sollte Ihnen auffallen, dass wir die CSS-Klasse aus der verknüpften CSS-Datei beim Laden der Webseite noch keinem Element zuweisen. Aber beachten Sie die Zeile `$("#c").addClass("mKlasse");`. Wie der Name der Methode `addClass()` offensichtlich macht, wird durch den Aufruf dieser Methode dem vorangestellten Element die benannte Style-Sheet-Klasse zugewiesen. Das geschieht dynamisch, aber ohne dass die Webseite irgendwie neu geladen wird. Die Funktionalität wird beim Klick des Anwenders auf den entsprechenden Button ausgelöst, wie Sie auf Grund der umgebenden `click()`-Methode sehen.

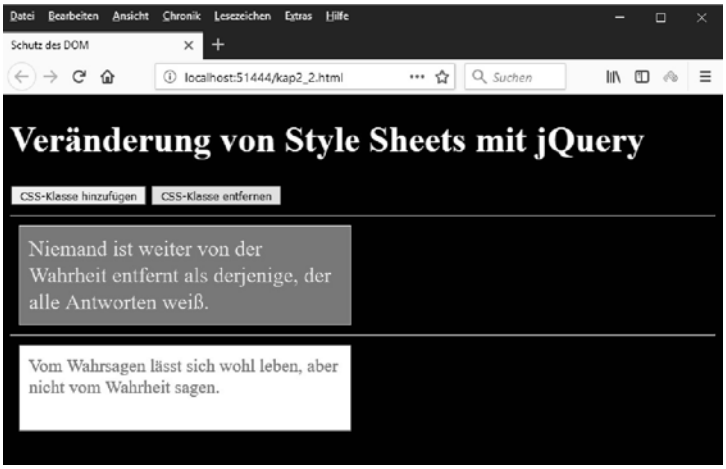


Bild 2.5 Die CSS-Klasse wurde zugewiesen.

In der Reaktion auf die zweite Schaltfläche können Sie erkennen, wie vollkommen analog die Klasse wieder entfernt wird. Dazu kommt die Methode `removeClass()` zum Einsatz. Wenn Sie das Beispiel testen, sehen Sie, dass Schrift und Hintergrund entsprechend verändert werden.

Praxistipp

Mit der Methode `toggleClass()` könnten wir das Beispiel mit analoger Funktionalität noch leichter schreiben. Damit wird eine CSS-Klasse entfernt oder hinzugefügt und zwar immer abhängig vom Zustand. Ist die Klasse bereits zugewiesen, wird sie entfernt und umgekehrt. Wir bräuchten also nur eine Schaltfläche.

2.3 Animiertes Verkleinern und Vergrößern eines Elements

Nun wollen wir mit jQuery ein Element animiert verkleinern und vergrößern und es damit aus- beziehungsweise wieder einblenden. Zuerst schauen wir uns die externe CSS-Datei an. Darin wird eine Eigenschaft definiert, die auf die folgenden Animationen konkrete Auswirkungen hat (*kap2_3.css*):

Listing 2.5 Die CSS-Datei

```
body {
  background: black; color: white; font-size: 20px;
}
#b2 {
  width: 300px;
}
#h2 {
  background: white; color: #0000FF; font-size: 18px;
  padding: 10px; margin: 10px;
}
```

Die für das folgende Beispiel interessante Festlegung betrifft die Breitenangabe der Id `#b2`. Die als Selektor verwendete ID referenziert ein Bild. Die Breitenangabe wird die Art der folgenden Animation beeinflussen. Oder anders ausgedrückt – bei den anderen Bildern, bei denen die Breite nicht festgelegt ist, wird die Animation anders ablaufen.

Bild 2.6 Das Originalaussehen

Aber widmen wir uns zuerst noch der Webseite selbst. Darin finden Sie im Wesentlichen drei Bilder und einen Text darunter. Alle vier Elemente sollen animiert werden (*kap2_3.html*):

Listing 2.6 Drei Bilder und einen Text verkleinern oder vergrößern

```
...
  <script type="text/javascript">
    $(document).ready(function () {
      $("#toggle1").click(function (event) {
        $('#b1').slideToggle('slow');
      });
      $("#toggle2").click(function (event) {
        $('#b2').slideToggle('slow');
      });
      $("#toggle3").click(function (event) {
        $('#b3').slideToggle(10000);
        $('#h2').slideToggle('slow');
      });
    });
  </script>
</head>
<body>
  <h1>Ein Bild und Text mit jQuery animiert ein- und ausblenden</h1>
  <button id="toggle1">Toogle Bild 1</button>
  <button id="toggle2">Toogle Bild 2</button>
  <button id="toggle3">Toogle Text und Bild 3</button><hr />
  <img src="images/b1.jpg" id="b1" /><img src="images/b2.jpg" id="b2" />
  <img src="images/b3.jpg" id="b3" /><hr />
  <h2 id="h2">Animierte Bilder und Texte</h2>
</body>
</html>
```

Im Zentrum dieser Animation steht die Methode `slideToggle()`. Auch dieser Name ist sehr sprechend. Mit diesem Effekt lassen sich Objekte je nach aktuellem Zustand ein- oder ausblenden beziehungsweise verkleinern oder vergrößern. Es wird also der aktuelle Zustand umgeschaltet. Sie sehen die Anwendung gleich vier Zeilen mit Animationsaktivitäten. Wie Sie sicher erkennen, taucht als Parameter eine Zeitangabe auf. Diese legt fest, wie lang die Animation benötigen soll. Man kann in allen Animationen in jQuery solche Parameter für die Geschwindigkeit übergeben. Erlaubte Parameter sind `slow`, `normal`, `fast` oder die Angabe in Millisekunden. Die Angabe in Millisekunden wird aber dann in der Regel nicht in Hochkommata eingeschlossen.

Wenn Sie die Animation des ersten Bilds nachvollziehen, werden Sie sehen, dass das Bild beim Verkleinern in der Höhe und Breite reduziert wird und dann ganz verschwindet. Dabei wird kontinuierlich der rechts stehende Inhalt nach links verschoben, ohne dass „Lücken“ auftreten. Umgekehrt wächst das Bild von diesem Punkt aus nach oben und in der Breite und Höhe, wenn Sie erneut die Schaltfläche betätigen. Die beiden anderen Bilder werden dabei kontinuierlich nach links verschoben.

Bild 2.7 Das erste Bild wird nach unten und in der Breite zusammengestaucht.

Für dieses Verhalten ist massiv von Bedeutung, dass die Breite von diesem Bild **nicht** (!) über das `width`-Attribut beim `img`-Tag oder über CSS festgehalten wird.

Bild 2.8 Das erste Bild wurde ausgeblendet.

Beim zweiten Bild wird die Breite hingegen über die CSS-Regel für die ID b2 festgelegt. Die verhindert, dass auch die Breite verkleinert wird. Sie werden sehen, dass beim Verkleinern das Bild nur in der Höhe zusammenschnurrt und dann ganz verschwindet.

Bild 2.9 Bild 2 wird in der Höhe gestaucht.

Erst wenn das Bild 2 ganz verschwunden ist, wird Bild 3 schlagartig dessen ursprünglichen Raum einnehmen.

Bild 2.10 Bild 2 ist verschwunden.

Beachten Sie nun aber den Text und Bild 3, wenn Sie auf die dritte Schaltfläche klicken. Die Überschrift verschwindet wieder nur hinsichtlich der Höhe. Das Bild 3 hingegen, für das wieder die Breite nicht festgehalten wird, verändert sich in Höhe und Breite.

Bild 2.11 Text und Bild 3 werden unterschiedlich animiert.

Offensichtlich spielt es bei der Wirkung von `slideToggle()` eine Rolle, auf welche Art von Element die Animationstechnik angewendet wird, beziehungsweise es spielen auch noch die CSS- und teils auch formatierenden HTML-Regeln eine Rolle, die einem Element vorher zugewiesen werden. Beachten Sie, dass die Zeitspannen beim Klick auf die dritte Schaltfläche für die jeweiligen Animationen des Textes und des Bilds bewusst unterschiedlich gewählt wurden.

Die Animationen in dem Beispiel sind grundsätzlich unabhängig voneinander. Wenn Sie die Zeitspanne zum Ausführen der verschiedenen Animationen lang genug wählen, um schnell genug Klicks auf die drei Schaltflächen auslösen zu können, können Sie die Animationen parallel laufen lassen.

Die jQuery-Warteschlange

Aber was passiert bei dem Beispiel, wenn Sie die gleiche Schaltfläche **mehrfach** anklicken? Das ist vielleicht etwas überraschend. Die Ereignisse werden kumuliert. Das bedeutet, sie werden nacheinander ausgeführt, wobei ein Folgeereignis erst dann ausgeführt wird, wenn das vorangehende vollständig abgearbeitet wurde. Das ist ein explizites Feature in jQuery – eine Warteschlange (die jQuery-**Queue**). Ein erneuter Klick auf die Schaltfläche führt also nicht dazu, dass die laufende Animation abgebrochen und die neue unmittelbar gestartet wird. Das müsste man gegebenenfalls explizit programmieren.

2.4 Attribute dynamisch verändern

Wir wollen in einem abschließenden Beispiel durchspielen, wie man mit jQuery Attribute bei einem Element der Webseite dynamisch verändern kann. Dazu stellt jQuery die flexible und nützliche Methode `attr()` zur Verfügung. Damit können Sie eines oder mehrere Attribute eines Elements dynamisch verändern. Sie setzen in geschweiften Klammern ein Wertepaar als Parameter, wobei zuerst das Attribut spezifiziert wird, dann folgen ein Doppelpunkt und anschließend ein String mit dem neuen Wert. Alternativ können Sie auch zwei String-Parameter angeben. Bei der Variante stehen der erste Parameter für den Attributnamen und der zweite Parameter für den Wert (in dem Fall können Sie aber nur ein Attribut ändern). Wollen Sie nur den Wert eines Attributs abfragen, geben Sie nur den Namen des Attributs als String-Parameter an.

Praxistipp

Wir werden der Einfachheit halber im folgenden Beispiel nur ein Attribut verändern, aber wenn Sie gleichzeitig mehrere Attribute ändern wollen, brauchen Sie in den geschweiften Klammern nur durch Komma getrennt weitere Wertepaare notieren.

Für unser Beispiel wollen wir ein Bild in der Webseite austauschen, indem wir den Wert des Attributs *src* eines <*img*>-Tags verändern (*kap2_4.html*):

Listing 2.7 Attribute mit jQuery manipulieren

```
...
  <script type="text/javascript">
    $(document).ready(function () {
      $("#toggle1").click(function () {
        $("img").attr({
          src: "images/b1.jpg"
        });
      });
      $("#toggle2").click(function () {
        $("img").attr(
          "src", "images/b2.jpg"
        );
      });
    });
  </script>
</head>
<body>
  <h1>Ein Bild austauschen</h1>
  <button id="toggle1">Bild 1</button><button id="toggle2">Bild 2</button>
  <hr /><img src="images/b1.jpg" />
</body>
</html>
```

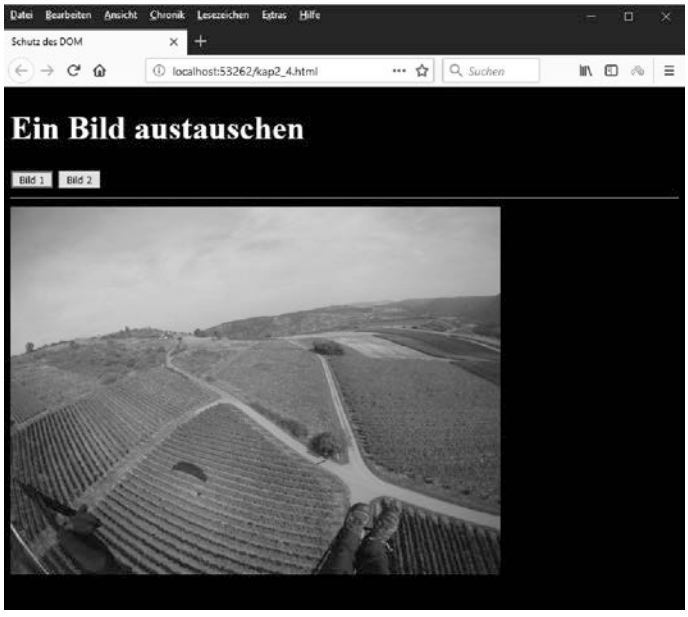


Bild 2.12 Das Bild vor dem Austausch

Wir ändern einmal mit der Notation in den geschweiften Klammern den Wert und einmal mit den zwei String-Parametern. Wie oben beschrieben wird jeweils der Wert von `src` ausgetauscht.

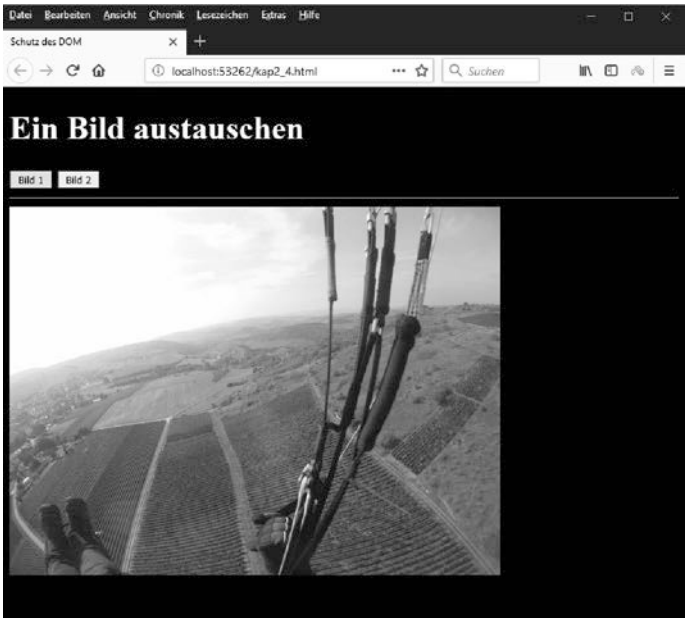


Bild 2.13 Auf die zweite Schaltfläche wurde geklickt.

2.5 Zusammenfassung

Sie haben in dem Kapitel erst einige wenige Beispiele gesehen, die aber schon recht gut entscheidende Schlüsselfaktoren von jQuery demonstriert haben. Sie sollten sich insbesondere die Funktion `$()` und die `ready()`-Methode merken. Aber auch Techniken zur Spezifizierung von Reaktionen wie die `click()`-Methode sind elementar wichtig. Und Animationstechniken wie `addClass()`, `toggleClass()`, `removeClass()` oder `slideToogle()` werden Ihnen auch später in der Praxis bei DHTML-Effekten hilfreich sein. Auch das Verändern von Attributwerten (`attr()`) haben Sie kennengelernt. Richtig verständlich werden die Techniken in den weiteren Kapiteln des Buchs, wenn Sie in das Gesamtkonzept von jQuery tiefer eingestiegen sind.

Index

Symbole

$() 64
$.ajax() 357
$.ajaxPrefilter() 365
$.ajaxSetup() 354
$.ajaxTransport() 367
$.Callbacks() 383
$.data() 84
$(document).ready() 69, 278
$.each() 88, 204
$.Event 269
$.fn 466
$.get() 333
– Deferred Object 380
$.getJSON() 342
$.getScript() 344
– Deferred Object 381
$.load() 346
$.map() 204
$.param() 336
$.parseJSON() 342
$.post() 333
– Deferred Object 380
$.proxy() 297
$.ready()
– jQuery Mobile 450
$.removeData() 84
*.jquery.json package manifest
– Plugin 474
.ui-menu 422

A

abort()
– XHR 327
accepts 358
Accordion 390, 411
Achsen 96
adaptive Webseite 489
add() 210
– Callbacks 383
addClass() 35, 218, 255
after() 177
ajax() 357
Ajax 46, 325
– Mehrere Anfragen ausführen und synchronisieren 382
ajaxComplete() 355
ajaxError 356
ajaxError() 355
Ajax Events 354
ajaxSend 356
ajaxSend() 355
ajaxSetup() 354
ajaxStart 357
ajaxStop 357
ajaxSuccess 357
ajaxSuccess() 355
ajaxTransport() 367
aktuelle Systemzeit 91
always
– animate() 317
always()
– Deferred Object 377
Ancestor 97
andSelf() 212
– deprecated 57
animate() 316, 423

animated 111
Animationen 303
– endlos 307
Animationsfilter 113
Animationsrate 304
Annahmen
– QUnit 482
Anzeigen 309
Apache 25
append() 166, 209, 255
appendTo() 74, 171
application/x-www-form-urlencoded 336, 359
Aptana 17
Array 500
– assoziiertes 500
– durchsuchen 91
– sortieren 91
Arrayliteral 501
Array-Notation 203, 500
asserts
– QUnit 482
async 358
attr() 41, 158
Attribute 158
– löschen 195
Attributfilter 128
Auslöser
– Ereignis 267
Außenabstand 245
autocomplete() 413
Autocomplete 390, 413
axis 96

B

BarCamp 3
Basisfilter 111
Basisselektoren 97
before() 177
beforeSend 358
– Ajax 355
bind() 283
blur() 279
Bootstrap 13
border 245
Bower 11, 462
Brackets 17
Browser 14
Bubble-Phase 265
button 133
Button 391, 415

C

cache 358
Caching 358
– Ajax 333
callback=?
– JSONP 332
Callback-Hölle 268
Callbacks 498
– add() 383
– disable() 383
– disabled() 384
– empty() 383
– fire () 383
– fired() 384
– fireWith() 383
– has() 384
– lock() 383
– locked() 384
– remove() 383
Callbacks Object 383
cancelBubble 276
cancelBubble() 276
Cascading Style Sheets 53
catch()
– Deferred Object 377
– Promises 373
CDN 12, 29
– jQuery Mobile 430
– jQuery UI 398
change() 279
checkbox 133
Checkboxradio 391
checked 137
Child 97
children() 196
clearQueue() 68, 306
click() 32, 272, 279
clone() 195
cloneNode() 195
closest() 198
collapsible 452
Color()
– jQuery 318
complete 358
– Ajax 355
– animate() 317

contains() 91, 118, 141
Content Distribution Network 12
contents 358
contents() 201
contentType 358
– Ajax 336
context 90, 359
contextmenu() 279
controlgroup 391, 442
converters 332, 359
Core
– jQuery 58
createElement() 73
crossDomain 359
Cross-Domain-Zugriff 327
crossorigin 29
CSS 53
css() 66, 217, 255
cssHooks 248
CSS-Klassen
– jQuery-Zugriff 218
CSS-Template 253
currentTarget 270

D

data 359
– Event 270
– Textknoten 151
Data binding 267
data() 84
data-ajax 435
data-direction 436, 438
dataFilter 359
data-fullscreen 444
data-icon 440
data-iconpos 441
data-inline 441
data-rel 436, 439
data-role 432 f., 442 f., 445, 447, 452
– jQuery Mobile 429
data-theme
– jQuery Mobile 431
data-transition 438
data-type 442
dataType 360
Date 256
Datenbindung 262, 267, 283
– Praxis 416
Datenkapselung 51
Datenspeicherung 84
Datentypen 495
– JavaScript 496
Datepicker 391, 415
Datumskomponente 251
dblclick() 279
defaultChecked
– prop() 159
defaultSelected
– prop() 159
Deferred Object 372, 376
– JSONP 381
– Laden von Skripts 380
– then() 380
deferred.isRejected()
– deprecated 57
deferred.isResolved()
– deprecated 57
deferred.pipe()
– deprecated 57
Deklaration 496
delay() 68, 306
delegate() 293
– deprecated 57
Delegaten 268
delegateTarget 270
Delegation 268
deprecated 57, 261
dequeue() 68, 306
Descendant 97
Design
– reaktionsfähig 488
Designvorlagen 489
detach() 194
DHTML 53, 215
dialog() 416
Dialog 391, 406, 416
– jQuery Mobile 439
die() 297
– deprecated 57
disable()
– Callbacks 383
disabled 137, 448
disabled()
– Callbacks 384
disable-Methode 448
Document Object Model 27, 52
Document Type Definition 159
document.createElement() 73
document.getElementById() 94

document.getElementsByClassName() 94
document.getElementsByName() 94
document.getElementsByTagName() 94
document.ready()
– jQuery Mobile 450
Dokumentationstools 23
DOM 27, 52
done
– animate() 317
done()
– Deferred Object 377
DOT-Notation 203, 500
Download
– jQuery UI 393
Download Builder
– jQuery Mobile 430
– jQuery UI 393
Drag & Drop 390, 400
Draggable 410
draggable() 401
Droppable 411
DTD 159
duration
– animate() 317
Duration 304

E

each() 148, 203, 209
easing
– animate() 317
Easing 307, 314
Eclipse 17
effekt() 423
Effekte 303
– jQuery UI 391, 423
Eigenschaften 51, 158
empty 118
empty() 188
– Callbacks 383
Emulator 426
enabled 137
enable-Methode 448
end() 212
Endlosanimationen 307
eq() 111, 140, 148
equal()
– QUnit 484
Ereignisbehandlung 261
Ereignisobjekt
– jQuery 269
Ereignisquelle 268
Ereignisse
– eigene erstellen 269
error 360
– Ajax 355
error() 279
– deprecated 57
eval() 50, 342
even 111
Event 262
Event-Bubbling 266
Eventhandler 262
– Ajax 354
Event-Helper 278
Event-Objekt 264
event source 268
extend() 470
extends() 470

F

Factory
– ajax() 367
fadeIn() 312
fadeOut() 312
fadeTo() 312
fail
– animate() 317
fail()
– Deferred Object 377
Farbanimationen 423
Farben
– animieren 391
fast 304
Feld 157
file 133
FileZilla 25
Filter 96
– load() 347
filter() 140, 142, 201
Filterausdrücke 110
Filtermethoden 139
find() 201
finish() 306
fire()
– Callbacks 383
Firebug 24

fired()
– Callbacks 384
fireWith()
– Callbacks 383
first 111
first() 140
first-child 125
first-of-type 125
Fluid Layout 489
focus 111
focus() 279
focusin() 279
focusout() 280
Formularelemente
– Filter 133
– jQuery Mobile 446
Formularfilter 133
Framework 2
fulfilled
– Promises 373
Funktionen 497
Funktionsaufruf 498
Funktionsreferenz 498
fx.interval 304

G

Geschwister 199
get() 83, 147, 333
getAllResponseHeaders()
– XHR 327
getElementById() 53, 94
getElementsByClassName() 94
getElementsByName() 53, 94
getElementsByTagName() 53, 94
getJSON() 342
getResponseHeader()
– XHR 327
getScript() 73, 327, 344
Getter-Methoden 160
Git 23, 473
Gleiteffekte 309
global 360
globalEval() 91
grep() 91
Größenänderungen 390
Grunt 489
Gruppierung
– Button 442
gt() 111

H

has() 119, 140, 201
– Callbacks 384
hasClass() 228
hasData() 84
Hash 64
header 111
headers 360
height() 241
hidden 121
hide() 309
Hijax 435
Historie
– jQuery Mobile 436
history.back() 436
holdReady() 73, 277
hover() 280
html() 32, 151

I

IETester 15
ifModified 360
image 133
inArray() 91
Indexbereich 140
Information Hiding 51
Inhaltsfilter 118
Innenabstand 245
innerHeight() 245
innerHTML 32, 151
innerText 152
innerWidth() 245
input 133
insertAfter() 177
insertBefore() 177
integrity 29
interval
– jQuery.fx 304
is() 140, 144
isArray() 91
isDefaultPrevented() 274
isEmptyObject() 91
isFunction() 91
isImmediatePropagationStopped() 276
isLocal 360
isNumeric() 91
isPlainObject() 91
isPropagationStopped() 276

isWindow() 91
isXMLDoc() 91

J

Java Development Kit 19
Java Runtime Environment 19
JavaScript
– Versionsangabe 49
JavaScript Object Notation 50
Java Virtual Machine 19
JDK 19
jQuery
– Download 7
jQuery Foundation 3
jQuery Migrate Plugin 8
jQuery Mobile 425
jQuery UI 389
– Download 393
jQuery UI CSS Framework 396
jQuery UI Position utility 409, 421
jQuery Upgrade Guide 8
jQuery XMLHttpRequest 331
jQuery() 64
jQuery.ajax() 357
jQuery.ajaxPrefilter() 365
jQuery.ajaxTransport() 367
jQuery.boxModel
– deprecated 57
jQuery.browser 276 f.
– deprecated 57
jQuery.Callbacks() 383
jQuery.Color() 318
jQuery.contains() 91
jQuery.cssHooks 248
jQuery.data() 84
jQuery.Deferred() 372
jQuery.each() 163, 204
jQuery.Event 269
– Eigenschaften 270
– Methoden 274
jQuery.extend() 469 f.
jQuery.fn 466
jQuery.fn.extends() 469 f.
jQuery.fx.interval 304
– deprecated 57
jQuery.fx.off 306
jQuery.get() 333
jQuery.getJSON() 342
jQuery.getScript() 344
jQuery.globalEval() 91
jQuery.grep() 91
jQuery.holdReady() 73, 277
jQuery.inArray() 91
jQuery.isArray() 91
jQuery.isEmptyObject() 91
jQuery.isFunction() 91
jQuery.isNumeric() 91
jQuery.isPlainObject() 91
jQuery.isWindow() 91
jQuery.isXMLDoc() 91
jQuery.makeArray() 91
jQuery.map() 91, 204
jQuery.merge() 91
jQuery-Namensraum 62
jQuery.noConflict() 82
jQuery.noop() 91
jQuery.now() 91
jQuery.param() 336, 353
jQuery.parseHTML() 91
jQuery.parseJSON() 91, 342
jQuery.parseXML() 91
jQuery.post() 333
jQuery.proxy() 297
jQuery-Queue 41
jQuery.removeData() 84
jQuery.sub()
– deprecated 57
jQuery.support
– deprecated 57
jQuery.then() 68
jQuery.trim() 91
jQuery.type() 91
jQuery.uniqueSort() 91
jQuery-Warteschlange 41, 67
jQuery.when() 68
jqXHR 64, 331
JRE 19
JSFiddle 23
JSLint 23
JSON 50
JSON with Padding 327
jsonp 360
JSONP 327, 330, 362
– Deferred Objects 381
JSON.parse() 50, 342
jsonpCallback 361
JUnit 482
Just-in-time-Compiler
– JavaScript 47
JVM 19

K

keydown() 280
keypress() 280
keyup() 280
Kindfilter 125
Klonen 195
Knoten 58, 96
– verbundene 270
Knotentest
– XPath 97
Kompressoren 23
Konstruktor 499
Kontext 66, 90, 297, 359
Kontextmenü
– Reaktion 279
Kontrollkästchen 133
Konverter
– ajax() 366

L

Laden von Skripts
– Deferred Objects 380
landscape 449
lang 111
last 111
last() 140
last-child 125
Last-Modified
– Headerfeld 360
last-of-type 125
Leerzeichen
– entfernen 91
Lesezugriff 160
Less 489
Listen 445
Listener 268
listview 445
Literale 495
Live Event 292
live() 292
– deprecated 57
load() 280
– deprecated 57
– Filter 347
localhost 335
lock()
– Callbacks 383
locked()
– Callbacks 384
Löschen von Attributen 195
lt() 112

M

makeArray() 91
Map 64, 145
map() 91, 140, 145, 203, 209
margin 245
MariaDB 25
Media Queries 489
Mehrere Ajax-Anfragen
– ausführen und synchronisieren 382
Member 51
menu() 422
Menu 391
Menü 421
merge() 91
metaKey 270
Methoden 51, 497
mimeType 361
Mixins 490
Mobile
– jQuery 425
Mobile-First-Ansatz 489
mousedown() 281
mouseenter() 281
mouseleave() 281
mousemove() 272, 281
mouseout() 272, 281
mouseover() 281
mouseup() 281
Multithreading 367
MySQL 25

N

Nachfolger 201
Namensraum 466
– Ereignis 270
– jQuery 62
namespace
– Event 270
navbar 443
Navigationsbars 443
navigator.userAgent 277

Nebenläufigkeit 368
next() 199
nextAll() 199
nextUntil() 201
noConflict() 82
Node Package Manager 11, 457
Node.js 11, 23, 457
nodeName 90
– prop() 159
nodes 96
nodeType
– prop() 159
nodeValue
– Textknoten 151
noop() 91
normal 304
not() 112, 117, 140
Notepad++ 17
notEqual()
– QUnit 484
notify()
– Deferred Object 377
notifyWith()
– Deferred Object 377
notOk()
– QUnit 484
now() 91
npm 11, 457
nth-child() 125, 128, 144
nth-last-child() 125
nth-last-of-type() 125
nth-of-type() 125
NuGet 462
NYC 3

O

Objekt
– JavaScript 499
– verzögert 372
Objektfelder 94
Objektliteral 499
odd 112, 255
off() 285
Offset 228
offset() 234
offsetParent() 198, 229
on() 285
onclick 32
one() 288
onerror
– window 279
onload 32
only-child 125, 127
only-of-type 125
onmessage
– Web Worker 369
onreadystatechange
– XHR 328
open()
– XHR 327
option 133
Optionen 64
Optionsfelder 133
orientationchange 449
outerHeight() 245
outerWidth() 245
ownerDocument
– prop() 159

P

package.json-Datei 474
padding 245
page
– jQuery Mobile 433
pagebeforecreate 450
pagebeforehide 450
pagebeforeshow 450
pagecreate 450
pagehide 450
pageshow 450
pageX 270
pageY 270
Paketmanager 11
Paketmanifest
– Plugin 474
param() 336, 353
parent 97, 119
parent() 196
parents() 196
parentsUntil() 196
parseHTML() 91
parseJSON() 91, 342
parseXML() 91
password 133, 361
pending 378
– Promises 373
phpMyAdmin 25, 337

Plugin 455
– erstellen 465
– Validierung Webformular 458
– veröffentlichen 473
Plugin-Methoden
– Formularelemente in jQuery Mobile 448
Polsterung 245
portrait 449
position() 229
Positionierung 228
– jQuery UI 409
post() 333
postMessage()
– Web Worker 369
Präfixfilter 131
Prefilter
– ajax() 365
prepend() 166, 209
prependTo() 171
prev() 199
prevAll() 199
preventDefault() 274, 285
prevUntil() 201
processData 336, 361
progress
– animate() 317
Progressbar 416
promise() 379
– Deferred Object 377
Promises 268, 373
– JSONP 381
– Laden von einem Skript 381
prop() 158
proxy() 297
Punktnotation 203, 500

Q

queue 67
– animate() 317
queue() 68, 306
QUnit 13, 482

R

radio 133
Rahmen 245
range
– Slider 447
rangeslider 447
Range Slider 447
ready() 32, 69, 278
– jQuery Mobile 450
ready-Event 71
readyState
– XHR 328
reaktionsfähiges Design 488
Registerblätter 417
reject()
– Deferred Object 377
rejected
– Promises 373
rejectWith()
– Deferred Object 377
Rekursion 114
rel= 435
relatedTarget 270
Remote Requests
– Ajax 330
remove() 188
– Callbacks 383
removeAttr() 195
removeAttribute() 195
removeClass() 36, 227
removed 57
removeData() 84
replaceAll() 182, 186
replaceWith() 182
requestAnimationFrame() 305
reset 133
Resig, John 3
Resizable 411
resizable() 411
resize() 281
resolve()
– Deferred Object 377
resolveWith()
– Deferred Object 377
responseText 360
– XHR 328
responseXML 360
– XHR 328
Responsive Design 13, 488
Responsive Webdesign 488
result
– Event 270
Rollensystem
– jQuery Mobile 432
root 97, 112
RWD 488

S

Same-Origin-Policy 327
Sandbox 327
Sass 489
Schaltflächen
– jQuery Mobile 440
Schreibzugriff 160
scriptCharset 361
scroll() 281
Scrollen 238
scrollLeft() 238
scrollstart 449
scrollstop 449
scrollTop() 238
select 133
select() 281
Selectable 411
selectable() 411
selected 137
selectedIndex
– prop() 159
Selectmenu 391
selector 90
– deprecated 57
Selektion 390
Selektor 95
– CSS 55
– hierarchischer 98
send()
– XHR 328
Serialisieren 351
serialize() 351
serializeArray() 353
setMimeType()
– XHR 328
setRequestHeader()
– XHR 328
Setter-Methoden 160
setTimeout() 303
settled
– Promises 373
show() 309
Sibling 97
siblings() 199
Sichtbarkeitsfilter 121
Simulator 426
size()
– deprecated 57
Sizzle 13, 479
Sizzle() 480
slice() 140
slideDown() 309
Slider 391, 416, 447
slideToggle() 38, 114, 309
slideUp() 309
slow 304
Sortable 411
sortable() 411
Sortieren 390
specialEasing
– animate() 317
Spinner 391, 421
spinner() 421
SQL 96
SRI 29
start
– animate() 317
state() 378
– Deferred Object 377
status
– XHR 328
statusCode 361
statusText
– XHR 328
step
– animate() 317
stop() 306
stopImmediatePropagation() 276
stopPropagation() 276, 285
Style Sheets 53
style-Objekt 216
submit 134
submit() 281
Subresource Integrity 29
success 361
– Ajax 355
swipe 449
Systemzeit 91

T

Tabs 391, 417
tagName
– prop() 159
tap 449
taphold 449
target 112
– Event 270
TDD 482
Templates 251, 489

terminate()
– Web Worker 369
Test Driven Development 482
test()
– QUnit 483
testgetriebene Entwicklung 482
text 134
text() 151
textarea 133
Themen-Framework
– jQueryUI 391
ThemeRoller 391, 395
– jQuery Mobile 431
Themes
– jQuery Mobile 431
then() 68
– Deferred Object 377, 380
– Promises 373
Thenable-Objekte 377
this 90
Threads 368
timeout 361
timeStamp
– Event 270
toggle() 281, 312
– deprecated 57
toggleClass() 36, 227, 287
Toolbars 443
Toolkit 2
Tooltip 391, 420
traditional 362
Transparenz 121
Transparenzeffekte 312
Transporter
– ajax() 366
Trigger 165, 262, 267
trigger() 288
triggerHandler() 291
trim() 91
type 362
– Event 270
type() 91

U

Übergänge
– jQuery Mobile 438
ui-state-disabled 422
Ummanteln 177
unbind() 283
– deprecated 57
undelegate() 297
– deprecated 57
uniqueSort() 91
unload() 282
– deprecated 57
unwrap() 182
url 362
URL-Kodierung 351
userAgent 277
username 362

V

val() 156
Variablen 495
verbundene Knoten 270
Verkettung 305
Versatz 228
Version
– JavaScript 49
Versprechen 268
Verwandtschaftsbeziehungen
– DOM 94
verzögertes Objekt 372
viewport 434
VirtualBox 15
visible 121
Visual Studio 21
VMWare Player 15
Vorlagen 251

W

Warteschlange 306
– jQuery 41, 67
Web 2.0 46, 325
Web Tools Platform Project 17
Web Worker 367
Webformular
– Validierung 458
Webseite
– adaptive 489
Webserver 25
Wegblenden 309
when() 68
– Deferred Object 377

which
– Event 270
Widgets 390
– jQuery UI 389
width() 241, 255
window.onerror 279
window.setTimeout() 303
Worker 368
wrap() 177
wrapAll() 180
wrapInner() 181
WTP 17
Wurzel 97

X

XAMPP 25, 337
xhr 362
XHR 326
xhrFields 362
XHR-Objekt
– Eigenschaften 328
– Methoden 327
XML Path Language 96
XMLHttpRequest 326
XPath 96
xUnit 13
xUnit-Testing 482
– Ergebnis 482

Z

Zugangsverifizierung 336
Zusicherungen 268