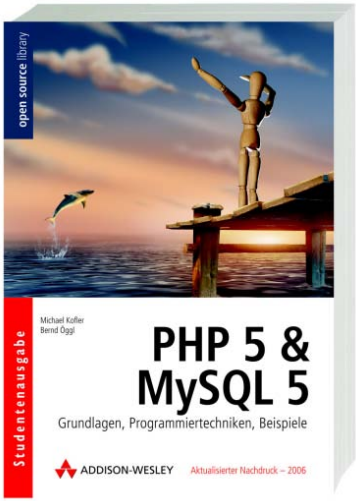
AJAX
mit Java-Servlets und JSP

open source library

Open Source Software wird gegenüber kommerziellen Lösungen immer wichtiger. Addison-Wesley trägt dieser Entwicklung Rechnung mit den Büchern der **Open Source Library**. Administratoren, Entwickler und User erhalten hier professionelles Know-how, um freie Software effizient einzusetzen. Behandelt werden sowohl Themen wie Betriebssysteme, Netzwerke und Sicherheit als auch Programmierung.

Eine Auswahl aus unserem Programm:



Dieses praxisorientierte Buch liefert nach einem kurzen Grundlagenteil eine ganze Sammlung von PHP- und MySQL-Rezepten: objektorientierte Programmierung mit PHP 5, XML-Funktionen, prepared statements, stored procedures, SQL-Grundlagen und -Rezepte, GIS-Funktionen, mysqli-Schnittstelle etc. Anschließend demonstrieren mehrere umfangreiche Beispielprojekte das Zusammenspiel von PHP und MySQL. Ein Kapitel über TYPO3 zeigt exemplarisch, wie im Internet kostenlos verfügbare PHP/MySQL-Projekte installiert und für eigene Zwecke adaptiert werden.

PHP 5 & MySQL 5
Michael Kofler, Bernd Öggl
696 S.
Euro 39,95 (D), 41,10 (A)
ISBN-13: 978-3-8273-2392-7
ISBN-10: 3-8273-2392-4

AJAX steht für Asynchronous Javascript and XML. Basis sind HTML/XHTML und CSS, das Document Object Model, XML und XSLT, JavaScript sowie das neue XMLHttpRequest-Objekt, um Daten auf asynchroner Basis mit dem Web-Server austauschen zu können. AJAX ist ein Einsteigerbuch der neueren Generation: Editieren und Programmierung werden in einem Zug gelehrt. Von Ihnen wird dabei nur etwas Vorwissen in HTML und JavaScript verlangt – nach der Dosis der über 350 Seiten, von denen keine verzichtbar ist, werden Sie sich aber wahrscheinlich kaum noch wiedererkennen: Dieses Wissens-Paket macht auch aus Ihnen einen gewandten Meister der Erstellung schneller Webapplikationen, aus denen sich das neue Web 2 rekrutiert. AJAX ist kein »Basteln Sie sich schnell Ihre eigene Homepage«-Buch, sondern eine Anleitung zum tieferen Hineintauchen in die Zukunft des Internets. Und den Einstieg gestaltet Ralph Steyer auf eine sehr angenehme Weise mit guten Erklärungen.

AJAX mit PHP
Ralph Steyer
312 S.
Euro 24,95 (D), 25,70 (A)
ISBN-13: 978-3-8273-2358-3
ISBN-10: 3-8273-2358-4

PEARSON Education

Ralph Steyer

ADDISON-WESLEY

open source library

AJAX

mit Java-Servlets und JSP

So bringen Sie Speed in Ihre Webpräsenz

ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Die Deutsche Bibliothek – CIP-Einheitsaufnahme

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Umwelthinweis:
Dieses Produkt wurde auf chlorfrei gebleichtem Papier gedruckt.

10 9 8 7 6 5 4 3 2 1

08 07 06

ISBN-13: 978-3-8273-2418-4
ISBN-10: 3-8273-2418-1

ein Imprint der Pearson Education Deutschland GmbH
Martin-Kollar-Straße 10–12, D-81829 München/Germany

Einbandgestaltung: Marco Lindenbeck, webwo GmbH (mlindenbeck@webwo.de)
Lektorat: Brigitte Alexandra Bauer-Schiewek, bbauer@pearson.de
Fachlektorat: Dirk Frischalowski
Korrektorat: Petra Kienle
Herstellung: Monika Weiher, mweiher@pearson.de
Satz: reemers publishing services gmbh, Krefeld
Druck: Kösel, Krugzell (www.koeselbuch.de)
Printed in Germany

Inhaltsübersicht

Inhaltsverzeichnis

Vorwort

AJAX! Ein Schlagwort macht im Web Karriere. Jedoch nicht als Name eines griechischen Heerführers im Trojanischen Krieg, nicht als niederländischer Fußballverein und schon gar nicht als Haushaltsreiniger. Die derzeit heißeste Bedeutung von AJAX steht für die Abkürzung von **Asynchronous JavaScript and XML**. Als ich mich Mitte 2005 auf mein erstes Buch zu AJAX einließ, hoffte ich natürlich, dass das Thema erfolgreich sein würde. Viele Anzeichen deuteten auch darauf hin. Was dann aber bei Erscheinen des Buchs **AJAX mit PHP** zur CeBIT 2006 an Resonanz kam, übertraf die Erwartungen. Es sieht so aus, als ob die Internetgemeinde nur auf AJAX gewartet hätte und es seit Jahren wieder eine Entwicklung gibt, die das World Wide Web wirklich voranbringt.

Wie die ausgeschriebene Version von AJAX verdeutlicht, besteht die Basis dieser Technologie aus JavaScript und XML. Aber ebenso stellen HTML/XHTML und CSS oder XSLT, das Document Object Model sowie das neue `XMLHttpRequest`-Objekt zum Austausch von Daten zwischen einem Webbrowser und einem Webserver auf asynchroner Basis Kernelemente dar. Dazu benötigen Sie noch eine – weitgehend beliebige – Programmiertechnologie auf dem Server (etwa PHP oder ein auf Java aufsetzendes Verfahren wie JSP, Java Servlets oder EJBs), eventuell ergänzende Programme auf dem Server wie ein Datenbankmanagementsystem und schon ist AJAX zum Umkrempeln bekannter Webgewohnheiten bereit.

Mit den aufgeführten Schlagwörtern werden zwar im Wesentlichen Technologien beschrieben, die im WWW bereits seit geraumer Zeit etabliert sind. Dennoch scheint erst die Verbindung unter der Losung AJAX für eine breite Anwenderschaft hochinteressant zu sein. Ob Kritiker nun bemängeln, alleine der griffige Name mache die wirkliche Neuerung von AJAX aus (manche giften geradezu, es handele sich nur um alten Wein in neuen Schläuchen) oder tatsächlich echte Neuigkeiten damit verbunden sind, ist vollkommen gleichgültig. Möglicherweise ist die Verbindung bekannter Techniken sogar der Hauptvorteil für einen rasanten Durchbruch, wie ihn AJAX gerade auf breiter Front erlebt. Denn eine Etablierung von AJAX setzt nur wenige vollständig neue Techniken voraus. AJAX kann deshalb ohne großen Aufwand umgesetzt werden. Durch die breite Resonanz in Medien und im Web fokussiert sich ein Standard, auf den sich in kurzer Zeit alle Beteiligten der Webprogrammierung und Webdarstellung einigen müssen und werden. Die große Resonanz auf AJAX zwingt die Beteiligten im Web (sowohl Browser-Hersteller als auch Webseitenersteller) buchstäblich zu ihrem Glück.

Aber was ist eigentlich der Kick für das Web, der diese AJAX-Welle auslöst? Mit einem auf breiter Front unterstützten AJAX beginnt sich endlich ein standardisiertes Konzept der Datenübertragung zwischen einem Server und dem Browser zu etablieren, bei dem eine Webseite nicht bei jeder HTTP-Anfrage komplett wieder neu geladen werden muss. Nur gewisse Teile einer HTML-Seite werden bei Bedarf sukzessive nachgeladen. Und das bedeutet **Speed**! Lange Wartezeiten bei einfachen Änderungen in einer Webseite, die durch eine Anwenderaktion auf einer bereits geladenen Webseite ausgelöst werden, können in vielen Situationen erheblich reduziert werden. AJAX erlaubt in Zukunft die Erstellung von Webpräsenzen, die mit einem Anwender nahezu in Echtzeit interagieren können. Auch wenn vorher noch nicht alle Daten beim Client vorliegen. Es lassen sich mit AJAX endlich auch im Web interaktive, Desktop-ähnliche Anwendungen realisieren. Und darauf haben die Webanwender lange warten müssen. Wenn auch Sie solche Highspeed-Webanwendungen erstellen wollen, folgen Sie mir in die faszinierende Welt von AJAX.

Eppstein, im Sommer 2006

Ralph Steyer

www.rjs.de und www.ajax-net.de

1 Einleitung

AJAX revolutioniert das WWW. Und Sie sind mit von der Partie. Ich auch. Und ich habe Spaß dabei. Den werden auch Sie haben, denn AJAX bringt so viel frischen Wind in die Webentwicklung, dass ich – als derzeit noch ziemlicher unsicherer Anfänger im Paragliden – mit meinem Gleitschirm sicher nicht starten würde.

1.1 Wer ich bin

Mein Name ist Ralph Steyer. Ich bin Diplom-Mathematiker und dennoch ganz sportlich. In der Regel freue ich mich, wenn mir jemand sagt, ich sähe nicht wie ein Mathematiker aus. Aber das liegt wohl daran, dass die meisten Leute falsche Vorstellungen von Mathematikern haben ;-). Von Leichtathletik über Rasenkraftsport[1], Mountainbiken, Motorradfahren bis zum Paragliding mache ich in meiner Freizeit Dinge, zu denen jüngere Leute vor lauter Gameboy- und Computerspielen kaum kommen[2]. Nach dem Studium habe ich einige Jahre bei einer großen Versicherung im Rhein-Main-Gebiet als Programmierer gearbeitet. Seit 1995 verdiene ich meinen Lebensunterhalt als Freelancer, wobei ich fliegend zwischen der Arbeit als Fachautor, Fachjournalist, EDV-Dozent und Programmierer wechsle. Das macht aus meiner Sicht einen guten Mix aus, bewahrt vor Langeweile und hält mich sowohl in der Praxis als auch am Puls der Entwicklung. Insbesondere habe ich sowohl das Vergnügen als auch die Last, mich permanent über neue Entwicklungen in der EDV 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. 1995 konnte ich mein erstes HTML-Buch zum gerade aufkeimenden Boom des WWW schreiben und bereits 1997 zu Java. Und AJAX ist gerade wieder ein Highlight, das äußerst anregend ist. Wie ich bereits im Vorwort erwähnt habe, ist dies nicht das erste Buch zum Thema AJAX. Es gibt bereits ein Schwesterbuch in dieser Reihe, das sich allerdings auf Serverseite auf PHP konzentriert. In diesem Buch werden wir dagegen Java als Basis nehmen und auch etwas mehr den Fokus auf die Serverseite legen. Die Resonanz auf das Schwesterbuch war so großartig, dass ich Mitte März 2006 ein Portal

1 *Was zur Hölle ist das? Die Frage höre ich immer wieder. Aufgabe: Schauen Sie im Internet nach, was das ist ;-).*

2 *Und ich wegen meiner Familie und nicht zuletzt der unangenehmen Notwendigkeit des Geldverdienens natürlich auch nicht permanent ;-).*

zu AJAX gegründet habe. Sie erreichen es unter *http://www.ajax-net.de*. Dort finden Sie sowohl die Quelltexte zum vorliegenden Buch als auch zum Schwesterbuch sowie aktuelle Neuigkeiten und viele andere aktuelle Informationen zum Thema AJAX.

1.2 Wer sind Sie?

Sie haben jetzt den Vorteil, dass Sie nun etwas von mir wissen. Ich kenne Sie natürlich nicht. Und sowohl die privaten als auch die beruflichen Details werden mit absoluter Sicherheit ziemlich differieren. Aber ich habe eine gewisse Vorstellung, wer sich warum mit AJAX beschäftigen wird und dieses Buch lesen könnte. Und wie Sie sich von der Zielgruppe unterscheiden könnten, die AJAX mit PHP betreiben möchte. Ich denke, Sie haben bereits Webseiten erstellt und sind irgendwann den Einschränkungen einer statischen HTML-Seite überdrüssig geworden. Vielleicht haben Sie bereist erste Erfahrungen mit dynamisch generierten Webseiten und/oder Content Management Systemen (CMS)[3] sowie per JavaScript und Style Sheets optimierten Seiten. Möglicherweise haben Sie auch schon auf dem Server programmiert, aber das muss nicht unbedingt der Fall sein. Ich gehe jedoch davon aus, dass Sie im Land von Java und der objektorientierten Programmierung kein absoluter Neuling sind.

Nun haben Sie wahrscheinlich Webanwendungen gesehen oder davon gehört, die ein Antwortverhalten quasi wie Desktop-Anwendungen aufweisen. Schlagwörter wie das **Web 2.0**, **semantisches Web** oder **interaktives Web** machen Sie neugierig. Statt dem Anwender nur ein passives Verhalten (maximal mit reiner Anforderung von einer neuen Webseite oder einer Formulareingabe) zuzuordnen, wird sich das Web immer mehr zu einem bidirektionalen (oder vielleicht multidirektionalen) Medium entwickeln, bei dem der Besucher von Webseiten bezüglich der dargestellten Inhalte immer stärker eingebunden wird. Sei es ein Wiki, bei dem ein jeder Anwender zum Inhalt beitragen kann, sei es ein Blog, bei dem Betreiber ohne großes technischen Hintergrundwissen seinen Teil zum weltweiten Wissen beiträgt, sei es ein Dienst, bei dem Inhalte optimal an jeden Besucher angepasst werden (unter Umständen in Echtzeit auf Aktivitäten des Anwenders reagierend). Ebenso wird das WWW immer mehr strukturiert und die Vermischung von Information, Funktionalität und Layout aufgebrochen, was eine viel bessere Informationssuche gestattet. Und ich vermute, da wollen Sie dabei sein. Sie wollen den Zug erwischen, der in die Zukunft des Internets fährt.

1.3 Was behandelt das Buch?

Das vorliegende Buch soll Ihnen den Einstieg in die Webprogrammierung mit AJAX ermöglichen. Dazu werden im Laufe des Buchs alle relevanten Techniken samt vieler praktischer Beispiele durchgespielt. Das Buch wendet sich im Wesentlichen an Web-

3 *Tipp: Ein sehr interessantes, einfaches und dennoch leistungsfähiges OpenSource-CMS ist Joomla!. Damit wird auch das AJAX-Portal AJAX-NET.de betrieben.*

seitenersteller, Webdesigner und Webprogrammierer, die schon Erfahrung mit (X)HTML haben und zumindest Grundlagen von CSS und JavaScript bereits beherrschen. Erfahrungen mit XML und Kenntnisse von serverseitiger Programmierung werden nicht vorausgesetzt, schaden aber selbstverständlich nicht. Im Schwesterbuch **AJAX mit PHP** werden jedoch auch explizit Leser angesprochen, die grundsätzlich in der Programmierung (also auch in JavaScript und serverseitigen Programmiertechniken wie PHP) nicht so geübt sind. Dieses Buch legt die Latte geringfügig höher. Sie haben hier zwar ein Buch für Einsteiger in die AJAX-Welt vorliegen, das nicht Freaks und Programmierprofis als Leser im Auge hat. Allerdings kann man nur schwerlich Java als serverseitige Technologie verwenden, wenn Sie keinerlei Erfahrungen der objektorientierten Programmierung (OOP) und/oder Java haben. Deshalb werden gewisse einfache Grundkenntnisse in der OOP und Java vorausgesetzt. Oder zumindest wird von Ihnen als Leser erwartet, dass Sie den knapp gehaltenen Erklärungen zu den Java-Grundlagen folgen können und sich bei Bedarf in spezieller Java-Literatur zu Details informieren. Sie müssen wirklich kein Profi in Java sein, aber es hilft für das Verständnis unseres Kapitels zu JSP und Servlets doch sehr, wenn Sie bereits ein wenig Erfahrung mit Java haben. Dennoch brauchen Sie jetzt keine Angst zu haben, denn es handelt sich trotzdem wie gesagt ausdrücklich um ein Einsteigerbuch.

Im ersten Kapitel nach dieser Einleitung erfahren Sie zunächst, wozu AJAX eigentlich da ist. Welche Gründe gibt es für AJAX und dessen Erfolg? Warum und wie löst AJAX Probleme der konventionellen Webprogrammierung? Dazu soll bereits in einer sehr frühen Phase und ohne lange Vorrede ein erstes praktisches Beispiel durchgespielt werden, welches AJAX einsetzt. Die konkreten Hintergründe erarbeiten wir dann Schritt für Schritt. In den folgenden Kapiteln werden dazu elementare Grundlagen behandelt, die für einen erfolgreichen Einstieg in die Erstellung von AJAX-Applikationen notwendig sind. Dieses Grundwissen für AJAX-Entwickler umfasst (X)HTML, Style Sheets allgemein und CSS im Besonderen, JavaScript, DOM, XML sowie serverseitige Programmierung samt Datenbankzugriff. Denn AJAX bezeichnet wie schon erwähnt im Wesentlichen den Zusammenschluss dieser etablierten Technologien aus dem Webumfeld. Diese müssen Sie als AJAX-Entwickler – zumindest in Grundzügen – beherrschen. In den Kapiteln 3 bis 9 behandeln wir das Grundwissen zu diesen Einzeltechnologien, soweit Sie dieses für AJAX benötigen. Dies umfasst folgende Einzelthemen:

- Grundlagen zu (X)HTML samt HTML-Eventhandlern und DOM (Document Object Model) aus Sicht von HTML
- Grundlagen zu Style Sheets (im Wesentlichen CSS – Cascading Style Sheets)
- Vertiefende Einblicke in JavaScript und DOM
- Grundlagen zu HTTP und XMLHttpRequest sowie deren genaue Details, soweit sie für AJAX unumgänglich sind
- Wichtige Techniken zu DHTML (Dynamic HTML)
- Grundlagen zu XML (Extensible Markup Language) sowie ein Überblick zu XPath

- Serverseitige Programmierung im Allgemeinen und Java (JSP und Servlets) im Besonderen. Dazu besprechen wir den Umgang mit Webdatenbanken anhand von MySQL.

Bei der Vielzahl der Themen ist offensichtlich, dass wir die Quadratur des Kreises versuchen müssen. Auf der einen Seite ist klar, dass in einem recht kompakten AJAX-Buch keine vollständige Behandlung aller Details zu den jeweiligen Einzelthemen erfolgen kann. Sie sollen mit diesem Buch kein HTML, CSS oder JavaScript von Grund auf lernen. Über die verschiedenen Kapitel zu den Grundlagentechniken wird versucht, so kompakt wie möglich und so ausführlich wie nötig die jeweiligen Fundamente zu behandeln, wie Sie sie – ausgehend von den beschriebenen Vorkenntnissen – zur Programmierung von AJAX-Anwendungen benötigen. Dabei legen wir in jedem Kapitel den Schwerpunkt auf die Fakten, die bei der jeweiligen Technologie für den AJAX-Blickwinkel hauptsächlich von Interesse sind. Mit anderen Worten – auch wenn Sie eine der Technologien schon beherrschen, lohnt sich ein Blick in den Abschnitt, um die spezifischen Abhandlungen zum AJAX-Bezug nicht zu verpassen. Zumal in den jeweiligen Kapiteln auch Beispiele zu AJAX aus Sicht der jeweiligen Technologie vorgestellt werden. Auf der anderen Seite wird für bestimmte weitergehende Details auf weiterführende Spezialliteratur verwiesen werden müssen.

Im vorletzten Kapitel sollen Ihnen dann noch verschiedene Lösungen zur asynchronen Datenübertragung vorgestellt werden, deren Ansätze es schon einige Zeit gibt und die letztendlich in AJAX münden. Dies umfasst den Umgang mit Frames und IFrames zum Nachladen von Daten, Cookies zur Verfolgung von Websitzungen, Vorratshaltung von Daten mit JavaScript-Objekten und auch eine spezifische Mircosoft-Lösung mit so genannten XML-Dateninseln in Webseiten. Zum Abschluss betrachten wir noch sehr kompakt ein paar Frameworks für AJAX, um das Buch mit einem Anhang und einem umfangreichen Index zu beschließen.

1.4 Schreibkonventionen

In diesem Buch finden Sie verschiedene Schreibkonventionen, die Ihnen helfen sollen, die Übersicht zu bewahren. Wichtige Begriffe werden **hervorgehoben**. Vor allem sollten Sie erkennen können, ob es sich um normalen Text oder `Programmcode` handelt. Ebenso werden Tasten (Alt, Strg oder Shift) und noch einige weitere Besonderheiten gekennzeichnet. Diese Formatierungen werden konsequent verwendet. Und ebenso finden Sie im Buch Bereiche, die über die Markierung mit verschiedenen Symbolen besondere Aufmerksamkeit erzeugen sollen.

Hinweis

Das ist ein besonderer Hinweis, den Sie an der Stelle beachten sollten.

Tipp

Das ist ein Tipp, der Ratschläge oder besondere Tricks zur jeweiligen Situation zeigt.

Achtung

Hier droht Gefahr.

1.5 Was benötigen Sie?

Sie benötigen natürlich zu einer erfolgreichen Arbeit mit dem Buch einen Computer. An diesen werden ebenso wie an das **Betriebssystem** jedoch keine besonderen Anforderungen gestellt, solange beide halbwegs modern sind. Das Betriebssystem sollte ein 32-Bit-System oder besser sein. Empfehlenswerte Betriebssysteme sind alle grafischen Systeme wie Linux, Windows oder MacOS. Die konkrete Wahl spielt – wie meist im Web – keine Rolle. Sie sollten allerdings einen Internetanschluss zur Verfügung haben, denn wir behandeln hier ja mit AJAX eine Internettechnik. Und für den Einsatz »in the wild« benötigen Sie in jedem Fall Zugang zum Internet. Ebenso werden viele Quellen auf das Internet verweisen. Da wir uns auf Serverseite im Java-Umfeld bewegen, benötigen Sie eine virtuelle Maschine und die entsprechenden Java-Pakete. Allerdings erhalten Sie diese entweder mit einer Java-Entwicklungsumgebung oder

Sie haben sie automatisch mit Ihrem Betriebssystem installiert oder mit einem entsprechenden Server, der als Java-Container fungiert (siehe unten). Auf diese Details gehen wir allerdings noch genauer ein.

Hinweis

Da wir uns in diesem Buch mit Java-Programmierung auf Serverseite auseinander setzen wollen, müssen bei Ihnen einige Voraussetzungen für die erfolgreiche Programmierung mit Java erfüllt sein. Um diese Details kümmern wir uns allerdings erst an den passenden Stellen.

Ansonsten brauchen Sie noch einen Editor, mit dem Sie gut umgehen können. Ein solcher wird bei jedem Betriebssystem mitgeliefert, etwa Notepad unter Windows oder Kedit unter Linux. Wenn Sie unterstützende Programme für die Erstellung von Style Sheets, HTML, XML, Java oder JavaScript verwenden wollen, ist das natürlich bequemer (aber wie gesagt nicht notwendig).

Was Sie freilich zwingend brauchen, ist mindestens ein Webbrowser, der AJAX unterstützt. Dies sind derzeit unter anderem alle Browser, die zum aktuellen Microsoft Internet Explorer[4] kompatibel sind, Opera 7.6 und höher, für Linux-Anwender neuere Versionen des Konqueror oder Nautilus, auf einer aktuellen Gecko-Engine basierende Browser wie Firefox, Mozilla oder Netscape ab der Version 7.1 und für Apple-Anwender Safari 1.2 und höher. Sie sollten **auf jeden Fall** sogar mehrere Browser zur Verfügung haben, denn leider gilt auch unter AJAX, dass man darauf basierende Webapplikationen **unbedingt** in allen relevanten Browsern testen **muss**. Sie können ja schon bei einer etwas anspruchsvolleren normalen Webapplikation niemals sicher sein, wie diese in verschiedenen Browsern aussieht bzw. sich verhält. Es gibt auch heutzutage zahlreiche Sonderfälle, die Sie testen müssen. Und in der Regel wissen Sie nicht, mit welchen Browsern Ihre »Kunden« arbeiten. Für Ihre Tests sind sowohl Browser von verschiedenen Herstellern notwendig als auch verschiedene Versionen eines Browsers (ältere sowie die derzeit topaktuellen Browser gehören für Webdesigner und Webprogrammierer zur Pflichtausstattung).

4 *Im Grunde sogar bereits ab der Version 4.0.*

Tipp

Sollten Sie Windows-Anwender sein, sollten Sie sich auf jeden Fall Linux als weiteres Betriebssystem zulegen. Es ist interessant zu sehen, wie sich verschiedene Webapplikationen unter verschiedenen Betriebssystemen verhalten. Nun brauchen Sie keine Angst zu haben, dass sie parallel zu Ihrem Betriebssystem ein weiteres Betriebssystem installieren oder sich gar einen neuen Rechner kaufen müssen. Es gibt hervorragende Live-CDs, von denen Sie Linux direkt starten können, ohne dass auf Ihrer Festplatte irgendeine Änderung durchgeführt wird. Ein ganz hervorragendes Live-Linux ist **Knoppix**, was Sie sich ohne Schwierigkeiten aus dem Internet laden können. Die Knoppix-CD enthält u.a. den Konqueror als Webbrowser sowie einige weitere Browser, deren Gegenstück Sie unter Windows oder MacOS wahrscheinlich kennen.

AJAX basiert darauf, dass Daten von einem Webserver angefordert werden. Deshalb ist es für die praktische Erstellung von AJAX-Applikationen unabdingbar, dass Sie entweder Zugang zu einem Webserver im Internet haben (inklusive der Möglichkeit zum Datentransfer – meist mit FTP) und dort Programme oder Skripte ausführen können. Das wird letztendlich auch für ein AJAX-Projekt in der Praxis notwendig sein. Allerdings ist es in der Praxis ebenso unüblich, schon bei der Entwicklung direkt auf einem Webserver im Internet zu arbeiten – vor allem, wenn Sie nur Dinge testen wollen. Sie sollten sich also eine Testumgebung mit einem Webserver auf einem lokalen Rechner oder in einem lokalen Netzwerk schaffen.

Tipp

Als Webserver bietet sich ein Rundum-Sorglos-Paket wie **XAMPP** an, das Sie für verschiedene Betriebssysteme aus dem Internet laden können (unter *http://www.apachefriends.org/de/*)[5]. Dieses Paket bezeichnet eine Sammlung an Programmen mit dem Webserver **Apache** samt **MySQL** und **PHP**-Unterstützung sowie einigen weiteren Webtechnologien. Sie brauchen dieses Paket nur mit einem einfachen Assistenten zu installieren und schon haben Sie einen voll funktionstüchtigen Webserver in einer Grundkonfiguration zur Verfügung. Beachten Sie aber, dass diese Pakete in der Grundeinstellung ausschließlich für lokale Testzwecke konfiguriert sind. Um die Sache möglichst einfach zu halten, sind in der Grundeinstellung sämtliche Sicherheitseinstellungen extrem niedrig eingestellt. Da wir allerdings auf Java als serverseitige Technologie in diesen Buch setzen, benötigen wir ein Serverprogramm,

5 *Hinter XAMPP steckt ein Non-Profit-Projekt namens Apache Friends, das sich die Förderung des Apache Webservers und verbundener Technologien wie MySQL, PHP und Perl auf die Fahne geschrieben hat.*

das Java Servlets oder Java Server Pages ausführen kann. Der Referenzserver schlechthin ist **Tomcat**, der im Rahmen des Apache-Projekts vorangetrieben wird und den reinen Apache-Webserver um die Fähigkeit der Ausführung von serverseitigen Java-Applikationen ergänzt. Im nächsten Kapitel besprechen wir die Einrichtung von Tomcat sowie XAMPP, um eine Testumgebung für die ersten Beispiele aufzubauen. Natürlich können Sie auch andere Webserver wie den **IIS** (**Internet Information Server**) von Microsoft oder **Sambar** verwenden, wenn Sie eine andere serverseitige Technologie einsetzen wollen. Der Webserver muss nur die von Ihnen favorisierte serverseitige Programmiersprache unterstützen.

Da wir in diesem Buch Java als serverseitige Technologie verwenden, bietet es sich an, dass Sie eine IDE für Java zur Verfügung haben. Meine Empfehlung ist da **Eclipse**, aber es gibt natürlich noch eine ganze Reihe weiterer sehr guter Java-IDEs. Auf sämtliche Details rund um Java, Tomcat und Eclipse gehen wir ausführlich in dem entsprechenden Kapitel ein. Für den Anfang können wir dieses Thema hinten anstellen.

2 AJAX stellt sich vor

Sie interessieren sich offensichtlich für AJAX, denn sonst würden Sie diese Zeilen hier nicht lesen. Aber warum befinden Sie sich damit zurzeit in so großer und illustrer Gesellschaft? Wenn Sie AJAX und dessen immense Bedeutung für die Zukunft des Webs näher kommen wollen, helfen Ihnen die Arbeitsweise und vor allem die Probleme der konventionellen Webprogrammierung beim Verständnis. Und wie AJAX Ihnen und dem Web hilft, die bisherigen Probleme im Web zu lösen.

2.1 Wozu AJAX?

Als das WWW 1990 aufkam, war dies ein riesiger Schritt, um aus dem bis dato rein textbasierenden Internet mit isolierten Daten eine multimediale Welt zu gestalten, die Inhalte miteinander verknüpft. Anwendern wurden mit einem Male Inhalte optisch interessant aufbereitet und verfügbar gemacht, ohne dass sie deren genauen Ort wissen mussten. Stattdessen konnten die Personen, die Inhalte bereitstellten, Verknüpfungen in Form von Hyperlinks bereitstellen, die ein Anwender nur anklicken musste. Sie kennen das natürlich alles.

2.1.1 Der Datentransfer per Paketvermittlung

Der tatsächliche Austausch von Daten zwischen einem anfordernden Client und einem Server erfolgt im WWW genau wie im restlichen Internet. Die gesamte Internetkommunikation basiert auf der Datenübertragung in Form einer so genannten **Paketvermittlung**, die mit TCP/IP[1] (Transmission Control Protocol/Internet Protocol) als **Transportprotokoll** realisiert wird. Bei einer Paketvermittlung werden alle zu übertragenden Daten zwischen zwei Kommunikationspartnern in kleinere Datenpakete aufgeteilt. Diese agieren als abgeschlossene und vollständige Transporteinheiten, die unabhängig voneinander vom Sender zum Empfänger gelangen. Es werden also einfach Datenpakete hin und her verschickt, die nur aufgrund von Adressangaben (IP-Nummern) und einiger ergänzenden Informationen wie einem Port (eine Art Sendekanal) zugeordnet werden. Dabei wird während der Kommunikation keine Verbindung im eigentlichen Sinn zwischen dem Server und dem Client aufgebaut

1 *Als Familie zu verstehen (auf die exakte Beschreibung und Unterscheidung zu UDP – User Datagram Protocol – etc. soll verzichtet werden, da sie für AJAX nicht interessant ist).*

oder gehalten. Bei einer Paketvermittlung unterscheidet man dennoch allgemein zwischen einer **verbindungsorientierten** und einer **nichtverbindungsorientierten** (oder auch **zustandslosen**) Form. Die konkrete Kommunikationsform wird jedoch nicht auf der Transportebene, sondern jeweils mit spezifischen Protokollen realisiert, die auf TCP/IP oder anderen Transportprotokollen aufsetzen. Solche höheren Protokolle sind etwa **Telnet**, **FTP** (**File Transfer Protocol**) oder **HTTP** (**Hyper Text Transfer Protocol**), wie es im WWW eingesetzt wird. Diese auf dem eigentlichen Transportprotokoll aufsetzenden Protokolle werden **Dienstprotokolle** genannt.

Verbindungsorientierte Datenkommunikation

Bei einer verbindungsorientierten Datenkommunikation wird auf Basis der Datenpakete in der Ebene der Dienstprotokolle zwischen Sender und Empfänger eine temporäre virtuelle/logische Verbindung aufgebaut. Das bedeutet, obwohl rein physikalisch auf Ebene des Transportprotokolls keine dauerhafte Verbindung zwischen den Kommunikationspartnern vorhanden ist, wird über ein darauf aufsetzendes Dienstprotokoll ein Mechanismus bereitgestellt, um eine dauerhafte Verbindung zu gewährleisten. Das ist zum Beispiel bei einer Echtzeitkommunikation wie **VoIP** (**Voice over IP** – Internettelefonie) oder **Chatten** der Fall. Aber auch beim Fernsteuern von einem Remote-Rechner (zum Beispiel mit **Telnet** oder **SSH** – **Secure Shell**).

Zustandslose Datenkommunikation

Bei der nichtverbindungsorientierten Form der Datenübertragung sendet man einfach Datenpakete mit allen notwendigen Informationen zum Empfänger und kümmert sich nicht um eine virtuelle Verbindung. Jede Anfrage durch einen Client an einen Server erzeugt dort genau eine Sendung einer Antwort, die durchaus in viele Einzelpakete zerlegt sein kann, die erst beim Empfänger wieder zusammengefügt werden. Danach »vergessen« beide Kommunikationspartner die Verbindung. Das zur Kommunikation von einem Webserver und einem Webclient (in der Regel ein Webbrowser) entwickelte Protokoll HTTP ist nun genau so ein zustandloses Protokoll. Es erlaubt einen einfachen Datenaustausch zwischen den beteiligten Parteien, bei dem der Client Daten vom Server anfordert und als Antwort vom Server eine Webseite gesendet bekommt[2]. Aber es wird keine Verbindung zwischen Webserver und Webbrowser aufrechtgehalten. Damit kann der Webserver beispielsweise bei einer zweiten Anfrage durch einen Webbrowser nicht erkennen, ob dieser bereits vorher eine Anfrage geschickt hat[3]. Ein Szenario wäre, dass sich ein Anwender bei einem Webserver mit seinen Zugangsdaten anmeldet und dann eine geschützte Seite angezeigt bekommt. Fordert er nun vom gleichen Webserver wieder eine neue geschützte Seite an, kann der Webserver rein auf Basis von HTTP nicht erkennen, ob der Anwender sich bereits vorher legitimiert hat. Natürlich wissen Sie aus Ihrer Erfahrung im Web, dass eine Anwendersitzung durchaus verfolgt werden kann und Sie nach einer

2 *Jede Anforderung (der so genannte **Request**) bewirkt als Antwort das Senden einer vollständigen neuen Webseite (**Response**).*

3 *Zumindest nicht rein auf Basis des HTTP-Protokolls.*

Anmeldung in einem geschützten Bereich nicht bei jeder Anforderung einer neuen Seite erneut Ihre Zugangsdaten eingeben müssen. Das wird aber nicht so einfach von HTTP gewährleistet, sondern durch die Webapplikation, die dazu verschiedene Techniken einsetzen kann[4].

Anfordern und Senden

Kommen wir zurück auf den reinen Zyklus »Anfordern – Senden der Webseite«. In der Anfangsphase des Webs war so ein Anfordern einer Webseite durch einen Client und deren Darstellung nach Erhalt der Antwort vollkommen ausreichend. Aber bereits nach wenigen Jahren genügten vielen Leuten rein statische Webseiten nicht mehr. Insbesondere die mangelnden Möglichkeiten der Interaktion mit einem Betrachter einer Webseite stellten ein Dilemma dar. Aus Ihrer Erfahrung im Web ist Ihnen sicher klar, dass jede Eingabe einer neuen Webadresse in der Adresszeile des Browsers oder der Klick auf einen Hyperlink mit einer Verknüpfung zu einer anderen Webseite als Anforderung an einen Webserver geschickt wird, dem Browser diese neue Seite zu schicken. Der Besucher will ja in dieser Situation bewusst eine neue Information haben, die bis dato noch nicht auf seinem Rechner verfügbar ist.

Aber auch jede beliebige andere Anwenderaktion, auf die in irgendeiner Form reagiert werden muss, ergab zu dieser Zeit eine Anforderung an einen Webserver, eine neue Webseite zu schicken. Nehmen wir folgende Situation: Ein Anwender soll in einem Eingabefeld eines Webformulars seine Telefonnummer eingeben. Er verwechselt dabei bei der Eingabe die Zahl 0 mit dem Buchstaben O. Alleine die primitive Plausibilisierung dieser einen Eingabe in einem einzelnen Feld erfordert das Versenden der kompletten Formulareingaben an den Webserver und die Rücksendung einer vollständigen neuen Webseite, in der zu einer Fehlermeldung zudem alle anderen bisherigen Eingaben des Anwenders erhalten bleiben sollten – zumindest sofern diese korrekt waren. Allgemein stellte zu dieser Zeit jede Reaktion auf eine Anwenderaktion einen solchen Zyklus aus der Übermittelung von Formulardaten an einen Webserver, Verarbeitung auf dem Webserver und der Sendung einer vollständigen Webseite als Antwort dar. Aufgrund der Tatsache, dass Daten unter Umständen mehrfach um die Erde kreisen und ein Webserver also bei jeder Anfrage durch einen Nutzer eine vollständig neue Webseite erzeugen und übermitteln muss, ist das Verhalten einer typischen Webanwendung sehr träge und zudem sehr belastend für den Webserver sowie die Kommunikationswege.

2.1.2 Clientseitige Webprogrammierung als erste Optimierung

Um der mangelnden Performance von Webanwendungen bei gleichzeitiger Belastung der Kommunikationswege und des Servers sowie dem Brachliegen der Client-Ressourcen zu begegnen, entstanden etwa ab 1995 verschiedene clientseitige Programmiertechniken. Auf der einen Seite waren es Skriptsprachen wie **JavaScript** oder **VBScript**, die im Webbrowser ausgeführt werden. Auf der anderen Seite entwickel-

4 *Unter anderem auch die Erweiterung von HTTP um spezifische Header-Felder.*

ten verschiedene Firmen Möglichkeiten, vollständige Programme in eine Webseite zu implementieren, die im Rahmen eines Webbrowsers ausgeführt werden konnten – etwa in Form von **Java-Applets** oder **ActiveX-Controls**. Dazu kamen vielfältige Multimediatechniken. Die Hauptfunktion von allen clientseitigen Programmiertechniken war eine Verlagerung von der Funktionalität vom Server auf den Client, die dort bereits erledigt werden kann. Denken wir noch einmal an Daten, die auf dem Client in einem Webformular erfasst werden. Wenn dort ein Anwender wie gesagt statt Zahlen Buchstaben einträgt, werden die fehlerhaften Daten – unter Umständen rund um die Welt – zum Server gesendet, um dann die banale Fehlermeldung als Antwort zurück zum Client zu schicken. Mit clientseitiger Programmierung ist es möglich, dass der Fehler bereits beim Client bemerkt wird und die fehlerhaften Daten überhaupt nicht zum Server gesendet werden müssen. Clientseitige Skripte und Programme in Webseiten erlauben es einem Webbrowser in vielen Fällen, selbst auf eine intelligente Weise mit Situationen umzugehen, die sonst ein Programm auf dem Webserver oder eine Aktion durch den Anwender erforderlich machen. Wesentlicher Effekt ist damit eine Beschleunigung vieler Vorgänge, da der Browser keine Anfrage an den Server schicken und die Antwort nicht abwarten muss.

2.1.3 Probleme bei clientseitiger Programmierung

Die Entstehung clientseitiger Programmiertechniken machten Webserver auch abgesehen vom Versenden von Dateien natürlich nicht arbeitslos. So lassen sich nicht alle Aufgaben bei einer Webapplikation auf einen Client verlagern. Alle Aktivitäten mit zentraler Datenhaltung müssen zum Beispiel zwingend auf dem Server erledigt werden. Ebenso sollten sicherheitsrelevante Dinge wie die Überprüfung von Zugangsdaten nicht im Client stattfinden. Dummerweise hat der Einsatz von clientseitiger Programmierung aber noch einen gravierenden Nachteil. Es muss gewährleistet sein, dass das Skript oder Programm im Rahmen der Webseite beim Client auch läuft. Und diese grundlegende Forderung hat sich bei clientseitiger Programmierung als riesiges Problem erwiesen. Aus unterschiedlichsten Gründen bekam clientseitige Programmierung insbesondere ab etwa dem Jahr 2000 immer mehr Probleme. Einige clientseitige Programmiertechniken erwiesen sich zum Beispiel als gravierend unsicher. Deshalb wurde von vielen Leuten eine Art Sippenhaft für clientseitige Programmierung und Client-Techniken im Allgemeinen gefordert (auch für weitgehend ungefährliche Techniken wie JavaScript, Java-Applets oder Cookies). Immer öfter konnte man lesen, die Anwender sollten jegliche Form clientseitiger Programmierung in ihren Browsern deaktivieren. Ein weiteres Problem clientseitiger Programmierung bestand darin, dass es einige proprietäre Standards gab, die nicht in allen Webbrowsern implementiert wurden, und einige sehr gute Techniken durch marktpolitische Spielchen großer Softwarefirmen unter die Räder kamen – zum Beispiel Java-Applets, aber auch erste Ansätze für asynchrone Datenübertragung von der Firma Microsoft.

2.1.4 Rückverlagerung auf den Server

Die nicht zu leugnenden Probleme bei clientseitiger Programmierung – unabhängig davon, ob hausgemacht, real oder nur aufgebauscht – bewirkten etwa ab dem Jahr 2000 die zunehmende Rückverlagerung von Aktivität auf den Webserver, obwohl im Client viele dieser Aufgaben bereits im Grundsatz gelöst werden konnten. Der damalige Standpunkt der Anbieter von Webinhalten ist freilich mehr als verständlich. Wenn diese etwa auf Java-Applets gesetzt hätten und die Masse der Besucher zeitgleich einen Webbrowser einsetzte, der diese nicht richtig unterstützt, hätten die Anwender Probleme bekommen. Und einem Besucher einer Webseite ist es egal, ob das Problem durch einen inkompatiblen Browser oder ein schlecht gemachtes Webangebot verursacht wird. Es ist vollkommen klar, dass einen Besucher die Ursache für ein Problem nicht interessieren muss – er wird und darf natürlich eine Webseite fordern, die einfach funktioniert. Arrogante Webseiten mit Meldungen wie »Sie brauchen ein Flash-Plugin, sonst sehen Sie nichts!«, »Laden Sie Java, sonst machen wir keine Geschäfte mit Ihnen« oder »Die Seite ist für den Internet Explorer optimiert« waren zwar kurzfristig ein Versuch einiger Anbieter, Anwender zu einem bestimmten Verhalten zu zwingen. Aber die Anbieter haben mittlerweile ihre Arroganz entweder aufgegeben oder selbst aufgegeben, da sie keine Besucher mehr haben ;-).

Aufgrund dieser Situation im Web kann man rückblickend (etwas überspitzt) festhalten, dass das Web ab ungefähr dem Jahr 2000 für einige Zeit auf das Niveau zurückfiel, das es bereits vor 1995 hatte. Die Webserver »schafften sich den Wolf«, die Netzleitungen schmolzen ob unnötiger Anfragen und Unmengen an redundanten Datentransfers und die Client-Rechner starben vor Langeweile – zwar alles auf einem technisch höheren Niveau als knapp ein Jahrzehnt zuvor, aber im Grunde nicht anders.

2.1.5 Der Status quo

Mittlerweile ist jedoch eine Normalisierung zwischen den Extremen zu beobachten. Serverseitige Webprogrammierung wird über Sprachen wie PHP immer populärer, aber die Hysterie bezüglich der Gefahren clientseitiger Programmierung ist abgeklungen. Ebenfalls halten sich die meisten Browser-Hersteller so langsam an diverse Standards, die für die Darstellung und Funktion einer fortschrittlichen Webseite notwendig sind. Und clientseitige Programmierung erobert sich den Teil an Aktivität im Webbrowser zurück, der dort sinnvollerweise schon getan werden sollte. Zwar ist als einziger Vertreter der clientseitigen Zunft nur noch JavaScript bzw. dessen Microsoft-Clon JScript[5] übrig geblieben, aber der wird mittlerweile wieder flächendeckend eingesetzt. Schauen Sie sich heute einmal populäre Webseiten im Internet an. Keine einzige der heutzutage angesagtesten Webseiten kommt ohne JavaScript oder Style Sheets daher. Dementsprechend sind auch unqualifizierte Empfehlungen zur Deakti-

5 *JScript ist eine Variante von JavaScript, die zur Unterstützung von JavaScripts im Internet Explorer von Microsoft entwickelt wurde.*

vierung von JavaScript im Browser absolut praxisfern und verstummen immer mehr. Denn kaum ein Anwender möchte auf den Nutzen populärer Webangebote wie Amazon, ebay, Wikipedia oder Yahoo verzichten.

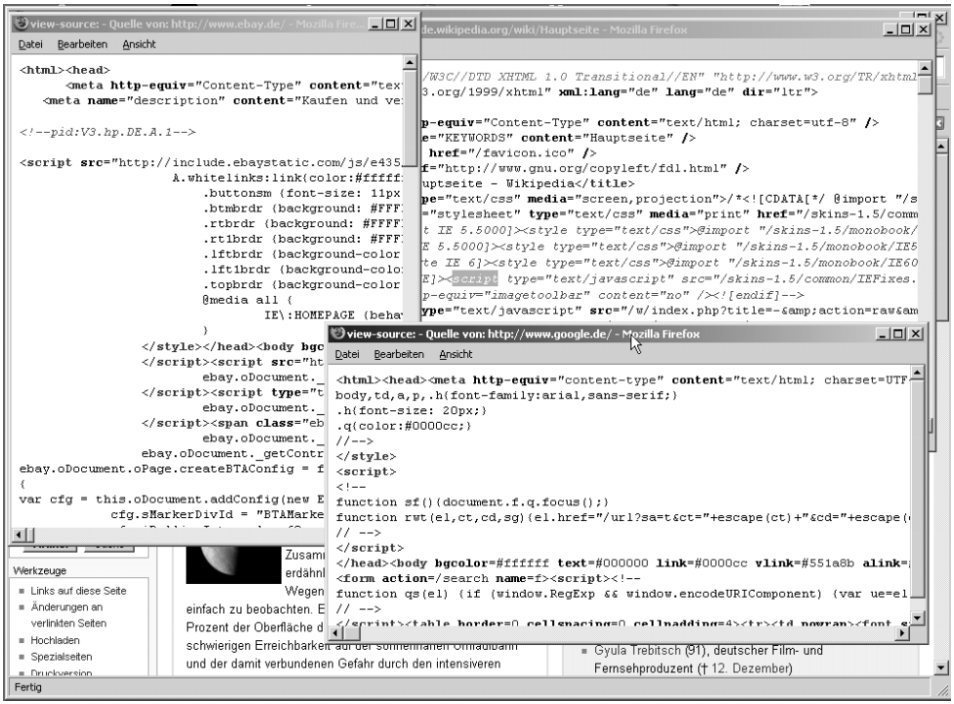


Abbildung 2.1: Alle angesagten Webpräsenzen setzen auf CSS und JavaScript – schauen Sie sich einfach einmal die Quelltexte an.

Als Ersteller von Webangeboten können Sie im Umkehrschluss also JavaScript und CSS samt zusammenhängenden Techniken heutzutage wieder bei den meisten Clients voraussetzen, was wir für AJAX auch machen müssen.

2.1.6 Wozu brauchen wir AJAX?

Wenn wir nun aber heutzutage bei einer typischen, etwas aufwändigeren Webapplikation in der Regel eine vernünftige Arbeitsteilung zwischen dem Webserver und dem Client sehen, wozu dann noch AJAX? Nun, das Problem von HTTP ist weder durch einseitige Programmierung auf Server oder Client noch durch eine Arbeitsteilung zu lösen: die Notwendigkeit, bei der Reaktion auf eine Anfrage durch einen Webbrowser immer eine vollständige Webseite als Antwort zu senden. Betrachten Sie als Beispiel für eine problematische Situation ein Webformular, das in einem Listenfeld die Auswahl aller Bundesländer in der Bundesrepublik Deutschland erlaubt.

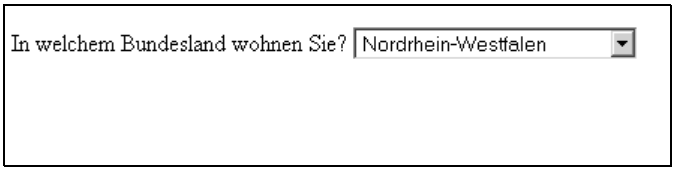


Abbildung 2.2: Ein Listenfeld, über das eine Auswahl der Bundesländer vorgenommen werden kann

Je nach gewähltem Bundesland soll nach einer Auswahl durch den Anwender in der Webseite eine ergänzende Information angezeigt werden. Dies kann ein weiteres Listenfeld sein, das alle Landkreise des Bundeslands zur Auswahl anbietet oder auch nur die Anzeige der jeweiligen Landeshauptstadt.

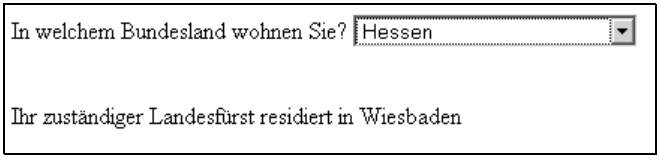


Abbildung 2.3: Nach einer Auswahl im Listenfeld werden Zusatzinformationen in der Webseite angezeigt.

Sie können nun bereits beim ersten Anfordern des Webformulars auf Vorrat alle Landkreise oder Landeshauptstädte von Deutschland dem Client senden, diese Informationen per JavaScript vorhalten[6], dann aufgrund der Auswahl verwenden und mit DHTML-Techniken in der Webseite anzeigen[7]. Aber es ist offensichtlich, dass zuerst einige überflüssige Daten zum Client geschickt werden müssen, die später niemals verwendet werden. Wenn der Anwender zum Beispiel als Bundesland Hessen auswählt, wird er niemals die Landkreise oder Landeshauptstädte von den übrigen Bundesländern benötigen. Das Verschicken von Daten auf Vorrat ist also ineffektiv. Vor allem, wenn das für mehrere Situationen in einer Webseite notwendig ist oder allgemein große Datenmengen betroffen sind, ist so eine Vorrathaltung kaum sinnvoll. Besser wäre es, benötigte Daten erst dann anzufordern, wenn sie auch wirklich gebraucht werden. In unserer Beispielsituation würde ein Anwender also auf ein Bundesland klicken und erst dann werden ihm vom Server die Landkreise, die Landeshauptstädte oder sonst eine spezifische Information geschickt. Nur besteht hier das schon mehrfach angesprochene Dilemma, dass dann eine Anforderung an den Webserver geht und dieser eine komplette Webseite als Antwort schickt. Und das wäre das komplette, möglicherweise sehr umfangreiche und schon sehr weit ausgefüllte Webformular, das mit allen bisherigen Selektionen durch den Anwender sowie den bereits eingegebenen Werten reproduziert werden muss. Die Webapplikation bekommt das träge, wenig an Desktop-Anwendungen erinnernde Verhalten.

6 *In einem JavaScript-Datenfeld zum Beispiel.*
7 *Wir werden das später im Buch noch weiter verfolgen.*

Ein weiteres Problem bei HTTP ist die Verfolgung von längeren Sitzungen. Diese besteht in der Regel aus vielen verschiedenen Webseiten, die nacheinander vom Client angefordert werden und einem bestimmten Besucher zugeordnet werden müssen. Wenn sich ein Anwender beispielsweise für einen geschlossenen Bereich legitimiert hat und dort immer neue Inhalte anfordert, bedeutet das bei jeder neuen Anforderung eine neue Seite. Es muss mühsam auf dem Client oder in der Regel dem Server verfolgt werden, ob die neue Anforderung auch wirklich von einem bereits legitimierten Besucher erfolgt. Oder wenn Sie in einem Onlineshop ein Produkt zur Bestellung auswählen und dann weitere Produkte auswählen wollen. Wenn Sie dabei auf weitere Seiten surfen, muss der Onlineshop Ihre bisherige Auswahl ja behalten, um abschließend eine Bestellung aller ausgewählten Produkte entgegennehmen zu können.

AJAX for president

Und hier betritt AJAX die Bühne! Über die Zeit haben Webentwickler zahlreiche Techniken entwickelt, entweder mit besagter Vorratshaltung von Daten eine Applikation zu beschleunigen oder den Webserver auszutricksen, um bei einer Anforderung von Daten nicht eine vollständig neue Webseite gesendet zu bekommen[8] – oder aber eine Sitzung zu verfolgen. AJAX beschreibt nun eine sukzessive immer weiter standardisierte Vorgehensweise, wie man ohne Vorratshaltung von Daten auskommt und dennoch eine Reaktion einer Webapplikation in (nahezu) Echtzeit gewährleisten kann, obwohl neue Daten vom Webserver angefordert werden. Und wie man eine Sitzung sehr einfach verfolgen kann. Statt der Anforderung einer vollständigen Webseite wird eine AJAX-Datenanfrage dazu führen, dass nur die neuen Daten vom Webserver geschickt und diese dann in die bereits beim Client geladene Webseite »eingebaut« werden. Für den Fall der Verfolgung einer Sitzung bedeutet das, dass von einem Besucher nur eine einzige Seite angefordert wird und damit gar keine wirkliche »Verfolgung« einer Sitzung aus mehreren Seiten notwendig ist.

Vereinfacht gesagt werden Daten bei der Verwendung von AJAX also erst dann angefordert, wenn sie benötigt werden[9], dann für JavaScript verfügbar gemacht[10] und anschließend in die bestehende Webseite eingebaut[11]. Dabei wird in vielen Fällen die normale Interaktion des Benutzers mit der Webanwendung nicht durch ein Laden einer neuen Seite unterbrochen. Vergegenwärtigen Sie sich nun noch einmal, dass das **A** in AJAX für *asynchron* seht. Die Kommunikation zwischen Browser und Server, die der Benutzer wahrnimmt, ist nur die, die durch seine Aktionen ausgelöst wird, zum Beispiel sein Anklicken eines Hyperlinks oder das Versenden von Formulardaten. Die offensichtlichen Schritte der Kommunikation zwischen Client und Server gibt der Benutzer vor. Bei der konventionellen Webprogrammierung kommunizieren Browser und Server sonst normalerweise auch nicht weiter. Bei AJAX ist das anders. Unabhängig von einer offensichtlichen Benutzeraktion können der Browser und der Web-

8 Wir werden auch diese Alternativen zu AJAX im Kapitel 10 kurz besprechen.
9 Oft in Form eines XML-Dokuments, das mithilfe einer serverseitigen Programmiertechnik wie PHP aufbereitet wird – das ist aber nicht zwingend.
10 Über ein geeignetes Objekt.
11 Unter Verwendung von Style Sheets, DOM und JavaScript – also DHTML.

server weitere Nachrichten austauschen. Eben asynchron zum Verhalten der Webapplikation an der Oberfläche. Ein Skript, das über einen beliebigen Eventhandler oder ein Ereignis im Allgemeinen aufgerufen wird, kann zum Beispiel eine Kommunikation außerhalb des Benutzertakts auslösen. Dies kann und wird im Extremfall so weit führen, dass ein Anwender in einem Webformular über ein Listenfeld ein Zeichen auf der Tastatur eingibt und direkt nach Eingabe des Zeichens bereits eine ergänzende Information vom Server nachgeladen wird. Das kann man beispielsweise mit dem Eventhandler `onKeyUp` oder `onKeyPress` erledigen[12]. So etwas kennen Anwender bereits von vielen Desktop-Anwendungen mit Suchfunktionen. Aber im Web ist diese komfortable Unterstützung des Anwenders bisher daran gescheitert, dass immer eine vollständige Webseite ausgetauscht werden musste.

Hinweis

Bei der bisherigen Webprogrammierung fristeten sämtliche Eventhandler, die bei Tastatureingaben ausgelöst wurden (dem vollständigen Druck einer beliebigen Taste, dem Herunterdrücken einer Taste und dem Loslassen einer Taste – alle drei Situationen bilden eigenständige Ereignisse), ein tristes Schattendasein. Die Reaktion auf das Betätigen einer Taste auf der Tastatur wird zwar vom W3C schon recht lange offiziell unterstützt. Die Unterstützung in geeigneten Reaktionsmodellen war jedoch bis zum Auftauchen von AJAX nicht sonderlich wichtig, da das WWW vom Anwender nahezu ausschließlich mit der Maus bedient wurde (mit Ausnahme der URL-Eingabe in der Adressleiste des Browsers) und Daten vom Server fast ausschließlich aufgrund dieser Interaktion mit dem Anwender angefordert wurden.

Hinweis

Das immer populärer werdende **Google Suggest** (*http://www.google.com/webhp?complete=1&hl=en*) arbeitet genau auf diese Weise. Google Suggest bezeichnet eine Erweiterung des konventionellen Suchdiensts. Während der Anwender im Webformular in dem Eingabefeld einen Suchbegriff eingibt, werden ihm unter dem Eingabefeld Zeichen für Zeichen verbesserte Vorschläge in einem Listenfeld präsentiert. Dabei werden ihm solche Begriffe vorgeschlagen, die mit den bisher eingegebenen Zeichen beginnen und damit in Verbindung mit dem gedachten Suchbegriff des Anwenders stehen könnten. Natürlich sind diese mit einem gewissen Ranking versehen, das auf der Serverseite durch die – möglicherweise auch sehr komplexe – Geschäftslogik gegeben ist (suggest – vermuten). So etwas kennen die meisten Anwender natürlich schon von Desktop-Applikationen, aber im Web ist es neu.

12 *Der Eventhandler* `onKeyUp` *eignet sich am besten, denn zu dem Zeitpunkt ist der Wert der gedrückten Taste bereits bekannt und kann verwertet werden.*

Wir werden im Laufe des Buchs die Funktionsweise von Google Suggest nachprogrammieren.

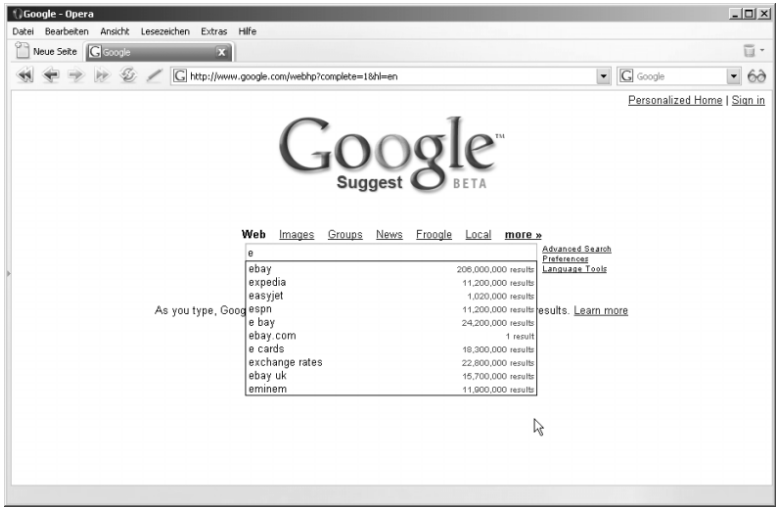


Abbildung 2.4: Wenn Sie einen Buchstaben eingeben, schlägt Ihnen Google Suggest bereits zahlreiche Begriffe vor, die damit beginnen.

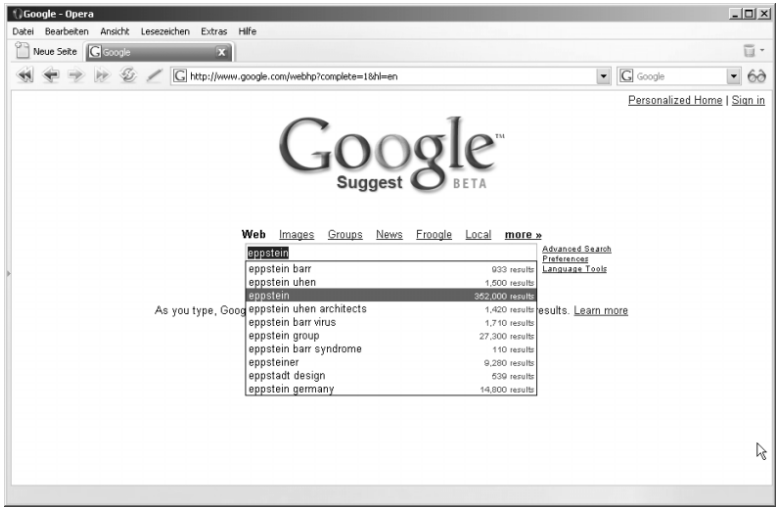


Abbildung 2.5: Je genauer die eingegebenen Begriffe werden, desto besser werden auch die angebotenen Vorschläge.

Allgemein wird ein Server auf jede Anfrage dieser Art mit Daten antworten, die dann im Client via JavaScript über ein DOM-Objekt zur Verfügung stehen. AJAX-Anwendungen können dabei in allen konformen Webbrowsern ausgeführt werden, wobei in

der Regel die meiste Programmlogik oder der Prozessfluss der Anwendung zum Generieren der Daten auf dem Server bleibt. Hier kommen dann weitgehend beliebige serverseitige Techniken wie die in diesem Buch behandelten Java Servlets und Java Server Pages, aber auch EJBs (Enterprise Java Beans), .NET-Komponenten oder Skripte wie PHP zum Einsatz. Diese serverseitige Geschäftslogik kann beliebig komplex werden, aber die konkrete Umsetzung ist nicht vorgegeben. AJAX fordert zwar im Client bestimmte Techniken, jedoch keine explizit einzuhaltenden Voraussetzungen auf dem Webserver. Das ist sicher ein Schlüssel zum Erfolg von AJAX, da auf Serverseite keinerlei Erweiterungen oder Umstellungen erfolgen. Nur müssen Daten so zum Client geschickt werden, dass dieser damit etwas anfangen kann. Am besten eignet sich dazu in komplexeren Fällen als Datenformat zwar XML, aber jede Form von Klartext (reiner Text, HTML, Style Sheets, RSS etc.) kann in bestimmten Situationen ebenso sinnvoll sein.

Hinweis

Per AJAX an den Client gesendete Daten sind dort nicht einfach so im Quelltext der Webseite zu sehen, wie es direkt vom Browser angeforderte Klartextdaten sind. Das beseitigt das Problem, dass alle in Form von Klartext zum Client gesendeten Informationen einem halbwegs versierten Anwender frei zugänglich sind, wenn er sich den Quelltext über den Browser ansieht. Das `XMLHttpRequest`-Objekt, das jede Datenanforderung von AJAX kapselt, verbirgt die Struktur der zum Client gesendeten Daten.

Allerdings kann natürlich jeder halbwegs versierte Anwender die per HTTP übertragenen Daten mit einem Sniffer analysieren. Es handelt sich ja um ein unverschlüsseltes Protokoll, das reinen Klartext als Nutzlast beinhaltet.

Abbildung 2.6: Die Webseite zeigt ein Listenfeld und einen per AJAX geschickten Text an.

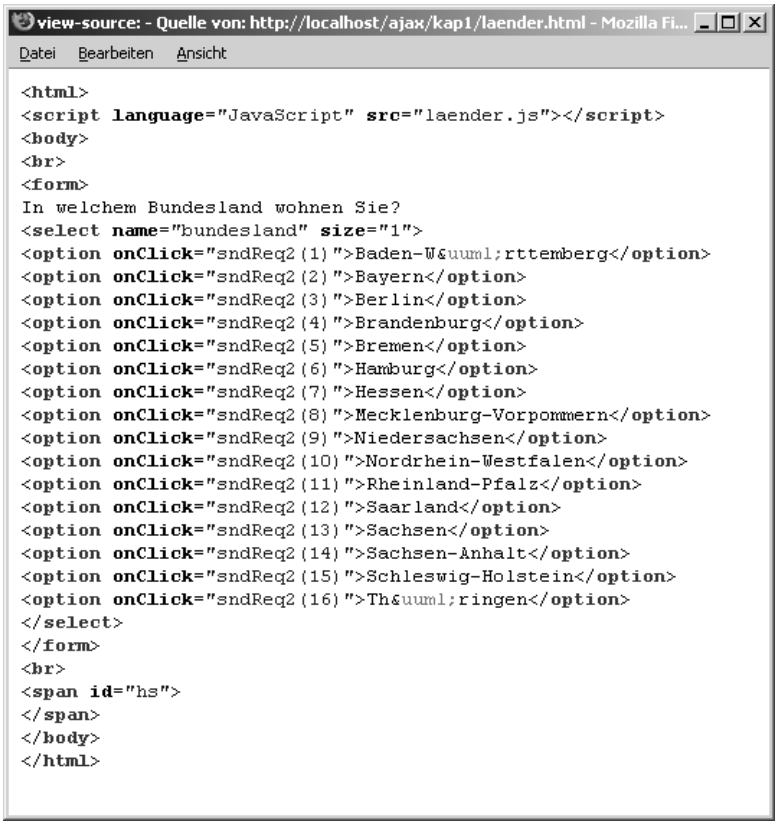


Abbildung 2.7: Im Quelltext ist die Antwort vom Server nicht zu sehen.

AJAX und Web Services

AJAX-Applikationen sind auch prädestiniert, um so genannte **Web Services** zu nutzen. Die Technologie von Webdiensten (oder Web Services) bezeichnet Anwendungen, die über das Web oder im Prinzip eine beliebige andere Netzwerkform mit einer eindeutigen Adresse erreichbar und deren Schnittstellen zum Aufruf im XML-Format definiert sind. Web Services stellen einem Aufrufer über das Netz eine bestimmte Funktionalität bereit, die dieser unmittelbar in seiner Applikation nutzen kann. Dieser Form des Leistungsangebots wird im B2B[13]-Bereich eine große Zukunft versprochen. Ein Web Service ist dabei nicht zur unmittelbaren Kommunikation mit einem Anwender gedacht, sondern der direkten Interaktion mit anderen Programmen, etwa einer AJAX-Applikation, die davon Daten anfordert. Dabei werden XML-basierende Nachrichten zum Austausch der Informationen verwendet, was Sie mit AJAX-Applikationen ja quasi von Natur aus machen können. Die Grundlagen von Web Services

13 *Business-to-Business*

bilden ein Verzeichnisdienst zur Registrierung von Webdiensten (**UDDI – Universal Description, Discovery and Integration**)[14], eine auf XML basierende plattform-, programmiersprachen- und protokollunabhängige Sprache zur Beschreibung der unterstützten Methoden eines Webdiensts (**WSDL – Web Services Description Language**) und ein Kommunikationsprotokoll zum eigentlichen Aufruf der bereitgestellten Methoden (**SOAP – Simple Object Access Protocol**[15]), das wie AJAX als Unterbau gewöhnlich auf HTTP oder HTTPS aufsetzt.

2.1.7 Die Geschichte von AJAX

Wie bereits mehrfach erwähnt, bezeichnet AJAX im Wesentlichen Technologien, die schon lange im Web etabliert sind, beginnend bei HTML und HTTP über JavaScript und CSS bis hin zu XML. Auch die asynchrone Nachforderung von Daten, die in eine Webseite integriert werden sollen, gibt es schon seit ungefähr 1998. Allgemein wurde das Verfahren zur asynchronen Nachforderung von Daten früher mit dem Begriff **XMLHttpRequest** beschrieben. Die ersten Techniken zur clientseitigen Anforderung von Daten auf diese Weise gehen auf das Outlook Web Access Team von Microsoft zurück und wurden sowohl in den Microsoft Exchange Server als auch den Internet Explorer integriert[16]. Später folgten weitere isolierte Anwendungen, die sich aber nicht flächendeckend durchsetzen konnten. Der Zusammenschluss dieser also schon einige Zeit bekannten, aber teilweise noch nicht standardisierten Techniken unter dem Begriff AJAX ist relativ neu. Wem genau die Erfindung dieses Begriffs zuzuordnen ist, ist nicht eindeutig. Allerdings hat ihn ein Mitarbeiter der Agentur Adaptive Path mit Namen **Jesse James Garrett** in seinem Essay AJAX: A New Approach to Web Applications[17] maßgeblich geprägt und bekannt gemacht. Richtig populär begann AJAX als Begriff im Jahr 2005 zu werden. Das lag sicher nicht zuletzt daran, dass Google das Verfahren für einige bekannte Anwendungen wie beispielsweise Google Groups, Google Maps, Gmail und im schon angesprochenen Google Suggest verwendet und die AJAX-Unterstützung in der Gecko-Engine, auf der viele wichtige Browser basieren, erheblich weiterentwickelt wurde. Diese unabdingbare Unterstützung ist mittlerweile in allen modernen Webbrowser integriert.

2.2 Sprung ins kalte Wasser

Ihnen sollte nun einigermaßen klar sein, was AJAX für Sie tun kann. Na, dann tun Sie es doch. Oder noch motivierender – ab ins kalte Wasser ;-). Wir erstellen unsere erste AJAX-Webapplikation. Dabei ignoriere ich bewusst, dass Ihnen zahlreiche Details zu

14 *Damit lassen sich Web Services für bestimmte Zwecke finden.*

15 *Die ausgeschriebene Form Simple Object Access Protocol für SOAP wird seit der Version 1.2 offiziell nicht mehr verwendet, da SOAP nicht (nur) mehr dem einfachen Zugriff auf Objekte dient.*

16 *Wir werden uns in diesem Zusammenhang noch XML-Dateninseln in Webseiten ansehen (siehe Kapitel 10).*

17 *Im Internet unter http://www.adaptivepath.com/publications/essays/archives/000385.php nachzulesen.*

den verwendeten Techniken noch unklar sein können[18]. Aber Sie werden den Schritt-für-Schritt-Anweisungen sicher folgen können und anhand des praktischen Beispiels bekommen Sie ein Gefühl für AJAX. Und nicht nur das – obwohl das nachfolgende Beispiel ob der Klarheit und der möglichst einfachen Programmierung nicht besonders elegant und universell noch umfangreich oder komplex sein wird, werden nahezu sämtliche (!) relevanten Schritte zum Umgang mit AJAX bereits vertreten sein.

2.2.1 Aufbau einer Testumgebung

Um AJAX-Beispiele in der Praxis zu testen, brauchen Sie zwingend eine Testumgebung, die den Gegebenheiten im realen Internetumfeld entspricht. Das erfordert die Installation bzw. Einrichtung einiger Programme.

Der Webserver bzw. Java Application Server

Auf der einen Seite benötigen wir für AJAX nur einen beliebigen Webserver, von dem wir Daten anfordern können. Auf diesem müssen Sie dann aber auch mit einer geeigneten Programmiersprache programmieren können. Und da wir auf Serverseite sowohl im folgenden Beispiel als auch in den weiteren Beispielen im Buch ausschließlich Java einsetzen werden, sollte der Server Java-Applikationen ausführen können. In der Grundform können das Webserver wie Apache aber nicht. Sie benötigen also einen so genannten **Java Application Server** oder einen **Java-Container**, der den eigentlichen Webserver unterstützt. Wenn Sie bereits so einen Server zur Verfügung haben, dieser eingerichtet ist und Sie damit umgehen können, können Sie den nun folgenden Abschnitt überfliegen. Aber das wird sicher nicht für alle Leser der Fall sein, denn Erfahrung in serverseitiger Programmierung oder gar die Administration von Servern wird in dem Buch nicht vorausgesetzt.

Nun kann bereits ein Java-Container[19] wie Tomcat im Grunde fast alle Aufgaben erledigen, die ein reiner Webserver wie Apache normalerweise erledigt. Also im Wesentlichen auf Anfragen von einem Webbrowser beliebige Arten von Dokumenten zum Client schicken – seien es Webseiten, Bilder, multimediale Dateien. Dennoch ist es in der Praxis aus verschiedenen Gründen sinnvoll, sowohl einen reinen Webserver wie Apache als auch einen Server wie Tomcat parallel zu betreiben. Für spezielle Dinge einer Webanforderung kann ein spezialisierter Webserver doch mehr Dinge als ein Server, der auf die Ausführung von Java-Applikationen spezialisiert ist. Ein Webserver kann zudem einem Java Application Server viel Arbeit abnehmen, wenn keine Java-Applikationen ausgeführt werden müssen. Vor allen Dingen dann, wenn die verschiedenen Serverprogramme auf verschiedenen Rechnern ausgeführt werden können. Wozu bei einer leichten Erkältung einen hoch spezialisierten Arzt aufsuchen,

18 Ich hoffe das sogar, denn sonst bräuchten Sie das Buch gar nicht weiterzulesen ;-).

*19 Oder auch (teils nicht ganz treffend) **JSP Engine**, **Java Application Server**, **Servlet-Container** oder **J2EE Application Server** genannt. Im Grunde muss man die Details zwar unterscheiden und was Tomcat so genau ist, aber für unsere Zwecke sind diese Spitzfindigkeiten zu weit führend.*

wenn der Hausarzt das ebenso behandeln kann und Ihnen vielleicht sogar mehr Zeit und Aufmerksamkeit widmet? Auch in der Praxis werden bei Webprojekten, die auf Java zurückgreifen, in vielen Situationen Anfragen von einem Webbrowser an einen Webserver wie Apache gestellt, und dieser reicht nur bei Bedarf gezielt Anfragen an einen Java-Container wie Tomcat weiter.

Installation von Apache bzw. XAMPP

Sie können Apache aus dem Internet in Form reiner Quelltexte laden (immerhin ist es ein OpenSource-Projekt) und diese Quelltexte selbst kompilieren und auf Ihrem Rechner einrichten. Dies ist für den Betrieb in der Praxis sicherlich auch ein sinnvoller Weg, vor allem unter Unix oder unter Linux. Für Windows gibt es allerdings auch einen bereits kompilierten Installer und auch unter Linux wird Apache bei vielen Distributionen bereits mitinstalliert oder steht in Form von spezifischen Paketen wie RPM bereit.

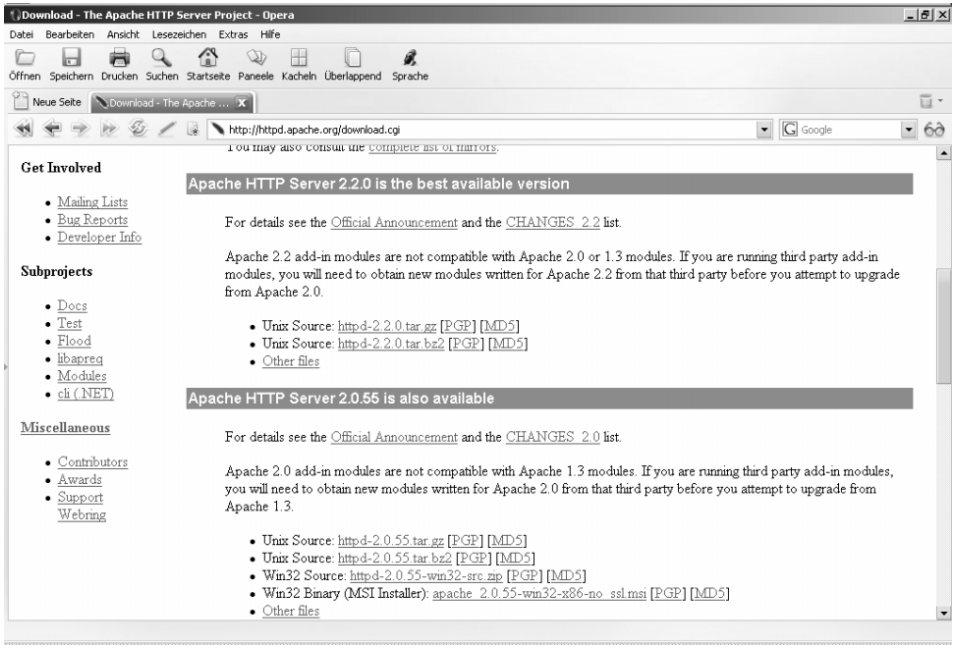


Abbildung 2.8: Apache gibt es in Form von Quelltext oder als Win32 Binary.

Aber es geht noch komfortabler. Ich hatte Ihnen zur Bereitstellung eines reinen Webservers in Kapitel 1 ein Rundum-Sorglos-Paket wie **XAMPP** (*http://www.apachefriends.org/de/*) nahe gelegt. Von XAMPP gibt es eine Version für Linux-Systeme (getestet für Ubuntu, SuSE, RedHat, Mandrake und Debian), eine Version für Windows 98, NT, 2000, 2003 und XP sowie Beta-Versionen für Solaris SPARC (entwickelt und getestet mit Solaris 8) und Mac OS X. Wenn Sie sich die XAMPP-Installationsdatei für Ihr Betriebsystem aus dem Internet geladen haben und ausführen, wird Ihnen ein typi-

scher Installationsdialog bereitgestellt. Damit ist die Installation des Webserver Apache sowie von einigen weiteren Webtechnologien ein absolutes Kinderspiel. Zur Sicherheit lesen Sie bei Bedarf die beigefügten Hinweise. Besonders interessant ist bei XAMPP auch für den Java-Fall, dass damit bereits **MySQL** inklusive **phpMyAdmin** zur Administration installiert und eingerichtet wird. Diese OpenSource-Webdatenbank ist extrem weit verbreitet und auch wir werden sie bei unseren Beispielen mit Datenbankzugriff verwenden. Diesen Komfort der automatischen Installation und Konfiguration werden wahrscheinlich viele Leser zu schätzen wissen, selbst wenn sie Java auf der Serverseite verwenden wollen.

Sobald die Installation beendet ist, können Sie entweder Apache manuell starten oder aber auch so einrichten, dass Apache als Dienst bzw. Prozess in Ihrem Betriebssystem integriert und sogar automatisch beim Start des Rechners aufgerufen werden kann. Der Webserver steht Ihnen nun nach dem Start über die Angabe von *localhost* in der Adresszeile des Webbrowsers zur Verfügung, wenn Sie ihn auf dem gleichen Rechner laufen lassen, wie Ihren Webbrowser. Andernfalls geben Sie in der Adresszeile einfach die IP-Nummer oder den Namen des Rechners ein, auf dem der Webserver läuft.

Abbildung 2.9: Zugriff auf den Webserver über localhost – hier sehen Sie die Administrationsoberfläche von XAMPP.

Tipp

Unter Windows finden Sie Ihre IP-Nummer in der Konsole mit dem Befehl `ipconfig` heraus, unter Linux/Unix mit `ifconfig`. Unter Linux/Unix benötigen Sie aber unter Umständen root-Rechte.

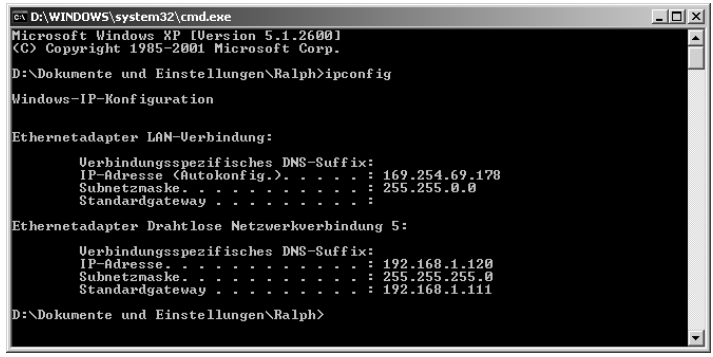


Abbildung 2.10: Ermitteln der IP-Adresse

Wenn Ihr Server auf einem anderen Rechner läuft, können Sie mit dem Befehl `ping [IP-Adresse/Name]` testen, ob der Rechner im Netz erreichbar ist. Kommt der ping-Befehl durch und der Webserver ist dennoch nicht erreichbar, behindert Sie eine Firewall oder Ihr Webbrowser ist falsch eingestellt oder der Webserver ist einfach nicht gestartet.

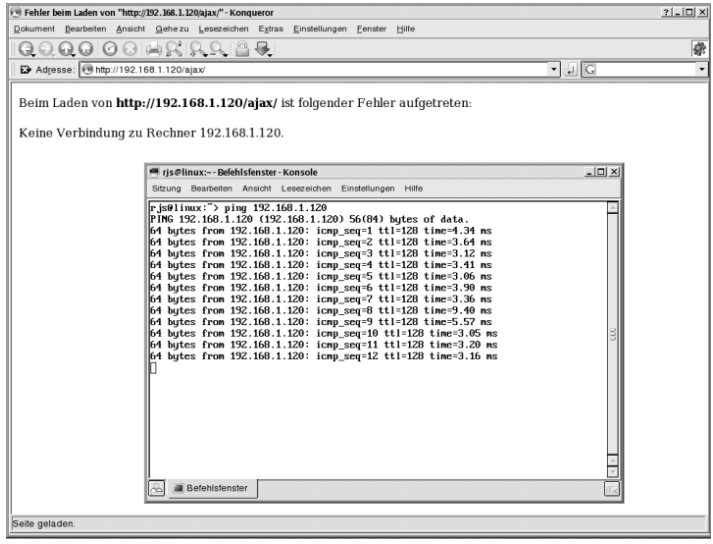


Abbildung 2.11: Der Host ist physikalisch erreichbar, aber der Webserver nicht.

Tipp

Sollten Sie bereits einen Webserver auf Ihrem Rechner betreiben und diesen nicht für AJAX-Experimente verwenden wollen oder können, können Sie ohne Probleme mehrere Webserver parallel auf einem Rechner installieren und sogar gleichzeitig laufen lassen. Diese dürfen dann nur nicht auf dem gleichen **Port** (in der Standardeinstellung Port 80) laufen. Webserver lassen sich in der Regel einfach über den Administrationsdialog oder die Konfigurationsdateien auf einen anderen Port legen und bei verschieden eingestellten Ports können Sie mehrere Webserver parallel laufen lassen. Dann müssen Sie aber beim Aufruf im Webbrowser als auch den Pfadangaben in den Programmierbeispielen den Port des Webservers explizit angeben, der nicht auf Port 80 läuft. Das machen Sie mit einem Doppelpunkt abgetrennt hinter der Host-Adresse. Beispiel:

Listing 2.1: Der Webserver horcht auf Port 4512

```
http://localhost:4512
```

Der Port wird auch der Schlüssel sein, wie Sie Apache und Tomcat auf einem Rechner parallel betreiben können. Die beiden Server werden einfach mit verschiedenen Ports arbeiten.

Tipp

Um herauszubekommen, welche Ports auf Ihrem System bereits belegt sind, bietet sich das Tool `netstat` an. Der Parameter `-a` zeigt alle Verbindungen und abhörenden Ports und *-n* Adressen und Portnummern numerisch an.

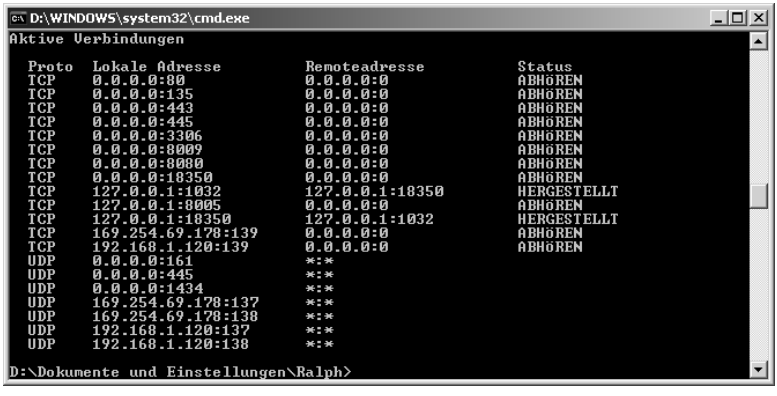


Abbildung 2.12: netstat hilft bei der Untersuchung von Ports

Achtung

Wie schon erwähnt, konfiguriert die Setup-Routine von XAMPP den Webserver in der Grundeinstellung ausschließlich für lokale Testzwecke. Sie sollten auf keinen Fall das XAMPP-System ohne manuelle Anpassung für externe Zugriffe zur Verfügung stellen. Die Grundeinstellungen sind extrem unsicher.

Bereitstellen von Daten über den Webserver

Bei Apache werden Daten in der Regel über das Verzeichnis *htdocs* bereitgestellt.[20] Dies ist das Wurzelverzeichnis aller Zugriffe über den Webserver. Oder mit anderen Worten: Wenn ein Anwender die Adresse des Webservers ohne weitere Verzeichnis- oder Dateiangaben angibt, bekommt er den Inhalt dieses Verzeichnisses angezeigt.

Hinweis

In der Praxis wird genau genommen eine Vorgabedatei[21] aus diesem Verzeichnis angezeigt, da ein Webanwender den Inhalt des Verzeichnisses aus Sicherheitsgründen nicht direkt sehen sollte. Die Details kann man meist individuell einrichten. In Apache erfolgt das zum Beispiel in der Datei *httpd.conf*.

Wurzelverzeichnis eines Webservers bedeutet auch, dass sämtliche Verzeichnisangaben auf einem Host ab diesem Verzeichnis aus gesehen werden. So würde der URL *http://localhost/ajax/meindatei.html* bedeuten, der Besucher bekommt die Datei *meinedatei.html* aus dem Verzeichnis *ajax* geschickt, welches wiederum ein Unterverzeichnis von *htdocs* ist. Wo konkret sich *htdocs* im Verzeichnisbaum des Rechners befindet, kann je nach Konfiguration von Apache individuell angegeben werden.[22] Bei XAMPP unter Windows befindet es sich zum Beispiel in der Grundeinstellung unter *[Installationslaufwerk]:\apachefriends\xampp*, wobei das jedoch auch von der Art der Installation und der Versionsnummer abhängen kann.

20 Bei anderen Webservern können Sie diese Information in der Dokumentation nachschauen.

21 Meist mit Namen index.html oder index.php.

22 Genau genommen muss man nicht einmal htdocs verwenden, aber die Ausführungen gehören hier nicht her.

Tipp

Wir werden in der Folge aus Gründen der einfacheren Infrastruktur meist nicht Apache als Webserver benutzen, sondern Anfragen direkt an Tomcat schicken, auch wenn wir nicht explizit Java Server Pages oder Java Servlets verwenden wollen, sondern nur reine HTML- oder XML-Dateien unmittelbar anfordern. Sie können aber natürlich alle Dateien für die folgenden Beispiele, die keine JSP oder Servlets darstellen, dennoch direkt in *htdocs* erstellen und von da nur die Java-Aufrufe an Tomcat weitergeben. Dann ist es zu empfehlen, in *htdocs* ein eigenes Verzeichnis für unsere Experimente zu erstellen. Dazu ist es oft sinnvoll, einzelne Projekte in weitere Unterverzeichnisse zu verlagern. Wenn wir explizit Java Servlets oder Java Server Pages verwenden, müssen Sie diese aber auf jeden Fall über Tomcat bereitstellen (siehe unten).

Achtung

Beachten Sie, dass sowohl bei Pfadangaben unter Apache als auch bei Tomcat Groß- und Kleinschreibung relevant ist. Am besten benennen Sie alle Verzeichnisse und Dateien konsequent klein.

Installation von Tomcat

Bisher war die Geschichte extrem einfach, finden Sie nicht? Keine Angst – jetzt wird es härter ;-). Aber nur ein bisschen. Die Installation von Tomcat ist nicht ganz so einfach. Tomcat benötigt sowohl eine Reihe von Anpassungen als auch Voraussetzungen auf der Server-Maschine. Und da wir letztendlich auch Java programmieren wollen, erschlagen wir in diesem Schritt gleich mehrere Fliegen. Denn zum Erstellen von Java-Applikationen benötigen wir viele Dinge, die Tomcat zur Ausführung von JSP und Servlets ebenfalls benötigt.

Die Installation von Tomcat unterscheidet sich natürlich auf verschiedenen Betriebssystemen. Wir berücksichtigen hier aber sowohl die Installation unter Windows als auch Linux. In jedem Fall setzt Tomcat auf einer vorhandenen Java-Laufzeitumgebung bzw. einer Java-Entwicklungsumgebung auf. Jetzt soll bei Ihnen vorausgesetzt werden, dass Sie sich schon ein wenig in Java auskennen, aber kein Profi sein müssen und sich auch noch nicht zwingend mit der serverseitigen Welt in Java beschäftigt haben. Damit die Leser, die in Java noch nicht so vertraut sind, nicht vollständig allein gelassen werden, folgt zudem zuerst ein kleiner Exkurs zu den Grundlagen von Java und was es mit dem JDK und dem SDK und Java-IDEs auf sich hat.

Was ist Java?

Java bezeichnet eine objektorientierte, plattformneutrale, von der Syntax an C/C++ angelehnte Programmiersprache bzw. eine ganze Plattform mit zahlreichen Tools und Softwarepaketen zur Erstellung und zum Ausführen von stabilen, sicheren und leistungsfähigen Applikationen. Diese Applikationen laufen unabhängig vom zugrunde liegenden Betriebssystem und können sowohl als Client- als auch als Server-Applikationen ausgeführt werden. Dabei muss man **unabhängig** etwas genauer beschreiben. Java-Applikationen laufen auf allen Plattformen, für die es eine Laufzeitumgebung (virtuelle Maschine) gibt. Erfunden wurde Java Anfang der 90er Jahre von der Firma Sun Microsystems (*http://www.sun.com* bzw. *http://java.sun.com*). 1995 trat Java im Rahmen des boomenden WWW in Form von **Java-Applets** seinen Siegeszug an. Applets ebneten den Weg für die gesamte Familie an Java-Applikationen – unter anderem auch solche, die auf einem Server-System wie Tomcat ausgeführt werden können. Das so genannte **JDK** (**Java Development Kit**) wird als wesentlichster Bestandteil der so genannten **Java-2-Plattform** bzw. des **Java 2 SDK** gesehen. **SDK** steht dabei für **Software Development Kit**. Dieses JDK, das Sie von den Java-Seiten von Sun kostenlos herunterladen können, erlaubt Ihnen bereits die Erstellung von Java-Applikationen, die Sie dann als Desktop-Anwendungen ausführen können[23]. Alle wichtigen Grundbibliotheken von Java und Tools, die Sie beim Erstellen von Java-Applikationen benötigen, sind enthalten, etwa ein Compiler, ein Debugger und ein Interpreter. Allerdings sind die JDK-Tools alle auf Befehlszeilenebene zu verwenden, wenn Sie diese direkt einsetzen wollen. Und das JDK enthält keinen eigenen Editor. Deshalb bietet sich bei der Programmierung von Java-Applikationen auf jeden Fall eine IDE (Integrierte Entwicklungsumgebung) an. **Eclipse** ist hier eine sehr gute Wahl. Beachten Sie, dass mit dem JDK zwar plattformneutrale Java-Programme erstellt werden können, Sie das JDK aber für Ihre spezielle Entwicklungsplattform spezifisch benötigen.

Nun sollten Sie noch über drei weitere Abkürzungen im Java-Umfeld Bescheid wissen. Da gibt es einmal den Begriff **J2SE**. Dies steht für **Java 2 Standard Edition** und bezeichnet vereinfacht gesagt den Fall von Java, bei dem einfache eigenständige Java-Applikationen erzeugt werden. Dabei sind eine bestimmte Menge an Tools (im Grunde das JDK) und Java-Pakete bzw. Java-Klassen gemeint. Der zweite Begriff

23 *Ebenso die Erstellung von Java-Applets, aber diese Form von Java-Applikationen hat heutzutage kaum noch eine Bedeutung. Eine der wenigen Ausnahmen finden Sie unter http://www.de.map24.com – einem Routenplaner. Hier kommt explizit ein Java-Applet zum Einsatz (Stand April 2006).*

nennt sich **J2EE**. Dies steht für **Java 2 Enterprise Edition**. Hierunter fallen alle Tools, Pakete und Klassen, die für Anwendungen, die im Unternehmensumfeld entwickelt werden, Verwendung finden. Dazu zählt unter anderem die Erstellung von serverseitigen Applikationen wie Servlets oder JSP. Letztendlich gibt es noch **J2ME** (**Java 2 Micro Edition**). Damit werden Applikationen für so genannte Embedded Systems (Handy, PDA etc.) erstellt.

Die Plattformneutralität von Java basiert darauf, dass Java sowohl kompiliert als auch interpretiert ist. Der eigentliche Java-Quellcode, den Sie mit einem gewöhnlichen Klartexteditor erstellen können, wird in einen binären Zwischencode (so genannten **Bytecode**) kompiliert. Dieser bezeichnet ein architekturneutrales und noch nicht vollständiges Object-Code-Format. Er ist vor allem noch nicht auf einer jeweiligen Plattform lauffähig und muss in einer **Laufzeitumgebung** (diese nennt man **Java virtual machine** – **JVM**) interpretiert werden. Da jede Java-Laufzeitumgebung plattformspezifisch ist, arbeitet das endgültige ausgeführte Programm auf dieser spezifischen Plattform. Dort werden erst alle Elemente hinzugebunden, die für eine spezielle Plattform notwendig sind. Die Tatsache, dass der letzte Teil der Übersetzung des Bytecodes von einem plattformspezifischen Programm auf der Plattform des Endanwenders ausgeführt wird, nimmt dem Entwickler die Verantwortung, verschiedene Programme für verschiedene Plattformen erstellen zu müssen. Java-Programme können auf verschiedensten Systemen mit unterschiedlichen Prozessoren und Betriebssystemarchitekturen laufen, denn die JVM kümmert sich um die Anpassung des Bytecodes an die Besonderheiten der zugrunde liegenden Plattform. Dies ist unabhängig davon, ob eine Applikation auf einem Client oder einem Server läuft.

Wenn Sie das JDK aus dem Internet geladen haben, erfolgt die Installation natürlich spezifisch für jedes Betriebssystem. Sie ist jedoch grundsätzlich vollkommen unkompliziert und wird hier nicht weiter beschrieben. Mit dem JDK installieren Sie gleichzeitig eine so genannte **Java-Laufzeitumgebung** bzw. **Java Runtime Environment** (**JRE**). Das ist der Minimalstandard, um Java-Programme laufen lassen zu können. Es handelt sich um die Umgebung, die auf jedem Computer vorhanden sein muss, auf dem eine Java-Applikationen laufen soll. Die JRE besteht aus der JVM, den wichtigsten Kernklassen von Java und einem Java-Interpreter, ein paar ergänzenden Programmen sowie einigen dazu gehörenden Bibliotheken bzw. sonstigem lauffähigen Code. Insbesondere sind keinerlei Entwicklungstools enthalten. Die Setup-Routine des JDK installiert automatisch eine JRE mit. Die JRE wird im Unterverzeichnis *jre* installiert.

Sie werden übrigens in der Regel auf Ihrem Rechner diverse Java-Laufzeitumgebungen finden. Das ist einmal diejenige, die vom JDK in dessen Installationsverzeichnis angelegt wird (sofern sie dieses installieren), aber auch an unterschiedlichsten anderen Stellen werden Sie – teils differierende – Versionen finden. Die verschiedensten Programme installieren eine JRE – in den unterschiedlichsten Versionen.

Wenn Sie nun eine Java-Laufzeitumgebung bzw. das JDK installiert haben, ist schon eine der wichtigsten Voraussetzungen für die Installation von Tomcat erfüllt. Tomcat gibt es in verschiedqenen Versionen. Dabei unterstützen ältere Versionen von Tomcat Java Servlets und JSP nicht in den neuesten Versionen. Allerdings werden wir im Rahmen dieses AJAX-Buchs diese Details nicht ausreizen. Nur der Vollständigkeit halber – wir verwenden in diesem Buch Tomcat in der Version 5.5 als Referenzsystem und diese Version unterstützt die Servlet-Spezifikation 2.4 und die JSP-Spezifikation 2.0.

Achtung

Die verschiedenen Tomcat-Versionen selbst setzen bestimmte Versionen des J2SE-JDK voraus. Dies müssen Sie im Zweifelsfall für eine spezifische Tomcat-Version genau klären. Tomcat in der Version 5.5 benötigt das J2SE JDK 5 oder J2SE 1.4 mit Kompatibilitätspaket.

Tomcat selbst gibt es sowohl als gepacktes Archiv als auch – unter Windows – als Installationsdatei im .exe-Format (*http://tomcat.apache.org/*). Die Installationsdateien sind in etwa gleich groß (ca. 6 Mbyte).

Die ZIP-Datei enthält Batch-Dateien für Windows, aber auch Shell-Skripts für Unix bzw. Linux (im Wesentlichen Startaufrufe und Konfigurationseinstellungen) sowie den eigentlichen Code von Tomcat. Und dieser ist in Java geschrieben. Damit ist die ganze Geschichte auf verschiedenen Betriebssystemen einzusetzen.

Tipp

Die Verwendung einer *.exe*-Installationsdatei unter Windows ist komfortabler, wenn Sie Tomcat später als Dienst starten wollen. Die Konfiguration und Einrichtung als Dienst wird dann automatisch erledigt.

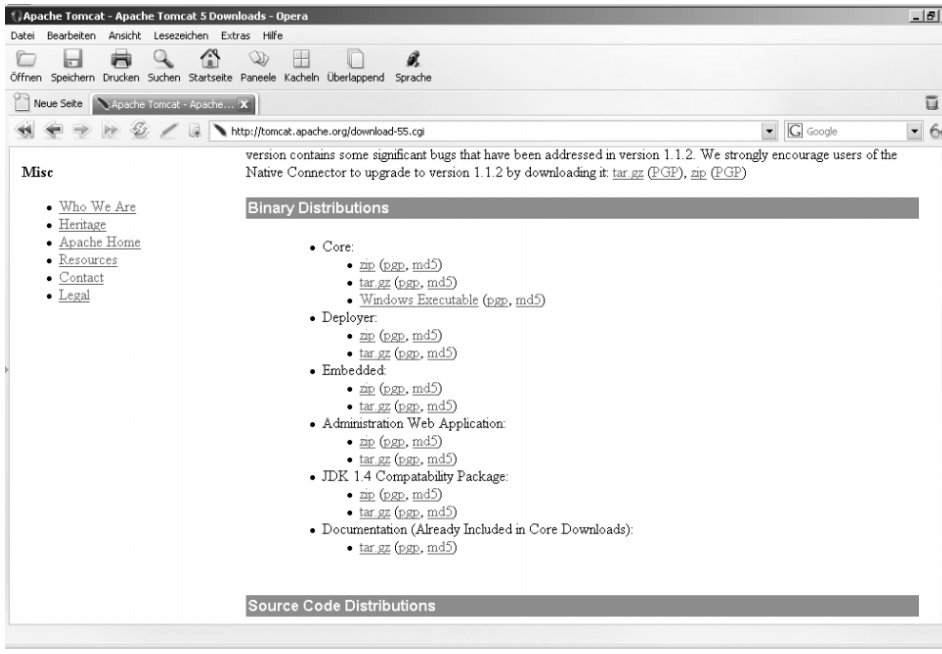


Abbildung 2.13: Der Download von Tomcat ist in verschiedenen Versionen möglich.

Der Installationsassistent selbst gibt wenig Rätsel auf. Eigentlich gibt es nur zwei Details zu beachten.

1. Die Zugangsdaten für den Administrator
2. Der Port, auf dem Tomcat lauschen soll

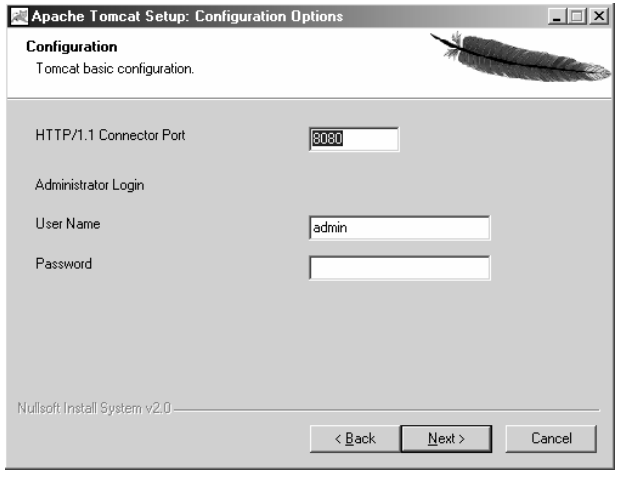


Abbildung 2.14: Port und Zugangsdaten für den Administrator

Wie schon für den Betrieb zweier verschiedener Webserver auf einen Rechner besprochen, ist der Port der Schlüssel, um auf Basis von TCP/IP auf einem Rechnersystem verschiedene Programme ansprechen zu können. Tomcat lauscht standardmäßig auf Port 8080. Dies können Sie aber sowohl bei der Installation ändern als auch später in den Konfigurationsdateien von Tomcat noch anpassen (wobei das selten notwendig ist). Falls Sie die Installationen über die komprimierten Archive durchführen wollen, expandieren Sie diese einfach in ein Verzeichnis Ihrer Wahl. Im günstigsten Fall können Sie Tomcat danach sofort starten. Dann sind Sie jetzt schlicht und einfach fertig. Herzlichen Glückwunsch. Sie können jetzt in dem Abschnitt weitermachen, indem der Start von Tomcat beschrieben wird.

Hinweis

Sollten Sie übrigens Linux als Betriebssystem benutzen, sind sie in der Regel in der glücklichen Situation, dass sowohl Java als auch Tomcat (und auch Apache sowie MySQL) bei vielen Linux-Distributionen aus dem distributionseigenen Setuptool (etwa YaST bei der SuSE-Distribution) installiert werden können.

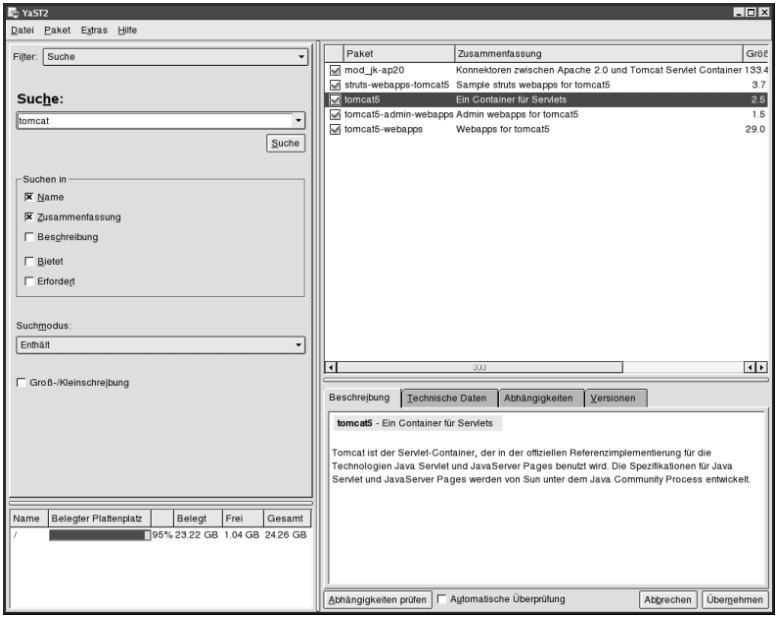


Abbildung 2.15: Installation von Tomcat unter SuSE-Linux mit YaST

Allerdings ist man dann auf die Java- und Tomcat-Versionen festgelegt, die der Distribution beiliegen oder aber von den Distributoren für ein Update später bereitgestellt werden.

Anpassung von Tomcat

Gerade unter Windows ist es im günstigsten Fall nicht notwendig, dass Sie für Tomcat nach der Installation Anpassungen vornehmen müssen. Wenn Tomcat problemlos läuft, sollten Sie die nachfolgenden Einstellungen auch so belassen, wie sie voreingestellt sind.

Aber im Problemfall sind das die Stellen, wo Sie nachschauen sollten. Zum Beispiel müssen einige **Umgebungsvariablen** (Environment-Variablen) korrekt gesetzt sein. Diese zeigen Tomcat vor allem, wo sich die Java-Laufzeitumgebung sowie die notwendigen Java-Klassen und -Pakete befinden. Dazu können Sie die Konfigurations- und Startdateien von Tomcat manuell anpassen oder aber diese Umgebungsvariablen global setzen.

Tipp

Unter Windows XP können Sie die Umgebungsvariablen über den Arbeitsplatz und dann EIGENSCHAFTEN/ERWEITERT/UMGEBUNGSVARIABLEN ändern.

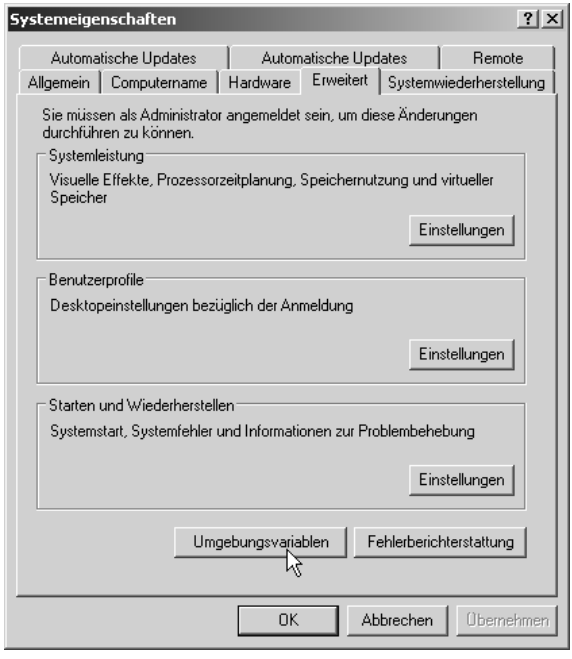


Abbildung 2.16: Anpassen der Umgebungsvariablen unter Windows XP

Bei den Umgebungsvariablen sind nun einige Stellen interessant, damit Tomcat seine Java-Umgebung korrekt finden kann.

Der so genannte **Klassenpfad** (**classpath**) zeigt dem Java-System, wo sich bestimmte Klassen befinden, die zur Kompilier- bzw. Laufzeit benötigt werden. In früheren Java-Versionen wurde dieser Klassenpfad explizit gesetzt. Das macht man in neueren Versionen nicht mehr. Im Gegenteil – in vielen Fällen ist es eine Quelle großer Probleme, wenn der Klassenpfad explizite Verzeichnisangaben enthält. Sie sollten in der Regel darauf achten, dass für den Klassenpfad nur ein einfacher `.` gesetzt ist.

Eine wichtige Angabe für Tomcat ist `JAVA_HOME`. Der Wert dieser Umgebungsvariablen sollte auf das Installationsverzeichnis des Java-SDK eingestellt sein. Die Umgebungsvariable `CATALINA_HOME` gibt an, wo sich Tomcat selbst befindet. Und letztendlich können Sie auch die Pfadangabe des Betriebssystems `Path` um das Verzeichnis erweitern, in dem sich die Tools des Java-SDK befinden. Allerdings sollten Sie beachten, dass dies dann zu einem Problem wird, wenn Sie mehrere Versionen von Java gleichzeitig auf Ihrem Rechner haben. Wenn einige Programme eine ganz bestimmte Version benötigen und der Pfad auf das falsche Verzeichnis verweist, wird unter Umständen die falsche Java-Version ausgewählt.

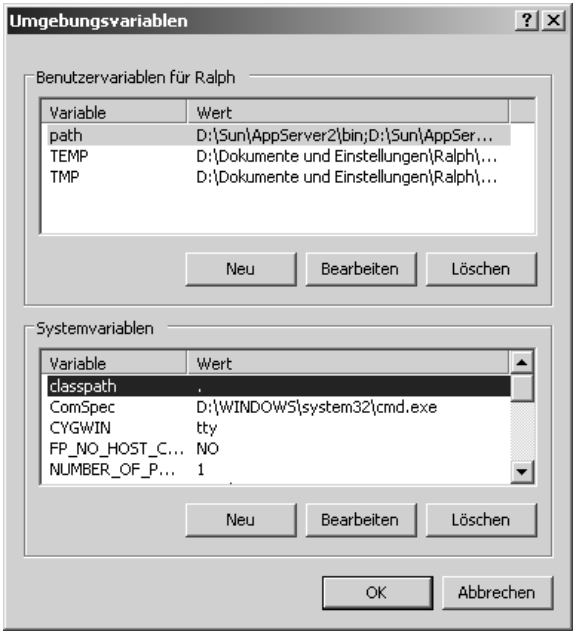


Abbildung 2.17: Einstellen der Umgebungsvariablen

Mit diesen Anpassungen sollte Tomcat nun eingerichtet sein.

Starten von Tomcat

Je nach Tomcat-Version und Installationsart gibt es verschiedene Möglichkeiten, den Server zu starten. Unter Windows steht Ihnen nach erfolgreicher Installation zum Beispiel eine Batch-Datei zur Verfügung. Unter Unix bzw. Linux ist das eine Shell-Datei.

Diese heißen überraschenderweise ;-) *startup.bat* bzw. *startup.sh*. Sie finden sie im *bin*-Verzeichnis. Der Stopp von Tomcat erfolgt über das Schließen des Konsolenfensters (was man eigentlich nicht machen sollte) oder über den Aufruf einer anderen Batch-Datei mit Namen *shutdown.bat* bzw. unter Unix/Linux die Shell-Datei *TomcatStop.sh*. Auch diese finden Sie im *bin*-Verzeichnis. Wenn Sie Tomcat mit dem Windows-Installer installiert haben, kann es sein, dass es im Windows-Startmenü Einträge zum Starten, Stoppen und Verwalten von Tomcat gibt. Ist Tomcat als Windows-Dienst installiert, wird Tomcat eventuell beim Booten bereits automatisch gestartet. Ebenso können Sie Tomcat natürlich in einem Linux/Unix-System als Daemon automatisch starten.

Tomcat aus einem Browser aufrufen

Wenn Sie nun Tomcat installiert und gestartet haben, können Sie ihn über einen Browser ansprechen. Dazu geben Sie in der Adresszeile des Webbrowsers das Protokoll *HTTP* wie gewohnt ein, dann die Adresse des Rechners (zum Beispiel *localhost*, wenn Tomcat auf dem gleichen Rechner wie Ihr Webbrowser gestartet ist), dann ein *:* gefolgt von *8080* (oder dem Port, den sie für Tomcat eingestellt haben). Das sieht zum Beispiel so aus:

http://localhost:8080

Sie erhalten die Willkommensseite von Tomcat angezeigt.

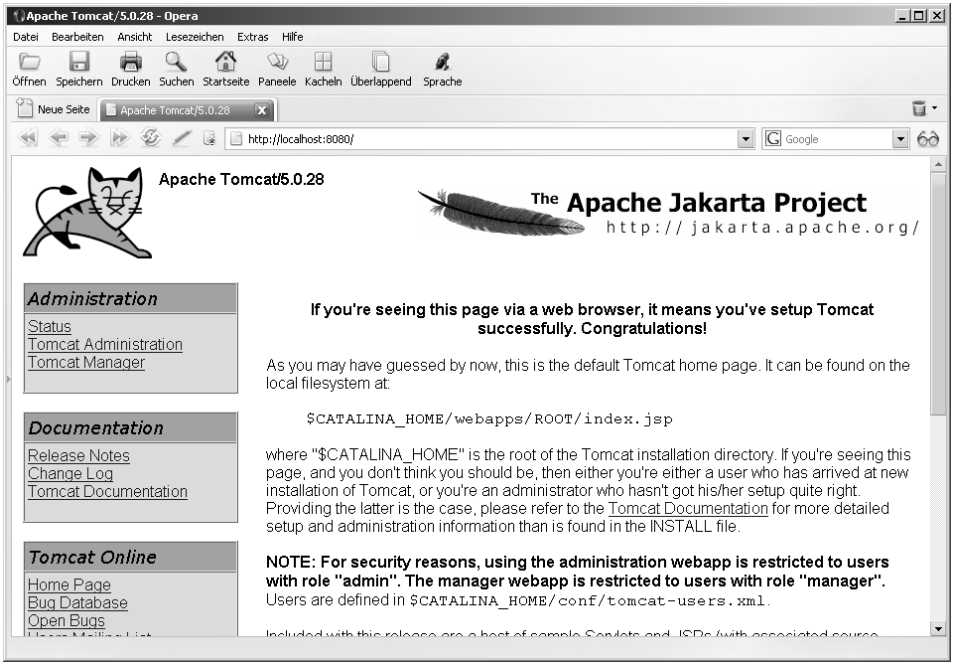


Abbildung 2.18: Der Begrüßungsdialog von Tomcat

Tomcat administrieren

Dieses ist nun auch die Stelle, über die Sie an die Administration von Tomcat gelangen. Und obwohl dies kein Buch zu Tomcat und tiefer gehenden Details von Java Servlets und JSP ist, kommen Sie nicht umhin, sich ein wenig mit der Administration von Tomcat zu beschäftigen. Im Gegensatz zu PHP-Skripts, die einfach in ein bestimmtes Verzeichnis in der Apache-Struktur kopiert werden und in der Regel sofort ausführbar sind, müssen Sie bei JSP und vor allem Java Servlets unter Tomcat gewisse Einstellungen vornehmen, bevor Sie diese Applikationen über den Server ausführen können. Das macht einerseits den Umgang mit Tomcat und Servlets bzw. JSP etwas umständlicher. Auf der anderen Seite ist das System dadurch sicherer und leistungsfähiger, wie wenn man Skripte ohne weitere Voraussetzungen ausführen kann.

Die Konfiguration von Tomcat beruht auf XML-Dateien. Wir werden in diesem Buch uns um XML ja noch genauer kümmern. Dies zwar in einem anderen Zusammenhang (beim Versenden von Daten über AJAX), aber XML gewinnt auch an vielen anderen Stellen – so etwa bei der Konfiguration von Programmen – zurzeit immer mehr an Bedeutung. Im Wurzelverzeichnis von Tomcat finden Sie einen Ordner *conf*. Darin gibt es eine Datei *tomcat-users.xml*. In dieser Datei werden die Anwender von Tomcat inklusive spezifischer Rollen, die ihnen zugedacht sind, verwaltet. Die wichtigsten Anwender für den Betrieb von Tomcat für unsere Zwecke sind **admin** und **manager**. Ihre Angaben zum Administrator haben Sie möglicherweise schon bei der Installation gesetzt. Andernfalls können Sie einen User an dieser Stelle zum Administrator und zum Manager machen. Das geht beispielsweise wie folgt:

Listing 2.2: Einrichten eines Benutzers als Administrator und Manager

```
<role rolename="admin"/>
<role rolename="manager"/>
<user username="MeinName" password="MeinPasswort" roles="admin,manager"/>
```

Wenn Sie jetzt Tomcat neu starten, können Sie Tomcat über *http://localhost:8080/manager/html* administrieren. Den Link dazu finden Sie auf der Begrüßungsseite links oben unter dem Administrationslink.

Hinweis

Die Managementseite von Tomcat ist sehr wichtig, denn darüber können Sie Java Servlets und JSP starten, neu laden oder beenden. Das ist immer dann von Bedeutung, wenn Sie ohne Neustart von Tomcat selbst eine neue Applikation unter Tomcat zur Verfügung stellen wollen oder eine bestehende überarbeitet haben und austauschen. Wenn Sie dies nicht machen, kann es sein, dass Tomcat bei einer folgenden Anfrage die veraltete Applikation verwendet.

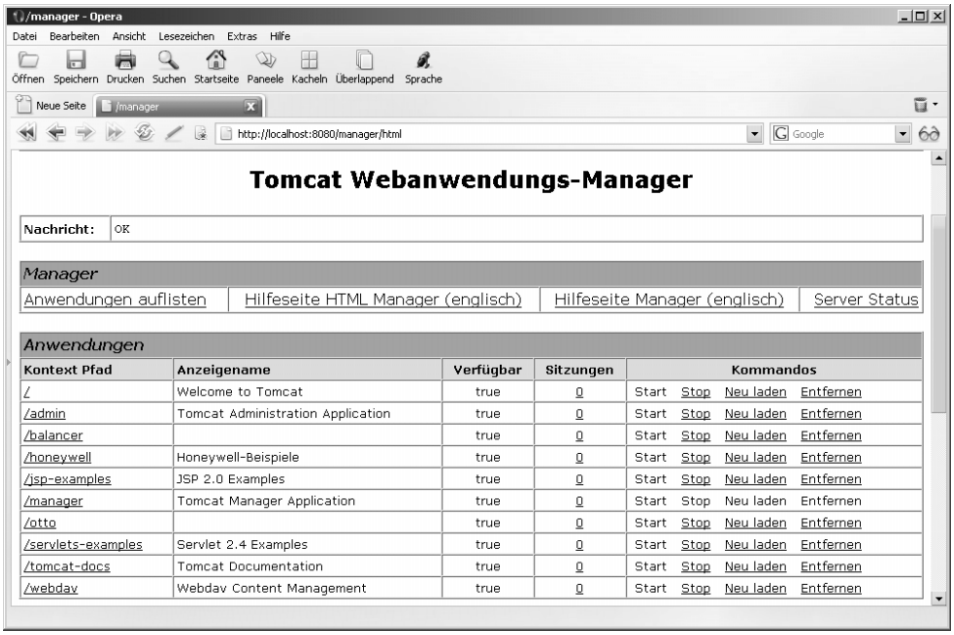


Abbildung 2.19: Das Management der Applikationen, die über Tomcat ausgeführt werden sollen

Tipp

Wenn Sie Tomcat in der Standardeinstellung betreiben, arbeitet der Server auf Port 8080. Sie können die Portnummer ersetzen, indem Sie in der Datei *server.xml* (ebenfalls zu finden unter dem Wurzelverzeichnis von Tomcat im Ordner *conf*) die Zeile `<Connector port="8080" >` anpassen.

Bereitstellen von Daten über Tomcat

Um bei Apache Dateien bereitzustellen, kopieren Sie diese in das Verzeichnis *htdocs* oder ein Unterverzeichnis davon und rufen sie über den Server aus dem Browser heraus auf. Bei Tomcat ist das Verfahren ähnlich – außer, dass Sie normalerweise beim Aufruf von Dateien den Port mit angeben. Das Verzeichnis, über das Sie bei Tomcat Daten bereitstellen, heißt *webapps*. Beachten Sie, dass viele Ressourcen in festgelegten Unterstrukturen abgelegt werden müssen. Java Servlets, Java-Beans und andere serverseitige *.class*-Dateien müssen zum Beispiel immer in der Unterstruktur */WEB_INF/classes* liegen, die sich direkt in *webapps* oder einem Unterverzeichnis darin befinden muss. Unter *WEB_INF/lib* befinden sich *.jar*-Archive. Andere Dateien wie Grafiken oder HTML-Dateien werden parallel zu *WEB_INF* abgelegt.

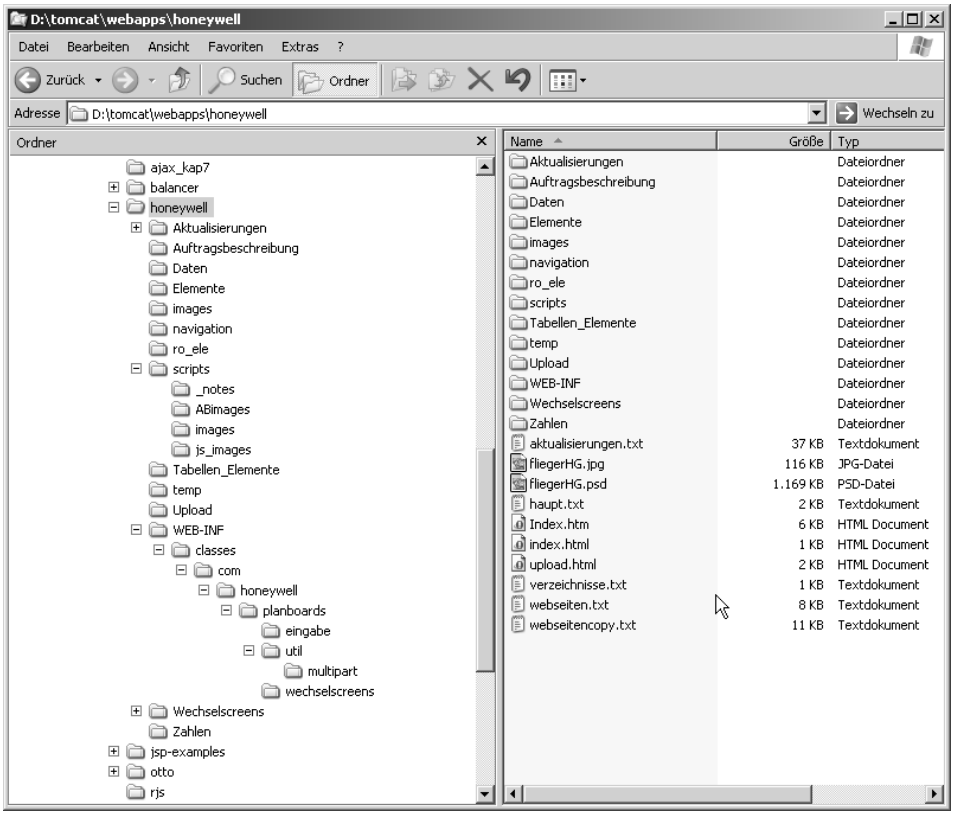


Abbildung 2.20: Eine typische webapps-Struktur von Tomcat

JSP-Dateien haben in der Regel die Dateierweiterung *.jsp*. Diese *.jsp*-Dateien werden direkt wie HTML-Dateien oder Grafiken in *webapps* bereitgestellt und im Webbrowser aufgerufen. Sie können in *webapps* auch Unterverzeichnisse erzeugen. In vielen Fällen werden JSP-Dateien zusammen mit allen anderen benötigten Ressourcen (*.html-*, *.xml-*, *.properties-* oder *.class*-Dateien, JavaBeans, Sprachressourcen, Bildern ...) zu einer einzigen so genannten **Web Application** in Form einer *.war*-Datei[24] zusammengebunden. Diese *.war*-Datei kann dann ebenfalls über das *webapps*-Verzeichnis bereitgestellt werden.

Im Fall von Java Servlets werden die *.class*-Dateien unterhalb von *webapps* in der Unterstruktur *WEB-INF\classes* bereitgestellt. Auch hier lassen sich Unterverzeichnisse anlegen. Allerdings müssen Sie bei Servlets unbedingt eine XML-Datei mit Namen *web.xml* in dem Verzeichnis *WEB-INF* anpassen, was derzeit noch zu weit führt. Darauf kommen wir jedoch in Kapitel 9 zurück.

24 *Web Application Archive*

So weit sollte nun die serverseitige Umgebung zum Ausführen unserer AJAX-Applikationen vorbereitet sein, zumindest solange wir JSP einsetzen. Nun fehlt noch als Baustein ein geeigneter Webbrowser.

Browser

Für den Anfang genügt einer der im ersten Kapitel genannten Webbrowser, aber Sie sollten letztendlich für die Praxis mindestens Opera, Internet Explorer und einen Browser der Mozilla-Familie (Mozilla, Firefox etc.) zu Testzwecken installieren.

Editor

Zur Erstellung von HTML-, Javascript- und CSS-Code genügt – wie schon mehrfach erwähnt – ein reiner Texteditor. Im Falle der serverseitigen Programmierung mit Java wird das auf lange Sicht nicht genügen, vor allem, wenn Sie Java Servlets erstellen (ich hatte Ihnen an anderer Stelle bereits eine IDE wie Eclipse empfohlen). Da wir uns in unserem ersten Beispiel jedoch auf JSP beschränken, kommen Sie vorerst auch mit einem reinen Klartexteditor aus.

2.2.2 Das konkrete Beispiel

Als erstes AJAX-Beispiel realisieren wir das vorher in dem Kapitel bereits angesprochene Webformular, das einem Anwender in einem Listenfeld die Auswahl aller Bundesländer erlaubt und in dem je nach gewähltem Bundesland eine ergänzende Information angezeigt werden soll. Dies ist der Einfachheit halber hier nur die Anzeige der jeweiligen Landeshauptstadt in einem Teil der Webseite. Aber diese Information soll erst dann vom Server angefordert werden, wenn der Anwender durch einen Klick in der Webseite ein konkretes Bundesland ausgewählt hat. Und das ist echtes AJAX. In unserer Beispielsituation klickt ein Anwender also auf ein Bundesland in dem Listenfeld und erst dann wird ihm vom Server die zugehörige Landeshauptstadt geschickt (siehe dazu auch die Abbildungen 2.2 und 2.3 in diesem Kapitel).

Achtung

Beachten Sie, dass sämtliche Beispiele nur dann funktionieren, wenn Sie unmittelbar über den Webserver oder Tomcat arbeiten. In anderen Worten – wenn Sie eine HTML-Seite in einen Browser direkt laden, werden die Beispiele nicht funktionieren (das können Sie oft auch daran erkennen, dass in der Adresszeile des Browsers in der Regel das Protokoll `file://` auftaucht). Es ist kein wesentlicher Aufwand, die HTML-Seiten, die wir in diesen Beispielen verwenden, so anzupassen, dass Sie auch direkt in den Browser geladen werden können (sie müssen bloß die Pfadangaben auf die serverseitigen Skripte bzw. Servlets entsprechend mit absoluten Pfaden versehen),aber das ist Aufwand und in der Praxis auch nicht üblich. In der Regel befinden sich alle Dateien, die zu einer Webapplikation gehören, auf dem gleichen Rechner und oft sogar im gleichen Verzeichnis.

Dann arbeitet man bei Referenzen auf andere Dateien mit relativen Pfaden. Dieses Verfahren setzt jedoch voraus, dass auch die HTML-Datei, die ein Servlet oder ein Skript aufruft, vom Webbrowser über den Server angefordert wird. Sie müssen also in der Adresszeile des Browsers die Adresse des Servers, gegebenenfalls einen Port und dann Verzeichnis und Dateiname der HTML-Seite angeben. Für unsere Beispiele werden wir allerdings in der Regel so arbeiten, dass sowohl die Servlets bzw. JSP-Skripte als auch die reinen HTML-Seiten über Tomcat bereitgestellt werden. Das erkennen Sie dann daran, dass der Aufruf einer absolut angegebenen Ressource mit dem Port 8080 notiert wird.

Die HTML-Datei

Erstellen wir zuerst eine HTML-Datei, die nur ein Webformular mit einem einzeiligen Listenfeld bereitstellt. Diese sollte in Ihrem Arbeitsverzeichnis unter *webapps* gespeichert werden[25]. Schauen wir uns zuerst den Quelltext an und besprechen dann einige Details, soweit sie in dieser Phase schon sinnvoll erklärt werden können:

Listing 2.3: Eine HTML-Datei mit einem Webformular (laender.html)

```
01 <html>
02 <script language="JavaScript" src="laender.js"></script>
03 <body>
04 <br />
05 <form name="f">
06 In welchem Bundesland wohnen Sie?
07 <select name="bundesland" size="1" onClick="sndReq()">
08 <option>Baden-W&uuml;rttemberg</option>
09 <option>Bayern</option>
10 <option>Berlin</option>
11 <option>Brandenburg</option>
12 <option>Bremen</option>
13 <option>Hamburg</option>
14 <option>Hessen</option>
15 <option>Mecklenburg-Vorpommern</option>
16 <option>Niedersachsen</option>
17 <option>Nordrhein-Westfalen</option>
18 <option>Rheinland-Pfalz</option>
19 <option>Saarland</option>
20 <option>Sachsen</option>
21 <option>Sachsen-Anhalt</option>
22 <option>Schleswig-Holstein</option>
23 <option>Th&uuml;ringen</option>
24 </select>
```

25 *Ich nenne das Arbeitsverzeichnis ajax_kap2 und die HTML-Datei laender.html. Aber das können Sie natürlich frei wählen.*

```
25 </form>
26 <br />
27 <span id="hs"></span>
28 </body>
29 </html>
```

Die Webseite enthält ein HTML-Grundgerüst ohne Kopfbereich. Ansonsten gibt es drei wichtige Bereiche, die wir uns anschauen:

1. In Zeile 2 wird eine externe JavaScript-Datei referenziert, die wir gleich betrachten. Darin wird sich die Funktionalität befinden, um Daten vom Webserver anzufordern und in diese über die AJAX-Engine (den Verarbeitungsmechanismus für eine AJAX-Datenanfrage) in die Webseite zu integrieren, ohne eine neue Webseite laden zu müssen.
2. Von Zeile 5 bis 25 erstreckt sich das Webformular. Das Formular erhält über den Parameter `name` den Namen `f` (für **F**ormular), über den wir es in einer JavaScript-Funktion später ansprechen werden (Zeile 5). Das einzeilige Listenfeld wird mit einem `<select>`-Tag erzeugt. In Zeile 7 beginnt der Container und er endet in Zeile 24. Das Listenfeld bekommt ebenfalls über den Parameter `name` einen Namen für den späteren Zugriff aus JavaScript – `bundesland`. Beachten Sie in Zeile 7 den Eventhandler `onClick="sndReq()"`. Damit wird bei einem Klick des Anwenders auf das Listenfeld (genauer: einen Eintrag in dem Listenfeld) die JavaScript-Funktion `sndReq()` aufgerufen. Diese ist in der externen JavaScript-Datei definiert. Die einzelnen `<option>`-Container stellen die Einträge in dem Listenfeld dar und zeigen die 16 Bundesländer von Deutschland zur Auswahl an.
3. In Zeile 27 sehen Sie einen `<span>`-Container. Derzeit ist er noch leer. Aber er bezeichnet den Bereich, der im Folgenden mit AJAX gefüllt werden soll. Der `<span>`-Container hat ein Attribut `id` und der Wert dieses Attributs ist der Schlüssel zum Zugriff.

Die externe JavaScript-Datei

Betrachten wir nun die JavaScript-Funktionen, die wir benötigen werden. Die externe JavaScript-Datei mit Namen *laender.js*, die in der Webseite referenziert wird, enthält zuerst die Erzeugung des so genannten Request-Objekts als globale Variable. Darüber werden in den nachfolgend definierten Funktionen Daten vom Server angefordert und mit JavaScript wieder bereitgestellt. Bei AJAX-Anwendungen, die über so ein Request-Objekt Daten anfordern, verarbeitet der Webbrowser die Antwort des Servers nicht direkt, sondern er schleust diese durch eine AJAX-Engine. Im Endeffekt bedeutet das für Sie erst einmal nur, dass Ihnen ein Request-Objekt Methoden und Eigenschaften bereitstellt, die Sie aus JavaScript heraus verwenden können. Die Erzeugung sieht in einer einfachen Form[26] so aus:

26 *Wir sehen später noch eine allgemeiner und universeller einsetzbare Variante, die zudem die Erzeugung in einer eigenen Funktion kapselt. Aber dazu werden noch einige JavaScript-Techniken (etwa Ausnahmebehandlung) vorausgesetzt, die Ihnen erst im nächsten Kapitel vorgestellt werden sollen und hier vom wesentlichen Aspekt nur ablenken. Vorerst genügt uns die einfache Form.*

Listing 2.4: Erzeugen eines Request-Objekts

```
01 var resObjekt;
02 if(navigator.appName.search("Microsoft") > -1){
03  //resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
04   resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
05 }
06 else{
07  resObjekt = new XMLHttpRequest();
08 }
```

Hier wird ein neues Objekt für die gewünschte Anfrage an den Server erzeugt. Es handelt sich um ein so genanntes `XMLHttpRequest`-Objekt, das ursprünglich von Microsoft erfunden und als ActiveX-Objekt (Typ `ActiveXObject`) implementiert wurde. Andere Browser erzeugen so ein `XMLHttpRequest`-Objekt jedoch auf andere Weise, denn die extrem sicherheitskritische ActiveX-Technologie wurde in Nicht-Microsoft-Browser niemals integriert. Und wie so oft kann man dementsprechend keine universelle Lösung wählen, sondern muss je nach Browser spezifisch arbeiten.

Das `XMLHttpRequest`-Objekt muss für den Internet Explorer über den Konstruktor[27] `ActiveXObject()` erzeugt werden. Als Argument wird dabei eine Zeichenkette übergeben, die das gewünschte Objekt bezeichnet. Dabei muss man je nach Browser-Version und Systemumgebung sowie installierter DLL für das XML-Parsen entweder die ältere Form `new ActiveXObject("Microsoft.XMLHTTP")`[28] oder die neuere Variante `new ActiveXObject("MSXML2.XMLHTTP")`[29] nehmen. Wir sehen später im Buch eine Variante zur Erzeugung eines `XMLHttpRequest`-Objekts, die diese Auswahl automatisch macht. Für unsere Zwecke ist es derzeit ausreichend, dass eine der beiden Varianten auskommentiert ist (im Beispiel hier die ältere Variante). Wenn Sie Windows mit dem Internet Explorer verwenden, können Sie auf Ihrer Plattform einfach eine der beiden Möglichkeiten testen und wenn es funktioniert, soll das für den Anfang genügen. Die Zeile 2 (`if (navigator.appName.search("Microsoft") > -1)`) stellt eine **Browser-Weiche** dar und gewährleistet, dass die Erzeugung auf diese Art und Weise nur von den Browsern gemacht wird, die sich als Microsoft-Browser (als Internet Explorer) identifizieren. Der Grund ist, dass alle anderen Browser das `XMLHttpRequest`-Objekt über `new XMLHttpRequest()` erzeugen und das wird im `else`-Zweig der Browser-Weiche gemacht.

27 *Ein Konstruktor ist eine spezielle Methode zum Erzeugen eines Objekts. Sie wird in JavaScript wie auch einigen anderen Sprachen wie Java mit vorangestelltem Schlüsselwort new aufgerufen. Im Kapitel zu JavaScript gehen wir darauf genauer ein.*

28 *Über msxml.dll zur Verfügung gestellt.*

29 *Über die neuere Version des XML-Parsers msxml2.dll zur Verfügung gestellt.*

Achtung

Browser-Weichen, die einen direkten Vergleich auf einen Wert wie `if (navigator.appName=="Microsoft Internet Explorer")` oder `if (navigator.appName=="Netscape")`ausführen, sind in der Praxis weitgehend unbrauchbar, obwohl man sie sehr oft findet. Der Wert der `navigator`-Eigenschaft `appName` kann auf vielfältige Weise geändert werden, zum Beispiel durch einen Sponsor, der einen Browser ausliefert und die Kennung aus Werbezwecken etwa in `Netscape von D-Offline` verändert. Aber auch eine verbesserte Browser-Weiche der oben verwendeten Art mit dem Suchen einer charakteristischen und in jedem (bekannten) Fall vorkommenden Kennung in `appName` kann Ihre Webapplikation in eine Sackgasse manövrieren. Und zwar können sich einige bessere Browser wie Opera gegenüber dem Server temporär als anderer Browser ausgeben. Also kann der Anwender durch eine Browser-Einstellung eine Applikationslogik bewusst oder unbewusst austricksen.

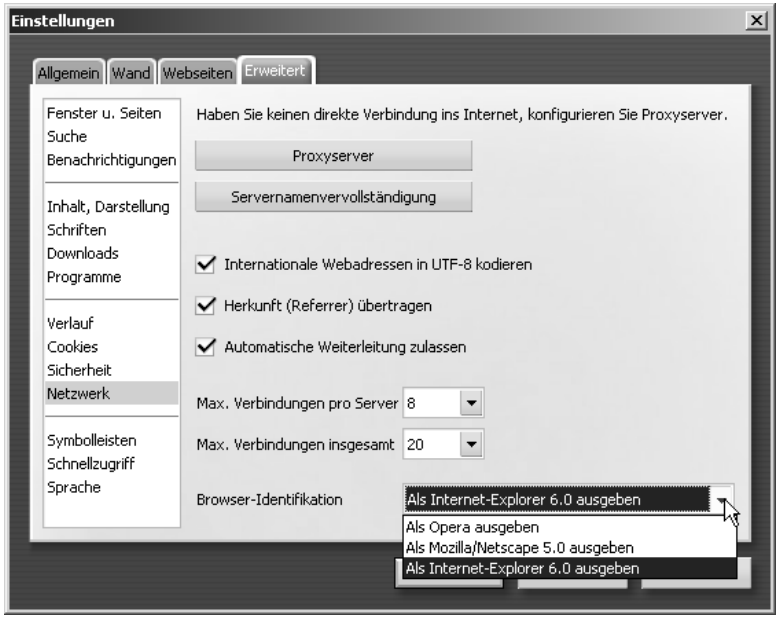


Abbildung 2.21: Der Anwender kann in einigen Browsern die Kennung verändern – hier in Opera.

Wenn sich nun also ein Browser wie Opera als Internet Explorer ausgibt, wird nach der Logik einer Browser-Weiche die Erstellung des `XMLHttpRequest`-Objekts mit dem Konstruktor `ActiveXObject` versucht. Und das wird scheitern, da Opera den anderen Weg geht. Die später noch demonstrierte Funktion zum universellen Erzeugen eines `XMLHttpRequest`-Objekts verzichtet deshalb auf eine Browser-Weiche und testet stattdessen, ob die Erzeugung auf einem ausgewählten Weg funktioniert. Andernfalls werden sukzessive die anderen Möglichkeiten ausprobiert. Um das zu realisieren, benötigen wir jedoch eine JavaScript-Technik mit Namen **Ausnahmebehandlung**. Wenn ich davon ausgehe, dass Sie schon etwas weiter mit Java vertraut sind, werden bei Ihnen die Alarmglocken läuten. In der Tat wurde in den neueren Versionen von JavaScript das Ausnahmekonzept von Java implementiert. So weit die gute Nachricht. Die schlechte ist, dass es nicht exakt genauso funktioniert wie unter Java, aber glücklicherweise sehr ähnlich – es gibt nur ein paar Feinheiten zu beachten, sonst fallen Sie auf die Nase.

Hinweis

Glücklicherweise ist die Art der Erzeugung des `XMLHttpRequest`-Objekts nicht relevant für die nachfolgend bereitgestellten Eigenschaften und Methoden des generierten Objekts. Mit anderen Worten – sobald Sie ein passendes `XMLHttpRequest`-Objekt erzeugt haben, können Sie in der Folge bei allen Browsern analog vorgehen. Dies vereinfacht den Umgang mit AJAX doch erheblich!

Da das `XMLHttpRequest`-Objekt in unserem Beispiel über eine globale Variable verfügbar gemacht wird, steht im Folgenden also ein `XMLHttpRequest`-Objekt mit all seinen Eigenschaften und Methoden unter dem Namen `resObjekt` bereit. Betrachten Sie die nachfolgende Funktion:

Listing 2.5: Die Funktion zum Senden der Anfrage

```
09 function sndReq() {
10   for(i=1;i<=16;i++){
11     if (this.document.f.bundesland.options[i-1].selected){
12       resObjekt.open('get', 'laender.jsp?wo='+i,true);
13       resObjekt.onreadystatechange = handleResponse;
14       resObjekt.send(null);
15       break;
16     }
17   }
18 }
```

Die Funktion `sndReq()` wird – wie oben gezeigt – in dem Webformular immer dann aufgerufen (mit dem Eventhandler `onClick`), wenn ein Anwender einen Eintrag in dem Listenfeld selektiert. Die Objektrepräsentation einer Auswahlliste im Rahmen des DOM-Objektmodells besitzt nun die Eigenschaften `selectedIndex` und `selected`. Mit `selectedIndex` können Sie ermitteln, welcher Eintrag in einer Auswahlliste aktiviert ist. Mit der booleschen Eigenschaft `selected` können Sie für jeden Eintrag testen, ob dieser selektiert ist oder nicht. Und das machen wir hier. Die `for`-Schleife, die sich von Zeile 10 bis 17 erstreckt, läuft über alle 16 Einträge des Listenfelds (die 16 Bundesländer). Der Zugriff auf den Selektionszustand erfolgt über `this.document.f.bundesland.options[i-1].selected` in Zeile 11. Das Schlüsselwort `this` steht für das aktuelle Browser-Fenster, `document` ist die Objektrepräsentation der Webseite, `f` die Objektrepräsentation des Webformulars, `bundesland` die Objektrepräsentation des Listenfelds und `options` ein Objektfeld mit den jeweiligen Einträgen in diesem Listenfeld. Die Schleife beginnt mit dem Startwert 1. Da in JavaScript Datenfelder mit dem Index 0 beginnen, muss der Zählindex für den jeweiligen Eintrag um den Wert 1 reduziert werden.

Wenn nun der selektierte Eintrag des Listenfelds gefunden wurde (dann hat `this.document.f.bundesland.options[i-1].selected` den Wert `true`), wird zuerst in Zeile 12 eine HTTP-GET-Verbindung zum Webserver durch `resObjekt.open('get', 'laender.jsp?wo='+i,true)` initialisiert. Der erste Parameter in der `open()`-Methode spezifiziert die Art der HTTP-Anforderung[30]. In unserem Fall wird die `GET`-Methode gewählt. Der zweite Parameter ist die Zieladresse der Anfrage. Dies ist ein gewöhnlicher URL (eine relative Pfadangabe auf die Datei *laender.jsp*), der um ein Wertepaar ergänzt wird (wie es allgemein bei der `GET`-Methode der Fall ist).

Hinweis

Sie können auf dem Server ein Skript oder Programm aufrufen (wie wir es in diesem Fall machen – ein JSP-Skript) oder auch direkt eine beliebige Datei anfordern, die dann einfach als Antwort vom Server geschickt wird. Zum Beispiel eine reine Textdatei, ein HTML-Fragment oder auch eine vollständige XML-Datei. AJAX stellt hier ausdrücklich keine genauen Anforderungen an die Art der anzufordernden Daten. Das macht das AJAX-Konzept mächtig und flexibel.

Der Name des übergebenen Werts ist `wo`, der aktuelle Wert des Zählindex der `for`-Schleife. Der dritte Parameter gibt an, ob die Kommunikation asynchron erfolgen soll oder nicht. Der Wert `true` besagt, dass die Kommunikation asynchron laufen soll, der Wert `false` für den Parameter führt dazu, dass das Programm beim nachfolgenden Ausführen der `send()`-Methode bis zur Beendigung der Kommunikation blockiert. Andernfalls (was bei uns der Fall ist) läuft die Webapplikation einfach weiter. Die abschließende `break`-Anweisung bricht beim ersten Treffer die Schleife ab und sorgt damit dafür, dass überflüssige Durchläufe unterbleiben.

30 Die Details werden in den folgenden Kapiteln genauer erläutert.

Hinweis

Obwohl in AJAX das Wort »asynchron« steckt, **muss** sich eine AJAX-Applikation keinesfalls asynchron verhalten. Der Begriff bedeutet nur, dass die Applikation Daten anfordern **kann**, ohne die Kommunikation mit dem Benutzer zu unterbrechen. Ebenfalls kann eine AJAX-Datenanforderung natürlich synchron zur konventionellen Anforderung von Daten durch den Anwender ablaufen.

Sobald die HTTP-Antwort beim Client wieder eintrifft, muss man darauf reagieren können. Dazu definieren wir eine Funktion zum Umgang mit der Antwort. Diese wird allgemein vor dem Aufruf der `send()`-Methode[31] an die `onreadystatechange`-Eigenschaft des Request-Objekts gebunden (Zeile 13: `resObjekt.onreadystatechange = handleResponse;`). Das gesamte Verfahren werden wir natürlich noch genauer besprechen. Die hier angegebene Funktion `handleResponse()`[32] wird nur dann ausgeführt, wenn die Daten vollständig übermittelt wurden. Beachten Sie, dass Sie beim Binden an die `onreadystatechange`-Eigenschaft bei der Funktion keine Klammern angeben. Es handelt sich hier um eine so genannte **Funktionsreferenz** und keinen gewöhnlichen Funktionsaufruf.

Nachfolgend finden Sie das ziemlich einfache Listing der Funktion `handleResponse()`:

Listing 2.6: Die Funktion zum Behandeln der zurückgelieferten Daten

```
19 function handleResponse() {
20   if(resObjekt.readyState == 4){
21     document.getElementById("hs").innerHTML =
22       resObjekt.responseText;
23   }
24 }
```

In Zeile 20 wird zuerst mit der Eigenschaft `readyState` ein bestimmter Status der Datenübertragung überprüft. Der Wert 4 steht für `COMPLED` (vollständig). Wenn also die Daten vollständig übertragen wurden, greifen wir in Zeile 21 mit `document.getElementById("hs")` auf das Element der Webseite zu, dem der Parameter `id` mit dem Wert `hs` zugeordnet ist. Das ist in der Webseite der `<span>`-Container. Die Methode `getElementById()` ist im W3C-Objektmodell DOM definiert. Das Attribut `innerHTML` erlaubt den Zugriff auf den Inhalt dieses Containers. Der Wert wird in Zeile 22 über das Objekt `resObjekt` zugewiesen. Dieses Objekt enthält über die Eigenschaft `responseText` die Antwort des Webservers.

31 Diese schickt die Anforderung an den Server.

32 Die Wahl dieses Namens finden Sie sehr häufig im AJAX-Umfeld.

Die Serverseite

Zu guter Letzt schauen wir uns an, wie die Antwort von Server generiert wird. In der Funktion `sndReq()` sehen Sie in Zeile 12, dass ein JSP-Skript mit Namen *laender.jsp* aufgerufen und an dieses ein Wertepaar übergeben wird. Hier ist dessen Quelltext, den Sie einfach mit einem normalen Klartexteditor erstellen können:

Listing 2.7: Das aufgerufene JSP-Skript, das die Antwort an den Client generiert

```
01 <%@ page language="java" %>
02 Ihr zuständiger Landesfürst residiert in
03 <%
04   switch(new Integer(request.getParameter("wo")).intValue()) {
05     case 1: %> Stuttgart<%break;
06     case 2: %> M&uuml;nchen<%break;
07     case 3: %> Berlin<%break;
08     case 4: %> Potsdam<%break;
09     case 5: %> Bremen<%break;
10     case 6: %> Hamburg<%break;
11     case 7: %> Wiesbaden<%break;
12     case 8: %> Schwerin<%break;
13     case 9: %> Hannover<%break;
14     case 10: %> Düsseldorf<%break;
15     case 11: %> Mainz<%break;
16     case 12: %> Saarbrücken<%break;
17     case 13: %> Dresden<%break;
18     case 14: %> Magdeburg<%break;
19     case 15: %> Kiel<%break;
20     default: %> Erfurt
21 <%}%>
```

Wir gehen an dieser Stelle auf den Quellcode nur marginal ein. Die entscheidende Stelle sehen Sie in Zeile 4. Dort beginnt eine Auswahlanweisung, die mit `request.getParameter("wo")` den übergebenen Parameter an das JSP verwendet. Der Index `wo` ergibt sich aufgrund der Übergabe eines Felds mit diesem Namen mit `GET`. Der Wert von `request.getParameter("wo")` wird in einen primitiven Wert vom Typ `int` konvertiert und in einer `switch-case`-Konstruktion getestet. Je nach Wert wird ein spezifisches Ergebnis an den Client gesendet. Beachten Sie, dass wir in diesem einfachen Beispiel kein XML, sondern reinen Text (MIME-Typ `text/plain`) senden. Aber das ist keine Einschränkung der grundsätzlichen Funktionalität. Im späteren Verlauf des Buchs werden wir die Erweiterung auf HTML und XML vornehmen.

Tipp

Neben der oben gezeigten Variante könnten Sie für einige Browser in der HTML-Seite für das Listenfeld im Webformular auch folgende Variante notieren:

Listing 2.8: Eine alternative Variante des Listenfelds (laender2.html)

```
<select name="bundesland" size="1">
<option onClick="sndReq2(1)">Baden-Württemberg</option>
<option onClick="sndReq2(2)">Bayern</option>
<option onClick="sndReq2(3)">Berlin</option>
<option onClick="sndReq2(4)">Brandenburg</option>
<option onClick="sndReq2(5)">Bremen</option>
<option onClick="sndReq2(6)">Hamburg</option>
<option onClick="sndReq2(7)">Hessen</option>
<option onClick="sndReq2(8)">Mecklenburg-Vorpommern</option>
<option onClick="sndReq2(9)">Niedersachsen</option>
<option onClick="sndReq2(10)">Nordrhein-Westfalen</option>
<option onClick="sndReq2(11)">Rheinland-Pfalz</option>
<option onClick="sndReq2(12)">Saarland</option>
<option onClick="sndReq2(13)">Sachsen</option>
<option onClick="sndReq2(14)">Sachsen-Anhalt</option>
<option onClick="sndReq2(15)">Schleswig-Holstein</option>
<option onClick="sndReq2(16)">Thüringen</option>
</select>
```

Der wesentliche Unterschied zur ersten Version besteht darin, dass hier bei jedem `<option>`-Container ein eigener Eventhandler `onClick` notiert wird, über den der aufgerufenen Funktion ein jeweils individueller Parameterwert mitgegeben werden kann. Dies vereinfacht die JavaScript-Funktion zum Anfordern der Daten wie folgt:

Listing 2.9: Die vereinfachte JavaScript-Funktion zum Anfordern der Daten

```
function sndReq2(klick) {
  resObjekt.open('get', 'laender.jsp?wo='+klick,true);
  resObjekt.onreadystatechange = handleResponse;
  resObjekt.send(null);
}
```

In der JavaScript-Funktion muss nicht getestet werden, welchen Eintrag der Anwender angeklickt hat. Leider versteht der Internet Explorer aber nicht, mit Eventhandlern bei einem `<option>`-Tag umzugehen. Deshalb werden die JavaScript-Funktionen mit dem bereits individuell gesetzten Wert für den Übergabewert dort nicht aufgerufen. In Browsern wie Firefox oder Opera funktioniert diese Variante jedoch einwandfrei.

2.2.3 Ein paar Ideen zur Erweiterung bzw. Abwandlung

Neben dem gerade realisierten Beispiel lassen sich – sofern Sie HTML einigermaßen beherrschen oder zumindest unser bisheriges Beispiel verstehen und anpassen können – leicht Modifikationen finden, die in der Praxis sinnvoll sind. Hier sind noch zwei Ideen, die Sie bei Zeit und Muße versuchen können:

1. Nehmen Sie das Beispiel in diesem Kapitel als Basis und modifizieren Sie es so, dass Sie einem Anwender eine Auswahlliste mit Bildnamen anbieten. Je nach ausgewähltem Bild soll dieses dynamisch nachgeladen und in der Webseite angezeigt werden.
2. Als zweite Modifikation des Beispiels in diesem Kapitel könnten Sie eine Auswahlliste mit zwei Einträgen bereitstellen. Der Anwender soll auswählen können, ob er einen Tag einer Arbeitswoche (5 Tage) oder einer vollen Woche (7 Tage) auswählen möchte. Je nach Auswahl zeigen Sie im nächsten Schritt ein Listenfeld mit fünf (Montag bis Freitag) oder mit sieben Einträgen (Montag bis Sonntag) an, aus dem der Anwender dann wieder einen Eintrag auswählen kann. Die weitere Ausarbeitung dieses Benutzerdialogs brauchen Sie aber nicht zu verfolgen.

2.3 Einige kritische Bemerkungen zu AJAX

AJAX wird das Web und das Anwenderverhalten dort revolutionieren. Das steht außer Zweifel. Anwender werden in Zukunft auch von Webapplikationen ein Antwortverhalten erwarten, wie sie es von Desktop-Applikationen kennen. AJAX ist für diese zukünftige Erwartungshaltung Auslöser wie auch die Technologie zur Lösung. Aber insbesondere im Internet kann ein Antwortverhalten wie bei einer Desktop-Applikation nicht garantiert werden. Je nach Qualität der Netzwerkverbindung, der Belastung des Servers und einiger weiterer Faktoren kann eine AJAX-Anfrage einmal nahezu in Echtzeit eine Antwort liefern, aber ebenso gut auch wieder erheblich verzögert eintreffen. Das muss vom Ersteller einer Webseite einkalkuliert werden und sollte auch Besuchern auf smarte Art (Fortschrittsanzeigen oder entsprechende freundliche Hinweise bei größeren Datenmengen) kenntlich gemacht werden. Andererseits ist es aber auch ein Kernkriterium von AJAX, dass die Kommunikation zwischen Client und Server nicht von einer Reaktion oder Kenntnisnahme durch den Anwender beeinflusst sein muss und oft auch nicht soll. Denken Sie noch einmal an Google Suggest. Der Anwender drückt eine Taste und erhält Zusatzinformationen vom Server, ohne dass es ihm bewusst sein soll, ob die Information bereits lokal da war oder nachgeladen werden musste. Eine Eieruhr nach der Eingabe jedes einzelnen Zeichens (eventuell mit blockierter Tastatur) bei hängender Netzwerkverbindung ist sicher keine geeignete Vorgehensweise. Das Dilemma kann man auch mit »Wasch mich, aber mach mich nicht nass« umschreiben. Webentwickler werden hier neue Herausforderungen in der Benutzerführung zu lösen haben.

Und AJAX ist sicher nicht die Lösung aller Probleme im Web. Im Gegenteil, AJAX wirft sogar eine Reihe neuer Probleme auf. Da ist erst einmal das übliche Dilemma

beim Auftauchen einer neuen, hippen Technik. Zig Leute wollen zeigen, dass sie genauso hipp sind, dass sie den neuen supercoolen Zug erwischt haben und dass sie eine Menge davon verstehen ;-). Mit anderen Worten: Es werden in der Anfangszeit eine Menge Webseiten entstehen, die AJAX verwenden, obwohl es nicht notwendig ist. Doch AJAX ist kein Selbstzweck. AJAX wird – wie jede neue Technologie[33] – erst einmal durch eine Spiel- und Experimentierphase gehen müssen, bevor sich deutlich herauskristallisiert, wofür man AJAX einsetzen sollte und wofür nicht.

Konzeptionell werden AJAX-Applikationen zudem Lösungen für eine Reihe von Themen bieten müssen, die aufgrund der Art der Datenpräsentation im Client von Bedeutung sind. Das beginnt mit der Unterstützung des barrierefreien Internets (unter Umständen durch alternative Webseiten) und geht bis hin zur Indizierung durch Suchmaschinen, die mit dynamisch generierten Inhalten sowieso schon Probleme haben. Bei dynamisch veränderten Teilbereichen von Webseiten werden die Probleme noch wachsen. Ein echtes Problem der AJAX-Technologie ist die Tatsache, dass die Grundfunktionalität der ZURÜCK- Schaltfläche im Browser damit ausgehebelt wird. Das Drücken der Schaltfläche stellt in der Regel bei einer vorangegangenen AJAX-Datenanforderung nicht den vorherigen Zustand der Anwendung wieder her, da Browser für gewöhnlich nur statische Seiten in ihrer Historie verwalten. Um der Erwartungshaltung des Anwenders zu genügen (ein Drücken der ZURÜCK-Schaltfläche macht seine zuletzt ausgeführte Aktion wieder rückgängig), wurden über die Zeit verschiedene Lösungen entwickelt, die aber teilweise recht trickreich programmiert werden müssen. So genannte AJAX-Frameworks bieten hier Unterstützung[34]. Auch gibt es bei AJAX-Anwendungen unter Umständen Probleme, zuverlässige Lesezeichen im Browser zu setzen. Aber auch dazu existieren mittlerweile Workarounds in den verschiedenen AJAX-Frameworks.

Ein grundsätzliches Problem hat AJAX offensichtlich, da neben dem eigentlichen Prozessfluss einer Applikation auf einem Server auch clientseitige Programmierlogik in Form von JavaScript verwendet werden muss. Damit werden neben der reinen Datenanforderung und Bereitstellung auch Techniken aus dem DHTML-Umfeld programmiert. Wie bereits vorher erwähnt, ist zwar die Zeit vorbei, in der viele Anwender JavaScript deaktiviert haben. Aber eine JavaScript-Unterstützung kann immer noch nicht zu 100% garantiert werden. Ein Dilemma ist auch, dass AJAX-Applikationen sehr intensiv auf DHTML aufsetzen. Zwar werden DHTML-Effekte und Style Sheets in den heutigen Browsern recht gut unterstützt, aber es gibt einige Browser, in denen die Anwender bestimmte Effekte von Style Sheets individuell einstellen können. Das kann zu unvorhersehbaren Nebeneffekten in Applikationen führen, die das Layout (insbesondere Positionierung von Objekten und die Größeneinstellungen von

33 *Das möglicherweise schlimmste Beispiel für eine überwiegend überflüssige und nervige Verwendung einer im Grunde sehr guten Technologie ist Flash – zumindest in der Anfangsphase, als Flash total in Mode kam. Es hat lange gedauert, bis die Spieler kapiert haben, dass ein Flash-Intro auf einer Webseite fast immer nur nervt. Aber mittlerweile wird Flash meist dosiert eingesetzt und kann dann seinen vollen Nutzen entfalten.*

34 *Wir werden aber auch selbst ein Beispiel erstellen, mit dem die Funktionalität der ZURÜCK-Schaltfläche bei einer AJAX-Applikation nachgebildet wird (siehe dazu Kapitel 9).*

Schriften) über Style Sheets regeln. Und insbesondere im Fall des Internet Explorers muss bemängelt werden, dass die Erzeugung eines `XMLHttpRequest`-Objekts auf der extrem sicherheitskritischen ActiveX-Technologie beruht, die die meisten sicherheitsbewussten Anwender deaktiviert haben. Schlimmer ist aber, dass die Webbrowser leider auch heute noch zahlreiche Eigenheiten aufweisen, so dass jede AJAX-Applikation grundsätzlich umfangreich getestet werden muss.

Und zu guter[35] Letzt sollte jedem Anwender von AJAX-Technologie bewusst sein, dass bei einem Anwender ziemlich aktuelle Browser vorausgesetzt werden. Selbst keineswegs antike Browser wie der Netscape Navigator 6 oder der Internet Explorer 5.5 sind hier mit einer vollständigen Unterstützung sämtlicher Techniken überfordert (obwohl der reine asynchrone Datenaustausch auch in einigen älteren Browsern schon funktioniert).

2.4 Einige interessante AJAX-Projekte

AJAX findet in der Praxis mehr und mehr Anwendungen. Es gibt mittlerweile eine ganze Reihe größerer Webprojekte, die AJAX einsetzen oder deren hauptsächliche Funktionalität unmittelbar darauf beruht. Nachfolgend finden Sie eine Liste mit solchen Projekten. Diese Liste ist selbstverständlich unvollständig und stellt nur eine schlaglichtartige kleine Auswahl dar.

2.4.1 AJAXWhois

Whois ist ein schon sehr altes Protokoll bzw. ein bereits sehr lange gebräuchlicher Dienst im Internet. Damit lassen sich Informationen zu Internetdomains und IP-Adressen und deren Eigentümern abfragen. **AJAXWhois** gestattet den Zugang zu diesen Informationen nun auf Basis von AJAX.

2.4.2 Google Maps

Bei **Google Maps** werden Karten im Internet bereitgestellt, die mittels AJAX aktualisiert werden, wenn ein Anwender neue Informationen in der Karte sehen will, zum Beispiel wenn der Anwender die Karte verschiebt oder Teile der Karte vergrößert oder verkleinert. Derzeit werden aber nur Karten aus den USA und Kanada zur Verfügung gestellt.

35 Oder schlechter ;-)

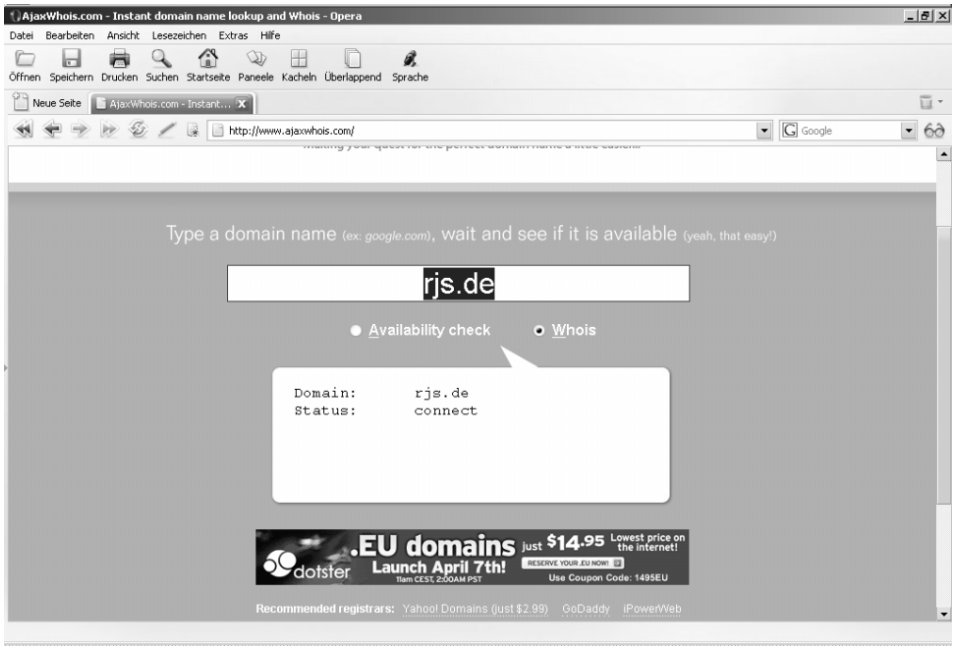


Abbildung 2.22: Whois mit AJAX

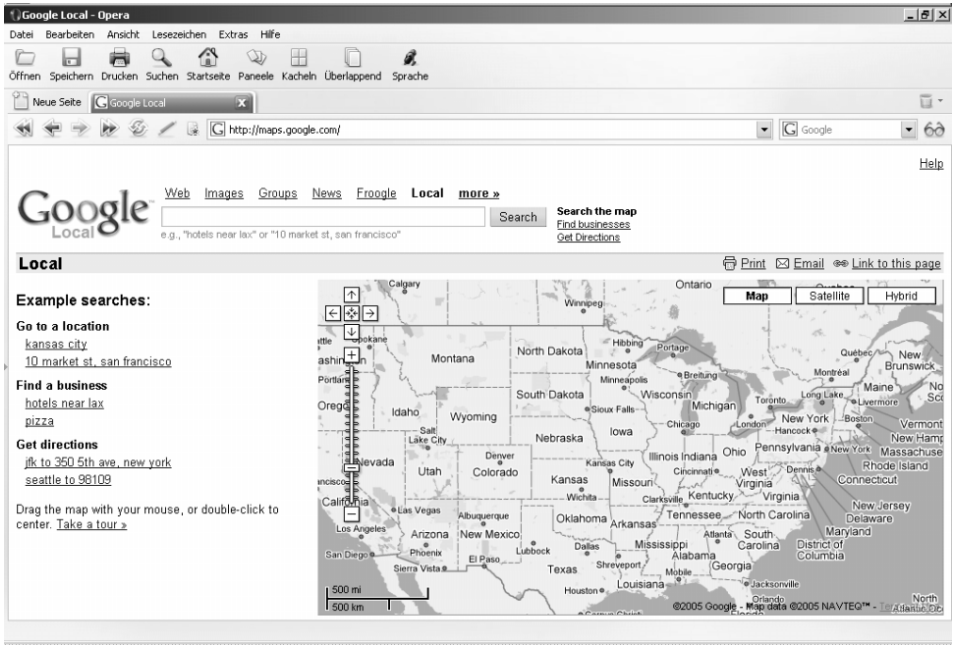


Abbildung 2.23: Auch Google Maps setzt auf AJAX.

2.4.3 Yahoo Maps

Was Google kann, möchte Yahoo schon lange können. So gibt es mit **Yahoo Maps** eine direkte Konkurrenz zu Google Maps von Yahoo. Auch hier werden dynamische Landkarten mittels AJAX realisiert. Das Projekt ist zum Zeitpunkt der Bucherstellung noch in einer Beta-Phase und nur in Englisch verfügbar. Vor allem gibt es so gut wie kein Kartenmaterial von Europa. Insbesondere Deutschland fehlt (noch).

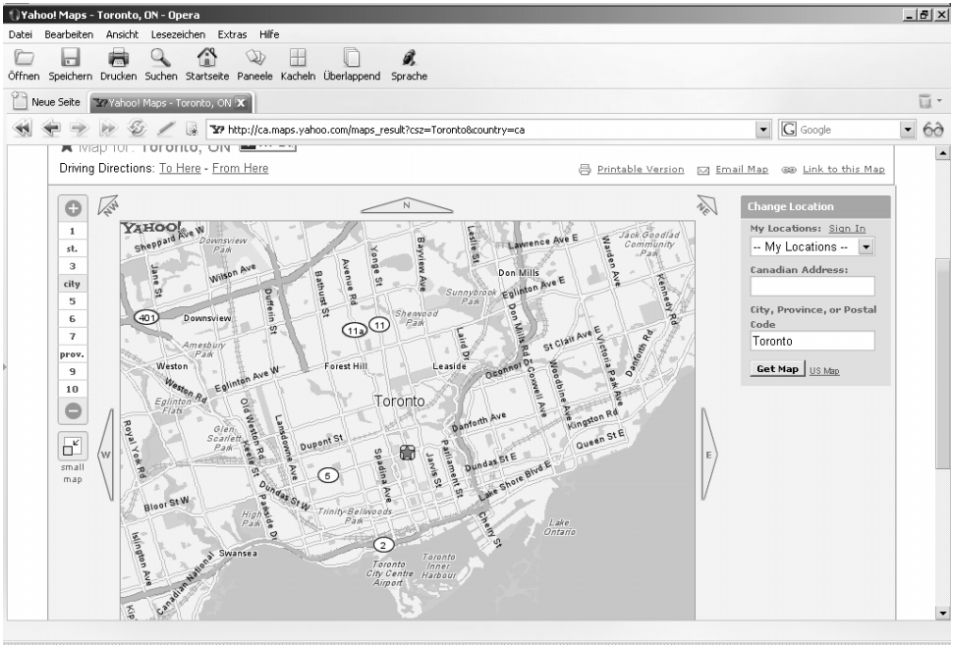


Abbildung 2.24: Yahoo Maps

2.4.4 Google Pages

Bei **Google Pages** handelt es sich um einen auf AJAX basierenden Editor für Webseiten (früher unter dem Begriff **Trogdor** bekannt). Mit Google Pages kann man online eine Website zusammenklicken, die auch gleich bei Google gehostet wird. Benötigt wird lediglich ein Google Account.

2.4.5 Writely

Writely ist eine webbasierende Textverarbeitung, die auf AJAX aufbaut, um die notwendige Performance zu erreichen. Man kann Microsoft- (*.doc*, *.rtf*), OpenOffice- (*.sxw*, *.odt*), Text- (*.txt*) und HTML-Dokumente hochladen und weiterbearbeiten. Writely gehört mittlerweile zum Google-Imperium.

2.4.6 Yahoo Flickr

Flickr ist eine kommerzielle Web-2.0-Webanwendung, die stark auf AJAX setzt. Benutzern ist es möglich, direkt digitale Bilder mit kurzen Kommentaren auf die Webseite zu stellen. Die Bilder können per E-Mail oder vom Fotohandy übertragen und auf andere Seiten verlinkt werden.

2.4.7 Google suggest

Schon mehrfach erwähnt wurde die Erweiterung der normalen Google-Suchmaschine mit Namen **Google suggest**. Es handelt sich dabei um eines der wichtigsten Referenzprojekte für AJAX überhaupt und damit den wahrscheinlich wichtigsten Auslöser für den AJAX-Boom.

2.4.8 Microsoft Live.com

Microsoft Live.com ist Microsofts neue Suchmaschine, die MSN ablösen soll. Auffälligstes Merkmal ist die Gestaltung mit AJAX und anderen Web-2.0-Elementen. Die Seite kann vom Anwender selbst verändert und per Drag&Drop verschoben werden. Zum Zeitpunkt der Bucherstellung steht nur ein eingeschränktes Beta-Stadium zur Verfügung. Viele wichtige Browser werden zurzeit schlecht oder gar nicht unterstützt.

2.4.9 Geonames.org

Geonames.org stellt Landkarten bereit, die ähnlich wie Google maps dynamisch per AJAX nachgeladen und angezeigt werden. Aber hier gibt es auch für Europa und Deutschland sehr genaues Kartenmaterial. Und die Applikation bietet noch mehr: eine frei verfügbare geografische Datenbank, die für eigene Anwendungen zahlreiche kostenlose geografische Webservices bereitstellt. Diese können direkt aus JavaScript heraus verwendet und damit in AJAX-Applikationen integriert werden.

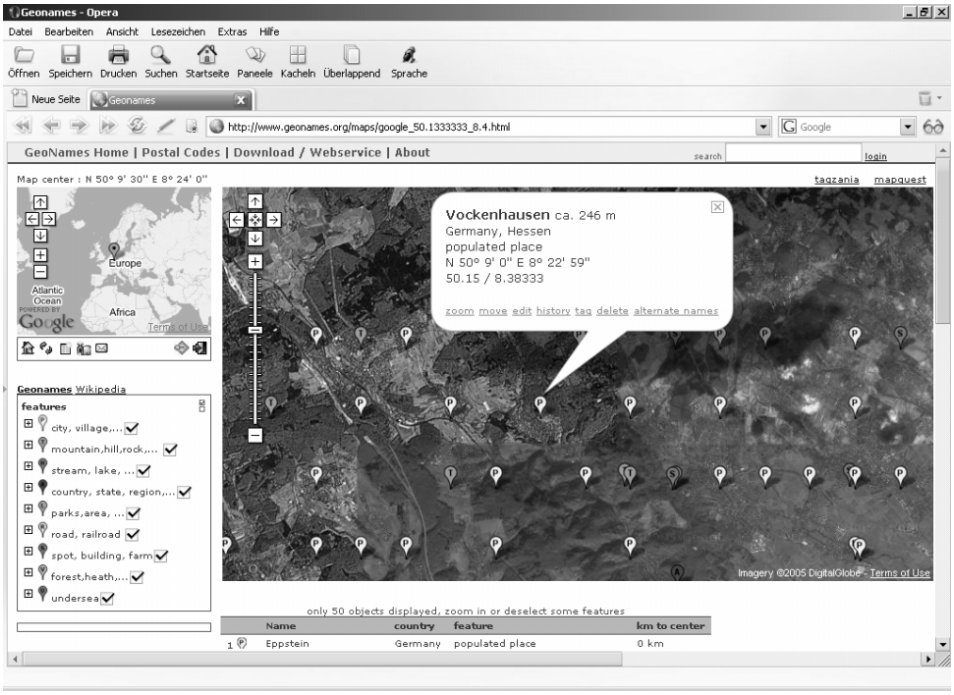


Abbildung 2.25: Geonames.org ist eine sehr interessante AJAX-Applikation, die auch für eigene Applikationen verwendet werden kann.

2.5 Zusammenfassung

Sie kennen nun die wichtigsten Argumente für AJAX und haben auch potenzielle Probleme gesehen. AJAX ist immer noch im Fluss und wird sich auch die nächsten Jahre permanent weiterentwickeln. Insbesondere die Probleme von AJAX werden im Laufe der Zeit durch Bibliotheken und Frameworks mit vorgefertigten Funktionen, die Entstehung von Communities und Foren sowie die immer weiter wachsende Unterstützung in den Browsern abnehmen. Aber das Fundament der Technologie steht und ist mittlerweile etabliert. Was die technischen Hintergründe von AJAX angeht, haben Sie mit einer ersten Beispielapplikation bereits alle wesentlichen Schritte gesehen. Auch wenn Ihre AJAX-Applikationen über die Zeit immer komplexer und leistungsfähiger werden – die Vorgehensweise ist stets identisch zu unserem recht primitiven ersten Beispiel. Sie werden eine weitgehend konventionelle Webseite erzeugen, die Daten über Eventhandler oder ein globales Eventhandling in bestimmten Situationen nachlädt. Die nachgeladenen Daten werden über DHTML-Techniken in der Webseite angezeigt. Für den Webserver ist es dabei vollkommen irrelevant, ob die Daten von einer AJAX-Engine angefordert werden oder dem Browser selbst. Auf Serverseite können Sie mit beliebigen Programmiertechniken arbeiten und die Daten so zum Client schicken, als ob Sie diese direkt in den Browser übermitteln wollen. Die

hauptsächliche Arbeit unter JavaScript besteht darin, ein `XMLHttpRequest`-Objekt zu erzeugen und dessen Methoden und Eigenschaften anzuwenden. Über dieses Objekt können Sie Daten am Browser vorbei vom Server anfordern und diese dann für einen Zugriff aus JavaScript bereitstellen.

3 (X)HTML) und DOM für AJAX-Programmierer

Die Basis des WWW ist ohne Zweifel HTML respektive dessen neuere Variante XHTML. Auch wenn Sie mit AJAX Daten von einem Webserver anfordern, machen Sie das aus einer (X)HTML-Seite heraus. Und letztendlich werden die angeforderten Daten wieder im (X)HTML-Kontext durch einen Webbrowser dargestellt, selbst wenn sie vom Webserver als reiner Text oder XML geschickt wurden. Kenntnis von HTML ist das absolut notwendige Fundament. Am besten sollten Sie sogar über Erfahrungen mit XHTML verfügen. Ein grundlegendes Verständnis von (X)HTML soll – wie bereits angedeutet – nun beim Leser jedoch vorausgesetzt werden. Wir fassen hier kurz die wichtigsten Details zusammen und arbeiten Unterschiede zwischen HTML und XHTML heraus. Schwerpunkt dieses Abschnitts wird der AJAX-Blick auf (X)HTML. Dies bedeutet die Behandlung der wichtigsten Container-Tags, wie sie aus AJAX zum Austausch von Inhalt verwendet werden, der Eventhandler, die zum Auslösen von Anfragen verwendet werden, sowie der Hintergründe zum Document Object Model (DOM) aus Sicht von (X)HTML.

3.1 Der Aufbau von (X)HTML-Dateien

(X)HTML-Dateien selbst bestehen immer aus reinem Text (ASCII bzw. Unicode). Damit sind (X)HTML-Dokumente insbesondere plattformunabhängig und Sie benötigen zur Erstellung nur einen primitiven Klartexteditor.

Tipp

Ein sehr beliebter, kompakter und kostenloser HTML-Editor ist Phase 5 (*http://www.ftp-uploader.de/*). Aber auch das OpenSource-Programm Nvu (ehemals Netscape Composer – *http://www.mozilla.nightrat.net/nvu/ bzw. http://www.nvu-composer.de/*) gewinnt immer mehr Freunde und geht in seiner Leistungsfähigkeit noch viel weiter. Es bietet unter anderem einen WYSIWYG[1]-Modus, aber auch einen hervorragenden CSS-Editor.

1 *What You See Is What You Get*

Abbildung 3.1: Der OpenSource-HTML-Editor Nvu

Eine (X)HTML-Datei muss jedoch im Browser interpretiert werden, um dem Dokument eine über reinen Text hinausgehende Bedeutung zu verleihen. Webbrowser wurden nun von Anfang an so konzipiert, dass sie im Fall von HTML nach dem **Prinzip der Fehlertoleranz** arbeiten. Damit können auch syntaktisch fehlerhafte Dokumente im Client weitgehend ausgewertet und optisch aufbereitet werden. Das Prinzip der Fehlertoleranz vereinfacht die Bereitstellung von Informationen erheblich und hat in der Anfangszeit ohne Zweifel erst den Erfolg des Webs ermöglicht. Allerdings verlieren solche nur lose reglementierten Klartextdokumente die Möglichkeit, zuverlässig von automatischen Analysesystemen (Parsern) ausgewertet zu werden. Und natürlich geht ein Stück Information verloren, wenn es keine Eindeutigkeit von Strukturen gibt. Bei XHTML wurde deshalb das Prinzip der Fehlertoleranz explizit abgeschafft!

(X)HTML verfügt nun im Gegensatz zu vollständigen Programmier- oder Skriptsprachen (wie etwa JavaScript) über keine Kontrollstrukturen in Form von Bedingungen, Sprüngen oder Schleifen. Ebenso werden Sie in (X)HTML keine Variablen finden[2]. Es gibt ebenso keine Befehle im Sinne von Befehlswörtern, die eine Aktion auslösen. Allerdings werden in HTML ab der Version 4 eine Reihe an Schlüsselwörtern bereitgestellt, die Voraussetzung für das Aufrufen von Funktionen sind (die angedeuteten Eventhandler, die wir gleich genauer betrachten). Besagte HTML-Version 4, die bereits seit 1997 verfügbar ist, ist auch der letzte Entwicklungsschritt von HTML gewesen und derzeit im Web trotz des schon antik zu nennenden Alters immer noch

2 *Im engeren Sinn. Allerdings können Sie Formularfelder als eine Form von Variablen sehen. Diese können aber nicht mit HTML selbst verwertet werden.*

aktuell. Allerdings gibt es seit Januar 2000 XHTML, das als Ablösung von HTML konzipiert wurde. XHTML 1.0, das im Grunde HTML 5 darstellt, ist eine auf XML basierende Neuformulierung von HTML 4.01 und enthält dabei alle Elemente von HTML 4.01. Nur ist die Syntax von XHTML bedeutend strenger als von HTML. Die Konsequenz ist, dass ein nicht XHTML-fähiger Webbrowser XHTML-Dokumente jederzeit richtig darstellen kann. Für ihn erscheinen sie als normales HTML. Im Fall einer unklaren Situation wird das Prinzip der Fehlertoleranz genutzt. Gleichzeitig kann XHTML von neueren Browser gemäß den strengen XHTML-Regeln verarbeitet werden. XHTML hat sich jedoch faktisch bis heute in der Web-Community **nicht** durchgesetzt, da die wesentlichen Vorteile von XHTML (strengere Syntax und Reduzierung nichtstandardisierter Anweisungen) dem normalen Webdesigner oder auch Hobby-Webseitenersteller kaum verständlich sind. Und da die Masse der Webseitenersteller einfach HTML statt XHTML (oder eine Mischform) einsetzt[3], werden sich Browser-Hersteller hüten, die HTML-Unterstützung einzustellen und die strengen XHTML-Regeln einzufordern.

3.2 Steueranweisungen

(X)HTML-Anweisungen bestehen aus Steueranweisungen, die aus so genannten **Tags** aufgebaut sind. Ein Tag beginnt immer mit einer geöffneten spitzen Klammer < und endet mit der geschlossenen spitzen Klammer >. Im Inneren der beiden Klammern befindet sich der konkrete Befehl. Ein Tag sieht von der Struktur her immer so aus:

Listing 3.1: Ein schematischer HTML-Tag

```
<Anweisung>
```

Hinweis

Ein Tag kann unter HTML sowohl klein als auch groß geschrieben werden. Auch Mischen von Groß- und Kleinbuchstaben ist erlaubt. Das hat keine Auswirkung. **Dies gilt jedoch nicht für XHTML**. Dort werden Anweisungen ausschließlich klein geschrieben. Um nun Webseiten konform zu den offiziellen Empfehlungen des W3C[4] und den strengen XHTML-Regeln zu erstellen, sollten Sie in neuen Seiten Tags ausschließlich klein schreiben.

3 *Und auch ich muss zugeben, dass ich mich nicht immer an die strengen XHTML-Regeln halte. Oft aus Faulheit, oft aber auch aus Vergesslichkeit. Wenn man nicht gezwungen wird und solche Lässigkeit keinerlei negative Konsequenzen hat, siegt halt die Schludrigkeit. Das ist menschlich ;-).*

4 *Das W3C – World Wide Web Consortium (http://www.w3c.org) – standardisiert nahezu alle relevanten Techniken im Web.*

In (X)HTML gibt es zwei Formen von Tags:

1. Einen einleitenden Tag (Anfangs-Tag oder Beginn-Tag genannt)
2. Einen beendenden Tag (Abschluss-Tag oder Ende-Tag genannt)

Der einleitende Tag eröffnet eine Anweisung, während der beendende Tag sie wieder schließt. Beide Tags sehen fast identisch aus, außer einem vorangestellten Zeichen, mit dem der beendende Tag zusätzlich beginnt – dem Slash (Schrägstrich) `/`. Wenn beide Tags angegeben werden, bilden sie immer einen Container (in XML **Element** genannt). Dies bedeutet, die im einleitenden Tag angegebene Anweisung (etwa eine Formatierung) wirkt sich auf sämtliche Dinge (Objekte) aus, die sich im Inneren des Containers befinden. Dies wird in vielen Fällen ein Text sein, es kann sich aber auch um Grafik oder andere Multimediaobjekte handeln. Der Ende-Tag hebt die Wirkung eines Anfangs-Tag auf.

In reinem HTML werden die meisten Tags paarweise vorkommen. Es gibt jedoch Situationen, wo HTML-Tags keine Container bilden. In einigen Fällen greift das Prinzip der Fehlertoleranz. Das betrifft die Tags, die nach offizieller Vorgabe paarweise auftreten müssten, aber deren Wirkungsende sich auch ohne Abschluss-Tag auf Grund einer anderen Situation eindeutig ergibt (etwa bei Listen – ein neuer Listeneintrag beendet den vorherigen). Der andere Fall betrifft die HTML-Tags, die gar keinen offiziellen Abschluss-Tag haben (etwa ein Zeilenvorschub mit dem Tag `<br>`). In XHTML muss aber jeder Tag mit einem Abschluss-Tag versehen werden oder als so genanntes **leeres Element** definiert werden[5] (also im Fall eines Zeilenumbruchs so: `<br />`).

Achtung

Der Zwang zur richtigen paarweisen Verwendung von Tags oder der expliziten Auszeichnung als leeres Element erweist sich vor allem dann als wichtig, wenn Sie eine Webseite mittels JavaScript und des DOM-Konzepts nachträglich manipulieren wollen. Falls hier keine korrekte Auszeichnung vorgenommen wurde, kann es zu Problemen kommen (hauptsächlich wenn Sie ein Ende-Tag weglassen – ein leeres Element wie einen `<img>`-Tag nicht als solches zu kennzeichnen, macht im derzeitigen WWW nach meiner Erfahrung keinerlei Schwierigkeiten[6]).

Container bzw. Elemente können beliebige – sinnvolle – andere Tags (**Kindelemente**) enthalten, aber die Reihenfolge der Auflösung sollte eingehalten werden!

5 Im Sinn von XML – das behandeln wir im XML-Abschnitt.

6 Rein um sich einen sauberen Stil anzugewöhnen, ist eine Kennzeichnung dennoch sinnvoll.

Beispiel:

Listing 3.2: Ein kursiver, unterstrichener Text

```
<i></u>Es wird Sommer</i></u>
```

Wenn ein Container weitere Container enthält, sollten diese wieder von innen nach außen beendet werden – in umgekehrter Reihenfolge der Einleitungs-Tags. **In XHMTL ist das auch zwingend.**

3.3 Attribute

Viele Tags sind erst dann sinnvoll einzusetzen, wenn sie genauer spezifiziert werden. Nicht jeder, denn beispielsweise ein Zeilenumbruch ist auch ohne die genauere Spezifizierung in (X)HTML durch `<br />` vollständig beschrieben. Aber nicht alle Anweisungen sind eindeutig. Parameter bzw. Attribute erweitern einen einleitenden (X)HTML-Tag und spezifizieren damit genauer die Bedeutung des Tag. Der Abschluss-Tag wird niemals mit Parametern erweitert. In HTML gibt es zwei Formen von Parametern:

1. Parameter mit einer Wertzuweisung
2. Parameter, die bereits einen Wert repräsentieren

Parameter mit einer Wertzuweisung bekommen über einen Zuweisungsoperator – das Gleichheitszeichen (=) – den entsprechenden Wert zugeordnet. Dies kann ein Text oder eine Zahl oder auch ein URL sein. Ein Tag mit einem Parameter mit einer Wertzuweisung sieht schematisch so aus:

Listing 3.3: Schema eines HTML-Tags mit Parameter und Wertzuweisung

```
<[Anweisung] [Parameter] = "[Wert]">
```

Viele Befehle lassen sich über mehr als einen Parameter spezifizieren. Diese werden dann einfach durch Leerzeichen getrennt aufgelistet. Bei mehreren Parametern spielt die Reihenfolge der Parameter keine Rolle.

Hinweis

In fast allen Fällen kann man bei der Wertzuweisung in einem HTML-Tag auf Hochkommata verzichten. Die Anweisung `<font color=red>` funktioniert in HTML uneingeschränkt. **In XHTML muss der zugewiesene Wert jedoch zwingend in Hochkommata eingeschlossen werden.**

Parameter, die bereits einen Wert repräsentieren, brauchen in HTML bloß durch ein Leerzeichen von der Anweisung oder anderen Parametern abgetrennt (!) in den Einleitungs-Tag geschrieben zu werden. Sie fungieren immer als Schalter. Wenn sie angegeben werden, wird eine Eigenschaft aktiviert, fehlen sie, ist die jeweilige Eigenschaft deaktiviert. Ein Tag mit einem Parameter, der bereits einen Wert repräsentiert, sieht schematisch so aus:

Listing 3.4: Schema für einen Parameter ohne Wertzuweisung

```
<[Tag] [Parameter]>
```

Ein Beispiel wäre `<table border>`, womit ein Tabellenrahmen gesetzt wird.

Achtung

In XHTML gibt es keine Parameter ohne Wertzuweisung.

Hinweis

Aus Sicht von AJAX ist ein Parameter besonders wichtig – `id`. Über die ID wird ein Bereich der Webseite identifiziert. Darüber kann z. B. dessen Inhalt ausgetauscht werden. Ebenso kann der Parameter in Zusammenhang mit Style Sheets eingesetzt werden. Ein weiterer wichtiger Parameter ist `name`. Damit vergeben Sie einen Namen für ein Element. Auch über diesen Namen können Sie ein Element einer Webseite aus DHTML heraus ansprechen. Zudem wird darüber bei Webformularen die Zuordnung von zu versendenden Werten vorgenommen. Und ebenso ist `class` wichtig, um eine Style Sheet-Klasse zuzuweisen.

3.4 Strukturierung und Gestaltung mit HTML

Obwohl HTML über die Zeit zahlreiche Gestaltungsmöglichkeiten für eine Webseite erworben und die Webdesigner diese in der Vergangenheit auch exzessiv ausgelebt haben, geht die aktuelle Tendenz der Webseitengestaltung dahin, möglichst alle Layoutfragen einer Webseite nicht mehr mit HTML zu lösen. Stattdessen wird das Layout vollkommen in – möglichst externe – Style Sheets (Formatvorlagen) ausgelagert oder zumindest mit Style Sheet-Parametern bei Tags gearbeitet. HTML wird nur noch zu Strukturierung einer Webseite verwendet und nicht mehr in dem Sinn, mit Befeh-

len wie <font size="7"> eine Schriftgröße festzulegen oder mit <b> einen Text fett auszuzeichnen. HTML-Befehle zur Angabe von Überschriften oder Absätzen strukturieren zwar immer noch eine Webseite, aber auch deren Layoutwirkungen werden über Style Sheets modifiziert oder gleich ganz ausgeschaltet. Dieser Schritt ist ein Teil des Bemühens, das WWW in ein semantisches Web umzuwandeln, in dem die reinen Inhalte sehr stark formalisiert und damit maschinenlesbar werden.

3.4.1 Gruppierung von Inhalt

Aus Sicht von AJAX sind bei der Strukturierung einer Webseite besonders Tags zur Gruppierung von Inhalt von Bedeutung. Gruppierung von Inhalt kann man in HTML beispielsweise mit Überschriften (<h1> bis <h6>) oder auch Absätzen (<p>) machen, aber besonders wichtig sind die Strukturelemente und . Man kann damit eine Unterstruktur innerhalb einer Webseite aufbauen und diesen Container verwenden, um den darin enthaltenden Inhalt zusammenzufassen und ihn formatieren oder ansprechen (etwa mit JavaScript) zu können. Genau das haben wir im letzten Kapitel gemacht, als wir mit der ID eines solchen Containers den Inhalt ausgetauscht haben (ursprünglich war er sogar leer).

Listing 3.5: Die Zeile 27 aus dem Webformular (laender.html) in Kapitel 2

```
<span id="hs"></span>
```

Darauf kommen wir gleich bei der Behandlung von DOM und später noch bei DHTML noch genauer zurück.

Tipp

Während die meisten Webgestalter meist den -Container bei DHTML-Effekten und beim Datenaustausch mit AJAX einsetzen, bietet sich in vielen Situationen eher der unverständlicherweise ziemlich unterschätzte -Container an. Dieser hat im Gegensatz zu (Zeilenvorschub) keinerlei Eigenwirkung und ist damit für eine Inline-Formatierung ideal. Er ist damit prädestiniert zur Auslagerung von Gestaltung in Style Sheets oder die Ansprache aus Programmier- und Skriptsprachen (etwa JavaScript). Der -Container ist somit logisch bedeutend radikaler bei der Trennung von Struktur, Layout und Funktionalität als der -Container, der unter seiner »Verschmutzung« durch eine HTML-Wirkung »leidet«. Allerdings hat der -Container den Vorteil, dass man Breite und Höhe angeben und damit das Layout einer Seite sehr flexibel gestalten kann. Man wird sinnvollerweise damit arbeiten, wenn sowieso komplette Absätze formatiert werden sollen.

3.5 Formulare zur Benutzerinteraktion

Ein – wenn nicht der – zentrale Aspekt von AJAX ist die Möglichkeit, die Benutzerinteraktion zu optimieren. Und die Interaktion mit dem Benutzer erfolgt in vielen Fällen über Webformulare. Besonders interessant ist für uns, dass nicht nur die Interaktion mit dem Anwender oft über ein Webformular erfolgt, sondern dass ebenso die Anforderung von Daten per AJAX auf der gleichen Technik wie die von Webformularen fußt. In AJAX wird ja in einer bestimmten Situation ein Eventhandler oder ein Ereignis allgemein ausgelöst und eine Anforderung nach Ergänzungsdaten per HTTP weggeschickt. Und das ist für den Browser genau der gleiche Vorgang, der beim Verschicken von Benutzereingaben in einem Webformular durchgeführt wird[7].

3.5.1 HTML-Formular-Tags

Ein Formular in einer Webseite wird im Wesentlichen aus Eingabefeldern, mehrzeiligen Textfeldern, Listen und Aufzählungen, Schaltflächen und beschreibendem Text bestehen. Das Formular besteht aus dem äußeren `<form>`-Container und darin enthaltenen Tags zur Spezifizierung der Eingabeelemente. Die meisten Eingabefelder basieren auf dem Tag `<input>`, der mit dem Attribut `type` genauer spezifiziert wird und damit verschiedene Ausprägungen annehmen kann:

- **Einzeilige Eingabefelder** werden ohne den `type`-Parameter oder mit `type="text"` erstellt.
- Mit `type="password"` definiert man ein **Passwortfeld** mit verdeckter Eingabe.
- Mit `type="button"` erstellen Sie eine **Schaltfläche** (über `value` erfolgt die Beschriftung).
- Mit `type="checkbox"` können Sie ein **Kontrollfeld** (das `value`-Attribut bezeichnet den Wert, der beim Versenden des Formulars übermittelt wird) erstellen.
- Die Typangabe `radio` spezifiziert einen **Radiobutton** (**Optionsfeld**).
- Eine **Schaltfläche zum Zurücksetzen** der Formulareingaben wird über `type="reset"` erstellt, eine **Schaltfläche zum Abschicken** des Formulars über `type="submit"`. Beide Schaltflächen erhalten eine individuelle Beschriftung über `value` (andernfalls wird eine Default-Beschriftung durch den Browser vergeben).

Mehrzeilige Eingabefelder werden mit einem eigenen HTML-Tag realisiert – dem `<textarea>`-Container. **Auswahllisten** in Formularen werden mit einem äußeren `<select>`-Container für die Listenstruktur und jeweils einen inneren `<option>`-Container für jeden Listeneintrag realisiert. Der **äußere** Container wird um das Attribut `name` erweitert, der Tag für den Listeneintrag benötigt keinerlei Parameter. Bei allen zu versendenden Formularfeldern ist das Attribut `name` wichtig. Darüber erfolgt beim Versenden von Benutzereingaben die Zuordnung, welche Information ein Anwender wo

7 *Oder aber auch bei der Anforderung einer Seite über die Adressleiste des Browsers.*

eingegeben hat. Es werden bei den meisten Browsern nur die Inhalte derjenigen Formularfelder verschickt, die mit dem `name`-Parameter spezifiziert sind. Mit anderen Worten – wenn in einem Webformular Formularfelder enthalten sind, die nicht mit dem `name`-Parameter spezifiziert sind, werden dort vorgenommene Eingaben beim Verschicken des Formulars nicht versendet.

Achtung

Sie sollten aufpassen, dass kein Name für ein Webformularelement mehrfach vorkommt. Sie können zwar die Reaktion nicht für jeden Browser vorhersehen, aber in der Regel werden die Inhalte der mehrfach benannten Felder versendet und dann besteht das Problem, auf Serverseite eine richtige Zuordnung vorzunehmen.

Tipp

Die Gestaltung von Formularen erweist sich durch die unterschiedliche Größe von Formularfeldern als etwas diffizil. Zur Anordnung von Formularen bieten sich deshalb Tabellen an. Dabei können sowohl das Formular in die Tabelle als auch die Tabelle in das Formular eingeschlossen werden. Alternativ sieht man in der letzten Zeit verstärkt die Gestaltung über Style Sheets, um vor allem den Anforderungen des barrierefreien Webs zu genügen.

3.5.2 Der ‹form›-Tag – remember AJAX

Innerhalb des `<form>`-Tags werden in der Regel einige, durch ein Leerzeichen getrennte Angaben gemacht, die Sie auch zum Teil bei den Funktionen im AJAX-Umfeld wiederfinden. Besonders wichtig sind folgende Parameter des `<form>`-Tags:

- Eine Adresse, wohin die eingegebenen Daten zur Verarbeitung geschickt werden. Dazu dient die Erweiterung `action=[URL]`. Diese Angabe eines URL finden wir auch bei der `open()`-Methode eines `XMLHttpRequest`-Objekts unter AJAX als zweiten Parameter wieder.
- Eine Methode, wie die Daten, die im Formular eingetragen wurden, zur Auswertungsstelle gelangen. Die Erweiterung `method=[Methode]` legt die Art der Übertragung fest. Die Art und Weise der Übertragung finden wir auch bei der `open()`-Methode eines `XMLHttpRequest`-Objekts unter AJAX als ersten Parameter wieder. Sie werden dort in der Praxis entweder `GET` oder `POST` als Wert finden. Beide Werte des `method`-Parameters beschreiben wir im Kapitel der AJAX-Grundlagen genauer. Die Werte spezifizieren unterschiedliche Möglichkeiten, mit HTTP Daten vom Client zum Server zu schicken.

3.6 Ab in den DOM – aus Sicht von HTML

Das **Document Object Model** (**DOM**) bezeichnet einen Standard des W3C. Er ist wie fast alle Spezifikationen des W3C plattform- und programmiersprachenunabhängig definiert. Das Konzept beschreibt ein Verfahren, bei dem ein Dokument nicht als statisch aufgebaute, fertige und nicht unterscheidbare Einheit, sondern als differenzierbare Struktur betrachtet wird. Deren einzelne Bestandteile sind Programmen und Skripten dynamisch zugänglich. Über das DOM lässt sich ein Dokument als Baumstruktur mit hierarchischer Verschachtelung der Elemente auffassen.

Dieser Ansatz ermöglicht bei Webseiten unter anderem die individuelle Behandlung von Bestandteilen der Webseite auch dann, wenn diese bereits in den Browser geladen ist. Jeder Container, jede Grafik, jede Referenz lassen sich nach dem Laden der Webseite wieder ansprechen. Das ist genau das, was wir in unserem ersten AJAX-Beispiel genutzt haben, als wir mittels der ID den Inhalt von einem `<span>`-Container ausgetauscht haben. Erinnern Sie sich an die Funktion zum Behandeln der zurückgelieferten Daten aus dem Beispiel in Kapitel 2:

Listing 3.6: Die Funktion zum Behandeln der zurückgelieferten Daten

```
19 function handleResponse() {
20   if(resObjekt.readyState == 4){
21     document.getElementById("hs").innerHTML =
22       resObjekt.responseText;
23   }
24 }
```

Das DOM ist aber kein Konzept, das speziell auf HTML-Dokumente gemünzt ist, sondern allgemein auf Klartextdokumente angewendet werden kann, die klare Strukturen aufweisen. Dies betrifft alle auf SGML zurückgehenden Dokumente und vor allem XML-Dokumente, deren Verarbeitung oder Validierung darüber möglich ist. Je nach Dokumenttyp und angewendeter Programmiertechnik nutzt man dabei unter Umständen verschiedene Facetten des DOM.

Unabhängig von der Art der strukturierten Klartextdatei werden beim Laden eines Dokuments durch einen DOM-kompatiblen Parser alle ihm im Rahmen des Konzepts bekannten und einzeln identifizierbaren Elemente bezüglich ihres Typs, ihrer relevanten Eigenschaften und ihrer Position innerhalb des Dokuments indiziert. Dies ist eine Art dynamische Datenstruktur im Hauptspeicher des Rechners, über die später alle indizierten Bestandteile des Dokuments in Form einer Objekthierarchie verfügbar sind. Im DOM werden alle Elemente eines Dokuments, vom Tag über Texte bis hin zum Attribut, als eine Menge zusammenhängender **Knoten** repräsentiert, die einen Baum abbilden und bei der man zwischen Knoten Eltern-Kind-Beziehungen beschreiben kann, wenn diese ineinander geschachtelt sind. Dies wird beispielsweise bei XML über eine Technik mit Namen **XPath** genutzt, um Wege von einem Knoten zu einem

anderen Knoten anzugeben. Aber auch bei CSS kann man angeben, dass sich eine Formatierung nur dann auf ein Element auswirkt, wenn dieses im DOM-Baum auf eine ganz bestimmte Weise verschachtelt ist.

Aber um es an dieser Stelle nicht zu kompliziert zu machen – das DOM ist kurz gefasst ein komplettes Modell, das jedes Element eines Dokuments beinhalten kann[8] und es erlaubt, diese nachträglich zu verwenden. Für JavaScript bedeutet dies, dass das DOM den Zugriff auf alle Bestandteile einer Webseite gewährleistet. Dies erfolgt mit geeigneten Methoden und Eigenschaften über verschachtelte Standardobjekte, benannte Elemente (über den Parameter `id`) oder standardisierte Datenfelder für bestimmte Elementtypen[9]. Um dies aber genauer besprechen zu können, brauchen wir JavaScript. Deshalb werden wir beim Abschnitt über JavaScript auf DOM zurückkommen.

3.7 Ein tolles Ereignis

Explizit zu HTML gehören auch **Eventhandler**. Das sind Schlüsselwörter, die als Attribute bei HTML-Tags notiert werden können und beim Auftreten von einem bestimmten Ereignis[10] eine nachfolgend notierte Funktion (so gut wie immer in JavaScript geschrieben) auslösen. Da Eventhandler vom W3C offiziell in den HTML-Sprachstandard aufgenommen wurden, spielt damit Groß- und Kleinschreibung bei der Notation keine Rolle. Es hat sich aber die Notation von Eventhandlern eingebürgert, klein zu beginnen und die zusammengesetzten Bezeichner bei jedem neuen Begriff mit einem Großbuchstaben zu notieren. Das W3C hat ebenfalls festgelegt, in welchen HTML-Tags welcher Eventhandler vorkommen darf, wobei sich einige Browser-Hersteller an diese Vorgaben nur sehr mangelhaft halten. Als Exempel für die manchmal unterschiedliche Unterstützung von Eventhandler können Sie wieder unser AJAX-Beispiel aus Kapitel 2 heranziehen. Dort hatte ich als Alternative für den Einsatz des Eventhandlers `onClick` bei einem Listenfeld auch die Möglichkeit demonstriert, diesen beim `<option>`-Tag direkt zu verwenden (etwa so: `<option onClick="sndReq2(7)">Hessen</option>`). Das unterstützen zum Beispiel Opera oder Firefox, aber nicht der Internet Explorer[11].

8 *Leider ist es nicht ganz eindeutig, welche Elemente von Webbrowsern bei einer Webseite tatsächlich indiziert werden. Beim Parsen einer Webseite merken sich manche Browser in der Tat jedes Element, andere dagegen nur ausgewählte Elemente. Besonders ältere Browser unterscheiden sich hier gewaltig.*

9 *Alle Formulare in einer Webseite werden etwa in ein Objektfeld mit Namen forms einsortiert, das dem DOM-Objekt document untergeordnet ist. Dieses wiederum ist window untergeordnet. Also erfolgt der Zugriff auf das dritte Formular in einer Webseite über* `window.document.forms[2]` *– die Indizierung beginnt bei 0.*

10 *Etwa wenn ein Anwender auf ein Element einer Webseite klickt oder mit dem Mauszeiger den Bereich eines Webseitenelements (zum Beispiel ein Bild) überstreicht.*

11 *Bei einem Einsatz von Eventhandlern hilft nur, diese in allen relevanten Browsern zu testen. Glücklicherweise passen sich aber die Verhaltensweisen der gängigen Browser immer mehr an.*

3.7.1 Die konkreten Eventhandler

Eventhandler stellen in jedem Fall das wichtigste Bindeglied zwischen HTML und JavaScript (und damit auch AJAX) dar. Ein Eventhandler sieht aus wie jedes gewöhnliche HTML-Attribut. Gemeinhin beginnen die Namen von Eventhandlern mit der Silbe `on`, gefolgt von einer sprechenden Beschreibung des Ereignisses. Hinter dem Namen des Ereignisses wird ein Gleichheitszeichen notiert, gefolgt von einer in Anführungszeichen notierten JavaScript-Anweisung oder -Funktion. Beispiel:

Listing 3.7: Ein Beispiel für den Aufruf einer Funktion mit einem Eventhandler

```
onClick="mFkt()"
```

Innerhalb der Regeln des W3C und der Unterstützung durch die verschiedenen Browser können Eventhandler relativ beliebig in einer Webseite eingesetzt werden, um die Reaktionslogik einer Webseite zu beschreiben. Die wichtigsten Ereignisse, bei denen eine Reaktion sinnvoll implementiert werden kann, sind

- das Laden (`onLoad`) und Verlassen (`onUnLoad`) einer Webseite,
- das Abschicken (`onSubmit`) und Zurücksetzen (`onReset`) von Formulardaten,
- das Überstreichen eines Bereichs der Webseite mit dem Mauszeiger (`onMouseOver` beim Erreichen des Bereichs und beim Verlassen (`onMouseOut`) oder auch allgemein bei der Bewegung der Maus (unabhängig davon, ob die Maustaste gedrückt ist oder nicht – `onMouseMove`),
- der Anwenderklick auf eine Referenz, ein Element der Webseite oder eine Schaltfläche (`onClick`)[12] und
- Änderung (`onChange`), Aktivierung (`onFocus`) und Verlassen (`onBlur`) eines Elements in einer Webseite – hauptsächlich in Zusammenhang mit Formularen.

Wie wir im letzten Kapitel schon angesprochen haben, werden insbesondere bei AJAX auch Tastatureingaben wichtig sein. Dabei kann man zwischen dem vollständigen Tastendruck (`onKeyPress`), dem Drücken einer Taste (`onKeyDown`) und dem Loslassen einer Taste (`onKeyUp`) unterscheiden. Tastaturbezogene Eventhandler werden hauptsächlich bei freien Eingabefeldern in einem Webformular ihren Einsatz finden.

12 *Es gibt natürlich nicht nur den einfachen Klick auf eine Referenz, sondern auch die Auswertung des gleichzeitigen Drückens einer Maustaste, des anschließenden Loslassens einer Maustaste und des Doppelklicks. Auch die jeweiligen Teilphasen bilden jeweils ein eigenes Ereignis. Diese werden über* `onMouseDown`, `onMouseUp` *und* `onDblClick` *beschrieben. Diese Verwendung (insbesondere eines Doppelklicks) spielt im Web aber in der Praxis keine Rolle.*

Hinweis

Mit einem Klick auf eine Referenz ist gemeint, dass Sie auf ein Element wie eine Schaltfläche in einem Formular mit der Maus klicken. Auch ein <a>-Element ist im engen Sinn eine Referenz. In einigen neueren Browsern gelten aber auch die meisten anderen Tags als Referenzen. Ob Anwender nun Überschriften, Bilder, Trennlinien etc. für einen Klick wählen. Im Grunde braucht der HTML-Tag nur noch einen Bereich zu beschreiben, auf den ein Anwender den Mauszeiger bewegen und klicken kann. Sie erweitern den entsprechenden Tag durch das Attribut `onClick`. Allerdings sollten Sie auf jeden Fall verschiedene Browser testen, ob der Eventhandler auch unterstützt wird.

Hinweis

Weitere Eventhandler zur Reaktion auf das Selektieren von Text (`onSelect`) oder die Unterbrechung eines Bildladevorgangs (`onAbort` und `onError`) sind in der Praxis von geringerer Bedeutung.

3.8 Zusammenfassung

Wir haben in diesem Kapitel die bedeutenden Fakten zu (X)HTML rekapituliert und abstrahiert zusammengefasst. Sie kennen nun die wichtigsten Unterschiede zwischen XHTML und HTML, die ersten Details zum DOM-Konzept und können Eventhandler zum Aufruf von JavaScript-Funktionen einsetzen.

4 Style Sheets für AJAX-Programmierer

Wie bereits erwähnt, reduzieren moderne Webseiten HTML fast vollständig auf die reine Strukturierung der Seite, während das Layout gänzlich von **Formatvorlagen** bzw. **Style Sheets** übertragen wird. Style Sheets stellen einmal den – mehr oder weniger erfolgreich umgesetzten – Versuch dar, den unterschiedlichen Interpretationen von Webseiten auf verschiedenen Plattformen einen Riegel vorzuschieben. Über Style Sheets können Sie in Formatregeln das Layout von Webseiten viel genauer als mit HTML festlegen. Die Verwendung von Style Sheets hat aber über die reine Erweiterung der Gestaltungsmöglichkeiten und der reduzierten Interpretationsspielräume durch Browser einen viel bedeutenderen Grund. Style Sheets heben das Vermischen von Gestaltungsbefehlen und Information auf. Es kann eine klare Trennung von Struktur und Layout erreicht werden. Nahezu alle bisherigen HTML-Gestaltungselemente werden bei konsequenter Anwendung von Style Sheets überflüssig.

Aber auch in Hinsicht auf AJAX ergibt der Einsatz von Style Sheets erhebliche Möglichkeiten. Sie können mittels JavaScript und Style Sheets (DHTML) gezielt Layout- oder Positionseigenschaften eines Elements der Webseite verändern. So lassen sich während des nachträglichen Anforderns von Informationen bestimmte visuelle Effekte anzeigen oder kontextbasierende Menüs an der aktuellen Position des Mauszeigers aufblenden, deren Inhalt per AJAX nachgeladen wurde. Es ist sehr nahe liegend, bei AJAX-Applikationen zur Gestaltung des gesamten Layouts der Webseite verstärkt auf Style Sheets und DHTML-Effekte zu setzen, da man ja bereits zum Austauschen und Anzeigen von neuen Informationen einen spezifischen DHTML-Effekt verwendet. Auf der einen Seite kann man also voraussetzen, dass Style Sheets und DHTML funktionieren, wenn AJAX funktioniert. Diese Welten sind untrennbar miteinander verbunden. Auf der anderen Seite gilt aber immer noch, dass Style Sheets im Web nicht **zuverlässig** funktionieren. Dabei ist es weniger das Problem, dass moderne Browser diese nicht unterstützen. Das ist in allen modernen Browsern mittlerweile einigermaßen vernünftig geregelt. Das Problem ist, dass Anwender in flexiblen Browsern wie dem Konqueror oder Mozilla Vorgaben von Style Sheets durch Browser-Einstellungen wieder aufheben können. Das wird bei allen Web-Applikationen ein Problem, die die Position von Objekten und die Größe von Schriften über Style Sheets einstellen. Allerdings muss man auch festhalten, dass man nicht jede

individuelle Veränderung eines Client-Programms durch einen Internet-Server kontrollieren kann und sollte. Man sollte sich heutzutage auch auf einen bestimmten Standard einfach verlassen können und nicht jede Eventualität einkalkulieren müssen.

4.1 CSS, XSL & more

Style Sheets bezeichnen keine eigene Sprache, sondern ein Konzept. Und es gibt nicht nur eine einzige Sprache, um Style Sheets zu definieren, sondern diverse Ansätze bzw. verschiedene Sprachen. Die genauen Regeln und die syntaktischen Elemente für die Style Sheets werden je nach verwendeter Sprache etwas differieren, aber oft ähnlich aussehen.

4.1.1 Von den Daten zur Darstellung

Allgemein liegen bei einer Anwendung von Style Sheets Daten in einer Rohform oder einer nicht gewünschten Darstellungsweise vor, die auf spezifische Weise verändert werden soll. Die Darstellung der Daten erfolgt dann in einer anderen Form, wobei die Informationen selbst erhalten bleiben. Unter Umständen werden allerdings im Ausgabedokument Daten der Quelle unterdrückt und/oder durch Zusatzdaten ergänzt. Die Beschreibung der Transformation bzw. Formatierung erfolgt in der Regel in Form einer externen Datei, kann aber auch in einigen Situationen direkt in die Datei mit den Daten notiert werden (etwa eine Webseite).

Style Sheets verleihen Informationen also ein neues Aussehen. Dazu werden die Daten und die Formatierungsinformationen von einem interpretierenden System zu einer neuen Darstellung verarbeitet. Im Fall des WWW verstehen moderne Webbrowser beispielsweise verschiedene Style Sheet-Sprachen und erzeugen aus (X)HTML bzw. XML und Style Sheets die tatsächliche Webseite. Bei allgemeinen XML-Dokumenten kommen so genannte **Prozessoren** zum Einsatz.

4.1.2 XSL

Eine sehr wichtige Sprache zum Definieren von Style Sheets nennt sich **XSL** (**eXtensible Stylesheet Language**). XSL geht auf XML zurück und fasst zwei Technologien zur Erstellung von Formatvorlagen zusammen – **XSL-FO** (**XSL Formating Objects**) für die Beschreibung eines Dokuments als Baum mit Formatierungsanweisungen und Stilangaben und **XSLT** (**XSL Transformation**) für die Transformation eines beliebigen Dokuments in einen anderen Baum. Indirekt wird auch **XPath** für die Adressierung von Baumbestandteilen zum engeren XSL-Umfeld gezählt. Insbesondere XPath werden wir genauer untersuchen, denn damit kann man bei AJAX Inhalte sehr genau spezifizieren und dies ist der Schlüssel, um Teile einer Webseite auszutauschen.

Einsatzgebiete von XSL

XSL-FO, XSLT und XPath lassen sich gemeinsam oder unabhängig voneinander verwenden:

1. XSL-FO definiert den grundsätzlichen Entwurf neuer Layouts, analog anderer Style Sheet-Sprachen wie CSS. Allerdings wird XSL-FO in der Hauptsache zur Druckvorbereitung verwendet.
2. XSLT kann Daten eines Formats in jede beliebige XML-basierte Form oder aber auch in Formate, die nicht XML-basierend sind, transformieren. Dabei können auch Formatregeln in CSS oder XSL-FO integriert werden. Oft wird mit XSLT ein Datendokument transformiert und in einem zweiten Schritt mit XSL-FO für ein spezifisches Medium aufbereitet.
3. XPath bildet die Basis für die Angabe von Knoten in einen Dokumentenbaum und wird beispielsweise in XSLT für XSLT-Templates oder auch bei XQuery verwendet.

4.1.3 CSS

Im Web kommen derzeit weniger XSL, sondern hauptsächlich die so genannten **CSS** (Cascading Style Sheets) zum Einsatz. Auch darüber können detailliert Formatangaben beschrieben werden. CSS sind älter als XSL. Bereits 1997 hatte das W3C das CSS-Konzept zum Einsatz von Style Sheets im Web als eine verbindliche Empfehlung vorgestellt und auf diese werden wir uns in diesem Buch im Folgenden auch beschränken. CSS gibt es mittlerweile in der Version 2, aber für unsere Zwecke brauchen wir zwischen CSS1 und CSS2 nicht zu unterscheiden.

Hinweis

In noch nicht allzu ferner Vergangenheit bewegten sich die Anwender von Style Sheets auf sehr dünnem Eis. Der Einsatz führte ob der unterschiedlichen Unterstützung in den verschiedenen Browsern zu zahlreichen Problemen, mittlerweile werden Style Sheets aber von fast allen modernen Browsern (halbwegs) einheitlich unterstützt – zumindest CSS1 und solange keine komplexeren Techniken wie Kindelementselektoren oder Nachelementselektoren (s. u.) verwendet werden. Dennoch – der Einsatz von Style Sheets erzwingt immer noch einen gründlichen Test und die Unterstützung von CSS2 differiert teilweise extrem zwischen Microsoft-kompatiblen Browsern und Browsern, die auf der Gecko-Engine aufsetzen (Firefox, Mozilla).

4.2 Die Verwendung von Style Sheets in einer Webseite

Style Sheets bestehen – wie (X)HTML – aus reinem Klartext, der Regeln zur Formatierung von Elementen beschreibt. Um Style Sheets wie CSS in einer Webseite zu verwenden, müssen Sie diese einer HTML-Seite hinzufügen. Dies kann darüber geschehen, dass Sie Style Sheets in eine Seite **einbetten** oder aus einer externen Datei **importieren**. Eine externe Lösung ist vorzuziehen, denn erst damit erreichen Sie die gewünschte Trennung von Formatierung und Inhalt. Insbesondere kann eine externe Style Sheet-Datei von mehreren Webseiten gemeinsam verwendet werden. Das reduziert die Arbeit bei der Erstellung und erleichtert vor allem die Wartung eines Webprojekts erheblich.

4.2.1 Interne Style Sheets

Die Einbetttung eines internen Style Sheet in eine Webseite erfolgt über den `<style>`-Tag, einen reinen HTML-Container. In dessen Inneren werden alle Elemente definiert, denen ein besonderes Layout zugewiesen werden soll[1]. Das Style Sheet wird konkret so eingebunden:

Listing 4.1: Ein ‹style›-Container

```
01 <style type="text/css">
02 <!--
03 ... irgendwelche CSS-Formatierungen ...
04 -->
05 </style>
```

4.2.2 Externe Style Sheets

Einen Verweis auf ein externes Style Sheet können Sie mithilfe des `<link>`-Tags und der folgenden Syntax vorzunehmen:

Listing 4.2: Die Referenz auf eine externe Style Sheet-Datei

```
<link type="text/css" rel="stylesheet" href="[URL einer CSS-Datei]">
```

Die externe Style Sheet-Datei ist eine Klartextdatei mit Stilinformationen (bei CSS meist mit der Erweiterung *.css* versehen), in der ohne ein Gerüst einfach sequentiell die Stilinformationen notiert werden.

1 *Meist in Kommentare eingeschlossen, damit für ältere Browser die Anweisungen verborgen werden.*

4.2.3 Inline-Definition

Wenn Sie bei nur einem Element eine individuelle Stilinformation angeben wollen, können Sie eine Style Sheet-Anweisung auch als **Inline-Definition** eines HTML-Elements verwenden. Dies bedeutet, über einen zusätzlichen `style`-Parameter innerhalb des HTML-Tags wird ein Attributwert gesetzt und die Stilregel gilt ausschließlich innerhalb des definierten Containers. Hier ist ein Beispiel, in dem die Schriftfarbe in dem Absatz auf blau gesetzt wird:

Listing 4.3: Eine Inline-Definition für ein Style Sheet

```
<p style="color: blue">
```

Hinweis

Die meisten modernen HTML-Editoren formatieren Elemente mit Inline-Definitionen, wenn sie im WYSIWYG-Modus arbeiten.

Optional kann (wie bei den anderen beiden Formen der Einbindung) über den `type`-Parameter der Typ der Style Sheets definiert werden. Beispiel:

Listing 4.4: Eine Inline-Definition für ein Style Sheet mit Angabe der Style Sheet-Sprache

```
<p type="text/css" style="font-size: 12pt; color: red">
```

Tipp

Browser, welche Style Sheets verstehen, haben eine Style Sheet-Sprache als Vorgabe eingestellt. Dies ist in der Regel CSS1 oder CSS2. Falls die Angabe des `type`-Parameters unterbleibt, wird in der Voreinstellung diese Sprache verwendet. Eine explizite Deklaration ist jedoch zu empfehlen, zumal über kurz oder lang auf XML basierende Style Sheets (etwa XSL) weitere Verbreitung finden werden.

4.2.4 Kaskadierende Wirkung von Stilinformationen

Jede CSS-Stilinformation in einer Webseite, die im Widerspruch zu einer HTML-Angabe steht, wird die HTML-Angaben überschreiben. Ansonsten gelten in einer Webseite erst einmal alle Angaben aus HTML weiter. Angenommen, Sie legen mit HTML die Hinter- und Vordergrundfarbe einer Webseite fest und spezifizieren dann

mit einem Style Sheet den Hintergrund mit einer anderen Farbe. Dann wird die Vordergrundfarbe unverändert aus den HTML-Definitionen wirken, während die HTML-Definition des Hintergrunds über die Style Sheet-Definition geändert wird. Diese Form der Modifizierung von Layoutbeschreibung nennt man **Kaskadierung**. Diese wirkt sich auch auf Style Sheets selbst aus. Abhängig von der Art und Reihenfolge der Stilvereinbarungen ergänzen[2] oder überschreiben[3] sich Regeln. Dabei gilt, dass die Wirkung einer Regel aus einer externen Style Sheet-Datei durch eine widersprechende Regel aus einem eingebetteten Style Sheet und diese wiederum von einer widersprechenden Inline-Regel überschrieben wird. Wenn auf der gleichen Ebene widersprechende Stilinformationen zu finden sind, ist das Verhalten von Browsern leider nicht ganz einheitlich. In der Regel wird jedoch die zuletzt notierte Stilvereinbarung als gültig gesehen.

4.3 Die konkrete Syntax von CSS-Deklarationen

Die Syntax einer CSS-Deklaration sieht immer gleich aus. Es gibt immer den folgenden Aufbau:

Listing 4.5: Eine schematische CSS-Formatangabe

```
[name]: [wert]
```

Dabei gibt `name` die zu formatierende Eigenschaft an und `wert` die konkrete Stilinformation. Das zu formatierende Element (der Selektor – s.u.) wird vorangestellt. Mehrere Deklarationen für ein Element werden mit Semikolon getrennt. In vielen Situationen werden die Werte auch in geschweifte Klammern gesetzt. Beispiel:

Listing 4.6: Eine CSS-Festlegung mit zwei Formatierungen

```
{
border-width: 2; text-align: center;
}
```

Tipp

Für die letzte Stilvereinbarung einer Regel kann das Semikolon weggelassen werden. Allerdings ist es besser, dort das Semikolon dennoch zu setzen. Wenn Sie später eine Regel erweitern und vergessen, bei der bis dato letzten Stilvereinbarung das nun fehlende Komma davor zu ergänzen, handeln Sie sich einen Fehler ein und die Browser-Reaktion ist nicht vorhersehbar.

2 *Solange eine Formatregel noch gar nicht festgelegt wurde.*
3 *Eine niedriger priorisierte Regel widerspricht der neuen Formatregel.*

Tipp

Zur Erstellung einer CSS-Deklaration bzw. -Datei benötigen Sie im Grunde nur einen reinen Klartexteditor. Allerdings bietet sich die Unterstützung durch einen CSS-Editor gerade bei komplexeren Fällen an. Einen sehr guten CSS-Editor bietet z. B. das bereits im letzten Kapitel empfohlene OpenSource-Programm Nvu.

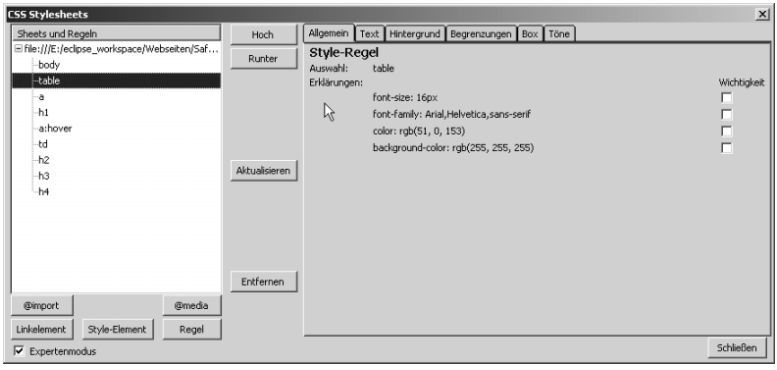


Abbildung 4.1: Der CSS-Editor von Nvu

4.3.1 Selektoren

Selektoren einer Regel bezeichnen die Elemente oder Attribute, die durch die definierten Stilvereinbarungen formatiert werden.

Elementselektoren

Im einfachsten Fall ist ein Selektor der Name eines Elements (**Elementselektor**), das in der Webseite vorkommt[4]. Mit einem Style Sheet können Sie somit jeden HTML-Tag modifizieren. Jeder Tag des formatierten Typs in der Webseite wird dann die Formatvereinbarungen berücksichtigen, die das Style Sheet zugeordnet hat. So würde zum Beispiel eine Überschrift der Ordnung 1 formatiert:

Listing 4.7: Eine CSS-Festlegung – für die Überschrift der Ordnung 1 wird ein Rahmen definiert und der Text zentriert ausgerichtet.

```
h1 { border-width: 2; text-align: center; }
```

4 *Also der Name eines HTML-Tags oder im Fall von XML allgemein eines XML-Elements.*

Tipp

Sie können auch mehrere Elementselektoren durch Kommata getrennt notieren und eine gemeinsame Stilvereinbarung zuweisen, wenn diese für alle Elemente gelten soll. Beispiel:

Listing 4.8: Eine CSS-Festlegung – für die Überschriften der Ordnung 1, 2 und 3 – zentrierte Ausrichtung

```
h1, h2, h3 { text-align: center; }
```

Achtung

Die Zuweisung von einem Wert zu einer Stilfestlegung muss über den **Doppelpunkt** erfolgen. Leider beachtet der Internet Explorer diese Regel nicht und unterstützt auch Stilangaben, bei denen der Wert fälschlicher Weise mit einem Gleichheitszeichen zugewiesen wurde. Dem Ersteller fällt dann der Fehler oft nicht auf und er veröffentlicht fehlerhafte Style Sheets, die in anderen, korrekt arbeitenden, Browsern nicht funktionieren.

Attributselektoren

Eine andere Form eines Selektors nennt sich **Attributselektor**. Das Verfahren ist zwar bezüglich der ansprechbaren Attribute recht allgemeingültig, aber in der Praxis haben sich bei Attributselektoren zwei Varianten etabliert, die nahezu ausschließlich zum Einsatz kommen und für die es Kurzschreibweisen in der CSS-Definition gibt[5] – den Punkt und die Raute. Sie spezifizieren hiermit spezielle Ausprägungen eines Elements.

Ein Tag kann in der Style Sheet-Definition mit einem Punkt abgetrennt werden, um die optionale Angabe einer Klasse zu erweitern. Dies ist die Festlegung der Eigenschaften für eine spezielle Instanz eines Elements. Beispiel:

Listing 4.9: Eine Angabe einer Stilinformation für eine spezielle Instanz der Überschrift h1

```
h1.neuefarbe { color : red}
```

Diese Klasse wird dem vorangestellten HTML-Tag in der Webseite über den Parameter `class` zugeordnet. Die Regel wirkt dann ausschließlich auf die HTML-Tags, die damit gekennzeichnet sind. Beispiel:

5 *Diese gibt es ähnlich im XML (oder genauer – XPath).*

Listing 4.10: Die spezielle Instanz von h1, der die Stilinformation zugeordnet wird

```
<h1 class="neuefarbe">Hallo</h1>
```

Damit ist es möglich, verschiedene Regeln für ein Element zu erstellen. Der ursprüngliche Elementselektor, welcher vor dem Punkt steht, ist das Elternelement zu dieser Instanz.

Sie können bei der Deklaration eines Style Sheets auch eine Klasse ohne direktes Elternelement angeben. In diesem Fall wird im Style Sheet der Attributselektor direkt mit einem vorangestellten Punkt notiert und bei der konkreten Verwendung in der Webseite wird ein beliebiges HTML-Element um die so benannte Regel mit dem `class`-Parameter erweitert. Beispiel:

Listing 4.11: Eine Angabe einer Stilinformation ohne direktes Elternelement

```
.neuefarbe { color : red}
```

Die Zuordnung der Klasse zu einem Element in der Webseite erfolgt wieder über den Parameter `class`. Allerdings ist das in diesem Fall bei **jedem beliebigen Tag** erlaubt**:**

Listing 4.12: Verschiedenen Tags wird die Stilinformation zugeordnet.

```
<h1 class="neuefarbe">Hallo</h1>
<h2 class="neuefarbe">Welt</h2>
```

Tipp

Die Festlegung von Klassen bei Stilvereinbarungen ist besonders deshalb interessant, weil Sie unter JavaScript über die Angabe `className` auf diese Klasse zugreifen können. Über `className` können Sie den Wert des HTML-Parameters `class` sowohl abfragen als auch setzen.

Die zweite Form eines Attributselektors beginnt mit dem Zeichen #. Damit können Sie Style Sheets auch über eine ID auswählen. Sie ordnen mit der Raute jedoch eine Regel genau einem bestimmten Element (keiner allgemeinen Elementklasse wie bei `class`) zu. Das funktioniert, indem bei der Deklaration der Stilinformation in der Style Sheet-Datei eine Raute statt eines Punkts vorangestellt wird (etwa `#meineID {color : red}`). In der HTML-Datei kann man diese Kennung anwenden, indem man statt `class` dem jeweiligen Element als Parameter `id` nachstellt und den Bezeichner der Regel als Wert

angibt (im Beispiel lautete das so: `<p id="meineID">`). In der Webseite ist – wenn die Seite fehlerfrei ist[6] – der Wert des ID-Parameters eindeutig und damit ergibt sich die eindeutige Zuordnung dieser Regel zu genau einem Element in der Webseite.

Universalselektor

Die dritte wichtige Form eines Selektors ist der **Universalselektor** `*`. Damit werden alle Elementknoten im Dokumentbaum ausgewählt – unabhängig vom Elementtyp.

Generationenselektoren

Neben den drei genannten Selektoren spezifiziert CSS auch Selektoren bezüglich einer Generationenbeziehung. Die Bezeichnungen finden sich so auch in XML bzw. XPath wieder. So gibt es Selektoren vom Typ **Nachfahren**. Hierbei werden mehrere Selektoren durch Leerzeichen getrennt aneinander gereiht. Etwa so:

Listing 4.13: Die Formatierungen gelten nur für Nachfahren des Elements li, die vom Typ a sind.

```
li a {
  ... irgendwelche CSS-Formatvereinbarungen
}
```

Dies spezifiziert alle Elemente vom Typ `a`, die direkte oder indirekte Nachfahren eines Elements vom Typ `li` sind. Also zum Beispiel alle Hyperlinks, die als Listeneintrag aufgeführt werden.

Ebenso gibt es die direkte Angabe von unmittelbaren **Kindelementen** als Selektoren. Dazu verwendet man den Operator `>`. Etwa so:

Listing 4.14: Ein Element muss unmittelbares Kindelement von einem <p>-Element sein, damit die Formatvereinbarung gilt

```
p > h1 {
  ... irgendwelche CSS-Formatvereinbarungen
}
```

Dies spezifiziert alle Elemente vom Typ `h1`, die unmittelbar von einem Absatz-Container umgeben sind. Ist die Überschrift der Ordnung 1 jedoch unmittelbar von einem

6 *Und da ist das Problem, weshalb diese Vorgehensweise vom W3C lange nicht empfohlen wurde. Da viele Webseiten sehr fehlerträchtig erstellt werden und das Prinzip der Fehlertoleranz gilt, kann die Eindeutigkeit des ID-Parameters nicht gewährleistet werden. Aber das Verfahren wird von allen wichtigen aktuellen Browsern unterstützt und Sie wissen ja: In einem anderen Zusammenhang wird die ID der Schlüssel zum Erfolg – beim Datenaustausch mit AJAX oder auch allgemein bei Zugriffen auf Bestandteile der Webseite mit DHTML. Unter gewissen Umständen kann man damit kompaktere Quellcodes schreiben, da die ID multifunktional eingesetzt wird, und bei solchen Seiten muss der Programmierer die Eindeutigkeit der ID zwingend sicherstellen.*

anderen Element wie einem -Container umgeben und dieser in einem Absatz-Container enthalten, gelten die Formatvereinbarungen für die Überschrift nicht.

Zu guter Letzt können Sie auch mit dem Operator + Selektoren vom Typ **Nachfolger** angeben. Das sind die Elemente, die auf der gleichen Hierarchieebene folgen wie der vorangestellte Selektor. Beispiel:

Listing 4.15: Die Formatierungen gelten für alle Absätze außer dem ersten Absatz.

```
p + p {
  ... irgendwelche CSS-Formatvereinbarungen
}
```

Pseudo-Klassen

Darüber hinaus spezifiziert die CSS-Definition auch so genannte **Pseudo-Klassen**, die mit einem Doppelpunkt beginnen. Bekannt sind beispielsweise :hover[7] oder :visited,[8] aber es gibt noch weitere Möglichkeiten, die auch Generationenbeziehungen angeben können (zum Beispiel :before und :after).

Achtung

Fast allen diesen erweiterten Möglichkeiten zur Festlegung von Selektoren (sowohl Generationen als auch Pseudo-Klassen) ist gemein, dass sie sehr unterschiedlich in verschiedenen Browsern unterstützt werden. Im Allgemeinen sollte man bei einem Einsatz sehr umfangreiche Tests durchführen oder ganz darauf verzichten (mit Ausnahme des Hover-Effekts – der funktioniert praktisch immer).

4.4 Positionierung und Ausblenden von beliebigen Elementen über CSS

Mit CSS können Sie auch Elemente in der Webseite beliebig positionieren, sowohl relativ als auch absolut. Dies wird bei AJAX-Applikationen (und nicht nur da) ein sehr wichtiges Mittel sein. Die absoluten Positionsangaben von Elementen (in der Regel von der linken oberen Ecke eines Fensters aus gesehen) funktioniert sehr einfach, während die relative Positionierung etwas abstraktes Vorstellungsvermögen bedarf. Bei der absoluten Positionierung benötigen Sie die Angaben position:absolute sowie left=[Position] und top=[Position]. Damit können Sie beliebige Elemente pixelgenau positionieren und nahezu alle modernen Browser kommen damit klar. Bei der relativen Positionierung geben Sie statt absolute das Schlüsselwort relative an und damit die Position in Bezug zu dem vorangehenden Element. Von Interesse für die

7 *Festlegung des Verhaltens beim Überstreichen mit dem Mauszeiger – der so genannte **Hover-Effekt**. Diese Pseudo-Klasse wird vor allem beim Tag <a> sehr zuverlässig unterstützt.*

8 *Festlegung einer Regel für Links, die von einem Besucher bereits besucht wurden.*

Darstellung von Inhalt bei AJAX-Applikationen ist auch das Schlüsselwort `display`. Mit `display : none` können Sie zum Beispiel festlegen, dass der Inhalt des entsprechend markierten Containers nicht angezeigt wird. Eine Alternative dazu ist `visibility : hidden`. Auch damit wird Inhalt explizit verborgen.

4.5 Zusammenfassung

Sie haben in diesem Kapitel sehr kompakt die wichtigen Grundlagen von Style Sheets im Allgemeinen und CSS im Besonderen erfahren. Diese sind bei zahlreichen AJAX-Effekten unabdingbar, denn rein Informationen in eine Seite zu übernehmen, genügt oft nicht. Dem Anwender muss visuell eine Veränderung deutlich gemacht werden und da bietet sich der Einsatz von CSS oder anderer Style Sheet-Sprachen an – insbesondere in Verbindung mit JavaScript, was dann ein Teil dessen darstellt, das als DHTML bekannt ist.

5 JavaScript für AJAX

Hauptfunktion von clientseitigen Skriptsprachen ist eine Verlagerung von Funktionalität vom Server auf den Client. Im Fall von AJAX ist JavaScript darüber hinaus der ultimative Schlüssel, um mit geeigneten Funktionen und Methoden ohne Neuladen der Webseite vom Client aus Daten vom Server nachzufordern und diese dann mittels DHTML in eine Webseite zu platzieren. JavaScript ist damit neben (X)HTML **die** Schlüsseltechnik des gesamten AJAX-Umfelds. In diesem Kapitel werden wir die Erzeugung eines `XMLHttpRequest`-Objekts neu beleuchten und eine JavaScript-Funktion vorstellen, mittels der die Erzeugung portabel über verschiedene Browser hinweg erfolgen kann. Dazu werden die JavaScript-Kenntnisse vertieft, die Sie zur effektiven Programmierung von AJAX-Applikationen benötigen und die teilweise auch nicht unbedingt zum einfachen JavaScript-Grundwissen gehören. Allerdings werden Sie bezüglich der reinen Syntax von JavaScript wenige Probleme haben, wenn Ihnen Java einigermaßen vertraut ist. Dennoch – JavaScript zählt zu den wohl am häufigsten unterschätzten Techniken im EDV-Umfeld. Sowohl in Hinsicht auf das, was man damit machen kann, aber vor allem auch in Hinsicht auf das, was man dazu wissen muss. Viele Laien (wobei explizit EDV-Profis aus anderen Gebieten gemeint sind) halten JavaScript-Kenntnisse für trivial und sehen in ihrer Ignoranz gar nicht, dass es unzählige Feinheiten in JavaScript zu beachten gibt. Dabei sind diese Feinheiten zugegebenermaßen vielfach mehr durch die interpretierenden Browser mit ihren unzähligen Unterschieden gegeben, denn durch JavaScript selbst. Sie sollten sich diesem Kapitel also auch dann widmen, wenn Sie mit komplexeren Sprachen wie Java, C, C++ etc. vertraut sind. Und insbesondere Java-Profis sollten auf die unzähligen (feinen) Unterschiede zwischen JavaScript und Java achten. Im Kapitel wird auf diese Unterschiede immer wieder hingewiesen.

5.1 Die Einbindung von JavaScript in eine Webseite

Bei Skriptsprachen handelt es sich allgemein um Interpretersprachen, die in normale Webseiten über spezielle Tags integriert werden und innerhalb des Clients von einem speziellen Interpreter verarbeitet werden. Aber auch so ein JavaScript-Interpreter ist – wie der (X)HTML-Interpreter – Teil jedes modernen Webbrowsers. JavaScript stellt auf Client-Seite eine reine Erweiterung des (X)HTML-Codes in Form von Klartext dar und ist dort nur als eingebundener Bestandteil eines (X)HTML-Gerüsts zu verwen-

den. Die verschiedenen Arten der Einbindung von JavaScript (oder auch anderer Skriptsprachen) in eine Webseite hat dabei durchaus Ähnlichkeit zur Einbindung von Style Sheets.

5.1.1 Die direkte Notation

Die Verbindung eines JavaScript-Bereichs mit einer Webseite kann einmal über eine **direkte Notation** eines JavaScripts in der Webseite erfolgen. Dazu werden JavaScript-Anweisungen einfach in die entsprechende (X)HTML-Datei als Klartext hineingeschrieben. Der Beginn eines Skripts muss dazu allerdings durch eine eigene Steueranweisung gekennzeichnet werden, die noch zu HTML gehört und die mit ihrem zugehörigen Abschluss-Tag einen Container für die Skriptanweisungen bildet. Über das `<script>`-Element wird angegeben, dass all das, was in dem eingeschlossenen Container steht, ein Skript ist. Über den Parameter `language` können Sie festlegen, um welche Skriptsprache es sich handelt[1]. Dabei ist die Groß- und Kleinschreibung für den Namen der Skriptsprache irrelevant. Das bedeutet, die Anweisungen `<script language="JavaScript">` und `<script language="javascript">` sind äquivalent. Wenn Sie eine andere Skriptsprache verwenden wollen (beispielsweise `VBScript`), geben Sie diese ebenso an.

Hinweis

Da bei der direkten Implementation von JavaScript in den Internet Explorer Lizenzgebühren an Sun/Netscape fällig gewesen wären, hat Microsoft dies nie gemacht. Stattdessen hat Microsoft eine Kopie namens **JScript** entwickelt und nur diese im Internet Explorer implementiert. JScript besitzt im Wesentlichen die gleiche Struktur wie JavaScript und versteht die gleiche Syntax. Allerdings ist der Befehlssatz in einigen Sprachversionen nicht vollständig deckungsgleich. Durch die nicht vorhandene direkte Implementation von JavaScript muss der Internet Explorer sämtliche JavaScripts in einer Webseite erst in JScripts übersetzen und diese dann ausführen. Das funktioniert allerdings recht zuverlässig und vom Anwender unbemerkt. Und auch bei der Referenzierung im `<script>`-Tag brauchen Sie JScript nicht explizit anzugeben. Im Gegenteil – Sie sollten es auch nicht machen, wenn es keine konkreten Gründe dafür gibt. In dem Fall schließen Sie nämlich alle Browser außer dem Internet Explorer von der Ausführung aus.

1 *Wenn Sie keine Sprache angeben, wird von allen bekannten Browsern als Standard JavaScript verwendet.*

Tipp

Die Angabe der **JavaScript-Version** – beispielsweise über die Angabe `JavaScript1.3` oder `JavaScript1.5` – ist unter Umständen sinnvoll (beachten Sie, dass die Versionsnummer ohne Leerzeichen angehängt wird). Wenn Sie diese Versionsangaben setzen, werden in dem Container folgende Anweisungen nur von den Browsern ausgeführt, die diese Version unterstützen. Man kann damit die fehlerhafte Ausführung von Skripts durch inkompatible (in der Regel ältere) Browser verhindern, wobei die Container logisch sinnvoll aufgebaut werden müssen, um beim Ignorieren des Containers keine anderen Probleme zu bekommen. Sie sollten sich bei einer Anwendung jedoch darüber klar sein, dass bestimmte Versionsangaben in einigen Browsern Probleme machen. Das macht dieses scheinbare banale Thema gefährlicher als es zuerst scheinen mag. So werden einige (sehr) alte Browser Versionsanweisungen grundsätzlich nicht beachten und versuchen, für sie möglicherweise unbekannte Anweisungen auszuführen. Das Problem ist jedoch kaum noch relevant, da so alte Browser mehr oder weniger ausgestorben sind. Beim Internet Explorer muss man jedoch beachten, dass die Angabe `JavaScript1.0` (was ja im Prinzip identisch zur Angabe `JavaScript` ist) Probleme macht. Er führt entsprechend gekennzeichnete Skript-Container einfach nicht aus. Jedoch können Sie nahezu jeden Browser mit der Angabe des MIME-Typs über `type="text/javascript"` dazu veranlassen, jeden JavaScript-Container unabhängig von der Versionsangabe auszuführen. Mit anderen Worten: Die gleichzeitige Angabe des MIME-Typs und einer Versionsangabe für JavaScript sollte unterbleiben. Was Sie noch beachten sollten, ist, dass sich die Angabe von einer JavaScript-Version in vielen Fällen schlicht und einfach als unbrauchbar erweist, weil sie zu restriktiv ist. Dies betrifft uns insbesondere im Fall von AJAX. Es gibt viele Anweisungen, die offiziell zu einer bestimmten Version von JavaScript gehören und die dennoch von Browsern verstanden werden, die einen bestimmten Standard nicht vollständig unterstützen und deshalb einen explizit mit einer Versionsangabe gekennzeichneten Container vollständig ignorieren werden. Das gilt vor allem für Techniken aus dem JavaScript-Standard 1.5 wie der von Java hergeleiteten Ausnahmebehandlung, die wir zur Erstellung einer universalen Funktion zum Erzeugen eines `XMLHttpRequest`-Objekts verwenden werden. Beispielsweise ignoriert der Internet Explorer 6 alle Container, die mit Versionsangaben jenseits von `javascript1.3` gekennzeichnet sind. Dabei unterstützt der Internet Explorer 6 den Standard 1.5 nahezu vollständig. Ebenso wird Opera 7.2 nur Container bis zur Angabe `javascript1.4` ausführen, obwohl JavaScript 1.4 im

Grunde nie ein implementierter Standard war und auch dieser Browser nahezu mit allen Anweisungen aus JavaScript 1.5 zurechtkommt (bereits in Vorgängerversionen). Für die Praxis kann man festhalten, dass ein entsprechender Container nur mit Versionsangabe bis zur JavaScript-Version 1.3 (inklusive) sinnvoll ist.

Der <script>-Tag kann an jeder beliebigen Stelle innerhalb eines HTML-Dokuments platziert werden. Die Skriptanweisungen werden einfach geladen, wenn die Webseite von oben nach unten in den Browser geladen wird. Entsprechend ist klar, dass unten in einer Webseite notierte Skriptanweisungen (Funktionen) erst dann zur Verfügung stehen, wenn die Seite bis dahin geladen ist. Eine übliche Vorgehensweise ist es deshalb, einen <script>-Container vor den Body (innerhalb des Headers einer Webseite oder direkt dahinter) zu platzieren und darin die Deklarationen von wichtigen Funktionen einzubetten. Diese Funktionen sind dann für das ganze Dokument verfügbar. Sie können weitere <script>-Container in anderen Abschnitten Ihrer Webseite platzieren oder Eventhandler verwenden, um diese Funktionen dort gezielt aufzurufen oder irgendwelche Informationen zur Verfügung zu stellen.

Tipp

Wenn ein Skript vor dem Webseitenkörper platziert wird, hat das den Vorteil, dass es bereits bereitsteht, bevor der sichtbare Teil der Webseite über das Netz geladen wird. Oft ist das sehr sinnvoll oder gar unumgänglich. Deshalb sollten Sie wichtige JavaScript-Funktionen möglichst weit oben in die Webseite schreiben (deklarieren) und gegebenenfalls später gezielt aufrufen. Dieser Code wird so lange verfügbar sein, wie das Dokument existiert. Es gibt allerdings auch Situationen, in denen Sie einen Skript-Container erst am Ende der Seite notieren – etwa dann, wenn Sie Daten in JavaScript-Variablen vorhalten wollen, die erst später in der Webseite angezeigt werden sollen. Dann kann das Laden der Informationen quasi im Hintergrund erfolgen, wenn der Besucher bereits alles sieht, was ihn am Anfang interessiert.

Bei einem Skript-Container steht in der Regel direkt nach dem einleitenden <script>-Tag ein HTML-Kommentar (<!--). Dieser wird unmittelbar vor dem abschließenden Tag </script> mit dem entsprechenden HTML-Kommentar-Endezeichen (-->) wieder geschlossen. Dadurch steht der gesamte JavaScript-Code innerhalb eines HTML-Kommentars. Dies ist zwar nicht zwingend, aber sicherer für den Fall, dass ein Browser die Seite lädt, der keine Skripts interpretieren kann[2]. Wenn Sie nun von Tools generierte <script>-Container sehen, werden Sie oft vor dem abschließenden HTML-Kommentar zwei Schrägstriche (//) sehen. Der Grund ist historisch bedingt. Die

2 *Wobei diese Browser mittlerweile ausgestorben sein sollten.*

Anwendung des HTML-Kommentars im Inneren des <script>-Containers hatte bei einigen älteren Browsern zu Problemen geführt. Man versteckt daher das Ende des HTML-Kommentars hinter der Zeichenfolge // (einen JavaScript-Kommentar[3]). Ein Skript-Container sieht also in der allgemeinsten Form so aus:

Listing 5.1: Ein Schema für einen JavaScript-Container

```
<script language="JavaScript">
<!--
   [Skriptanweisungen]
//-->
</script>
```

Tipp

Optional kann dem <script>-Container noch ein <noscript>-Container folgen. Darin notieren Sie Klartext und reines HTML mit Informationen, die dem Besucher bei deaktiviertem JavaScript angezeigt werden sollen. Wenn JavaScript bei einem Besucher aktiviert ist, werden die Anweisungen und der Text vom Browser ignoriert.

5.1.2 Die Einbindung einer externen JavaScript-Datei

Sinnvoller als die direkte Notation eines JavaScripts in die Webseite ist so gut wie immer der Aufbau einer Funktionssammlung in Form einer externen Datei, die in die Webseite eingebunden wird. Wie auch bei Style Sheets ist die Verwendung einer externen JavaScript-Datei unter anderem dann sinnvoll, wenn Sie gleiche Funktionalität in verschiedenen Webseiten verwenden wollen. Insbesondere ist die Wartung und Verwaltung des Codes leichter. Und vor allem wird die Trennung der Struktur und Funktionalität erreicht. Die externe JavaScript-Datei (in der Regel mit der Erweiterung *.js*) enthält im optimalen Fall Ihren gesamten JavaScript-Code mit Ausnahme der expliziten Aufrufe von JavaScript-Funktionen (diese werden in der Webseite meist mit Eventhandlern oder dem JavaScript-eigenen Eventhandling aufgerufen). Um eine externe JavaScript-Datei in eine Webseite einzubinden, müssen Sie den <script>-Tag nur um das Attribut src erweitern. Damit geben Sie den URL der externen JavaScript-Daten an. Dabei gelten beim Referenzieren von separaten JavaScript-Dateien die üblichen Regeln für URLs. Beispiel:

3 *Und auch Java-Kommentar – wie schon gesagt: Die reine Syntax von JavaScript und Java ist nahezu gleich. Die Konzepte (und das ist beim Verständnis einer Programmiersprache das Wichtige) sind jedoch grundverschieden.*

Listing 5.2: Referenz auf eine externe JavaScript-Datei

```
<script language="JavaScript" src="meineJS.js"></script>
```

Achtung

Der `<script>`-Container sollte auf jeden Fall leer sein (auch keine Leerzeichen), denn einige Browser machen Probleme, wenn darin Inhalt vorhanden ist. Diese Browser reagieren nicht etwa fehlerhaft, sondern es handelt sich hier im Sinn von XML um ein **leeres Element**. Sie können daher natürlich auch die Form `<script ... />` wählen.

5.1.3 Die Inline-Referenz

Es gibt einen Spezialfall, bei dem Sie JavaScript-Anweisungen ohne einen expliziten `<script>`-Container direkt als Parameterwert in eine HTML-Anweisung schreiben können. Dies ist die **Inline-Referenz**. Dazu müssen Sie nur der entsprechenden HTML-Anweisung das Schlüsselwort `javascript`, gefolgt von einem Doppelpunkt, als Attribut angeben.

Listing 5.3: Beispiel für eine Inline-Referenz

```
<a href="javascript:alert('Die Antwort ist 42')">
```

Dem Attribut `href` werden statt eines URL in Anführungszeichen eine oder mehrere JavaScript-Anweisungen zugewiesen. Mit dieser Technik können Sie gezielt eine JavaScript-Anweisung bei einer durch den Anwender initiierten Aktion – dem Klick – aufrufen. Dieses System hat aber Nachteile. Bei größeren Mengen Code ist die Sache nicht gut lesbar. Deshalb wird die Technik so gut wie ausschließlich zum Aufruf einer einzelnen JavaScript-Anweisung oder Funktion verwendet. Ebenso ist die Technik weitgehend auf die Tags `<a>` und `<area>` beschränkt. Zudem ist die hier ausschließlich unterstützte Aktion eines Klicks durch einen Anwender nicht die einzige denkbare Situation, bei der man JavaScripts ausführen möchte. Deshalb wird die Inline-Referenz in der Praxis nahezu vollständig durch viel flexiblere Eventhandler ersetzt.

5.2 Ist JavaScript bei einem Browser aktiviert?

JavaScript ist wie gesagt mit Abstand die wichtigste Technik zum Umsetzen des AJAX-Konzepts (wenn man HTML einmal als »gottgegeben« voraussetzt). Ist JavaScript bei einem Besucher nicht aktiviert, können Sie auch AJAX vergessen. Wie schon mehrfach erwähnt, wird JavaScript derzeit zwar wieder flächendeckend eingesetzt und die meisten Anwender haben es aktiviert. Dennoch sollten Sie sich nicht darauf verlassen. Ein kurzer Test auf die Unterstützung von JavaScript bei einem

Besucher ist aber nicht schwer. Dieser Test kann entweder ein einziges Mal beim Einstieg in ein Webprojekt ausgeführt werden oder auch bei jedem Laden einer Webseite Ihres Projekts (wobei das im Grunde viel zu aufwändig ist). Aber wie können Sie nun testen, ob bei einem Client JavaScripts überhaupt ausgeführt werden können oder nicht? Mit JavaScript selbst? Eine Abfrage per JavaScript, ob JavaScript aktiviert ist, ist ja nicht möglich, wenn es deaktiviert ist. Oder doch? Es geht indirekt. Man nutzt explizit die Tatsache, dass eine JavaScript-Aktion bei deaktiviertem JavaScript überhaupt nicht funktioniert. Das geht beispielsweise so: Mit JavaScript führen Sie eine Weiterleitung zu einem URL aus. Funktioniert der, muss auch JavaScript aktiviert sein. Funktioniert die Weiterleitung nicht, ist JavaScript deaktiviert.

Wenn Sie den Anwender nun komfortabel führen wollen, erstellen Sie eine Einstiegsseite in Ihr Webprojekt, das im Wesentlichen eine solche Weiterleitung enthält[4] und ihn für den Fall des deaktivierten JavaScript automatisch zu einem Projekt führt, das ohne JavaScript (und damit auch ohne AJAX) auskommt.

Eine solche Weiterleitung funktioniert auch ohne JavaScript, denn mit HTML sind automatische Weiterleitungen ebenso möglich. Dazu dient der Tag `<meta http-equiv="refresh" content="[Zeit];URL=[Ziel]">`. Beim Attribut `content` geben Sie zuerst die Zeitspanne an, nach der die Weiterleitung erfolgen soll, und dann das Ziel der Weiterleitung. Der entscheidende Trick beruht darauf, die Weiterleitung per HTML erst nach der Weiterleitung per JavaScript auszuführen. Das nachfolgende Beispiel zeigt eine solche automatische Trennung nach JavaScript und deaktiviertem JavaScript (*javascripttest.html*):

Listing 5.4: Trennung nach JavaScript-Unterstützung oder nicht

```
01 <html>
02 <head>
03   <meta http-equiv="refresh" content="5;URL=noAJAX.html" />
04 </head>
05 <script language="JavaScript">
06 <!--
07   location.href="mitAJAX.html";
08 //-->
09 </script>
10 <body>
11 <a href="noAJAX.html">Wenn die automatische Weiterleitung nicht funktioniert,↵
   klicken Sie bitte hier, um zum Webprojekt ohne AJAX zu kommen</a>
12 </body>
13 </html>
```

4 *Vielleicht noch mit einer Browser-Weiche gekoppelt (siehe dazu Seite 141).*

Wenn man diese Webseite lädt, erfolgt eine automatische Weiterleitung per JavaScript mit dem `location`-Objekt[5] (`location.href` – Zeile 7), das eine Eigenschaft des Objekts `window` ist. Nun wird diese Anweisung bei deaktiviertem JavaScript nicht ausgeführt. Im Header finden Sie in Zeile 3 einen HTML-`<meta>`-Tag, der aber erst nach einem Zeitintervall (hier 5 Sekunden und im Allgemeinen unbedingt länger als das Weiterleitungsintervall des JavaScripts) automatisch auf ein Teilprojekt ohne JavaScript und AJAX weiterleitet. Zur Sicherheit notieren Sie noch einen Hyperlink, womit die Weiterleitung auch vom Besucher manuell ausgelöst werden kann.

5.3 Elementare Grundstrukturen von JavaScript

Es wurde schon mehrfach angedeutet, dass ich bei Ihnen ein gewisses Fundament sowohl in Java als auch in JavaScript voraussetze. Sie müssen jedoch (noch) keine JavaScript-Experten sein. Aber ich gehe davon aus, dass Sie ein paar einfache JavaScripts und/oder Java-Applikationen schon erstellt haben, und wissen, was Schleifen, Entscheidungsanweisungen oder Variablen sind. Deshalb finden Sie hier nur einige Hinweise zu diesen elementaren Grundstrukturen von JavaScript. Dies sind logische Strukturen, die Sie in fast jeder Skript- und Programmiersprache so finden. Im Einzelnen umfasst das Variablen, Arrays, Datentypen, Operatoren, Ausdrücke, Anweisungen und Kommentare. JavaScript ist wie Java in den meisten Syntaxstrukturen an C angelehnt oder auch identisch[6]. Das bedeutet, viele Schlüsselwörter, aber auch Operatoren und Kontrollstrukturen finden Sie so auch in C und vielen anderen Sprachen wie PHP oder C#, die bezüglich ihrer Syntax auf C zurückgehen. Dies hat für jeden, der JavaScript lernen möchte, den wesentlichen Vorteil, dass syntaktische Kenntnisse aus einer Vielzahl von anderen Programmiersprachen direkt übernommen werden können.

5.3.1 Variablen und Datentypen – Vorsicht vor loser Typisierung

Variablen und Datentypen stellen in JavaScript kein besonders schwieriges Thema dar. Allerdings ist JavaScript eine **lose typisierte** Sprache, bei der Variablen »on the fly« durch Wertzuweisung entstehen und während der Laufzeit den Datentyp ändern können. Dies ist vollkommen unterschiedlich zu Java und wird reinen Java-Programmierern einen Kulturschock ;-) versetzen. Lose Typisierung bedeutet, dass von einem JavaScript-Programmierer zwar die Namen von Variablen festgelegt, dabei jedoch die Datentypen nicht deklariert werden. Dadurch bekommt eine Variable erst dann einen bestimmten Datentyp zugewiesen, wenn ihr der Wert eines bestimmten Typs zugewiesen wurde. Dies erfolgt vollkommen automatisch und ohne explizites Zutun des Programmierers. Sie können diese automatische Verwaltung der Variablen und Datentypen in JavaScript auch gar nicht abschalten. Dies ist bei vielen konventionellen Programmiersprachen wie C/C++ und Java anders. Wie

5 *Siehe dazu Seite 138.*
6 *Obwohl viele Strukturen aus C in JavaScript nicht vorkommen, etwa Zeiger.*

Ihnen wahrscheinlich vertraut ist, muss ein Java-Programmierer auf jeden Fall den Datentyp festlegen, bevor in einer Variablen ein Wert gespeichert werden kann. In der so festgelegten Variablen kann dann auch nur ein Wert des einmal fixierten Typs gespeichert werden. Jede Wertzuweisung eines anderen Datentyps wird einen Fehler erzeugen. Man nennt so etwas dann **statische Typisierung** von Variablen.

Achtung

JavaScript unterscheidet Groß- und Kleinschreibung. Das wirkt sich auf Variablenbezeichner, aber auch auf Funktionsnamen etc. aus.

Obwohl JavaScript bei der Deklaration von Variablen keine Angabe eines Datentyps fordert bzw. gestattet, verfügt JavaScript natürlich über vorgegebene Datentypen. Dies sind folgende:

Typ	Beschreibung	Beispiele
`Boolean`	Ein Wahrheitswert (`true` oder `false`)	`AJAXIstToll = true;`

`Number`	Dieser Typ kann einen Dezimal- oder einen Gleitkommawert (mit Punkt zum Abtrennen des Nachkommateils) enthalten und stellt eine Zahl dar. Neben der normalen Zahlenschreibweise gibt es die wissenschaftliche Notationsmöglichkeit über die Angaben `e` oder `E`.	`a = 123.45;`
		`b = 42;`
		`c = 1.2345E2;`

`Object`	Ein allgemeiner Objektdatentyp	`resObjekt = new XMLHttpRequest();`
`String`	Alphanumerische Zeichen (Text)	`a = "Felix";`

Tabelle 5.1: Die Datentypen in JavaScript

5.3.2 Arrays

In engem Zusammenhang mit Variablen sind **Datenfelder** bzw. **Arrays** zu sehen. Dies ist im Grunde eine Sammlung von Variablen, die alle über einen gemeinsamen Namen angesprochen werden können. Ein großer Vorteil von Arrays ist, dass man über Schleifen mit weniger Aufwand auf Werte zugreifen kann als es bei verschiedenen Variablennamen der Fall ist.

Hinweis

Ein weiterer Grund für die Bedeutung von JavaScript-Arrays ist, dass sich der Browser beim Laden der Webseite im Rahmen vom Document Object Model einzelne Bestandteile der Webseite merkt und einige davon über Datenfelder für den Zugriff über JavaScript (oder eine andere Technik) bereitstellt.

Ein Array wird in JavaScript etwas anders erzeugt als eine normale Variable. Sie müssen neben dem Namen kennzeichnen, dass ein Array vorliegt. Außerdem wird zur Erzeugung das Schlüsselwort `new` und ein **Konstruktor** verwendet. Das werden wir gleich noch in Zusammenhang mit der allgemeinen Erzeugung von Objekten genauer besprechen.

Achtung

Hier sollten Java-Programmierer besonders aufpassen, denn die Erzeugung von Arrays in JavaScript unterscheidet sich trotz der scheinbaren Ähnlichkeit fundamental von der Erzeugung in Java.

Ein Array erzeugen Sie in JavaScript zum Beispiel so:

Listing 5.5: Ein Datenfeld wird erzeugt.

```
a = new Array(30);
```

Achtung

Java-Programmierer sollten den Konstruktor genau betrachten. In Java sind Arrays klassenlose Objekte und es gibt keinen Konstruktor `Array()`!

Einem Element eines Arrays weisen Sie einfach einen Wert zu, indem Sie den Namen des Arrays, eine öffnende eckige Klammer, den Index und eine schließende Klammer angeben und dann wie gewöhnlich einen Wert zuweisen. Wir können also jetzt jedem Element des Arrays eine Zahl zuordnen:

Listing 5.6: Belegen der Felder im Array mit Werten

```
a[ 0 ] = 1 ;
a[ 1 ] = 2 ;
...
a[ 29 ] = 30 ;
```

Achtung

Im obigen Beispiel ist ein Array mit 30 Elementen erzeugt worden. Der Index beginnt mit 0. Da aber JavaScript – wie schon angedeutet – dynamisch Variablen entstehen lassen kann, kann man auch einen Index jenseits der so festgelegten Größe ansprechen. Die Zuweisung `a[500] = 8;` wäre also bei obigem Array kein Fehler. Das macht den Umgang mit Arrays in JavaScript bequem, jedoch auch gefährlich. Java ist hier als streng typisierte Sprache natürlich viel strenger. Das ist am Anfang unbequem, aber bei komplexen Applikationen viel, viel, viel, viel[7] sicherer.

Tipp

In einem JavaScript-Array lassen sich Elemente verschiedenen Datentyps speichern. Sie können beispielsweise in einem Array die Daten von einer Person aufnehmen, die teilweise aus Zahlen bestehen, teilweise aus booleschen Werten und aus Text. Auch dieses Verhalten haben Sie bei Java-Arrays erst einmal nicht. Sie müssen hier andere Konstruktionen anwenden.

5.3.3 Anweisungen

Eine Anweisung ist eine Quellcodezeile, die bestimme Befehle enthält. JavaScript-Anweisungen können über mehrere Zeilen im Editor reichen, denn sie enden immer erst mit einem Semikolon. Die Arten der Anweisungen sind in allen C-basierenden Programmiersprachen fast identisch wiederzufinden, so auch in JavaScript. Fassen wir die wichtigsten Details kurz zusammen:

- **Blöcke** werden mit geschweiften Klammern aufgebaut und fassen mehrere Anweisungen zusammen.
- **Ausdrücke** zur Manipulation von Daten werden über Operatoren zusammengesetzt und gegebenenfalls einer Variablen auf der linken Seite einer Zuweisung zugewiesen (Beispiel: `gewicht = 89 + 5;`).
- **Operatoren** in JavaScript sind eine Untermenge aller Opertoren, wie Sie diese auch in C finden.

7 *;-)*

- **Kommentare** werden in JavaScript direkt aus C übernommen. Mehrzeilige Kommentare beginnen mit einem Slash, gefolgt von einem Stern (`/*`) und enden mit einem Stern gefolgt von einem Slash (`*/`). Einzeilige Kommentare beginnen mit einem Doppel-Slash (`//`) und enden mit dem Zeilenende.
- In JavaScript haben Sie drei Arten von **Kontrollflussanweisungen** zur Verfügung:
  - **Entscheidungsanweisungen** leiten aufgrund einer Bedingung einen Programmfluss weiter. Es gibt die `if`- sowie die `if-else`-Bedingung und die `switch-case`-Fallunterscheidung[8].
  - **Schleifen** wiederholen Anweisungen. JavaScript kennt die `while`-, `do-while`- und `for`-Schleife. Die `while`- und die `for`-Schleife sind kopfgesteuerte bzw. abweisende Schleifen, während `do-while` eine fussgesteuerte oder annehmende Schleife bezeichnet. Das bedeutet, im Fall von `do-while` wird der Schleifeninhalt auf jeden Fall mindestens einmal ausgeführt.
  - **Sprunganweisungen** verlassen einen Anweisungsblock. JavaScript kennt unter anderem `break` und `continue`. Die Sprunganweisung `break` beendet eine Struktur (etwa eine Schleife oder eine Funktion) sofort, wenn diese Stelle im Quelltext erreicht wird. Mit `continue` können Sie an einer bestimmten Stelle unmittelbar den nächsten Schleifendurchlauf erzwingen und die nachfolgenden Anweisungen innerhalb der Schleife ignorieren. Als weitere Sprunganweisung ist `throw` zu erwähnen, was beim manuellen Auswerfen von Ausnahmen Verwendung findet. Ebenfalls eine Sprunganweisung ist `return`, um eine Funktion zu verlassen und gegebenenfalls einen Wert als Ergebnis zurückzugeben.

5.4 Funktionen, Prozeduren und Methoden

Die Zusammenfassung von Code wird in jeder modernen Programmiersprache über **Unterprogramme** realisiert. Statt Unterprogramm redet man aber in der Regel von **Funktion** oder **Methode**. Eine JavaScript-Funktion oder eine Methode können Sie in einer Webseite mit einem Eventhandler, einem JavaScript-eigenen Eventhandling oder aus einem Skript direkt aufrufen. In diesem Zusammenhang taucht zusätzlich der Begriff **Prozedur** auf. Allgemein sind Funktionen und Prozeduren erst einmal nur eine Reihe von Anweisungen im Quelltext, die zusammengefasst werden und einen Namen bekommen und dann jedes Mal über die Angabe des Namens aufgerufen werden, wenn diese Folge von Anweisungen benötigt wird. So als Funktionen definierte JavaScript-Anweisungen werden beim Laden der Webseite respektive des Skripts nicht direkt ausgeführt, sondern erst, wenn sie explizit aufgerufen werden.

8 *Beachten Sie, dass es sich bei der* `switch-case`*-Anweisung um eine so genannte Fall-Through-Anweisung handelt. Nach einem Treffer müssen Sie einen Block mit einer Sprunganweisung verlassen, wenn Sie nicht wollen, dass alle nachfolgenden Anweisungen der switch-case-Konstruktion abgearbeitet werden.*

Gibt es nun zwischen Prozedur und Funktion einen Unterschied? Es gibt normalerweise einen wesentlichen Unterschied. Eine Funktion erledigt eine Aufgabe und liefert immer einen Wert als Ergebnis zurück (einen **Rückgabewert**), eine Prozedur erledigt eine Aufgabe, ohne einen Rückgabewert als Ergebnis zurückzugeben. JavaScript unterscheidet aber nicht zwischen Funktionen und Prozeduren. Beide Formen eines Unterprogramms werden als Funktionen bezeichnet. Allerdings muss eine JavaScript-Funktion keinen Wert zurückgeben, der zwingend ausgewertet werden muss, wie es in einigen anderen (vor allem älteren) Programmiersprachen notwendig sein kann[9]. Damit sind in JavaScript die Unterschiede nahezu aufgehoben.

Eine spezielle Form einer Funktion ist eine **Methode**, die Java-Programmierer natürlich kennen. Methoden sind spezielle Unterprogramme, die einem Objekt zugeordnet sind und nur in Verbindung mit diesem vorkommen. Sie können in JavaScript – wie Eigenschaften eines Objekts – nur über die vorangestellte Angabe des Objekts angesprochen werden (die so genannte **Dot-Notation** oder **Punktnotation**). Das ist genau das gewesen, was wir mit `resObjekt.open('get', 'laender.jsp?wo='+i,true)` oder `resObjekt.send(null)` in Kapitel 2 gemacht haben. Sowohl `open()` als auch `send()` sind keine Funktionen, sondern Methoden eines Objekts vom Typ `XMLHttpRequest`.

Hinweis

JavaScript ist ausdrücklich hybrid. Im Gegensatz zu Java, das rein objektorientiert ist, kann man in JavaScript Objekte und Methoden verwenden, jedoch auch prozedural programmieren. Dafür besitzt JavaScript viele Dinge nicht, die eine echte objektorientierte Sprache haben muss, etwa echte Vererbung.

5.4.1 Definition eigener Funktionen

JavaScript besitzt eine ganze Reihe von vorgefertigten Funktionen, die Sie direkt verwenden können, indem Sie sie einfach aufrufen (eventuell mit geforderten Parametern). Diese gehören zum Sprachumfang von JavaScript. Beispiele sind `escape()` zum Kodieren von Zeichenketten und `unescape()` zum Dekodieren. Wenn Sie eigene Funktionen in JavaScript definieren wollen, notieren Sie bei der Deklaration zuerst das Schlüsselwort `function`, gefolgt von dem gewünschten Namen der Funktion (den Sie selbst weitgehend frei auswählen dürfen, der jedoch nicht mit einem Schlüsselwort identisch sein darf), einer öffnenden Klammer, optional einer Parameterliste, welche Variablen bezeichnet, über die Werte an die Funktion übergeben werden können (bei mehreren Parametern durch Kommata getrennt), und der abschließenden Klammer. Die eigentlichen Anweisungen der Funktion (die Implementation) werden in nachfolgende geschweifte Klammern eingeschlossen (einen Block). Die schematische Syntax zur Deklaration einer Funktion sieht also so aus:

9 *Zum Beispiel Pascal.*

Listing 5.7: Schema für die Deklaration einer Funktion

```
function [Name]([Parameterliste]) {
...Anweisungen
}
```

Tipp

Funktionen werden üblicherweise möglichst weit oben in einer Webseite deklariert oder über eine externe Datei eingebunden, deren Referenz vor dem Body einer Webseite steht. Damit kann gewährleistet werden, dass sie bereits vor dem Anzeigen einer Webseite vom Browser geladen und gegebenenfalls interpretiert wurden. Dies ist jedoch nicht zwingend. JavaScript-Funktionen können wie alle JavaScript-Anweisungen an jeder Stelle in einer HTML-Seite deklariert werden. Es kann dann nur vorkommen, dass ein Anwender eine Aktion auf der HTML-Seite auslöst und die dafür notwendige Funktion noch gar nicht geladen wurde.

5.4.2 Rückgabewerte einer Funktion mit return

Wenn eine Funktion einen Rückgabewert liefern soll, müssen Sie ihn über die Sprunganweisung `return` angeben. Das sieht formal so aus:

Listing 5.8: Schema für einen Rückgabewert

```
return [Rückgabewert];
```

Achtung

Die Anweisung `return` sollte immer die letzte Anweisung in einer Funktion sein. Sie verlässt unmittelbar die Funktion. Nachfolgend notierte Anweisungen in der Funktion werden nicht mehr ausgeführt. Sie können `return` allerdings vorher in Verbindung mit einer Entscheidungsstruktur verwenden. Beachten Sie, dass es in JavaScript im Gegensatz zu Java keine Möglichkeit gibt, **unerreichbaren Code** zu entdecken. Unerreichbarer Code entsteht beispielsweise dann, wenn Sie in einer Funktion eine auf jeden Fall auszuführende `return`-Anweisung notieren, der weitere Anweisungen folgen. Diese werden niemals ausgeführt und das ist dann unerreichbarer Code (unreachable code). In Programmiersprachen, die kompiliert werden (z.B. Java), kann der Compiler solche Probleme bereits im Vorfeld entdecken und den Programmierer vor der Erstellung solch fehlerhaften Codes bewahren. Durch das Interpreterprinzip bei JavaScript gibt es diese Möglichkeit rein technisch nicht.

Hinweis

Als Beispiele für selbst definierte Funktionen betrachten Sie noch einmal das erste AJAX-Beispiel in Kapitel 2. Dort haben wir die Funktionen `handleResponse()` und `sndReq()` erstellt.

5.4.3 Aufruf von Funktionen und Methoden

Der Aufruf einer Funktion oder einer Methode erfolgt – wie schon erwähnt – über eine Inline-Referenz, einen Eventhandler, ein JavaScript-Ereignis oder direkte Notation in einem `<script>`-Container in einer Webseite. Bei einer Methode müssen Sie natürlich zusätzlich beachten, dass der Methode das zugehörige Objekt vorangestellt werden muss[10]. Zudem müssen gegebenenfalls notwendige Parameter übergeben werden, wenn diese in der Deklaration gefordert werden. Dazu setzen Sie für jeden Parameter einen sinnvollen Wert, der in der Funktion bzw. Methode als Wert der lokalen, über den Parameternamen deklarierten, Variablen zur Verfügung steht.

Rekursion

Es ist in JavaScript auch möglich, dass sich eine Funktion wieder selbst aufruft. Das nennt man einen **rekursiven Aufruf**. Das grundsätzliche Verfahren ist unkompliziert, denn man notiert einfach im Inneren einer Funktion den Aufruf für die Funktion selbst. Schematisches Beispiel:

Listing 5.9: Rekursiver Aufruf der Funktion a()

```
function a(){
  ... irgendwelche Anweisungen
  a();
}
```

Sie sollten bei rekursiven Aufrufen jedoch beachten, dass irgendwann der rekursive Aufruf unterbrochen werden sollte bzw. muss. Mit anderen Worten – der Selbstaufruf sollte in der Regel an gewisse Bedingungen gekoppelt werden.

10 *In einigen Fällen kann man auf die Notation des vorangestellten Objekts verzichten. Darauf gehen wir im folgenden DOM-Abschnitt noch genauer ein.*

Hinweis

In komplexeren AJAX-Applikationen werden Sie recht oft mit rekursiven Funktionsaufrufen konfrontiert.

5.5 JavaScript und Objekte

Ein Großteil des Nutzens (wenn nicht gar der einzige Nutzen) von JavaScript resultiert aus der Tatsache, dass Sie damit Objekte nutzen können. JavaScript ist eine teilweise objektorientierte Sprache. Genau genommen nennt man sie **objektbasierend**, denn es fehlen – wie schon erwähnt – einige Fähigkeiten, die für eine echte objektorientierte Sprache unabdingbar sind. Die Details klären wir sukzessive. Aber zuerst soll kurz[11] rekapituliert werden, was ein Objekt und Objektorientierung bzw. objektorientierte Programmierung (OOP) überhaupt sind?

5.5.1 Was ist ein Objekt?

Unter einem **Objekt** stellt man sich in der Programmierung ein Softwaremodell vor, das ein Ding aus der realen Welt mit all seinen Eigenschaften und Verhaltensweisen beschreiben soll. Zum Beispiel ein Objekt Drucker oder Bildschirm. Auch Teile der Software selbst können als Objekt zu verstehen sein, der Browser beispielsweise oder ein Teil davon – zum Beispiel die Adresszeile des Browsers. Oder nur ein Teil eines Dokuments – eine Überschrift in einer Webseite oder ein `<span>`-Container. Oder auch ein Mechanismus, um aus einer Webseite heraus ohne Neuladen der Seite Daten vom Server nachfordern zu können. Eigentlich ist in dem objektorientierten Denkansatz alles als Objekt zu formulieren, was sich eigenständig erfassen und ansprechen lässt.

Wenn Sie nun mit Objekten in einem Programm oder Skript umgehen, unterscheidet man zwischen objektbasierender und objektorientierter Programmierung (OOP). Beide Arten der Programmierung erweitern die althergebrachte, prozedurale Programmierung. Dort hat man zur Strukturierung von Quelltext maximal kleine Codezusammenfassungen zur Verfügung und die Datenebene (Variablen und Konstanten) und die Anweisungsebene sind möglicherweise getrennt. Diese Vorgehensweise kann jedoch recht gefährlich werden, denn Änderungen in der Datenebene können Auswirkungen auf die unterschiedlichsten Programmsegmente haben. Außerdem entspricht eine solche Trennung nicht dem Abbild der realen Natur, wie es sich jeder Mensch vorstellt. Die reale Natur besteht ausschließlich aus Objekten, die durch ihre zusammengehörenden Eigenschaften, ihren Zustand und die Möglichkeiten, was man damit machen kann, gekennzeichnet sind. Diese drei Faktoren sind untrennbar mit einem realen Gegenstand verknüpft.

11 Wenn Sie von Java etwas Ahnung haben, sollten Ihnen die Details nicht unbekannt sein.

Denken Sie zum Beispiel einmal an ein Handy. Dessen Eigenschaften können Sie sich vergegenwärtigen, soweit diese Sie interessieren – Farbe, Form, Gewicht, Hersteller oder Größe. Uninteressante Eigenschaften (eventuell auch nur temporär) wie der Typ des Chips oder des Akkus sind zwar auf jeden Fall Eigenschaften des Objekts, aber Sie beachten diese einfach nicht. Ähnlich verhält es sich mit den Dingen, die Sie mit dem Handy machen können (Methoden). Sie können damit telefonieren[12], SMS schreiben, fotografieren oder spielen. Das Verschicken von MMS ist möglicherweise auch machbar, aber das interessiert Sie vielleicht nicht[13]. Zu guter Letzt sollte auch klar sein, dass bestimmte Funktionalitäten des Handys von einem Zustand abhängen können: im trivialen Fall, ob das Handy überhaupt angeschaltet ist, aber auch, wie der Ladezustand vom Akku ist, ob das Handy gesperrt ist oder ähnliche Dinge. Grundsätzlich charakterisieren diese drei Faktoren aber immer nur gemeinsam das Objekt und man wird bei der Wahrnehmung eines Objekts eine Abstraktion der tatsächlichen Eigenschaften, Methoden und Zustände auf das vornehmen, was wirklich von Interesse ist. Ein Objekt ist also eine Abstraktion eines in sich geschlossenen Elements der realen Welt. Dabei spricht man von Abstraktion, weil normalerweise weder sämtliche Aspekte eines realen Elements benötigt werden noch überhaupt darstellbar sind. Sie bedienen sich auch im realen Leben unzähliger Objekte, um Aufgaben zu erledigen, aber Sie wissen in der Regel nicht genau, wie die Objekte im Inneren funktionieren. Sie können die Objekte bedienen (Sie kennen die Methoden, um es verwenden zu können) und Sie wissen um die Möglichkeiten der Objekte (die jeweiligen Attribute/Eigenschaften).

Wesentlich an der objektbasierten und der objektorientierten Programmierung ist, dass die Methoden und die Daten (Attribute) gemeinsam einem Objekt zugeordnet und nur über dieses zugänglich sind. Dies bedeutet weiterhin, dass es keine Methoden oder Eigenschaften ohne ein zugehöriges Objekt gibt. Nach außen ist ein Objekt nur durch seine Schnittstellen zu den Methoden definiert, es ist gekapselt und versteckt seine innere Struktur vollständig vor anderen Objekten. Man nennt das Verstecken der inneren Struktur eines Objekts **Information Hiding** oder **Datenkapselung**. Der entscheidende Vorteil der Datenkapselung ist, dass ein Verwender weder wissen muss, wie ein Objekt im Inneren tickt und wie sich ein Objekt im Inneren, das heißt bezüglich seiner Eigenschaften und seiner inneren Attribute, vollständig verändern kann. Solange es sich nach außen unverändert zeigt, wird das veränderte Objekt problemlos in einem System funktionieren, wo es in seiner alten Form funktioniert hatte.

5.5.2 Methoden und Eigenschaften eines Objekts verwenden

Methoden und Eigenschaften eines Objekts werden in JavaScript – wie in den meisten Programmiersprachen – über die Punktnotation genutzt. Bei einer Methode sieht das so aus:

12 *Zumindest, wenn das Teil nicht zu neu ist ;-). Bei den meisten neuen Handys ist das Telefonieren ja nur noch Nebensache und die Möglichkeit zum Telefonieren wird gut versteckt ;-).*

13 *Millionäre einmal ausgenommen :-).*

Listing 5.10: Schema zum Verwenden einer Methode in der Punktnotation

```
Objekt.Methodenname(Argument)
```

Beispiel:

Listing 5.11: Die Verwendung einer Methode eines Objekts vom Typ XMLHttpRequest, wie wir es schon gemacht haben

```
resObjekt.open('get', 'laender.jsp?wo='+i,true);
```

Bei einer Eigenschaft sieht dies fast genauso aus. Sie verwenden eine Eigenschaft wie eine normale Variable. Nur stellen Sie auch hier ein Objekt voran:

Listing 5.12: Schema zum Verwenden einer Eigenschaft in der Punktnotation

```
Objekt.Eigenschaftenname
```

Beispiel:

Listing 5.13: Die Verwendung einer Eigenschaft eines Objekts vom Typ XMLHttpRequest

```
resObjekt.onreadystatechange
```

5.5.3 Objektbasierend versus objektorientiert

Obwohl JavaScript Objekte nutzen kann, ist JavaScript keine objektorientierte Sprache. JavaScript ist nur objektbasierend. Der Grund ist, dass eine objektorientierte Sprache gewisse Fähigkeiten bereitstellen muss, die JavaScript nicht hat. In der OOP werden ähnliche Objekte zu Gruppierungen zusammengefasst, die eine leichtere Klassifizierung der Objekte ermöglicht. Die Eigenschaften und Methoden der Objekte werden in den Gruppierungen gesammelt und für eine spätere Erzeugung von realen Objekten verwendet. Diese Beschreibungen oder Baupläne für konkrete Objekte nennt man **Klassen**, die Objekte selbst sind im OO-Sprachgebrauch **Instanzen** dieser Klassen. In der OOP kann man nun eine hierarchische Strukturierung der Klassen vornehmen – von allgemein bis fein. Gemeinsame Erscheinungsbilder sollen also in der objektorientierten Philosophie möglichst in einer höheren Klasse zusammengefasst werden (**Verallgemeinerung**). Erst wenn Unterscheidungen möglich bzw. notwendig sind, welche nicht für alle Mitglieder einer Klasse gelten, werden Untergruppierungen – untergeordnete Klassen – gebildet (so genannte **Spezialisierung**). Jede (gewöhnliche) Klasse kann eine Vielzahl von Unterklassen (**Subklasse**) und konkreten Instanzen (Objekten) haben. Eine übergeordnete Klasse wird **Superklasse** genannt. In der OOP bekommt jede Subklasse dafür freigegebene Eigenschaften und Methoden der Superklasse **vererbt**. Dieses Erbe sollte eine Subklasse um die Dinge erweitern, die die Superklasse noch nicht bereitstellt. So eine **Vererbung** ist nun in JavaScript nicht möglich. Ebenso können Sie mit JavaScript zwar vorgegebene Klas-

sen nutzen, aber keine Klassen im eigentlichen Sinn schreiben. Und damit fehlen bereits einige der wichtigsten Elemente einer objektorientierten Programmiersprache. Aber auch andere OOP-Techniken wie Polymorphismus funktionieren in JavaScript nur eingeschränkt[14].

5.6 Erzeugen von Objekten

Um mit JavaScript ein Objekt nutzen zu können, gibt es verschiedene Wege. Objekte können Ihnen zum Beispiel unmittelbar bereitstehen. Das sind dann entweder Standardobjekte der Laufzeitumgebung oder aber Objekte, die vom Browser beim Laden der Webseite aufgrund der (X)HTML-Struktur der Webseite automatisch angelegt und bereitgestellt werden. Letzteres sehen wir gleich beim erneuten Blick auf DOM und Ersteres zum Beispiel bei der Behandlung des Ausnahmekonzepts.

5.6.1 Explizite Objekterzeugung mit einem Konstruktor

Die andere Situation ist, dass Sie selbst explizit ein Objekt erzeugen müssen. Um in JavaScript aus einer Objektdeklaration (Klasse) explizit eine neue Objektinstanz anzulegen, verwendet man in der Regel wie in Java das reservierte Schlüsselwort `new`, gefolgt von dem Bezeichner der Objektdeklaration und anschließend einem Klammerpaar (eventuell mit Parametern darin). Dies sieht von der Syntax her so aus:

Listing 5.14: Die schematische Erzeugung einer Objektinstanz

```
new [Objektdeklaration]([optionale Parameter]);
```

Die Aufgabe dieser **Konstruktormethode** oder kurz **Konstruktor** ist die Erzeugung eines Objekts des angegebenen Typs. Dabei werden bei Bedarf Initialisierungen vorgenommen, sonstige notwendige Schritte ausgeführt und vor allem wird Speicherplatz für das Objekt reserviert. Sofern ein Objekt seine Arbeit nicht dadurch vollständig erledigt hat, dass es erzeugt wurde und einige Arbeitsschritte durchgeführt hat, weist man eine Referenz darauf einer Variablen zu. Über den Namen der Variablen können Sie dann bei Bedarf auf das Objekt zugreifen und die Methoden und Eigenschaften des Objekts nutzen. Die Zuweisung erfolgt meist in einem Schritt mit der Erzeugung. Das sieht dann im Allgemeinen so aus:

Listing 5.15: Schematisches Erzeugen einer Objektinstanz und Zuweisung zu einer Variablen

```
var [ObjektInstanz] =  new [Objektdeklaration]([opt Parameter]);
```

Beispiele:

14 *Das so genannte Prototyping, was einen gewissen Ersatz für fehlende OOP-Techniken in JavaScript darstellt, werden wir noch ansprechen.*

Listing 5.16: Beispiele für das Erzeugen von Objekten

```
resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
resObjekt = new XMLHttpRequest();
dat = new Date();
```

Hinweis

In JavaScript kann es zu einer Objektdeklaration durchaus mehr als eine Konstruktormethode geben. Die `Date`-Klasse zum Erzeugen eines Datumsobjekts besitzt beispielsweise verschiedene Konstruktormethoden.

5.7 Ausnahmebehandlung in JavaScript

Insbesondere aus Sicht von AJAX ist die **Ausnahmebehandlung** (**Exceptionhandling**) in JavaScript wichtig. Java-Programmierer werden nun auf vertrautem Terrain sein, aber dennoch diverse Unterschiede zu beachten haben. **Ausnahmen** bzw. **Exceptions** bezeichnen Situationen zur Laufzeit eines Programms oder Skripts, die eine unmittelbare Reaktion erzwingen. Andernfalls kann das Programm/Skript nicht sinnvoll weiter ausgeführt werden und muss in der Regel beendet werden. Eine Ausnahme können Sie sich als eine Störung des normalen Programmauflaufs aufgrund einer besonderen Situation vorstellen. Diese Störung macht eine isolierte und unverzügliche Behandlung notwendig, bevor der normale Programmablauf wieder fortgesetzt werden darf. Das Programm bzw. Skript wird also so lange unterbrochen, bis diese Ausnahme behandelt wurde. Ausnahmen sind allgemein ein leistungsfähiges und flexibles Instrument, um bestimmte Reaktionen eines Skripts unabhängig von einem vorgegebenen Zeitpunkt sicherzustellen. Im Fall von JavaScript können Ausnahmen entweder durch den JavaScript-Interpreter automatisch ausgeworfen werden oder aber von dem Programmierer selbst, wenn es notwendig erscheint. Letzteres werden wir hier aber nicht verfolgen.[15]

5.7.1 Was sind Ausnahmen?

Nicht jede Situation im Ablauf eines Programms bzw. Skripts ist vollkommen planbar, z.B. Benutzereingaben, Situationen, in denen sich eine mathematisch unsinnige Berechnung ergibt (etwa die Division durch 0) oder Zugriffsversuche eines Skripts auf das Netzwerk, obwohl der Computer oder der Browser gerade offline ist. Oder auch, wenn Sie ein Objekt erzeugen wollen und die Erzeugung nicht klappt. Solche Probleme müssen aber nicht zwangsläufig zum Abbruch eines Skripts oder Pro-

15 Dazu sei auf spezielle JavaScript-Quellen verwiesen.

gramms führen. Unter Umständen helfen geeignete Gegenmaßnahmen, das Skript oder Programm am Leben zu halten und sinnvoll weiter abarbeiten zu können.

Grundsätzlich kann man auch heutzutage im Web nur JavaScript in der Version 1.3 als etabliert ansehen, obwohl bereits seit recht langer Zeit die Version 1.5 verabschiedet wurde und bereits JavaScript 2 existiert. Die meisten neuen Features von JavaScript 1.5 werden daraus auch in den neuen Browsern unterstützt. Eine der wichtigsten Erweiterungen in JavaScript 1.5[16] ist nun das Ausnahmekonzept. Dieses Behandlungskonzept von nichtregulären Programmsituationen ist sehr stark von Java beeinflusst, obwohl es auch in anderen Sprachen wie C/C++ oder Delphi in verwandter Form vorkommt.

Achtung

So elegant sich die Verwendung der Ausnahmebehandlung in vielen Fällen einsetzen lässt – der Einsatz der Ausnahmebehandlung in JavaScript ist nicht unkritisch, denn es gibt nur wenige Möglichkeiten zum einfachen Schutz von alten Browsern, die mit der Ausnahmebehandlung als solches nicht klar kommen. Wenn Sie eine dieser Techniken einsetzen und ein solcher Browser lädt die Webseite, erhalten Sie einen JavaScript-Fehler.

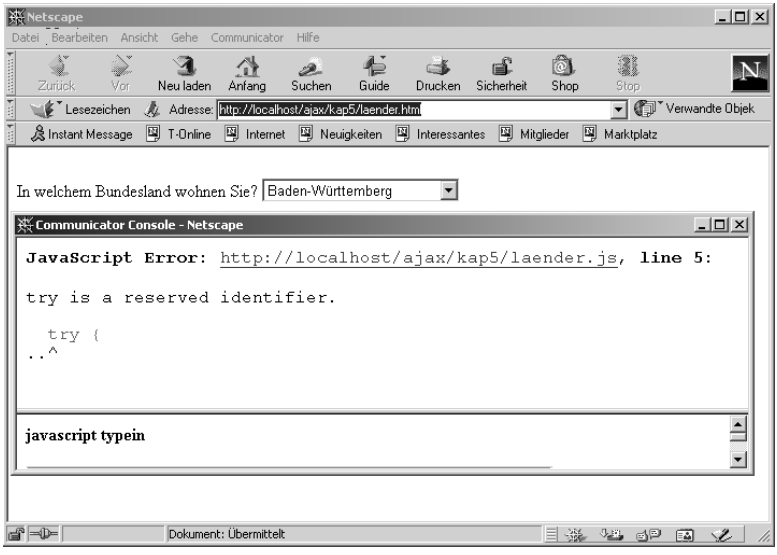


Abbildung 5.1: Ausnahmebehandlung wird in älteren Browsern einen JavaScript-Fehler auslösen – hier zugegebener Maßen der Antikbrowser Netscape Navigator 4.75.

16 *Genau genommen JavaScript 1.4, aber diese Version ist nie wirklich etabliert worden.*

Die Notation eines Skript-Containers mit Angabe der JavaScript-Version zum Abhalten von älteren Browsern ist untauglich, denn Sie müssen mindestens `JavaScript1.4` angeben und damit blockieren Sie beispielsweise den Internet Explorer 6 und auch den Opera 7, obwohl diese Browser hervorragend mit Ausnahmebehandlung zurechtkommen (und auch ein `XMLHttpRequest`-Objekt erzeugen können). Wenn Sie wirklich sichergehen wollen, kommen Sie kaum um eine positiv formulierte umfangreiche Browser-Weiche herum, in der Sie explizit alle Browser samt minimaler Version spezifizieren, für die Sie die Ausnahmebehandlung bereitstellen wollen (und auf jeden Fall auch selbst getestet haben)[17]. Alle anderen Browser sollten zur Sicherheit abgewiesen werden. Für diese stellen Sie konservative Anweisungen bereit. Auf der anderen Seite haben natürlich auch diejenigen nicht Unrecht, die argumentieren, dass alte Browser irgendwann aussterben und man seine Webpräsenz nicht ständig an dem kleinsten gemeinsamen Nenner einer verschwindend kleinen Minderheit orientieren kann. Ein guter Mittelweg zwischen Absicherung gegen alte Browser und dennoch nicht zu restriktiver Beschränkung ist die Angabe der JavaScript-Version 1.3. Darin ist zwar Ausnahmebehandlung noch nicht vorgesehen, aber die Gruppe der Browser, die dann Schwierigkeiten bekommt, ist recht klein.

Das Ausnahmekonzept hat in der »echten« Programmierung unter Sprachen wie Java eine weitaus größere Bedeutung als es in Skriptsprachen der Fall ist. Programme sind zum einen meist komplexer als Skripte und zum anderen gibt es viel mehr Situationen, die eine unmittelbare Reaktion des Programms erfordern (etwa fehlerhafte Zugriffsversuche auf Datenträger). Dennoch erleichtert die Ausnahmebehandlung auch in Skripten in vielen Situationen die Programmierung und sie setzt sich über kurz oder lang in allen Browsern in einheitlicher Form durch. Insbesondere wird AJAX das Konzept weiter etablieren, denn damit lässt sich eine universell einsetzbare Funktion zum Erzeugen eines `XMLHttpRequest`-Objekts erstellen. Allerdings muss man wie gesagt beachten, dass derzeit »in the wild« vielfach noch ältere Browser im Einsatz sind, die die Ausnahmebehandlung in JavaScript nicht unterstützen und zudem die verschiedenen Browser, welche offiziell mit Ausnahmen zurechtkommen, teilweise sehr unterschiedlich damit umgehen.

17 *Das ist jetzt kein Widerspruch dazu, dass eine Browser-Weiche umgangen werden kann, indem die Eigenschaft* `navigator.appName` *verändert wird. Es ist nur eine doppelte Sicherheit, bei der die Browser-Weiche alte Browser von der Ausnahmetechnik schützen soll und unsere nachfolgend noch erstellte universelle Erzeugungsfunktion im nächsten Schritt versucht, ein* `XMLHttpRequest`*-Objekt zu erzeugen. So ist das Verfahren zwar immer noch nicht zu 100% sicher, aber doch ziemlich sicher.*

Ausnahmen als spezielle Objekte

Grundsätzlich setzt eine Ausnahmebehandlung auf einem objektorientierten Konzept auf. Eine Ausnahme ist als Mitteilungsobjekt zu verstehen, das vom Laufzeitsystem bei Bedarf automatisch generiert wird und Informationen über die aktuelle Störung enthält. Beim Auftreten einer Ausnahme kann der Programmierer Gegenmaßnahmen ergreifen, wobei unter Umständen auch die spezifischen Informationen des Ausnahmeobjekts eine sehr genaue Steuerung von Maßnahmen gestatten. Eine Ausnahme ist aber noch mehr als nur ein Informationsobjekt. Das Auftreten eines Ausnahmeobjekts unterbricht wie gesagt unmittelbar einen normalen Ablauf und veranlasst das ausführende System, nach einer geeigneten Behandlungsstruktur zu suchen. Dabei entsteht in einem JavaScript ein Laufzeitfehler, der sich relativ standardisiert beschreiben lässt. **ECMAScript**[18] Edition 3 bzw. JavaScript 1.5 stellen ein Fehlerobjekt `Error` bereit und Instanzen von `Error`-Objekten werden als Ausnahmen generiert (ausgeworfen), wenn ein bekannter Laufzeitfehler auftritt. In der ECMAScript Language Specification Edition 3 gibt es neben `Error` selbst folgende nativen `Error`-Typen, die beim Auftreten des jeweiligen Laufzeitfehlers ausgeworfen werden.

Ausnahmetyp	Beschreibung
`EvalError`	Die JavaScript-Funktion `eval([Zeichenkette])` betrachtet die übergebene Zeichenkette als Zahlen mit zugehörigen Operatoren und berechnet das Ergebnis. Der String kann jede gültige JavaScript-Befehlsstruktur sein. Wenn bei der Evaluierung ein Fehler auftritt, wird zumindest nach den offiziellen Vorgaben diese Ausnahme ausgeworfen. Allerdings machen das die diversen Browser nicht zuverlässig.
`RangeError`	Obwohl JavaScript nur implizit mit Datentypen umgeht, besitzen die Datentypen natürlich Wertebereiche, die einen maximal erlaubten Wert festlegen. Wenn ein numerischer Wert den erlaubten Bereich überschreitet, wird diese Ausnahme ausgeworfen.
`ReferenceError`	Diese Ausnahme tritt bei einer ungültigen Referenz auf.
`SyntaxError`	Diese Ausnahme wird ausgeworfen, wenn syntaktische Fehler beim Parsing des Quelltexts entdeckt werden.
`TypeError`	Diese Ausnahme tritt auf, wenn der Typ eines Operanden von dem erwarteten Typ abweicht.
`URIError`	Diese Ausnahme tritt bei falscher Verwendung des globalen URI-Handlings (Uniform Resource Identifier) auf.

Tabelle 5.2: Typen von Ausnahmeobjekten in ECMAScript bzw. JavaScript

18 Was die Standardisierung von JavaScript festlegt.

Achtung

Selbst wenn ein Browser Ausnahmebehandlung in JavaScript unterstützt, ist die Umsetzung der spezifischen Ausnahmeobjekte in den verschiedenen Browsern sehr unterschiedlich realisiert. Auch wenn ein Browser mit Ausnahmen umgehen kann, bedeutet das noch lange nicht, dass spezifische Ausnahmeobjekte auch unterstützt werden. In den meisten Fällen sind Sie auf der besseren Seite, wenn Sie nur `Error` oder eine davon selbst abgeleitete Ausnahme (das werden wir aber nicht besprechen) verwenden. Andernfalls müssen Sie das Verhalten in jedem Browser testen und gegebenenfalls eine Browser-Weiche voranstellen. Das Verfahren wird auf jeden Fall sehr aufwändig.

5.7.2 Auffangen von Ausnahmen

Wenn eine Ausnahme in Ihrem JavaScript aufgetreten ist, müssen Sie darauf reagieren – das **Auffangen** der Ausnahme[19]. Sie können sowohl allgemein `Error` selbst auffangen (um sehr allgemein zu reagieren) oder ein spezielles Ausnahmeobjekt. Um nun eine ausgeworfene Ausnahme behandeln zu können, umgibt man den Aufruf einer Methode oder Struktur, die möglicherweise eine Ausnahme auswirft, mit einer `try-catch`-Struktur.

Listing 5.17: Die Struktur von try-catch

```
try{
  [kritische Anweisung]
}
catch ([Ausdruck]) {
  [Maßnahmen für den Ausnahmefall]
}
```

Alle potenziell kritischen Anweisungen stehen bei dieser Konstruktion im `try`-Zweig. Der Ausdruck in den Klammern des `catch`-Zweigs ist der Bezeichner für das konkrete Fehlerobjekt. In dem `catch`-Zweig selbst stehen einfach innerhalb geschweifter Klammern die Anweisungen, die beim Auftreten der Ausnahme durchgeführt werden sollen. Der Begriff `try` sagt übrigens bereits sehr treffend, was in diesem Block passiert. Es wird **versucht**, den Code innerhalb des `try`-Blocks auszuführen. Wenn ein Problem auftauchen sollte (sprich, es wird eine Ausnahme ausgeworfen), wird dieses sofort entsprechend im passenden `catch`-Block gehandhabt und alle nachfolgenden Schritte in diesem `try`-Block werden nicht mehr durchgeführt. Wenn also eine der Anweisungen innerhalb des `try`-Blocks ein Problem erzeugt, wird dieses durch die passenden

19 *Diese Bezeichnung erklärt sich aus dem Schlüsselwort, mit dem Ausnahmen manuell »geworfen« werden – `throw`. Für Details dazu sei jedoch auf weiterführende JavaScript-Literatur verwiesen. Oder Sie erinnern sich an Java.*

catch-Anweisungen aufgefangen und entsprechend behandelt (sofern die catch-Anweisung dafür die passende Behandlung enthält). Werden hingegen alle Anweisungen im try-Block ohne Probleme abgearbeitet, wird der catch-Block übersprungen.

Hinweis

Das Auftreten einer Ausnahme innerhalb eines try-catch-Konstrukts führt **nicht** dazu, dass das Skript beendet wird. Es läuft mit den Anweisungen hinter dem try-catch-Konstrukt weiter. Man sagt, die aufgetretene Ausnahme wurde **konsumiert**.

Achtung

Wenn Sie in JavaScript einen try-Block notieren, muss zwingend mindestens ein catch-Block folgen. Fehlt dieser, ist die Reaktion der Browser jedoch uneinheitlich. Während die ersten Browser mit Unterstützung des Ausnahmekonzepts noch eine Fehlermeldung beim Fehlen des catch-Blocks anzeigten, zeigen einige neuere Browser dummerweise gar nichts mehr an. Der try-Block wird einfach ignoriert.

Was Sie nun im catch-Teil konkret tun, ist nicht weiter festgelegt. Sie können jede Ihnen sinnvoll erscheinende Maßnahme ergreifen, die das Skript folgerichtig weiterlaufen lässt. Oft ist das eine Fehlermeldung mit anschließendem Rücksprung zu der Situation, die sich mit der ausgelösten Ausnahme sinnvoll umgehen lässt, etwa im Fall einer Benutzereingabe ein Fokussieren des kritischen Eingabefelds. Oder aber Sie können einen alternativen Weg zur Durchführung einer Aktion probieren, wenn der erste Weg gescheitert ist. In letzterem Fall werden Sie oft auch try-catch-Strukturen verschachteln.

Achtung

Wenn Sie das Ausnahmebehandlungskonzept in JavaScript einsetzen wollen, sollten kritische Aktionen also immer innerhalb eines try-Blocks durchgeführt werden. Aber noch einmal die Warnung – Sie sollten sicherstellen, dass der Besucher einen Browser verwendet, der mit Ausnahmebehandlung zurechtkommt. In vielen Fällen ist Ausnahmebehandlung zwar elegant, aber eine konservative Programmierung (etwa mit dem vorangestellten Testen von potenziell kritischen Variablenwerten) ist in der Praxis derzeit noch bedeutend besser.

Betrachten wir nun ein Beispiel zum Umgang mit Ausnahmen, das explizit in AJAX seine Anwendung findet. Wir optimieren die Erstellung eines `XMLHttpRequest`-Objekts.

5.7.3 Eine universelle Funktion zum Erzeugen eines XMHttpRequest-Objekts

Das Ausnahmekonzept wird der Schlüssel sein, um eine universelle, portable JavaScript-Funktion zum Erzeugen eines `XMHttpRequest`-Objekts zu schreiben. Gerade bei AJAX können Sie davon ausgehen, dass das Ausnahmekonzept eingesetzt werden darf. Denn damit Browser AJAX unterstützen können, müssen sie sowieso auf einem ziemlich neuen Stand der Technik sein. Auch sonst haben wir mit Funktionen, Konstruktormethoden zur Erzeugung von Objekten als auch der Objektthematik selbst jetzt alle Voraussetzungen geschaffen, um erneut einen Blick auf die Schritte zu werfen, die wir in Kapitel 2 zur Erzeugung eines `XMLHttpRequest`-Objekts durchgeführt haben. Da gab es einmal die Anweisungen `resObjekt = new ActiveXObject("Microsoft.XMLHTTP");` und `resObjekt = new ActiveXObject("MSXML2.XMLHTTP");`, um ein solches Objekt für Microsoft-kompatible Browser zu erstellen. Für andere Browser musste die Erstellung über `resObjekt = new XMLHttpRequest();` laufen.

Hinweis

Für den Internet Explorer können Sie bei der Instanzierung des Microsoft XML-Parsers auch eine bestimmte Version angeben, wenn Sie sicher sind, dass diese bei einem Anwender vorhanden ist. Etwa so:

Listing 5.18: Eine Versionsangabe bei der Instanzierung

```
resObjekt = new ActiveXObject( "Msxml2.XMLHTTP.5.0");
```

Derzeit sind die Angaben 3.0, 4.0 und 5.0 möglich. Allerdings gibt es kaum einen Grund, diese explizite Festlegung zu wählen. Sie handeln sich nur potenzielle Probleme ein.

Im Fall Microsoft-kompatibler Browser ist der Aufruf von `ActiveXObject()` mit vorangestelltem `new` die Anwendung einer Konstruktormethode, die dort ein Objekt erzeugt. Sie kann aber mit unterschiedlichen Werten für ihren Parameter verwendet werden (in Abhängigkeit von der installierten DLL für das XML-Parsen). Im Fall anderer Browser wird die Konstruktormethode `XMLHttpRequest()` ohne Parameter verwendet.

Die Unterscheidung zwischen den beiden Microsoft-Varianten haben wir bisher damit gelöst, einen der beiden Methodenaufrufe auszukommentieren. Das ist definitiv nicht sehr elegant und auch wenig praxistauglich. Und um zu gewährleisten, dass jeweils die richtige Konstruktormethode für die unterschiedlichen Browser-Welten

verwendet wird, kam eine Browser-Weiche zum Einsatz. Nun erinnern wir uns daran, dass Browser-Weichen[20] ziemlich unzuverlässig sein können, da der Wert der `navigator`-Eigenschaft `appName`[21], über die sich ein Browser identifiziert, auf vielfältige Weise geändert werden kann.

Wir werden nun als Alternative zu einer Browser-Weiche und dem Auskommentieren eines Teils vom Quelltext die Ausnahmebehandlung verwenden und einfach **versuchen**, die drei möglichen Konstruktoranwendungen nacheinander auszuführen. Der Trick beruht darauf, dass es entweder klappen kann. Dann wird ein `XMLHttpRequest`-Objekt erzeugt und wir können zufrieden sein. Oder aber die Erzeugung geht schief, weil ein falscher Browser oder ein falscher XML-Parser die Seite lädt. Dann wird eine Ausnahme ausgeworfen, die wir auffangen. Die Ausnahme an sich ist uninteressant. Aber im zugehörigen `catch`-Teil probieren wir einfach die nächste Variante. So werden im Fall eines Scheiterns der Objekterzeugung sukzessive die anderen Möglichkeiten ausprobiert. Betrachten Sie das nachfolgende Listing:

Listing 5.19: Universelle Erzeugung eines XMLHttpRequest-Objekts

```
01 function erzXMLHttpRequestObject(){
02   var resObjekt = null;
03   try {
04     resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
05   }
06   catch(Error){
07     try {
08       resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
09     }
10     catch(Error){
11       try {
12       resObjekt = new XMLHttpRequest();
13       }
14       catch(Error){
15         alert(
16           "Erzeugung des XMLHttpRequest-Objekts ist nicht möglich");
17       }
18     }
19   }
20   return resObjekt;
21 }
```

Die Funktion legt zuerst eine Variable `resObjekt` an. Diese soll eine Referenz auf das `XMLHttpRequest`-Objekt aufnehmen und als Rückgabewert der Funktion dienen. Am Anfang hat sie den Wert `null`. Damit wird sie von JavaScript implizit den Datentyp `Object` erhalten. In Zeile 4 versuchen wir das erste Mal, ein `XMLHttpRequest`-Objekt zu

20 Siehe dazu auch Seite 141.

21 Mit navigator sehen Sie ein Objekt, das Ihnen automatisch in JavaScript zur Verfügung steht – siehe dazu die folgende Betrachtung zu DOM.

erzeugen. Es ist im Grunde vollkommen ohne Belang, welche der drei zur Verfügung stehenden Varianten wir zuerst probieren. Hier wird mit `resObjekt = new ActiveXObject("Microsoft.XMLHTTP");` zuerst versucht, das Objekt mit dem älteren XML-Parser von Microsoft zu erzeugen. Hat dies funktioniert, wird der nachfolgende `catch`-Block komplett übersprungen und mit `return resObjekt;` (Zeile 20) das `XMLHttpRequest`-Objekt zurückgegeben.

Geht der erste Versuch schief, wird eine Ausnahme ausgeworfen und der Programmfluss macht mit der ersten Anweisung im ersten `catch`-Block (Zeile 7) weiter. Wir versuchen dort dann, das Objekt mit der neueren Version des XML-Parser von Microsoft zu erzeugen (Zeile 8 – `resObjekt = new ActiveXObject("MSXML2.XMLHTTP");`). Auch hier gilt wieder: Hat dies funktioniert, wird der nachfolgende `catch`-Block komplett übersprungen und mit `return resObjekt;` (Zeile 20) das `XMLHttpRequest`-Objekt zurückgegeben. Ist auch der zweite Versuch in die Hose gegangen, springt der Programmfluss in den zweiten `catch`-Block und fährt mit Zeile 11 fort. Dort starten wir eine dritten Versuch, das Objekt mit `new XMLHttpRequest();` zu erzeugen (Zeile 12).

Nun ist die weitere Abfolge eine Frage der Programmlogik. In unserem Fall packen wir auch den dritten Versuch in ein `try-catch`-Konstrukt. Das ist im Grunde nicht notwendig, denn wenn auch der dritte Versuch scheitert, sollte das Skript zum Nachladen von Daten per AJAX beendet werden. Eine nicht aufgefangene Ausnahme würde genau das bewirken. Wir gehen jedoch etwas eleganter vor. Im Erfolgsfall gilt wieder, dass wie gewünscht das Objekt zurückgegeben wird. Bei Misserfolg wird dem Besucher eine Fehlermeldung angezeigt und dann von der Funktion der Wert `null` zurückgegeben (das ist der Initialisierungswert – dieser wurde nicht verändert, wenn das Skript bis zu dieser Stelle kommt).

Abbildung 5.2: Wenn die Erzeugung des XMLHttpRequest-Objekts nicht funktioniert, bekommt der Anwender in dem Beispiel eine Fehlermeldung angezeigt – zumindest, wenn die Ausnahmebehandlung unterstützt wird.

Es ist natürlich eine Frage der Benutzerführung, ob so eine Fehlermeldung für den Anwender sinnvoll ist. Die Meldung können Sie gerne weglassen, aber falls das Skript oder die Webseite auch ohne die Erzeugung eines `XMLHttpRequest`-Objekts noch sinnvoll ist, unterdrücken Sie mit dem `catch`-Block die lästige und für den typischen Anwender ziemlich nichtssagende Standardfehlermeldung des Browsers. Stattdessen können Sie den Anwender darauf hinweisen, dass das Nachladen von Zusatzinformationen nicht funktioniert, Sie ihm aber eine etwas eingeschränkte Webseite zur Verfügung stellen. Und mit dem Rückgabewert `null` können Sie auf jeden Fall Ihr

Skript so steuern, dass es nach dem Aufruf der Funktion abgebrochen werden kann, wenn die Erzeugung des `XMLHttpRequest`-Objekts missglückt ist. Oder Sie leiten zu einer anderen Webseite weiter oder etwas Ähnliches.

Hinweis

Noch einmal der Hinweis, dass die erfolgreiche Erzeugung eines `XMLHttpRequest`-Objekts[22] bei Besuchern ziemlich neue Browser voraussetzt. Selbst keineswegs antike Browser wie der Netscape Navigator 6, der Internet Explorer 5.5 oder Opera 5.11 scheitern an der Erzeugung des Objekts. Obwohl diese bereits gut mit Ausnahmebehandlung umgehen können, also JavaScript 1.5 teilweise oder sogar ganz implementiert haben.

Hinweis

Unsere Funktion zum portablen Erzeugen des `XMLHttpRequest`-Objekts kann auch so formuliert werden, dass Sie statt `Error` selbst Ausnahmen von speziellerem Typen (Subklassen von `Error`) auffangen. Wenn das Erzeugen mit der `ActiveXObject()`-Methode nicht funktioniert, wird im Internet Explorer oder kompatiblen Browsern eine Ausnahme vom Typ `err_MSXML2` oder `err_Microsoft` ausgeworfen. Und auch diese können Sie individuell abfangen. Allerdings spricht kaum ein Argument für eine so spezielle Anwendung. Sie benötigen im Normalfall bei der Erzeugungsfunktion einfach keine Unterscheidung nach der Art der Ausnahme. Entweder hat die Erzeugung funktioniert oder nicht. Falls nicht, wird einfach durchprobiert und sich keine Gedanken gemacht, warum die Sache schief ging. Dafür sprechen aber mehrere Gründe dagegen, statt `Error` spezialisierte Subklassen aufzufangen. Sie können sich durch die Auswahl einer speziellen Ausnahmeklasse nur Probleme einfangen. Angenommen, ein Browser XYZ, der nicht Microsoft-kompatibel ist, versucht, die `ActiveXObject()`-Methode anzuwenden. Das wird schief gehen. Aber wie wahrscheinlich ist es, dass dieser Browser dann eine Microsoft-abhängige Ausnahme vom Typ `err_MSXML2` oder `err_Microsoft` auswerfen wird? Nicht sehr hoch. Sehr wahrscheinlich wird eine Ausnahme vom Typ `Error` oder einer speziellen Subklasse ausgeworfen, die für diese Browser-Plattform spezifisch ist. Sie müssten zig Spezialfälle betrachten und dennoch als Defaultfall `Error` selbst auffangen, um keinen Fall zu übersehen. Wenn Sie von vornherein allgemein nur `Error` auffangen, werden damit auch alle Ausnahmen aufgefangen, die vom Typ einer beliebigen Subklasse sind. Damit sind Sie aller Sorgen

22 *Und damit der grundsätzliche Einsatz von AJAX.*

ledig. Auch weitere spezielle Ausprägungen bei der Erzeugung des-Objekts sind im Grunde nur potenzielle Fehlerquellen. Sie sollten bei der Erzeugung des `XMLHttpRequest`-Objekts zwar unbedingt die Welt des Internet Explorers von der restlichen Welt abschotten, aber sonst so universell wie möglich arbeiten.

5.7.4 Behandlung von mehreren Ausnahmen

Wie wir gerade gesehen haben, ist ein Verschachteln von `try-catch`-Strukturen möglich, wobei dann auch außerhalb angesiedelte `catch`-Blöcke im Inneren nicht explizit aufgefangene Exceptions auffangen können. Damit können unterschiedliche Arten von Ausnahmen auch verschiedenartig – und damit sehr qualifiziert – gehandhabt werden. Wenn nun eine Ausnahme auftritt, wird die aufrufende Methode oder Funktion nach einer Ausnahmebehandlung durchsucht. Enthält die aufrufende Methode bzw. Funktion eine Ausnahmebehandlung, wird mit dieser die Ausnahme bearbeitet und das Ausnahmeobjekt beseitigt. Ist dort keine Routine vorhanden, wird die Suche in der Aufrufhierarchie nach oben fortgesetzt. Sofern eine aufgetretene Ausnahme nirgends aufgefangen wird, bricht der Interpreter normalerweise die Ausführung des Skripts ab. Dieses Weiterreichen der Behandlung über verschiedene Hierarchieebenen erlaubt sowohl die unmittelbare Behandlung ganz nahe am Ort des Problems als auch eine entferntere Behandlung, wenn dies sinnvoll ist – etwa in einer mehrere potenzielle Probleme umgebenden Struktur.

Hinweis

Die andere Version der Behandlung von mehreren Ausnahmen notiert einfach auf der gleichen Ebene mehrere `catch`-Blöcke. Das sieht schematisch so aus:

Listing 5.20: Mehrere catch-Blöcke hintereinander

```
try{
  [kritische Anweisung]
}
catch ([Ausdruck1]) {
  [Maßnahmen für den ersten Ausnahmefall]
}
catch ([Ausdruck2]) {
  [Maßnahmen für den zweiten Ausnahmefall]
}
... // weitere catch-Blöcke
catch ([Ausdruckn]) {
  [Maßnahmen für den n-ten Ausnahmefall]
}
```

So etwas ist etwa in **Java** gängige Praxis, in JavaScript aber **nicht** erlaubt. Stattdessen notieren Sie einfach getrennte `try-catch`-Strukturen für jeden einzelnen Fall.

5.8 Erstellen eigener Objekte und Prototyping in JavaScript

JavaScript ist zwar keine vollständig objektorientierte Sprache und sowohl das Schreiben von Klassen im eigentlichen Sinn als auch der Mechanismus der Vererbung sind nicht vorgesehen. Dennoch – neben der Verwendung der vordefinierten Standardobjekte über JavaScript kann man auch – eingeschränkt – eigene Objekte schreiben. Damit Sie in JavaScript nun ein eigenes Objekt anlegen können, sind effektiv zwei Schritte nötig.

1. Eine eigene Objektdeklaration muss erstellt werden.
2. Mithilfe dieser Objektdeklaration wird dann wie bei vordefinierten Objektdeklarationen eine konkrete Objektinstanz erstellt.

5.8.1 Erstellen einer Konstruktormethode

Um ein eigenes Objekt anzulegen, müssen Sie innerhalb von JavaScript einfach eine spezielle Funktion definieren, die als Deklaration des Objekts (als seine Konstruktormethode) fungiert. Nachdem die Objektdeklaration angelegt ist, können Sie an anderen Stellen innerhalb Ihres JavaScripts Instanzen dieses Objekts erzeugen. Dies machen Sie wie bei Standardklassen, indem Sie einer Variablen mit dem Schlüsselwort `new` eine Objektinstanz zuweisen. Dies sieht also von der Syntax her schematisch so aus:

Listing 5.21: Schema einer Objektdeklaration und das Erzeugen einer Objektinstanz

```
function [Funktionsname](){
 ... [Beschreibungen von Eigenschaften] ...
}
...
var [ObjektInstanz] = new [Funktionsname]();
```

this

In der Konstruktormethode ist ein JavaScript-Schlüsselwort von zentraler Bedeutung – `this`. So ein Schlüsselwort ist ein wesentliches Element jeder Programmiersprache, in der man Objekte erzeugen kann. Denn darüber hat man Zugriff auf das Objekt selbst, welches eine Methode aufruft. Wenn Sie in JavaScript in der Form `this.[variable]` in der Konstruktormethode eine Variable deklarieren, ist das ein Zugriff auf eine Eigenschaft von dem irgendwann daraus erzeugten Objekt. Beispiel:

Listing 5.22: Eine Deklaration einer Eigenschaft

```
function erzXMLHttpRequestObject(){
  this.resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Bei Funktionen (bzw. Methoden) gehen Sie ähnlich vor. Sie definieren in Ihrem Skript eine beliebige Funktion und verankern diese dann in der Konstruktormethode über `this`. Dabei müssen Sie aber die Funktion dort als **Funktionsreferenz ohne** Klammern notieren! Eine Funktionsreferenz ist ein Verweis auf eine Funktion und kein Aufruf. Beispiel:

Listing 5.23: Deklaration einer Methode

Listing 5.24: Verankerung als Funktionsreferenz in der Konstruktormethode

```
function a(){
  document.write(new Date());
}

function MeinObjekt(){
  this.a = a;
}
```

Hinweis

In unserer Funktion `sndReq()` in Kapitel 2 stand in Zeile 13 die Anweisung `resObjekt.onreadystatechange = handleResponse;`. Mit `handleResponse` geben Sie eine Funktionsreferenz an.

Ein Objekt erzeugen Sie dann einfach so:

Listing 5.25: Erzeugen einer Objektinstanz

```
new MeinObjekt();
```

Das nachfolgende Beispiel soll die oben definierte Funktion `erzXMLHttpRequestObject()` über das Schlüsselwort `this` als Funktionsreferenz in der Objektdeklaration `Erzeuge-AJAXObjekt()` verankern.

Hinweis

Beachten Sie die Wahl eines Großbuchstabens für den Beginn des Namens der Konstruktormethode. Allgemein beginnen Funktionen in JavaScript per Übereinkunft mit einem Kleinbuchstaben. Bei Konstruktormethoden ist es jedoch beispielsweise in Java üblich, dass diese mit Großbuchstaben beginnen (da dort der Name der Konstruktormethode zwingend der der Klasse ist und in Java Klassen per Übereinkunft immer mit einem Großbuchstaben beginnen). Es ist sehr sinnvoll, in JavaScript die Namenskonventionen aus Java zu übernehmen, die auch dort sinnvoll sind.

Listing 5.26: Verankerung als Funktionsreferenz in der Konstruktormethode

```
function ErzeugeAJAXObjekt(){
  this.erzXMLHttpRequestObject = erzXMLHttpRequestObject;
}
```

Damit wurde aus unserer universalen Funktion zum Erzeugung eines `XMLHttpRequest`-Objekts eine Methode generiert, die genau so ein solches Objekt erzeugt. Die Erzeugung und Anwendung der Objektdeklaration erfolgt so:

Listing 5.27: Erzeugen eines Objekts aus der selbst geschriebenen Objektdeklaration und Aufruf einer Methode

```
o = new ErzeugeAJAXObjekt();
resObjekt = o.erzXMLHttpRequestObject();
```

Sie werden sich möglicherweise nun fragen, was der Aufwand mit der Erzeugung eines eigenen Objekts soll? Die Frage ist nicht ganz einfach zu beantworten, weil mit JavaScript im Grunde kaum so komplexe Anwendungen geschrieben werden, dass die Vorteile offensichtlich werden. In der Praxis werden die überwiegende Anzahl der JavaScripts ohne die Erzeugung eigener Objekte auskommen. Aber grundsätzlich wird Code viel modularer und leichter wiederverwendbarer, wenn Sie ihn in Objekte strukturieren. Und ein Effekt des Aufbaus von Objekten ist die Möglichkeit der Erweiterung.

5.8.2 Prototyping – Erweiterung eines bestehenden Objekts bzw. einer Objektdeklaration

Vererbung zählt zu den Kernprinzipien der objektorientierten Programmierung, denn nur darüber lassen sich Spezialisierungen von bestehenden Strukturen ohne großen Aufwand realisieren. Der Sinn und Zweck ist, dass man in vielen Fällen in der Programmierung nicht jedes Mal das Rad neu erfinden, sondern bereits bestehende Funktionalitäten immer wieder nutzen und nur für eigene Zwecke anpassen möchte.

Allgemein haben Sie in der Programmierung nur zwei Möglichkeiten, bestehende Funktionalitäten für sich zu nutzen. Ein Weg führt darüber, dass Sie diese »kaufen«[23]. Wenn Sie in JavaScript ein Objekt (oder genauer – eine Eigenschaft oder Methode eines Objekts) verwenden, kaufen Sie dessen Funktionalität. Der Aufruf einer Funktion bedeutet das Gleiche. Der andere Weg führt speziell in der OOP über Vererbung, was eine Erweiterung einer bestehenden Funktionalität bedeutet. Diese wird also noch nicht explizit verwendet, sondern nach eigenen Vorstellungen angepasst. In JavaScript haben Sie – wie mehrfach erwähnt – keine Möglichkeit zur Vererbung. Aber Sie können in neueren Sprachversionen zumindest so genanntes **Prototyping** betreiben, was eine Art Ersatz für Vererbung darstellt. Das bedeutet, eine Objektdeklaration (die Konstruktormethode) bzw. ein Objekt wird – ähnlich wie bei der Vererbung – mit neuen Eigenschaften und/oder Methoden erweitert. Beachten Sie, dass in JavaScript ein Konstruktor selbst als Objekt betrachtet wird, der Eigenschaften bereitstellt.

Ein bereits fertiger Konstruktor oder auch eine der Standardklassen von JavaScript kann um eventuell fehlende Funktionalität erweitert werden (was ja bei Vererbung genau so geschieht). Besonders interessant ist Prototyping jedoch, wenn die Technik auf ein bereits existierendes Objekt angewendet wird. Für ein existierendes Objekt bedeutet Prototyping, dass es erweitert werden kann, nachdem (!) es bereits mit `new` erzeugt wurde. Die prototypbasierte Vererbung in JavaScript verkettet implizite Referenzen jedes neu erzeugten Objekts auf seinen Prototyp. Das JavaScript-Schlüsselwort (die Eigenschaft, die sowohl jedes erzeugte Objekt als auch der Konstruktor selbst bereitstellt) für diesen Zweck heißt `prototype` und die allgemeine Syntax dazu ist die folgende:

Listing 5.28: Erweiterung einer Eigenschaft per Prototyping

```
[Objekttyp].prototype.[Eigenschaft] = [Wert];
```

bzw.

Listing 5.29: Erweiterung einer Methode per Prototyping

```
[Objekttyp].prototype.[Methode] = [Methode];
```

Sobald diese Aktion durchgeführt wurde, besitzen alle zukünftig erzeugten, aber auch alle vorher bereits erzeugten Objekte die neue Eigenschaft bzw. Methode. Insbesondere ist zu beachten, dass nach der `prototype`-Anweisung erzeugte Objekte für die zugefügte Eigenschaft immer den über `prototype` angegebenen Vorgabewert besitzen, solange dieser nicht explizit mit einer Zuweisung verändert wurde. Betrachten Sie das nachfolgende Beispiel (*proto1.html*):

23 *Das ist in der OOP eine übliche Bezeichnung.*

Listing 5.30: Prototyping zur Erweiterung einer Objektdeklaration

```
01 <html>
02 <body>
03 <script language="Javascript">
04 function Ball(groesse, farbe) {
05   this.groesse = groesse;
06   this.farbe = farbe;
07 }
08 b1 = new Ball(40,"rot");
09 document.write("Der erste Ball ist " + b1.groesse +
10 " cm groß und " + b1.farbe + ". ");
11 Ball.prototype.besitzer = "Ralph";
12 document.write(
13   "Besitzer von Ball 1 ist " + b1.besitzer + ".<hr />" );
14 b2 = new Ball(30,"blau");
15 document.write("Der zweite Ball ist " + b2.groesse +
16   " cm groß und " + b2.farbe + ". ");
17 document.write(
18   "Besitzer von Ball 2 ist " + b2.besitzer + ".<hr />" );
19 b2.besitzer = "Florian";
20 document.write("Nun gehört Ball 2 " + b2.besitzer + ".<hr />" );
21 </script>
22 </body>
23 </html>
```

Die Konstruktormethode `Ball()` definiert zwei Eigenschaften für die Instanzen, die damit erzeugt werden – `groesse` und `farbe`. In Zeile 8 wird der erste Ball mit spezifischen Werten für diese Eigenschaften erzeugt. Zeile 9 und 10 geben die Werte der jeweiligen Eigenschaften aus. In Zeile 11 erfolgt die Erweiterung der Objektdeklaration um eine neue Eigenschaft (`Ball.prototype.besitzer = "Ralph";`). Beachten Sie, dass bei der nachfolgenden Ausgabe des Werts von dieser Eigenschaft über das bereits erzeugte Objekt `b1` der entsprechende Wert angezeigt wird (Zeile 12 und 13: `document.write("Besitzer von Ball 1 ist " + b1.besitzer + ".<hr />" );`).

In Zeile 14 wird ein neues Objekt erzeugt (`b2 = new Ball(30,"blau");`). Die nachfolgende Ausgabe in Zeile 15 und 16 (`document.write("Der zweite Ball ist " + b2.groesse +" cm groß und " + b2.farbe + ". ");`) zeigt die verwendeten Werte der Parameter. Aber die Ausgaben in den Zeilen 17 und 18 (`document.write("Besitzer von Ball 2 ist " + b2.besitzer + ".<hr />" );`) beweisen, dass der Wert von `b2.besitzer` immer noch den Vorgabewert der Prototyping-Aktion besitzt. Erst nachdem in Zeile 19 der Wert explizit mit `b2.besitzer = "Florian";` geändert wurde, hat `b2.besitzer` den neuen Wert.

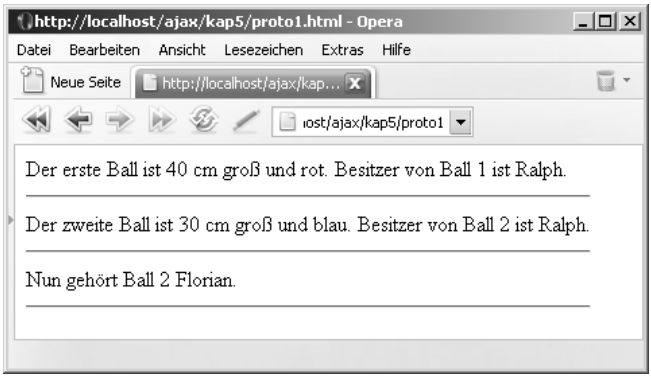


Abbildung 5.3: Die Objektdeklaration wurde erweitert

Hinweis

Eine praktische Anwendung von Prototyping im Rahmen von AJAX wäre der Fall, dass wir unsere universale Methode zum Erzeugen des `XMLHttpRequest`-Objekts erweitern oder ändern wollen. Das kann dann sinnvoll sein, wenn weitere Konstruktormethoden notwendig werden, um neue Browser abzudecken.

5.9 DOM aus Sicht von JavaScript und native JavaScript-Objekte

Wie wir gesehen haben, können Sie über JavaScript sowohl einige Standardobjekte der Laufzeitumgebung als auch selbstdefinierte Objekte verwenden. Die meisten Objekte, die Sie verwenden und die Ihnen in Form einer Objektbibliothek bereitgestellt werden, gehören aber gar nicht zu JavaScript, sondern zum DOM-Konzept, auf das wir schon an einigen anderen Stellen zu sprechen kamen. Ein DOM-fähiger Browser wird beim Laden einer Webseite alle ihm im Rahmen des Konzepts bekannten und einzeln identifizierbaren Elemente einer Webseite bezüglich ihres Typs, ihrer relevanten Eigenschaften und ihrer Position innerhalb der Webseite indizieren. Für Skripte verfügbar werden diese Objekte über spezielle Arrays (so genannte Objektfelder) oder ihren Namen, sofern bei ihnen im HTML-Tag das `name`-Attribut gesetzt wurde[24].

24 *Mit DOM Level 1 hatte das W3C ursprünglich eine abstrakte Schnittstelle geschaffen, über die sich allgemein hierarchisch strukturierte Dokumente unabhängig von einer konkreten Plattform beschreiben lassen. Die in Webbrowsern aus clientseitigen Skripten nutzbaren Objekte stellen eine Teilmenge der modular aufgebauten zweiten Version einer ganzen DOM-Familie dar. Diese umfasst neben der Kernspezifikation ein HTML-spezifisches Objektmodell und einen Mechanismus zur Behandlung von Ereignissen.*

Viele Objekte im DOM-Konzept sind in Form einer Objekthierarchie verfügbar. Allerdings sind nicht sämtliche Objekte in einer einzigen Hierarchiebeziehung miteinander verbunden. Wenn ein Objekt einem anderen untergeordnet ist, notiert man das in der DOT-Notation, indem man erst den Namen des oberen Objekts und dann den des darunter angesiedelten Objekts notiert. Wenn man beispielsweise eine Webseite nimmt, ist sie über das Objekt `document` aus JavaScript heraus verfügbar. Da sich die Webseite in einem Browser-Fenster befindet und dieses als `window` ansprechbar ist, erfolgt der Zugriff über `window.document`. Ein Formular in einer Webseite ist über ein `document` untergeordnetes Objektfeld mit Namen `forms` und einem Index verfügbar. In der DOT-Notation schreibt man das dann (für das erste Formular der Webseite) so:

Listing 5.31: Zugriff auf das dritte Formular in einer Webseite

```
window.document.forms[2]
```

Formularelemente sind nun wieder über das dem `forms`-Objektfeld untergeordneten Objektfeld `elements` verfügbar. Wenn man im dritten Formular auf einer Webseite das erste Element ansprechen will, geht das über die DOT-Notation so:

Listing 5.32: Zugriff auf das erste Formularfeld im dritten Formular in einer Webseite

```
window.document.forms[2].elements[0]
```

Das Verfahren setzt sich analog auf Eigenschaften und Methoden fort. Jedes Objekt stellt seine spezifischen Eigenschaften und objektgebundenen Methoden über die DOT-Notation bereit. Zugreifen auf Eigenschaften bedeutet Lesen der Werte, in vielen Fällen ist aber auch eine Veränderung der Eigenschaftswerte möglich. Wenn man Werte von Eigenschaften auslesen möchte, kann man die Objektnotation direkt an der Stelle im Skript notieren, wo sonst ein Literal oder eine Variable steht. Die Veränderung von Eigenschaften ist – falls das Objekt es zulässt – genauso möglich, wie Sie den Wert einer normalen Variablen ändern. Der Zugriff auf Methoden erfolgt einfach über den Namen des Objekts und dahinter durch einen Punkt getrennt den Namen der Methode samt Klammern und eventuell notwendigen Parametern.

Tipp

Unter gewissen Umständen kann die DOT-Notation um die Angabe der Objekte verkürzt werden, die offensichtlich sind. Sie können deshalb meist auf das Voranstellen des die Webseite selbst repräsentierenden Objekts `window` verzichten. Statt `window.prompt()` oder `window.alert()` notiert man einfach `prompt()` oder `alert()`. Oder auch direkt `document` statt `window.document`.

5.9.1 Verfügbare Klassen und DOM-Objekte

Aus JavaScript stehen unter anderem die folgenden DOM-Objekte bzw. nativen Klassen zur Erzeugung von Objekten für eine Verwendung in einer Webseite zur Verfügung. Beachten Sie, dass nicht alle Browser sämtliche Objekte unterstützen und es auch in der Art der Unterstützung zahlreiche Unterschiede gibt (hauptsächlich bei den DOM-Objekten). Wir trennen in der Tabelle nicht zwischen Objekten, die explizit zu DOM gehören, und Klassen, die nativ zu JavaScript gehören[25].

Objekt	Beschreibung
`all`	Das Objekt ermöglicht den direkten Zugriff auf alle Elemente einer HTML-Datei. Es gehört aber nicht zum offiziellen DOM-Standard, sondern ist eine Implementation für den Internet Explorer ab der Version 4.0. Das Objekt wird zwar von fast allen Browsern heutzutage ansatzweise unterstützt, aber den vollen Umfang sollte man nur im Internet Explorer voraussetzen! Sie sollten bei einem Einsatz auf jeden Fall alle relevanten Browser testen, die Sie unterstützen wollen.
`anchor`	Das Objekt beinhaltet eine Referenz auf einen Verweisanker in einer HTML-Datei.
`applet`	Das Objekt beinhaltet eine Referenz auf ein Java-Applet in einer HTML-Datei.
`Array`	Über diese Klasse werden Array-Objekte erzeugt. Deren Elemente können über einem gemeinsamen Bezeichner und einen Index angesprochen werden.
`Boolean`	Über diese Klasse wird ein Objekt mit Wahrheitswerten (`true` oder `false`) als Inhalt erzeugt.
`Date`	Über diese Klasse wird ein Objekt mit Informationen zu Datum und Uhrzeit erzeugt. Es gibt diverse Konstruktoren, um damit sowohl das aktuelle Systemdatum abzufragen als auch ein Datumsobjekt mit einem beliebigen Wert zu setzen. Darüber sind dann Datumsberechnungen möglich.
`document`	Dieses Objekt repräsentiert die Webseite selbst.
`event`	Ein Objekt, das bei Anwenderereignissen erzeugt wird und für die (zentrale) Ereignisbehandlung genutzt werden kann. Allerdings ist diese zentrale Ereignisbehandlung auch heutzutage inkonsequent in den verschiedenen Browser-Welten umgesetzt. Wir gehen im folgenden Abschnitt genauer darauf ein.
`form`	Das Objekt beinhaltet eine Referenz auf ein Objekt, das ein Formular in einer HTML-Seite repräsentiert.
`frame`	Das Objekt beinhaltet eine Referenz auf ein Frame in einer HTML-Seite.
`Function`	Ein Klasse zum Erzeugen eines Objekts mit JavaScript-Funktionen.
`history`	Dieses Objekt enthält Informationen über die URLs, die ein Anwender besucht hat.

Tabelle 5.3: Unter JavaScript verfügbare Objekte

25 *Die Unterscheidung ist aber einfach. Alle mit einem Großbuchstaben beginnenden Token bezeichnen native Klassen von JavaScript. Die mit einem Kleinbuchstaben beginnenden Token sind DOM-Objekte.*

Objekt	Beschreibung
`image`	Das Objekt beinhaltet eine Referenz auf eine Grafik in einer HTML-Datei.
`layer`	Das Objekt beinhaltet eine Referenz auf einen Layer in einer HTML-Datei. Das Objekt wurde im alten Netscape-DOM-Konzept unterstützt. Das neue Netscape-Modell, das bereits mit dem Navigator 6 eingeführt wurde, unterstützt es nicht mehr und auch andere Browser kommen damit nicht zurecht. Das Objekt ist veraltet und sollte nicht mehr eingesetzt werden.
`link`	Das Objekt beinhaltet eine Referenz auf Verweise in der aktuellen HTML-Datei.
`location`	Über das Objekt besteht Zugriff auf die Adresszeile des Browsers.
`Math`	Ein Objekt mit zahlreichen mathematischen Konstanten und Methoden
`mimeType`	Ein Objekt mit MimeType-Informationen
`navigator`	Die Objektrepräsentation mit Informationen über den verwendeten Browser des Besuchers
`node`	In neuen Varianten von DOM stellt dieses Objekt den Zugang zu einzelnen Elementen in einem baumartig strukturierten Dokument zur Verfügung. Es ist ein Schlüsselelement für den Einsatz von AJAX. Wir werden es noch umfangreich einsetzen.
`Number`	Ein Objekt mit numerischen Werten
`plugin`	Ein Objekt, das die vorhandenen Plug-ins in einem Browser repräsentiert
`RegExp`	Eine Klasse zum Erzeugen eines Objekts mit regulären Ausdrücken
`screen`	Ein Objekt mit Informationen über den verwendeten Bildschirm des Besuchers
`String`	Eine Klasse zur Erzeugung von Textobjekten samt diverser Manipulationsmöglichkeiten für Zeichen und Zeichenketten
`window`	Dieses Objekt enthält Statusinformationen über das gesamte Browser-Fenster. Jedes Fenster eines Browsers verwendet sein eigenes `window`-Objekt. Das `window`-Objekt ist das höchste Objekt in der Objekthierarchie der Objekte, welche den Browser direkt betreffen.

Tabelle 5.3: Unter JavaScript verfügbare Objekte (Forts.)

5.9.2 Objektfelder

Es gibt im Rahmen des DOM-Konzepts neben den hier aufgeführten Objekten weitere Objekte, die sich in der Notation ein wenig von den anderen Objekten unterscheiden, aber sonst ganz »normale« Objekte sind. Dies sind so genannte **Objektfelder**. Das bedeutet, es sind Datenfelder mit DOM-Objekten eines spezifischen Typs als Inhalt. Als typische Arrays werden enthaltene Objekte über einen Feldnamen sowie eine Indexnummer identifiziert. Ansonsten ist die Anwendung von Eigenschaften und Methoden vollkommen identisch wie bei anderen Objekten.

Die Objektfelder entstehen automatisch, wenn eine Webseite geladen wird und Elemente eines bestimmten Typs darin enthalten sind. Wenn beispielsweise eine Web-

seite ein Formular enthält, bedeutet dies, dass ein Objekt des Typs `form` darin enthalten ist. Wenn nun mehr als ein Formular in einer Webseite vorkommt, muss der Browser diese Formulare irgendwie identifizieren und speichern. Jedes Formular wird in einem Feld eines Objektfelds gespeichert, das automatisch generiert wird und das vom Bezeichner meist dem erzeugenden Objekt sehr ähnlich ist (im Fall von Formularen ist das beispielsweise `forms` – beachten Sie das `s`). Die Indexnummern entstehen automatisch, wenn der Browser das Objekt bei Abarbeitung der HTML-Seite erzeugt und in das Datenfeld einordnet. Das erste im Dokument auftretende Objekt jedes vorkommenden Typs erhält den Index 0, das zweite den Index 1 und so fort. Für den Fall von Formularen wird das erste Objekt vom Typ `form` im Datenfeldeintrag `forms[0]` gespeichert, das zweite in `forms[1]` usw.

Die nachfolgende Tabelle gibt die wichtigsten Objektfelder sowie eine kleine Beschreibung an.

Objektfeld	Beschreibung
`anchors`	Ein Datenfeld, das Referenzen auf alle Anker in einer Webseite enthält
`applets`	Ein Datenfeld, das Referenzen auf alle Java-Applets in einer Webseite enthält
`elements`	Ein Datenfeld mit Referenzen auf alle Eingabeelemente, welche sich in einem übergeordneten Formular befinden. Natürlich kann ein Webformular verschiedene Formularelemente enthalten (einzeilige Eingabefelder, Schaltflächen, Kontrollfelder etc.). Diese unterschiedlichen Formularelemente werden in JavaScript durch die folgenden Objekte repräsentiert: `Button`, `Checkbox`, `FileUpload`, `Hidden`, `Password`, `Radio`, `Reset`, `Select`, `Submit`, `Text` und `Textarea`.
`forms`	Ein Datenfeld, das Referenzen auf alle Formulare in einer Webseite enthält
`frames`	Ein Datenfeld, das Referenzen auf alle Frames in einer Webseite enthält
`images`	Ein Datenfeld, das Referenzen auf alle Bilder in einer Webseite enthält
`links`	Ein Datenfeld, das Referenzen auf alle Hyperlinks in einer Webseite enthält
`mimeTypes`	Ein Datenfeld, das Referenzen auf alle MIME-Typen in einer Webseite enthält
`options`	Ein Datenfeld mit allen Einträgen, die bei dem übergeordneten Formularelement vom Typ `Select` vorkommen
`plugins`	Ein Datenfeld, das Referenzen auf alle in dem Browser installierten Plug-in-Module enthält

Tabelle 5.4: Unter JavaScript verfügbare Objektfelder

5.9.3 DOM-Objekte aus JavaScript nutzen

Allgemein stellen Ihnen DOM-Objekte und -Objektfelder diverse nützliche Methoden und Eigenschaften bereit, die auch mittlerweile einigermaßen einheitlich in den verschiedenen Browsern unterstützt werden. So haben etwa alle Elemente einer Webseite, die in einem DOM-Baum abgebildet werden, ein Attribut `style`. Darüber haben Sie Zugang zu den Layoutmöglichkeiten des Objekts. Wir werden darauf im Kapitel

zu DHTML genauer zu sprechen kommen. Das DOM-Konzept stellt aber auch standardisierte Methoden sowie Eigenschaften bereit, über die Sie auf dem DOM-Baum navigieren und auch den DOM-Baum selbst manipulieren können. Diese sind zum Teil allgemein verfügbar, hängen aber auch vom jeweiligen Elementtyp ab. Die wohl mit weitem Abstand wichtigsten Methoden (insbesondere für AJAX) stellt das Objekt `document` bereit – die Methoden `getElementsByTagName()` und `getElementById()`. Darüber erhalten Sie den direkten Zugang zu einem über den Namen (HTML-Parameter `name`) oder die ID (HTML-Parameter `id`) spezifizierten Bestandteil der Webseite. In der Funktion zum Behandeln der zurückgelieferten Daten aus dem Beispiel in Kapitel 2 haben wir bereits `getElementById()` in Verbindung mit dem Attribut `innerHTML` eingesetzt. Auch darauf kommen wir im Kapitel zu DHTML genauer zurück. Allerdings werden wir in den folgenden Beispielen zu AJAX schon vorher `getElementById()` wieder einsetzen.

5.9.4 Wichtige JavaScript-Techniken

Sie werden für die Erstellung von AJAX-Applikationen teilweise einige JavaScript-Rezepte benötigen, die Sie nachfolgend finden.

Zugriff auf ein Webformular

AJAX werden Sie in vielen Fällen mit Benutzereingaben koppeln. Und diese erfolgen nahezu ausschließlich über Webformulare. Eine Webseite ist über das Objekt `document` aus JavaScript heraus verfügbar. Da sich die Webseite in einem Browser-Fenster befindet und dieses als `window` ansprechbar ist, erfolgt der Zugriff darauf über `window.document`. Ein Webformular in einer Webseite wird durch das Objekt `form` repräsentiert. Wenn eine Webseite mit Formularen geladen wird, wird für jedes Formular automatisch so ein `form`-Objekt erzeugt und in einem Objektfeld mit Namen `forms[]` gespeichert. Die Indexnummern entstehen automatisch, wenn der Browser das Objekt bei Abarbeitung der HTML-Seite erzeugt und in einen Element des Arrays einordnet. Das erste im Dokument auftretende Formular erhält den Index 0, das zweite den Index 1 und so fort. Der Zugriff über das Objektfeld erfolgt schematisch so:

Listing 5.33: Zugriff auf ein Formular in einer Webseite

```
window.document.forms[index]
```

Sollte ein Formular im HTML-Tag mit einem `name`-Attribut spezifiziert sein, können Sie statt eines Datenfeldelements des Objektfelds ebenso diesen Namen für den Zugriff verwenden. Dabei ist beim Namen Groß- und Kleinschreibung relevant. Obwohl der Wert im HTML-Container gesetzt wird und Groß- und Kleinschreibung des Namens eines HTML-Elements bei Zugriffen aus HTML (etwa bei einer `target`-Angabe, wenn Sie mit einem Link auf ein anderes Fenster verweisen) irrelevant ist, spielt es beim Zugriff aus JavaScript eine Rolle.

In mehreren Situationen (nicht immer) können Sie mit dem Schlüsselwort `this` und explizit dem `form`-Objekt auf ein Webformular zugreifen (`this.form`). Damit können Sie sowohl die recht lange Pfadangabe über `window.document` verkürzen als auch unhängig von einem konkreten Formular arbeiten. Über `this` sprechen Sie das gerade aktuelle Objekt an und von diesem aus können Sie in einer passenden Konstellation (das aktuelle Objekt muss die Webseite sein) auf das Formular zugreifen.

Es gibt für jede Form des Zugriffs auf ein Webformular spezifische Situationen, in denen jeweils die eine Form den anderen gegenüber im Vorteil ist. Ein Zugriff über das Objektfeld `forms[]` erweist sich beim Zugriff über Schleifen als besser. Ebenso ist es eine sinnvolle Möglichkeit zum Zugriff, wenn das Formular über keinen `name`-Parameter im HTML-Tag verfügt. Ein Zugriff über den Namen ist meist besser lesbar und toleranter gegenüber Änderungen des Aufbaus der Webseite. Wenn etwa vor einem Formular ein neues Formular in der Webseite eingefügt wird, müssen Zugriffe über einen Index angepasst werden, während Zugriffe über einen Namen keiner Änderung bedürfen. Der Zugriff über eine Referenz auf Basis von `this` ist am universellsten einsetzbar, ist aber nicht in jeder Konstellation möglich. Im Grunde bleibt es Ihnen in vielen Situationen selbst überlassen, welche Version Sie vorziehen.

Achtung

Beachten Sie, dass Sie nicht per JavaScript auf ein Webformular zugreifen können, bevor vom Browser die dazu notwendige HTML-Struktur abgearbeitet und damit die Objekte erzeugt wurden, die das Formular und seine Bestandteile repräsentieren. Mit anderen Worten – wenn eine JavaScript-Anweisung mit einer Referenz auf ein Webformular aufgerufen wird, bevor der Browser die HTML-Anweisungen zum Aufbau des Webformulars fertig interpretiert hat, wird der JavaScript-Interpreter einen Fehler melden. Dieses Problem tritt öfter auf, als man im ersten Moment vermuten mag. Immer wenn Sie dynamisch eine Webseite mit JavaScript neu schreiben (`document.write()`), sind Sie in einer kritischen Situation. Ober wenn Sie einen Skript-Container vor dem `<body>`-Tag notieren und in dem Skript-Container JavaScript-Anweisungen ausführen, die die Existenz eines Formulars bereits voraussetzen. Indem Sie statt der direkten Notation von Aufrufen in so einem Skript-Container den Eventhandler `onLoad` verwenden, können Sie das letzte Problem zumindest umgehen. Der Eventhandler wird erst ausgeführt, wenn die Webseite bezüglich ihrer HTML-Struktur vollständig geladen wurde.

Verschicken und Zurücksetzen von Formulardaten

Jede Objektrepräsentation eines Webformulars in einer Webseite besitzt eine Methode `submit()`. Deren Aufruf schickt die Werte in dem Formular ab. Das Verfahren hat durchaus Ähnlichkeit zum Aufruf der `open()`-Methode bei einem `XMLHttpRequest`-Objekt. Der Aufruf der Methode entspricht der Situation, wenn ein Anwender

auf einen SUBMIT-Button in einem Webformular klickt. Sie können damit das Abschicken der Formulardaten einem Hyperlink, einem einfachen Button oder jeder anderen Situation zuordnen, die sich mit einem Eventhandler beschreiben lässt. Beispiel:

Listing 5.34: Die Anwendung der submit()-Methode – Auslösung mit onClick bei einem gewöhnlichen Button

```
01 <html>
02 <body>
03 <form action="" method="get">
04 <input name="a" />
05 <br>
06 <input type="button" value="OK"
07   onClick="window.document.forms[0].submit()" />
08 </form>
09 </body>
10 </html>
```

Die Zeilen 3 bis 8 definieren ein einfaches Webformular. In der Zeile 7 wird mit dem Eventhandler `onClick` die Methode `submit()` aufgerufen und damit das Formular verschickt (`<input type="button" value="OK" onClick="window.document.forms[0].submit()" />`).

Tipp

Gerade die Methode `submit()` ist ein hervorragender Kandidat, um mit dem Schlüsselwort `this` zu arbeiten. Damit können Sie sowohl die recht lange Pfadangabe über `window.document` verkürzen als auch universeller arbeiten. Über `this` sprechen Sie wie gesagt das gerade aktuelle Objekt an. Sie können die Position und den Namen oder auch jedes andere Attribut des Formulars bzw. eines Formularelements verändern. Wenn Sie im obigen Beispiel die Zeile 7 betrachten, kann man `window.document.forms[0].submit()` auch kürzer als `this.form.submit()` schreiben. Das Schlüsselwort `this` bezieht sich hier auf das Element, das den Eventhandler `onClick` auslöst. Das ist in diesem Fall einfach die Webseite. Ebenfalls sehen Sie hier einmal explizit das `form`-Objekt und nicht das Objektfeld `forms` im Einsatz. Das `form`-Objekt entspricht also für das Beispiel `window.document.forms[0]`.

Jede Objektrepräsentation eines Webformulars in einer Webseite besitzt auch eine Methode `reset()`. Diese leert die Anwendereingaben in einem Webformular und setzt das Formular auf den Anfangszustand zurück[26]. Der Aufruf der Methode entspricht der Situation, wenn ein Anwender auf einen RESET-Button in einem Webformular

26 *Das heißt nicht zwingend, dass das Formular geleert wird. Wenn Formularfeldern mit dem Attribut* `value` *in HTML Vorgabewerte zugeordnet werden, werden diese beim Aufruf von* `reset()` *wieder reproduziert. Das passiert aber auch, wenn ein Anwender eine Reset-Schaltfläche anklickt.*

klickt. Sie können mit dieser JavaScript-Methode das Zurücksetzen der Formulardaten einem Hyperlink, einem einfachen Button oder jeder anderen Situation zuordnen, die sich mit einem Eventhandler beschreiben lässt.

5.10 JavaScript-Aufruf über JavaScript selbst

Der Aufruf von JavaScript-Funktionen und -Anweisungen ist von elementarer Bedeutung. Für die Anwendung von JavaScript im Allgemeinen, aber auch erst recht für AJAX. Immerhin wollen wir ja bei verschiedenen Situationen per AJAX Daten in die Webseite nachladen, und das in der Regel ohne explizite Aufforderung des Anwenders bzw. ohne dass sich der Besucher des Nachladens bewusst wird. Sie kennen bis hier natürlich schon verschiedene Varianten zum Aufruf einer JavaScript-Funktion oder -Anweisung. Aufrufen können Sie eine JavaScript-Funktion oder eine -Anweisung automatisch beim Laden einer Webseite, wenn Sie beispielsweise einen Funktionsaufruf direkt in einen Skript-Container in einer Webseite notieren. Alternativen zu dieser Form des Aufrufs sind die Inline-Referenz und natürlich die Verwendung von Eventhandlern bei einem HTML-Tag. Diese Techniken sind jedoch allesamt in HTML verankert. JavaScript stellt aber auch eine eigene globale Ereignisbehandlung sowie Möglichkeiten zum direkten Aufruf von JavaScript-Funktionen aus JavaScript selbst heraus bereit. Die Reaktionsmöglichkeiten einer Webpräsenz können damit von HTML losgelöst und sogar an zentraler Stelle in einer Webseite oder gar einem ausgelagerten Skript behandelt werden. Dies ist sehr sinnvoll, wenn eine konsequente strukturelle Trennung einer Seite erreicht werden soll. Vor allem ein globales Eventhandling ist eine logische Konsequenz der Bemühungen, Layout, Inhalt und Skriptfunktionalitäten in einer Webseite zu entflechten.

5.10.1 Der explizit Aufruf eines Eventhandlers aus JavaScript

Der wichtigste Schritt, um Layout, Inhalt und Skriptfunktionalitäten in einer Webseite zu entflechten, führt über den expliziten Aufruf eines Eventhandlers aus JavaScript, statt den Eventhandler im HTML-Tag zu notieren. Damit wird der Eventhandler nicht als Attribut in einen HTML-Tag geschrieben und dem HTML-Kontext zugeordnet, sondern Sie ordnen ihn explizit JavaScript zu. Eventhandler sind in dieser Form direkt aus JavaScript heraus als Eigenschaft eines Objekts anzusprechen und damit als Funktionsreferenzen in JavaScript zu notieren. Deshalb dürfen bei der Zuordnung der aufgerufenen Funktion keine (!) Klammern angegeben werden. Ansonsten würde die Funktion unmittelbar bei der Zuordnung ausgeführt und der Rückgabewert der Funktion zugewiesen.

Hinweis

Der Begriff »Eventhandler« wird also in mehrfacher Bedeutung verwendet. »Eventhandler« kann sowohl HTML-Parameter als auch JavaScript-Funktionsreferenzen bezeichnen. Beide Formen machen zwar das Gleiche, aber sie gehören unterschiedlichen Welten an – und müssen damit die dort jeweils gültigen Regeln befolgen (etwa die Berücksichtigung von Groß- und Kleinschreibung).

Diese Technik des Aufrufs von Funktionen und Anweisungen funktioniert bereits seit JavaScript 1.1 und ist damit in nahezu allen neueren Browsern[27] möglich. Und da wir uns hier bei der Angabe des Eventhandlers in JavaScript bewegen, ist Groß- und Kleinschreibung relevant. Der Eventhandler-Name (als Bestandteil des Skripts) muss vollständig in **Kleinbuchstaben** notiert werden!

Tipp

Wenn Sie sich grundsätzlich an die – im Grunde optionale – Regel halten, einen Eventhandler im HTML-Tag mit kleinem `on` zu beginnen und dann einen Großbuchstaben zu notieren, ist das rein von der Schreibweise her eine eindeutige Unterscheidungsmöglichkeit.

Schauen wir uns ein kleines Beispiel an:

Listing 5.35: Der explizite Aufruf eines Eventhandlers per JavaScript

```
01 <html>
02  <script language="JavaScript">
03   function mFunktion() {
04   alert("Herzlich Willkommen");
05  }
06  </script>
07  <body>
08   <form name="mF">
09   <input type="button" name="mB" value="OK" />
10   </formM>
11  <script language="JavaScript">
12  document.mF.mB.onclick = mFunktion
13  </script>
14  <body>
15 </html>
```

27 *Ältere Browser können Probleme machen. Aber diese Browser sollten so langsam der Vergangenheit angehören.*

Die in dem Beispiel verwendete Syntax beinhaltet eine Funktion, die in dem zweiten Skript-Container dem Formular `mF` und dort dem Element `mB` (einer Schaltfläche) zugeordnet wird (in Zeile 12). Wenn der Anwender den Formular-Button anklickt, wird die Funktion aufgerufen und ein kleines Mitteilungsfenster angezeigt.

Tipp

Ein wesentlicher Vorteil von Eventhandlern als Funktionsreferenz zeigt sich beim Einsatz bei Objektfeldern. Sie können so beispielsweise für alle Hyperlinks in einer Webseite mit einer Schleife eine Reaktion festlegen. Zum Beispiel einen Hovereffekt wie folgt:

Listing 5.36: Verwendung einer Funktionsreferenz bei einem Objektfeld

```
for (i = 0; i < document.links.length; i++)
document.links[i].onmouseover = hovereffekt;
```

Das event-Objekt

Die Reaktionsmöglichkeit auf Ereignisse per JavaScript basiert auf der Nutzung eines speziellen Objekts. Wir haben bereits beim Konzept der Ausnahmebehandlung gesehen, dass die Laufzeitumgebung von JavaScript unter gewissen Umständen Objekte generieren kann. So entsteht beim Auftreten eines Ereignisses (etwa ein Klick des Anwenders, ein Tastaturanschlag oder auch das Verlassen der Webseite) im Browser automatisch ein spezielles Objekt, auf das Sie in einem JavaScript direkt reagieren können – das `event`-Objekt. Dies ist ähnlich einem Ausnahmeobjekt als Mitteilungsobjekt zu verstehen. Ein `event`-Objekt beinhaltet Informationen über ein aufgetretenes Ereignis in einer Webseite samt den Randbedingungen, welche dabei eine Rolle gespielt haben. Dieses Objekt wird dann an einen Mechanismus zum Behandeln von `event`-Objekten in Form einer so genannten Message (Botschaft) weitergereicht.

Die Verwendung des `event`-Objekts zur Reaktion auf Ereignisse in einer Webpräsenz hat weitreichende Konsequenzen. Die Ereignisüberwachung wird direkt in JavaScript programmiert und damit wird die Verbindung von HTML (über die dort zugehörenden Eventhandler) und JavaScript in einer Zweckehe aufgelöst. Das ermöglicht sogar eine globale Ereignisbehandlung, wie wir sie unten sehen werden. Aber ebenso ergibt sich die Konsequenz, dass die gesamte Ereignisbehandlung nicht mehr funktionieren wird, wenn JavaScript bei einem Besucher nicht aktiviert ist. Allerdings können wir dann ja auch AJAX vollständig vergessen.

Betrachten wir als Beispiel für das Entstehen eines Ereignisobjekts die Situation, wenn ein Anwender mit der Maus in irgendeinen Bereich der Webseite klickt. Ein Mausklick erzeugt ein `event`-Objekt, das folgende Informationen enthält:

- Die verwendete Maustaste
- Eventuell gedrückte Zusatztasten `Strg`, `Alt`, `AltGr`, `Shift`
- Die Koordinaten des Klicks

Andere `event`-Objekte, die bei weiteren Ereignissen erzeugt werden, beinhalten natürlich andere Informationen, die dem Ereignis angepasst sind. So steht beispielsweise bei einem Tastendruck die gedrückte Taste als abzufragende Information bereit. Allgemein beinhaltet ein `event`-Objekt jedoch zahlreiche sinnvolle Informationen, die Sie zur Erstellung gut angepasster Applikationen nutzen können.

Die Reaktion auf ein event-Objekt

Sie müssen sich vergegenwärtigen, dass `event`-Objekte während der Anzeige einer Webseite im Browser permanent im Hintergrund erzeugt werden und auch überall im Code zur Verfügung stehen – unabhängig davon, ob Sie diese in einem Skript zur Kenntnis nehmen oder nicht. Sie sind einfach da. Wie eine Radiosendung, die unabhängig davon ausgestrahlt wird, ob nun darauf eingestellte Radioempfänger vorhanden und angeschaltet sind. Nun ist es sicher nicht Sinn und Zweck eines komplexen Modells, dass permanent Ereignisse abgefeuert und diese nie beachtet werden (sprich darauf reagiert wird). In bestimmten Situationen will man ja explizit eine Reaktion haben, d.h., es soll ein Skript aufgerufen werden. Auf welche Art und Weise auf welches erzeugte Ereignisobjekt reagiert werden soll, wird durch ein abstraktes Konzept geregelt – eben ein Ereignismodell, von dem es im Webumfeld verschiedene Versionen gibt. Darin sind aber allgemein sowohl die Verhaltensweisen auf Ereignisse festgelegt, die konkrete Syntax zur Umsetzung, aber auch erst recht diejenigen Ereignisse, welche überhaupt zur Kenntnis genommen werden können.

Wenn ein `event`-Objekt erzeugt wird, ist dieses nicht nur bei dem Element wahrzunehmen, über das es ausgelöst wird. Es bewegt sich durch die Objekthierarchie des Dokuments nach oben (die so genannte Bubble-Phase). An jeder Stelle dieses Wegs lässt sich das Ereignis abfangen. Testen Sie einmal das nachfolgende Beispiel in einem kompatiblen Browser:

Listing 5.37: Weiterreichen des event-Objekts

```
01 <html>
02 <body onclick="alert('body')">
03 <div onclick="alert('div')">
04 <h1 onclick="alert('h1')">
05 <span onclick="alert('span')">Klick</span></h1></div>
06 </body>
07 </html>
```

Wenn Sie auf die Überschrift klicken, erhalten Sie zuerst ein Mitteilungsfenster mit dem Inhalt *span*, dann *h1*, *div* und zum Schluss *body*.

Klicken Sie jedoch in einen freien Bereich der Webseite, erhalten Sie nur ein Mitteilungsfenster mit dem Inhalt *body*.

Laut W3C-DOM ist der Weg durch die Hierarchie sowohl von unten nach oben als auch wieder zurück zu beobachten. Allerdings nicht mit diesem Verfahren hier, sondern mit der nachfolgenden Technik eines so genannten Listener.

Abbildung 5.4: Der Eventhandler des innersten Containers wurde ausgelöst.

Abbildung 5.5: Stufe 3 hat gezündet.

Registrierung eines Listener

Es gibt in JavaScript eine Variante auf Ereignisse zu reagieren, die stark von Ereignismodellen beeinflusst ist, wie sie beispielsweise in Java eingesetzt werden. Man registriert **Listener** bei einem bestimmten Objekt. Das Konzept funktioniert von der Idee her so:

1. Ein Ereignis tritt bei einem Quellobjekt (der so genannte Event Source) auf.
2. Das entstandene Ereignisobjekt wird an einen registrierten Zuhörer (eben besagten Event-Listener oder kurz Listener) weitergeleitet.
3. Erst von dem Event-Listener wird über die konkrete Ereignisbehandlung entschieden und die Reaktion auch ausgelöst.

Im DOM-Eventmodell der Version 2 wird zur Registrierung eines Listener eine Methode `addEventListener()` definiert. Die schematische Syntax sieht so aus:

Listing 5.38: Schematische Registrierung eines Listener bei einem Element

```
Element.addEventListener("eventTyp", anweisung, wann)
```

Der `eventTyp` ist einer der Token, die wir auch bisher schon als Eventhandler gesehen haben – allerdings **ohne** vorangestelltes `on`, also etwa `click` oder `mouseover`. Mit `anweisung` ist eine Funktionsreferenz bezeichnet und `wann` ist ein Boolean-Wert, der festlegt, ob das `event`-Objekt auf dem Hin- (`true`) oder Rückweg (`false`) durch die Ereignishierarchie abgefangen werden soll. Beispiele:

Listing 5.39: Registrierung der Listener

```
document.addEventListener("click", kBody, false);
document.addEventListener("click", kSpan, true);
```

Die Technik der Listener ist zwar recht elegant, aber da ältere Browser sowieso nicht damit zurechtkommen, es auf einigen Betriebssystemplattformen Probleme gibt und der Internet Explorer sich grundsätzlich[28] der Methode `addEventListener()` verweigert, ist der Einsatzbereich in der Praxis derzeit noch sehr begrenzt. Zwar gibt es im Internet Explorer ein proprietäres Alternativkonzept mit einer Methode namens `attachEvent()`, aber zu deren Einsatz kann man auch kaum raten.

5.10.2 Globale Ereignisbehandlung mit dem event-Objekt

Der Aufruf von Eventhandlern per JavaScript trennt bereits den Aufruf einer JavaScript-Funktion oder -Anweisung von der HTML-Welt ab. Aber man kann noch weiter gehen. Sie können den Reaktionsmechanismus global implementieren, also losgelöst von einem einzelnen HTML-Tag oder einer per JavaScript bei einem Element aufgerufenen Aktion.

Wenn Sie sich nun fragen, wie Sie konkret über einen globalen Mechanismus auf ein durch den Browser schwirrendes `event`-Objekt reagieren können, dann kann man die Frage entweder ganz einfach beantworten (lassen Sie es besser) oder aber man muss es richtig kompliziert aufziehen. Der Anlass ist banal – im Grunde muss man für verschiedene Browser ein individuelles globales Eventhandling aufziehen und für jeden Browser, den man unterstützen will, testen, welches der Modelle ihm denn nun gefällt. Es gibt nämlich neben dem Fall, dass ein Browser gar keine globale Ereignisbehandlung unterstützt, zwei leider vollkommen inkompatible Ereignismodelle, die derzeit in der Praxis Verwendung finden. Diese Ereignismodelle gehen entweder auf Microsoft oder Netscape zurück. Manche Browser unterstützen sogar beide Varianten (zumindest in Teilen), aber nicht alle, und ein ausführlicher Test ist zwingend.

Die allgemeine Syntax zum Zugriff auf ein `event`-Objekt ist zumindest bei beiden konkurrierenden Ereignismodellen gleich. Vorangestellt wird das Objekt, das das `event`-Objekt generiert. Diesem folgt der Token `event` über die übliche Punktnotation:

28 Zumindest bis zur Version 6.0.

Listing 5.40: Grundsätzliches Ansprechen des event-Objekts

```
[Objekt].event
```

Beispiel:

Listing 5.41: Grundsätzliches Ansprechen des event-Objekts einer Webseite

```
document.event
```

Sie wollen bei einer Auswertung in der Regel dann natürlich eine ganz bestimmte Information eines Ereignisses verwenden. Dies sind bei einer objektorientierten Betrachtungsweise einfach die Eigenschaften des Objekts. So können Sie etwa einen Mausklick wie folgt ansprechen:

Listing 5.42: Ansprechen eines Mausklicks

```
document.event.CLICK
```

Diese Syntax steht vom Prinzip her sowohl im Ereignismodell von Microsoft als auch Netscape zur Verfügung. Dummerweise setzen sie aber die konkrete Implementierung in der Folge völlig unterschiedlich um. Auch in der Syntax zur Überwachung von Ereignissen (wie schon teilweise bei der Unterstützung der Eventhandler) unterscheiden sich beide Philosophien – und das, obwohl die grundsätzliche Logik gleich ist.

Globale Reaktion über das Netscape-Ereignismodell

Das Netscape-Ereignismodell wird grundsätzlich von Opera, Konqueror und allen auf Mozilla basierenden Browsern (Navigator, Firefox, Safari, Mozilla) unterstützt. Beim Netscape-Modell müssen wir aber zu allem Überfluss ein altes und ein neues Modell unterscheiden. In den ersten Versionen des Ereignismodells konnten Sie über eine zum `window`-Objekt gehörende Methode bewirken, dass alle dort als Parameter spezifizierten Ereignisse eines spezifizierten Typs innerhalb eines bestimmten Objekts (etwa einer ganzen Webseite) zentral behandelt werden. Dies ist die Methode `captureEvents()`, die wie folgt angewandt wurde:

Listing 5.43: Die grundsätzliche Verwendung von captureEvents()

```
[Objekt].captureEvents([Ereignistyp])
```

Wenn so eine Zeile im JavaScript-Code notiert wird, werden alle spezifizierten Ereignisse, die bei dem vorangestellten Objekt auftreten, aufgefangen und an eine zentrale Stelle weitergeleitet – der zentralen Ereignisbehandlungsroutine. Dies ist in diesem Fall einfach eine Funktion, welche als Argument das erzeugte `event`-Objekt mit der gewünschten Eigenschaft verwendet. Als Ereignistyp akzeptiert die Methode jedes bei einem Objekt von Netscape unterstützte Ereignis. Um das Auffangen eines Events zu ermöglichen, wird bei dem entsprechenden Objekt (etwa `window`) eine Anweisung der Art notiert:

Listing 5.44: Das Auffangen aller Klickereignisse im Fenster wird angezeigt

```
window.captureEvents(Event.CLICK);
```

Das Argument der Methode `captureEvents()` ist eine Eigenschaft des `event`-Objekts, die den Typ des Ereignisses spezifiziert. Der Aufbau ist einfach das vollständig groß geschriebene Ereignis, das per Punktnotation `Event` nachgestellt wird. Das abschließende Registrieren des Eventhandlers läuft so ab, dass das Ereignis, für das eine bestimmte Funktion als Eventhandler agieren soll, dem zugehörigen Objekt nachgestellt und auf der rechten Seite der Bezeichner der Eventhandler-Funktion zugewiesen wird. Dabei sollte beachtet werden, dass dort keine Klammern notiert werden dürfen, denn es handelt sich um Funktionsreferenzen!

So weit ist das globale Ereignisbehandlungskonzept sowohl einfach als auch effektiv. Die Verwendung von `captureEvents()` ist dennoch auf der Schutthalde der Geschichte gelandet, denn nicht nur der Internet Explorer und Microsoft-treue andere Browser verweigern grundsätzlich die Unterstützung. Auch neuere Varianten des Navigator respektive Mozilla und andere Gefolgsleute des Netscape-Ereignismodells zeigen keinerlei Lust, auf die alte Ereignisbehandlungsstruktur zu reagieren. Beim Browser-Wechsel auf den Navigator 6.0 hat Netscape eine ganze Reihe von hervorragenden Konzepten (so auch das Layer-Konzept) aufgegeben und ist mit fliegenden Fahnen zum Feind übergelaufen, um von diesem endgültig fertig gemacht zu werden ;-)[29]. Zwar wird in den Mozilla-treuen Browsern bei Verwendung von `captureEvents()` im Gegensatz zum Internet Explorer kein Fehler angezeigt, aber Ignoranz hilft dem Besucher auch nicht.

Dennoch ist das Netscape-Ereignismodell nicht tot. Sie verwenden nur alternativ einen Eventhandler als Funktionsreferenz, wie oben bereits besprochen. Die Spezialisierung ist nur, dass Sie in der referenzierten Funktion gezielt und global das `event`-Objekt überwachen. Das Ereignismodell von Netscape ist in Zusammenhang mit einem allgemeineren Konzept zu sehen, mit dem Netscape dynamische Webseiten realisiert – dem schon erwähnten Layer-Konzept (Schichten), das Netscape mit der Version 4.0 seines Browsers einführte und als Sprachelement von HTML verstanden hat. Mithilfe des `<layer>`-Containers war es ursprünglich möglich, beliebige Bereiche einer HTML-Datei exakt zu kontrollieren und Elemente zu positionieren. Neben der auf HTML beschränkten Funktion von Layern verwendet Netscape diese als Zielobjekt für Manipulationen unter JavaScript. Zwar hat Netscape das Layer-Konzept ab der Version 6 vom Navigator fast vollständig aufgegeben, aber der Kern des Eventmodells mit Funktionsreferenzen zu einer globalen Ereignisbehandlung ist »still alive and well«. Allgemein können Sie im derzeit unterstützten Netscape-Ereignismodell folgende Eigenschaften auswerten:

29 *Sie kennen die Geschichte der Browser-Kriege sicherlich und dass der – mit Verlaub grottenschlechte – Netscape Navigator 6.0 das Ende von Netscape nahezu besiegelt hat. Zumindest fast, denn Open-Source-Erben vom Netscape Navigator wie Mozilla und vor allem Firefox sind später wie Phönix aus der Asche gestiegen und haben die Erbsünde dieses inkompatiblen Versionswechsels fast vergessen gemacht.*

Eigenschaft	Beschreibung
`layerX` und `layerY`	Die relativen Koordinaten. Die Eigenschaft wird in keinem Browser außer alten Navigator-Varianten unterstützt. Führt in anderen Browsern entweder zu Fehlermeldungen oder wird ignoriert.
`modifiers`	Abfrage auf Sondertasten. Die Eigenschaft wird in keinem Browser außer alten Navigator-Varianten unterstützt. Führt in anderen Browsern entweder zu Fehlermeldungen oder wird ignoriert.
`pageX` und `pageY`	Koordinatenangaben relativ zum Fenster. Die Eigenschaften werden in nahezu allen Browsern unterstützt, die im Rahmen des neuen Ereignismodells funktionieren.
`screenX` und `screenY`	Die Bildschirmkoordinaten. Die Eigenschaften werden in nahezu allen Browsern unterstützt, die im Rahmen des neuen Ereignismodells funktionieren.
`which`	Der Tastaturcode bzw. der Maustastencode. Die Eigenschaften werden in nahezu allen Browsern unterstützt, die im Rahmen des neuen Ereignismodells funktionieren.
`type`	Die Art des Ereignisses. Die Eigenschaften werden in nahezu allen Browsern unterstützt, die im Rahmen des neuen Ereignismodells funktionieren.

Tabelle 5.5: Eigenschaften, die im Netscape-Ereignismodell auszuwerten sind

Schauen wir uns einige Beispiele an und wie verschiedene Browser darauf reagieren. Im ersten Beispiel soll erst die einfache folgende Situation realisiert werden: Der Besucher drückt eine beliebige Taste, der Tastaturcode wird mittels eines `XMLHttpRequest`-Objekts an den Server geschickt, dort wird eine Antwort generiert und diese mittels `getElementById()`, `innerHTML` und AJAX in der Webseite angezeigt. Das ist die HTML-Seite (*eventnc1.html*), in der die Ereignisbehandlung per Funktionsreferenz an das `window`-Objekt gebunden wird:

Listing 5.45: Globale Reaktion nach dem Ereignismodell von Netscape

```
01 <html>
02 <script language="JavaScript" src="eventnc1.js"></script>
03 <script language="JavaScript">
04   window.onkeypress = taste;
05 </script>
06 <body>
07 <h1>Drücken Sie eine Taste!</h1>
08 <span id="antwort"></span>
09 </body>
10 </html>
```

Im Beispiel wird zuerst in Zeile 2 eine externe JavaScript-Datei *eventnc1.js* eingebunden, die wir uns gleich ansehen. Von Zeile 3 bis 5 erstreckt sich ein innerer Script-

Container. Dort wird nur das Ereignis `onkeypress` beim `window`-Objekt überwacht (Drücken irgendeiner Taste). Wenn der Anwender eine Taste drückt, wird die Funktion `taste()` aufgerufen. Der Aufruf erfolgt als Funktionsreferenz, weshalb keine Klammern notiert werden (Zeile 4 – `window.onkeypress = taste;`). In Zeile 8 finden Sie wieder einen `<span>`-Container, dessen Inhalt wir austauschen wollen.

Schauen wir uns die externe JavaScript-Datei an:

Listing 5.46: Die externe JavaScript-Datei

```
01 var resObjekt = null;
02 function taste(ev) {
03   sndReq(ev.which);
04 }
05 function erzXMLHttpRequestObject(){
06   var resObjekt = null;
07   try {
08     resObjekt = new ActiveXObject("Microsoft.XMLHTTP");
09   }
10   catch(Error){
11     try {
12       resObjekt = new ActiveXObject("MSXML2.XMLHTTP");
13     }
14     catch(Error){
15       try {
16       resObjekt = new XMLHttpRequest();
17       }
18       catch(Error){
19         alert(
20           "Erzeugung des XMLHttpRequest-Objekts ist nicht möglich");
21       }
22     }
23   }
24   return resObjekt;
25 }
26 function sndReq(drueck) {
27     resObjekt.open('get', 'eventnc1.jsp?welcheTaste='+drueck,true);
28     resObjekt.onreadystatechange = handleResponse;
29     resObjekt.send(null);
30 }
31 function handleResponse() {
32   if(resObjekt.readyState == 4){
33     document.getElementById("antwort").innerHTML =
34       resObjekt.responseText;
35   }
36 }
37 resObjekt=erzXMLHttpRequestObject();
```

Die externe JavaScript-Datei ist in weiten Zügen mit der bisher schon verwendeten Datei identisch. Nur werden zur Generierung des `XMLHttpRequest`-Objekts die universelle Funktion mit der Ausnahmebehandlung eingesetzt sowie die übergebenen Parameter und die ID angepasst. Neu ist die Definition der Funktion `taste()`, in der das Ereignismodell von Netscape zum Tragen kommt (Zeile 2 bis 4). Die Funktion wird ja als Funktionsreferenz an einen Tastendruck gebunden. Der Parameter an die Funktion ist das `event`-Objekt. Dessen Eigenschaft `which` enthält den Tastaturcode. In der Funktion wird die Funktion `sndReq()` mit dem Tastaturcode als Parameter aufgerufen (`sndReq(ev.which);`). Dieser wird in der Funktion in Zeile 27 an das Skript *eventnc1.jsp* auf dem Server weitergereicht (`resObjekt.open('get', 'eventnc1.jsp?welcheTaste=' + drueck, true);`). Dabei wird der Pseudo-URL entsprechend der Regeln zusammengesetzt, die wir in Kapitel 3 besprochen haben. Schauen wir uns das einfache JSP der Vollständigkeit halber noch an. Sie werden es lesen können, auch wenn Sie noch kein JSP kennen:

Listing 5.47: Das JSP eventnc1.jsp zum Generieren einer Antwort

```
01 <%@ page language="java" %>
02   Sie haben die Taste
03 <%=  request.getParameter("welcheTaste") %>
04 gedrückt.
```

Im Skript wird der übergebene Wert zusammen mit einem vorgegeben Text genommen und als Antwort wieder an den Client geschickt. Dort wird in dem `<span>`-Element ausgegeben, welche Taste vom Anwender gedrückt wurde (der numerische Code, der über die Eigenschaft `which` an den Server weitergereicht wurde).

Abbildung 5.6: Die Webseite, nachdem der Anwender eine Taste gedrückt hat – hier im Firefox

Abbildung 5.7: Auch Opera kommt mit dem Netscape-Ereignismodell zurecht.

Der Internet Explorer verweigert die Reaktion auf diese Form der Ereignisbehandlung und leider werden wie gesagt einige der anderen Eigenschaften des `event`-Objekts nicht in allen Browsern unterstützt. Schauen wir uns noch ein Beispiel mit der Auswertung anderer Eigenschaften des `event`-Objekts an, die in nahezu allen Browsern unterstützt werden, die im Rahmen des neuen Ereignismodells funktionieren. Zuerst die HTML-Datei (*eventnc2.html*), in der dieses Mal die Ereignisbehandlung per Funktionsreferenz an das `document`-Objekt gebunden wird und auf das Loslassen der Maustaste reagieren soll:

Listing 5.48: Eine weitere Variante mit globaler Reaktion nach dem Ereignismodell von Netscape

```
01 <html>
02 <script language="JavaScript" src="eventnc2.js"></script>
03 <script language="JavaScript">
04   document.onmouseup = pos;
05 </script>
06 <body>
07 <h1>Klicken Sie auf die Webseite!</h1>
08 <span id="antwort"></span>
09 </body>
10 </html>
```

Die HTML-Datei unterscheidet sich nicht wesentlich von dem vorherigen Beispiel. Nur wird in Zeile 2 eine andere externe JavaScript-Datei *eventnc2.js* eingebunden und in Zeile 4 wird der Eventhandler `onmouseup` überwacht und die Funktionsreferenz auf die Funktion `pos()` referenziert. Schauen wir uns die externe JavaScript-Datei an, soweit sie sich geändert hat:

Listing 5.49: Die veränderten Parts der externen JavaScript-Datei

```
02 function pos(ev) {
03   sndReq(ev.pageX, ev.pageY, ev.screenX, ev.screenY , ev.type);
04 }
...
25 function sndReq(pageX, pageY, screenX, screenY, type) {
26   resObjekt.open('get',
27     'eventnc2.jsp?pageX=' + pageX + '&pageY=' + pageY
28     + '&screenX=' + screenX + '&screenY=' + screenY
29     + '&type=' + type, true);
30     resObjekt.onreadystatechange = handleResponse;
31     resObjekt.send(null);
32 }
...
```

Die externe JavaScript-Datei unterscheidet sich nur im Aufruf der Funktion `sndReq()` und deren Implementierung. Die Funktion bekommt in dieser Variante mehrere Parameter beim Aufruf übergeben (Zeile 03 – `sndReq(ev.pageX, ev.pageY, ev.screenX, ev.screenY , ev.type);`). Diese werden in der Funktion in Zeile 26 bis 29 dazu verwendet, den Pseudo-URL an das Serverskript *eventnc2.jsp* entsprechend der Regeln zusammenzusetzen, die wir in Kapitel 3 besprochen haben. Im Vergleich zur ersten Variante ist zwar etwas mehr Arbeit zu leisten, aber schwierig ist die Geschichte keinesfalls. Die einzelnen Wertepaare werden einfach mit `&` getrennt aneinander gefügt. Wir schauen uns natürlich auch hier das aufgerufene JSP an. Die übergebenen Werte werden wieder mit Texten und einigen HTML-Tags aufbereitet und dem Client zurückgegeben:

Listing 5.50: Das aufgerufene JSP eventnc2.jsp

```
01 <%@ page language="java" %>
02 X-Koordinate:
03 <%=  request.getParameter("pageX") %>
04 <br>Y-Koordinate:
05 <%=  request.getParameter("pageY") %>
06 <br>Bildschirm: X-Koordinate:
07 <%=  request.getParameter("screenX") %>
08 <br>Bildschirm: Y-Koordinate:
09 <%=  request.getParameter("screenY") %>
10 <br>Typ des Ereignisses:
11 <%=  request.getParameter("type") %>
```

Hinweis

Es sollte Ihnen auffallen, dass die Namen der Parameter über die gesamte Logik gleich bzw. ähnlich (bis auf das vorangestellte Objekt) bleiben. Das ist in der Praxis absolut üblich, um immer die eindeutige Zuordnung über Namensräume oder Unterprogramme hinweg zu haben.

Abbildung 5.8: Verschiedene Eigenschaften des Ereignisobjekts – hier das Resultat im Firefox

Das globale Microsoft-Ereignismodell beim event-Objekt

Das Microsoft-Objektmodell erlaubt wie das Netscape-Modell den unmittelbaren Zugriff auf das `event`-Objekt samt dessen Ereignisse, soweit diese im Microsoft-Ereignismodell verstanden werden. Das bedeutet, das `event`-Objekt stellt ähnliche Fähigkeiten bereit wie auch unter dem Netscape-Modell. Nur werden diese anders benannt. Mozilla-Browser kommen explizit nicht damit zurecht. Jedoch verstehen sowohl der Opera als auch der Konqueror auch diese Benennungen der Eigenschaften des `event`-Objekts, zumindest teilweise. Allerdings ist die Microsoft-Variante der globalen Ereignisbehandlung nicht nur mit andersartig benannten Eigenschaften des `event`-Objekts verbunden. Es gibt auch eine spezielle Syntax zur globalen Registrierung von Ereignissen. Und bei deren Verwendung steigen dann auch Opera und Konqueror aus. Die globale Ereignisbehandlung im Microsoft-Ereignismodell basiert auf der folgenden Erweiterung des Script-Elements:

Listing 5.51: Eine erweitere Variante des <script>-Containers zur globalen Ereignisbehandlung im Microsoft-Modell

```
<script FOR=[Objekt] EVENT=[Ereignis] language="JavaScript">
   ...[Konkrete Anweisungen]...
</script>
```

Als `[Objekt]` geben Sie zum Beispiel `document` an, wenn Sie die gesamte Webseite überwachen wollen. Oder `window` für das gesamte Browser-Fenster. Als `[Ereignis]` geben Sie einfach die Eventhandler an. Mehr ist nicht notwendig, um alle Ereignisse des spezifizierten Typs von dem angegebenen Objekt zentral zu verarbeiten. Innerhalb des Skripts können Sie dann auf alle Eigenschaften des `event`-Objekts zugreifen, die Microsoft respektive der Internet Explorer anerkennen. Das gesamte Prozedere ist unkompliziert, solange kein Browser die Seite lädt, der damit nicht klar kommt.

Im Microsoft-Konzept sind folgende Eigenschaften des `event`-Objekts vorgesehen:

Eigenschaft	Beschreibung

`altKey`
`ctrlKey`
`shiftKey`
	Die entsprechenden Sondertasten. Sie stehen für die Zusatztasten ALT, CRTL oder SHIFT. Wenn sie gemeinsam mit einer anderen Taste oder einem Mausklick gedrückt wurden, werden die jeweiligen Eigenschaften den Wert `true` enthalten.

`clientX` `clientY`	Die Bildschirmkoordinaten. Die Eigenschaften beinhalten die Information über die horizontalen Pixel (`clientX`) und die vertikalen Pixel (`clientY`) der Cursorposition relativ zur oberen linken Ecke des Anzeigefensters, wenn ein koordinatenabhängiges Ereignis (etwa eine Mausaktion) ausgelöst wurde.
`keyCode`	Der Tastaturcode. Die Eigenschaft speichert bei Tastaturereignissen den dezimalen Code (ASCII/ANSI-Wert) der gedrückten Taste.
`offsetX` `offsetY`	Die Koordinaten relativ zum Objekt. Über die Eigenschaften kann auf die horizontalen Pixel (`offsetX`) und die vertikalen Pixel (`offsetY`) der Cursorposition relativ zur oberen linken Ecke des Elements, das ein Ereignis ausgelöst hat, zugegriffen werden.
`x` `y`	Die Koordinaten relativ zum Elternelement. Die Eigenschaften speichern die horizontalen Pixel (`x`) und die vertikalen Pixel (`y`) der Cursorposition relativ zur oberen linken Ecke des Eltern-Elements von dem Element, das ein Ereignis ausgelöst hat. Wenn ein absolut positionierter Bereich das Eltern-Element ist, ist dessen obere linke Ecke der Bezugspunkt. Wenn das auslösende Element sonst kein Eltern-Element hat, gilt die linke obere Ecke des Dokuments als Koordinatenursprung.

Tabelle 5.6: Eigenschaften, die im Microsoft-Ereignismodell auszuwerten sind

Schauen wir uns ein Beispiel an, in dem die Position des Mauszeigers beim Überstreichen der Webseite ausgegeben und zudem überwacht wird, ob ein Klick auf die Webseite erfolgt. Falls ja, zeigt das Beispiel alle dabei gedrückten Sondertasten an. Nachfolgend finden Sie den HTML-Code:

Listing 5.52: Einsatz des event-Objekts im Microsoft-Modell

```
01 <html>
02 <script language="JavaScript" src="eventie1.js"></script>
03 <script FOR=document EVENT=onmousemove language="JavaScript">
04   status = event.clientX;
05   sndReq2(event.clientX, event.clientY, event.x, event.y);
06 </script>
07 <script FOR=document EVENT=onclick language="JavaScript">
08   sndReq(event.shiftKey, event.altKey, event.ctrlKey);
09 </script>
10 <body>
11 <h1>Bewegen Sie die Maus oder klicken Sie auf die Webseite!</h1>
12 <span id="antwort"></span>
13  </body>
14 </html>
```

Die Webseite zeigt dem Besucher zuerst nur eine Überschrift der Ordnung 1 mit der Aufforderung zur Bewegung der Maus oder einem Klick an.

Abbildung 5.9: Die Seite vor der ersten Anwenderaktion

Sie finden in dem Listing zuerst eine Referenz auf eine externe Skriptdatei. In Zeile 3 beginnt der spezielle `<script>`-Container, über den die Überwachung aller `onmousemove`-Ereignisse in der Webseite registriert wird. Im Inneren geben wir bei jeder Bewegung des Mauszeigers mit Zeile 4 den Wert einer Eigenschaft des `event`-Objekts in der Statuszeile aus (`clientX` – die X-Position des Mauszeigers). Das ist eine rein auf JavaScript und den Client beschränkte Aktion. Der Aufruf von `sndReq2()` in Zeile 5 (`sndReq2(event.clientX, event.clientY, event.x, event.y);`) bringt aber wieder AJAX ins Spiel. Die Werte der angegebenen Eigenschaften[30] werden – wie Sie sicher unschwer vermuten – an den Server weitergereicht und dort wird eine Antwort generiert.

30 *X- und Y-Position des Mauszeigers sowie die relativen Koordinaten zum Elternelement – das ist in diesem Beispiel identisch.*

Abbildung 5.10: Per AJAX wird die aktuelle Position des Mauszeigers in die Webseite geschrieben.

Der zweite spezielle `<script>`-Container von Zeile 7 bis 9 sorgt für die Überwachung aller Klick-Ereignisse in der Webseite. Im Inneren geben wir bei jedem Klick in der Webseite aus, welche Zusatztasten gedrückt waren. Der Aufruf von `sndReq()` in Zeile 8 (`sndReq(event.shiftKey, event.altKey, event.ctrlKey);`) reicht wieder per AJAX die Werte der angegebenen Eigenschaften an den Server weiter, der eine Antwort generiert.

Die beiden JavaScript-Funktionen sind vollkommen analog den bisherigen Varianten aufgebaut und sehen so aus:

Listing 5.53: Die angepassten Funktionen zum Anfordern der Daten vom Server

```
22 function sndReq(shiftKey, altKey, ctrlKey) {
23     resObjekt.open('get', 'eventie1.jsp?shiftKey=' + shiftKey +
24       '&altKey=' + altKey + '&ctrlKey=' + ctrlKey,
25       true);
26     resObjekt.onreadystatechange = handleResponse;
27     resObjekt.send(null);
28 }
29 function sndReq2(clientX, clientY, x, y) {
30     resObjekt.open('get', 'eventie1.jsp?clientX=' + clientX +
31       '&clientY=' + clientY + '&x=' + x + '&y=' + y,
32       true);
```

```
33     resObjekt.onreadystatechange = handleResponse;
34     resObjekt.send(null);
35 }
```

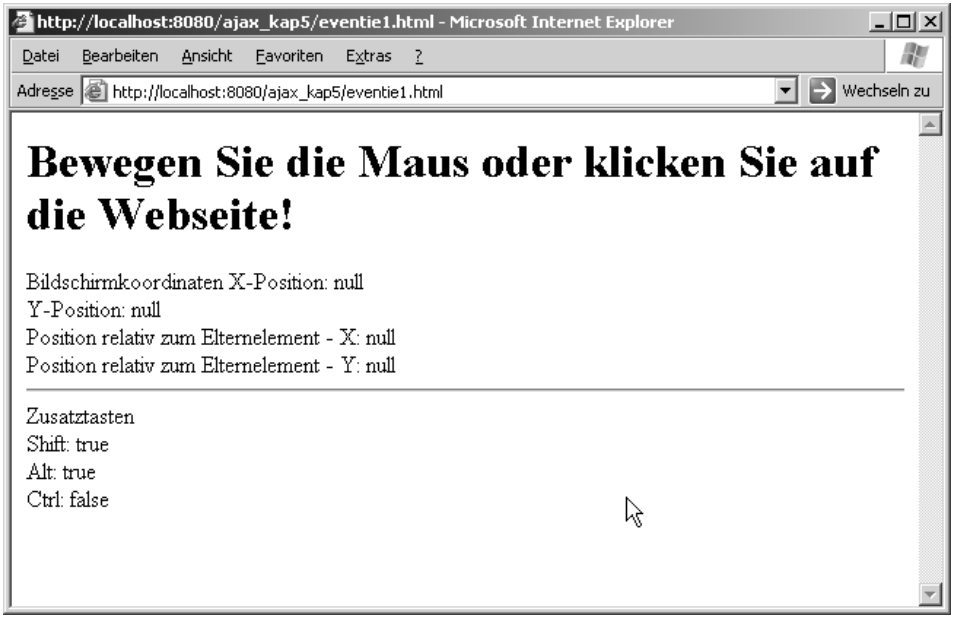


Abbildung 5.11: Beim Klick waren die Alt- und die Shift-Taste gedrückt

Und zum Schluss wieder das JSP, das die Antwort generiert. Hier wird sich nicht viel tun, solange wir JSP und serverseitige Programmierung nicht besprochen haben:

Listing 5.54: Das JSP zur Generierung der Antwort

```
01 <%@ page language="java" %>
02 Bildschirmkoordinaten X-Position:
03 <%=  request.getParameter("clientX") %>
04 <br>Y-Position:
05 <%=  request.getParameter("clientY") %>
06 <br>Position relativ zum Elternelement - X:
07 <%=  request.getParameter("x") %>
08 <br>Position relativ zum Elternelement - Y:
09 <%=  request.getParameter("y") %>
10 <hr>Zusatztasten<br>Shift:
11 <%=  request.getParameter("shiftKey") %>
12 <br>Alt:
13 <%=  request.getParameter("altKey") %>
14 <br>Ctrl:
15 <%=  request.getParameter("ctrlKey") %>
```

5.10.3 Fazit der Ereignisbehandlung

Globale Ereignisbehandlung ist im Grunde sehr viel versprechend. Und wenn ich als Java-Programmierer argumentiere, muss ich globale Ereignisbehandlung empfehlen. Sie bedeutet die vollständige funktionale Abtrennung von HTML. Und die damit einhergehende Modularisierung ist ein sehr wichtiger Schritt hin zu stabilen, besser wartbaren und professionellen Skripten. Außerdem trägt die Verwaltung von Skripten in einem eigenen Dokument der Forderung nach dem barrierefreien Web respektive JavaScript Rechnung.

Wenn ich mich allerdings auf meine Erfahrung als Webprogrammierer zurückziehe, rate ich bei den meisten Webpräsenzen zum Einsatz von HTML-Eventhandlern. Die unzähligen Probleme mit den verschiedenen Ereignismodellen, alten Browsern, unzuverlässigen Browser-Weichen, speziellen Einstellungen beim Anwender etc. führen dazu, dass ich in den meisten Fällen zu einem extrem konservativen Einsatz von Webtechniken rate. Grundsätzlich handeln Sie sich mit globaler Ereignisbehandlung unter Umständen viele Probleme ein. Sie sollten sorgfältig abwägen, ob eine globale Ereignisbehandlung wirklich den Aufwand wert ist. Wenn man globale Ereignisbehandlung für verschiedene Browser-Welten sicher machen möchte, kommt man um eine Trennung der Browser sicher nicht herum. Eventhandler machen schlicht und einfach weniger Probleme im Alltag – im einfachsten Fall, indem Sie diese direkt in HTML-Tags notieren.

Wenn ich nun aber an AJAX-Applikationen denke und Sie zumindest die Reaktion in JavaScript verlagern wollen, rate ich dazu, explizit einen Eventhandler als Funktionsreferenz aufzurufen. Eine Webapplikation unter Einsatz von AJAX funktioniert sowieso nur dann vernünftig, wenn JavaScript in einer neuen Version unterstützt wird und sich die Browser auf einem modernen Stand befinden. Dann gibt es auch keine Probleme mit JavaScript-Eventhandlern.

Wie Sie sich auch entscheiden – Sie sollten sorgfältig darüber nachdenken, im Fall von globaler Ereignisbehandlung doppelt und dreifach testen und vor allem Mischkonstellationen vermeiden. Es gilt dringend ein Entweder-Oder. Das `event`-Objekt kann selbstverständlich auch in Verbindung mit HTML-Eventhandlern eingesetzt werden. Nur führt das dann wieder den Versuch ad absurdum, die Struktur und die Funktionalität streng zu trennen.

5.11 Zusammenfassung

Sie haben in diesem recht umfangreichen Kapitel nun alle JavaScript-Details gesehen, die für eine erfolgreiche Programmierung von AJAX-Webseiten notwendig sind – von der einfachen Einbindung (die aber durchaus ein paar Tücken haben kann) über den Test, ob JavaScript bei einem Browser überhaupt aktiviert ist, die elementaren Grundstrukturen von JavaScript samt Funktionen und Methoden, den Umgang mit Objekten in JavaScript einschließlich des Erstellens eigener Objekte und Prototyping sowie die Nutzung des DOM-Konzepts, die Technik der Ausnahmebehandlung bis hin zur Ereignisbehandlung, die vollkommen in JavaScript verlagert wird.

6 HTTP und die Interna des XMLHttpRequest-Objekts

Sowohl Segen als auch Fluch des WWW ist das Dienstprotokoll **HTTP** (HyperText Transfer Protocol)[1], über das grundsätzlich die Kommunikation zwischen einem Webbrowser und einem Webserver erfolgt. Auf der einen Seite ist HTTP schnell, einfach und zuverlässig. Auf der anderen Seite ist es jedoch zustandslos, was die Verfolgung von Sitzungen und die ressourcenschonende Nachforderung von Daten erschwert. Kenntnisse in HTTP sind nun auf der einen Seite nicht unbedingt notwendig, um AJAX-Schablonen *anwenden* und *anpassen* zu können, jedoch unerlässlich, wenn Sie den Grund für die Notwendigkeit von AJAX-basierten Anwendungen und deren genaue Hintergründe sowie die Eigenschaften und Methoden eines `XMLHttpRequest`-Objekts verstehen wollen. Und Letzteres ist auch die Basis dafür, anspruchsvollere AJAX-Applikationen selbst zu entwickeln.

6.1 Datenübertragung per HTTP

Erinnern Sie sich an den `<form>`-Tag in HTML und dort den `method`-Parameter oder auch den ersten Parameter der `open()`-Methode eines `XMLHttpRequest`-Objekts unter AJAX? Die Angaben dort legen bei der Datenübertragung per HTTP die genaue Art und Weise fest, wie die eingegebenen Daten zum Server gelangen und dort zu behandeln sind. Dabei wird gewöhnlich eine von den nachfolgenden zwei[2] Methoden verwendet:

- GET
- POST

Diese beiden Angaben spezifizieren unterschiedliche Möglichkeiten, mit HTTP Daten vom Client zum Server zu schicken. HTTP ist ein zustandsloses Protokoll. Das bedeutet, zu jeder Anfrage eines Clients (der so genannte **Request**) wird vom Empfänger

1 *Die genauen Spezifikationen sind in dem RFC2616 und RFC 1945 beschrieben.*

2 *Zusätzlich gibt es noch Methoden wie* `PUT`*, die zum Hinzufügen einer Ressource zum Datenbestand des Servers verwendet wird, und* `HEAD`*. Allerdings wird in der Praxis so gut wie immer* `POST` *oder* `GET` *verwendet. Etwas weiter unten werden aber die Methoden der Datenübertragung mit HTTP erläutert.*

(dem Webserver) genau eine Antwort (englisch: **Response**) generiert. Ein solcher HTTP-Request wird zum Beispiel von einem Webbrowser generiert, wenn Anwender einen URL in die Adresszeile des Browsers eingegeben oder auf einen Hyperlink in der Webseite geklickt haben. Ist die Anfrage erfolgreich, dann enthält die Antwort des Servers die angefragte Ressource zur Darstellung. Bei einer klassischen Webseite ohne Anwenderinteraktion ist der Frage-Antwort-Zyklus damit beendet. Ein solcher Zyklus läuft aber ebenso ab, wenn Benutzereingaben in einem HTML-Formular durch den Client abgeschickt oder auch per AJAX Daten vom Server angefordert werden. Und das soll ja möglicherweise unbemerkt vom Anwender erfolgen. Die so genannten **HTTP-Header** (**HTTP Request Header** für die Anfrage und **HTTP-Response Header** für die Antwort) werden dabei bei jedem Datenaustausch mit übertragen und bestehen aus reinem Klartext.

Bevor die Daten vom Client an den Webserver gesendet werden, werden sie zunächst vom Browser zu einer einzigen Zeichenkette verpackt. Die Art und Weise der weiteren Behandlung wird bei einem Formular durch den Parameter `method` gesteuert, den wir wie gesagt auch bei der `open()`-Methode eines `XMLHttpRequest`-Objekts unter AJAX als ersten Parameter wiederfinden. Erinnern Sie sich an Zeile 12 in unserer externen JavaScript-Datei aus dem Beispiel in Kapitel 2:

Listing 6.1: Zeile 12 der Funktion zum Senden der Anfrage

```
resObjekt.open('get', 'laender.jsp?wo='+i,true);
```

Bei der `GET`-Methode werden die Daten vor dem Versenden zusammen an das Ziel der Datenübertragung (den URL) angehängt, wobei die Daten durch ein `?` von dem ursprünglichen URL abgetrennt werden. Diesen entstehenden **Pseudo-URL** sieht ein Anwender **nach dem Abschicken** eines Formulars im Adressfeld des Browsers.

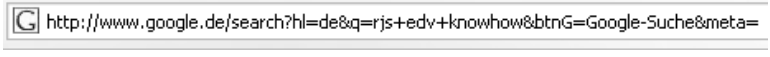


Abbildung 6.1: Bei der GET-Methode sieht der Anwender nach dem Abschicken die versendeten Daten im Adressfeld seines Browsers.

So einen Pseudo-URL können Sie natürlich auch manuell im Adressfeld des Browsers eingeben. Lesezeichen mit vorgefertigten Anfragen basieren auch auf diesem Prinzip.

Hinweis

Unabhängig von der verwendeten Versandmethode wertet das empfangende Programm bzw. Skript auf dem Server den Inhalt dieser Umgebungsvariablen aus, wobei einige Arbeit zu leisten ist. Der Pseudo-URL mit der Übergabe von Werten an ein Skript bzw. Programm beinhaltet die Zeichen der Benutzereingabe nicht unverändert. Bei der Generierung des Pseudo-URLs wird in der Regel der Prozess der **URL-Kodierung** durchlaufen, was beim Versenden von

Formulardaten automatisch erfolgt. Beim manuellen Versenden von Daten, wie es zum Beispiel beim Aufruf der `open()`-Methode der Fall ist, müssen Sie die URL-Kodierung von Hand erledigen[3]. Diese URL-Kodierung erfolgt hauptsächlich, damit bei der Übertragung von Sonderzeichen über das Internet/Intranet keine Probleme entstehen. Dabei werden alle Leerzeichen in dem String, der aus den Benutzereingaben zusammengesetzt wird, durch Pluszeichen ersetzt. Zusätzlich werden sämtliche reservierten Zeichen, die ein Benutzer beispielsweise im Formular eingegeben hat (etwa das Gleichheitszeichen oder das kaufmännische Und), in hexadezimale Äquivalente konvertiert. Ein so konvertiertes Zeichen wird jeweils mit dem Prozentzeichen eingeleitet. Danach folgt der Hexadezimalcode.

In dem String des Pseudo-URL können also weder das Gleichheitszeichen, das Prozentzeichen, das oben erwähnte Fragezeichen noch das Pluszeichen oder das kaufmännische Und (Ampersand) durch Benutzereingaben vorkommen. Aber auch zahlreiche andere Zeichen wie deutsche Umlaute oder das ß werden verschlüsselt.

Wenn die Daten an ein auswertendes Skript bzw. Programm übermittelt werden, kommen alle Daten als ein Satz von Name-Wert-Paaren an. Der Name ist jeweils der, der in dem entsprechenden Tag auf der HTML-Seite festgelegt wurde. Die Werte sind das, was der User eingetragen oder ausgewählt hat. Dieser Satz von Name-Wert-Paaren wird in einem Teilstring des URL übermittelt, den ein Webskript wieder auflösen muss. Ein solcher String ist ungefähr so aufgebaut:

Listing 6.2: Ein Satz mit Wertepaaren

```
name1=wert1&name2=wert2&name3=wert3
```

Der String muss vom auswertenden Skript bzw. Programm beim kaufmännischen UND (`&`) sowie dem Gleichheitszeichen (`=`) in einzelne Stücke zerlegt werden. Die entstandenen Teilstücke müssen dann noch weiterverarbeitet – sprich dekodiert – werden. Dies umfasst unter anderem die Rückwandlung aller Pluszeichen in Leerzeichen und aller `%xx`-Sequenzen (entstanden aus der Hex-Übersetzung in den Pseudo-URL bei bestimmten Sonderzeichen) in einzelne Buchstaben mit dem ASCII-Wert `xx` (Beispiel: `%3D` wird wieder zu `=`). Wenn Sie das von Hand machen wollen, kann das sehr mühsam sein. Aber einige serverseitige Sprachen stellen Ihnen auch Standardfunktionen bereit (etwa JavaScript, das Sie auf dem Server im Rahmen von Active Server Pages anwenden können, mit `unescape()`) oder aber die

3 *Dazu nutzt man in JavaScript die Funktion* `escape()`.

Rückkodierung erfolgt sogar vollautomatisch. Die gesamte Rückwandlung der kodierten Zeichen als auch die Zerlegung der Wertepaare erfolgen hier vollautomatisch. Das macht PHP besonders bei Einsteigern so beliebt, aber Java (was ja unsere Server-Technologie ist) lässt sich hier nicht lumpen.

So bequem `GET` ist – in vielen Fällen kann es angebracht sein, eine andere Methode als `GET` einzusetzen. Das betrifft unter anderem alle Aktionen, die auf Datenbanken zugreifen. Hier kommt in der Regel `POST` zum Einsatz. Wenn Sie beim Versenden von Daten die Methode `POST` verwenden, werden die zu verschickenden Daten in eine Umgebungsvariable gepackt, die im HTTP-Rumpf versteckt ist[4]. Das empfangende Programm bzw. Skript wird die Daten wie eine auf der Kommandozeilenebene erfolgte Benutzereingabe behandeln. Es gibt daher kein EndOfFile-Signal (EOF) und die Länge der übermittelten Daten muss vom empfangenden Programm bzw. Skript aus einer weiteren Standardumgebungsvariable des HTTP-Headers entnommen werden. Für den Anwender hat `POST` die Konsequenz, dass nach dem Versenden von Formulardaten in der Adresszeile des Browsers keine der verschickten Eingaben zu sehen ist. Das spielt allerdings bei einer AJAX-Anfrage keine Rolle, da hier in der Adresszeile des Browsers grundsätzlich nichts zu sehen ist (die Kommunikation läuft ja explizit am Browser vorbei).

Hinweis

Die `GET`-Methode wird von den meisten Browsern beim Versenden von Formularen standardmäßig verwendet, wenn Sie den `method`-Parameter nicht angeben. Allerdings eignet sich die Methode `GET` durch die Abhängigkeit von der maximalen Länge eines URL nicht zur Übertragung größerer Datenmengen! Ebenso stellt die Anzeige der Benutzerdaten in der Adresszeile bei sensiblen Daten ein Sicherheitsrisiko dar (wobei nicht der Trugschluss erfolgen darf, dass `POST` eine sichere Übertragung gewährleistet – die Daten werden nur nicht in der Adresszeile des Browsers angezeigt). Suchmaschinen verwenden sehr oft die `GET`-Methode zur Übertragung von Suchbegriffen, denn ein Ergebnis-URL lässt sich als Lesezeichen im Browser speichern. Damit ist die Suche selbst über ein Lesezeichen aufrufbar und muss bei einer späteren Suche nicht mehr manuell eingeben werden.

4 *Diese Daten werden bei AJAX an die* `send()`*-Methode übergeben. Mit der Methode* `setRequestHeader()` *kann sichergestellt werden, dass der Medientyp* `applicationjx-www-form-urlencoded` *der übertragenen Daten auf dem Server korrekt verarbeitet werden kann.*

6.1.1 HTTP-Interna

HTTP wurde 1989 von Tim Berners-Lee zusammen mit der Technik des URL und HTML entwickelt und ist ein Dienstprotokoll, das auf der konkreten Datenübertragung via einem Transportprotokoll aufsetzt. Im Internet bzw. Intranet kommt dazu bekanntlich TCP/IP zum Einsatz.

Hinweis

Derzeit gibt es von HTTP zwei Versionen – HTTP/1.0 und HTTP/1.1. Wesentlicher Unterschied ist, dass bei HTTP/1.1 bei Bedarf im Gegensatz zur Version 1.0 mehrere Anfragen und Antworten in einer einzigen TCP-Verbindung gesendet und abgebrochene Übertragungen fortgesetzt werden können. Zudem lassen sich Dateien vom Client zum Server übertragen oder dort auch löschen. Für die konkrete Arbeit mit AJAX ist das allerdings meist irrelevant.

HTTP wird in den offiziellen RFCs von TCP der Standard-Port 80 zugeordnet. Zusätzliche Informationen für die Kommunikation (Übertragungsart, Browser-Typ, Server-Typ, Sprache etc.) können im **HTTP-Header** untergebracht werden. Gegebenenfalls können von einer Applikation auch eigene Header-Felder ergänzt werden. Ein HTTP-Header wird immer von einer Leerzeile begrenzt.

HTTP-Request

Bei einem typischen HTTP-Request durch einen Client besteht die gesamte Anfrage ausschließlich aus dem HTTP-Header. Schauen wir uns so einen typischen HTTP-Request exemplarisch an:

Listing 6.3: Ein vollständiger HTTP-Request

```
01 GET /ajax/ HTTP/1.1\r\n
02 Connection: Keep-Alive\r\n
03 User-Agent: Mozilla/5.0 (compatible; Konqueror/3.4; Linux) KHTML/3.4.2 ➥
   (like Gecko)\r\n
04 Accept: text/html, image/jpeg, image/png, text/*, image/*, */*\r\n
05 Accept-Encoding: x-gzip, x-deflate, gzip, deflate\r\n
06 Accept-Charset: utf-8, utf-8;q=0.5, *;q=0.5\r\n
07 Accept-Language: de, en\r\n
08 Host: 192.168.1.120\r\n
09 \r\n
```

Deutlich sind in Zeile 1 die Methode der Datenanforderung (hier `GET`), ein relativer URL und die Protokollversion (hier HTTP 1.1) zu erkennen. Auf die Startzeile des Request folgen eine Reihe von Feldern in mehr oder weniger beliebiger Reihenfolge, die die Anfrage genauer beschreiben. Jedes Header-Feld besteht aus einem Namen und einem Wert. Das Wertepaar wird durch einen Doppelpunkt voneinander

getrennt. Die Werte gleichnamiger Header-Felder können in einem Header zusammengefasst werden, indem sie mit einem Komma getrennt werden. Die wichtigsten standardisierten Header-Felder in diesem Request sind folgende (wobei Sie auch eigene Felder definieren können – der Webserver muss diese nur auch verstehen):

Header-Feld	Beschreibung
`Accept`	Eine Liste mit der Angabe der erlaubten **MIME-Typen** (Multipurpose Internet Mail Extensions). MIME steht für einen Internetstandard zur Spezifizierung von Dateitypen bei der Kommunikation zwischen Servern und Browser im WWW. Sowohl der Server als auch der Browser kennen bestimmte Dateitypen. Beim Übertragen vom Server zum Browser wird über das HTTP-Protokoll der MIME-Typ mit übertragen. Aufgrund seiner Liste mit MIME-Typen kann der Browser respektive der Server eine Datei eines bekannten Typs korrekt behandeln. MIME-Typen werden nach folgendem Schema angegeben: `Hauptkategorie/Unterkategorie` Hauptkategorien sind etwa `text`, `image` oder `audio`. Unterkategorien von `text` sind beispielsweise `plain` (eine reine Textdatei), `javascript` (beim Einbinden von JavaScripts) oder `html` (eine HTML-Datei). Unterkategorien von `image` sind beispielsweise `gif` oder `jpeg`.
`Accept-Charset`	Die Spezifikation der akzeptierten Zeichenkodierungen, in denen der Inhalt vorliegen darf. Für XML-Dokumente sollte hier zumindest UTF-8 erlaubt werden, um größtmögliche Kompatibilität zu erhalten. Jeder Eintrag kann mit einem Parameter `q` mit Werten zwischen 0 und 1 zur Gewichtung versehen werden.
`Accept-Encoding`	Eine Liste der zur Übertragung des Inhalts erlaubten Kodierungen. Insbesondere kann eine komprimierte Übertragung angegeben werden.
`Accept-Language`	Eine Liste der Sprachen, die der Benutzer als Präferenz in seinem Browser eingestellt hat. Auch hier ist wieder eine Gewichtung über den Parameter `q` möglich.
`Connection`	Über `Connection` kann die Art der Verbindung angegeben werden. `Keep-alive` ist der Versuch, eine Verbindung offen zu halten.
`Host`	Der Server mit der angefragten Ressource. Gegebenenfalls mit zusätzlicher Portnummer, falls nicht der Standard-Port 80 für HTTP verwendet wird.
`User-Agent`	Der Browser des Besuchers. In dem Beispiel steht `Mozilla/5.0 (compatible; Konqueror/3.4; Linux) KHTML/3.4.2 (like Gecko)` für den Konqueror der KDE 3.4 unter Linux.

Tabelle 6.1: Wichtige HTTP-Header-Felder

HTTP-Response

Bei der Antwort des Webservers wird neben den Header-Informationen, die wie die vom Request aussehen und oftmals gleich benutzt werden, der tatsächliche Inhalt als Nutzlast transportiert. Das sind dann Teile der Datei, die ein Client vom Webserver angefordert hat. Eine typische Antwort eines Webservers sieht so aus:

Listing 6.4: Ein typische Webserver-Antwort per HTTP

```
01 HTTP/1.1 200 OK\r\n
02  Request Version: HTTP/1.1
03  Response Code: 200
04 Date: Mon, 19 Dec 2005 12:36:47 GMT\r\n
05 Server: Apache/2.0.52 (Win32) mod_ssl/2.0.52 OpenSSL/0.9.7c PHP/5.0.3RC2-dev\r\n
06 X-Powered-By: PHP/5.0.3RC2-dev\r\n
07 Content-Length: 44\r\n
08 Keep-Alive: timeout=15, max=100\r\n
09 Connection: Keep-Alive\r\n
10 Content-Type: text/html; charset=ISO-8859-1\r\n
11 \r\n
12 Line-based text data: text/html
13 Sie haben die Taste mit dem Code 74 gedr\374ckt
```

Eine Antwort des Servers beginnt mit einer Statuszeile, die neben der Identifikation des Protokolls und der Versionsnummer einen dreistelligen Statuscode (siehe unten) sowie optional eine Beschreibung des Ergebnisses in Textform enthält. Die Meldung kann gegebenenfalls auch aufgespalten werden. Der Statuszeile der Antwort folgt wie bei der Client-Anfrage eine Reihe von Header-Feldern, die weitere Informationen enthalten. Den wichtigsten Teil der Antwort stellen in den meisten Fällen aber die Inhaltsinformationen dar, die nach dem Block mit den Header-Feldern folgen und durch eine Leerzeile vom Header abgetrennt werden.

6.1.2 Meldungen des Webservers

Eine Datenanforderung an einen Server kann selbstverständlich schief gehen. Der Server sendet dann im HTTP-Header in der ersten Zeile eine Fehlermeldung zurück. Aber auch sonst gibt es **HTTP-Statuscodes**, die von Interesse sind, um die Antwort des Servers zu verstehen. Dabei hängen die Meldungen teilweise von der HTTP-Version ab.

HTTP-Statuscode	Beschreibung
1xx	Mit einer 1 beginnende Statuscodes kennzeichnen eine vorläufige Antwort. Die Transaktion wird mit einer lxx-Antwort nicht beendet und die Bearbeitung der Anfrage dauert noch an. In AJAX wird das vom `XMLHttpRequest`-Objekt als `INTERACTIVE` oder `LOADING` weitergegeben.
2xx	Mit einer 2 beginnende Statuscodes kennzeichnen eine erfolgreiche Operation. Allerdings muss je nach Situation das genaue Vorgehen weiter vom Client geregelt werden. Der Statuscode 200 steht etwa für `OK` und bedeutet, dass eine Anfrage erfolgreich bearbeitet und das Ergebnis der Anfrage in der Antwort übertragen wurde. In AJAX wird das vom `XMLHttpRequest`-Objekt als Status `COMPLETED` oder `LOADED` weitergegeben. Statuscode 201 bedeutet, eine angeforderte Ressource wurde vor dem Senden der Antwort erstellt, und 202 besagt, dass eine Anfrage zwar akzeptiert wurde, aber erst zu einem späteren Zeitpunkt ausgeführt werden kann.
3xx	Mit einer 3 beginnende Statuscodes kennzeichnen eine Umleitung und erfordern weitere Schritte seitens des Clients. So steht 300 für eine mehrfach verfügbare Ressource, deren konkrete Auswahl in einem Folgeschritt benannt werden muss. 301 und 302 ist die Kennzeichnung, dass eine Ressource unter einer neuen Adresse zur Verfügung steht (301 dauerhaft und 302 temporär).
4xx	Mit einer 4 beginnende Statuscodes kennzeichnen eine fehlerhafte Anfrage. So steht 400 für `Bad Request` (fehlerhafter Aufbau der Client-Anfrage), 401 für `Unauthorized` (Authentifizierung ungültig), 403 für `Forbidden` (fehlende Berechtigung des Clients) oder 404 für `Not Found` (die angeforderte Ressource wurde nicht gefunden).
5xx	Mit einer 5 beginnende Statuscodes kennzeichnen einen Server-Fehler. Dies umfasst Statuscode 500 (`Not Implemented`), 502 (`Bad Gateway`) oder 505 (`HTTP Version not supported`).

Tabelle 6.2: Statuscodes eines Webservers

Tipp

Zum Beobachten der Kommunikation zwischen einem Client und dem Webserver eignen sich hervorragend so genannte Sniffer (das sind Programme zur Netzwerkanalyse wie zum Beispiel **Ethereal** – *http://www.ethereal.com*).

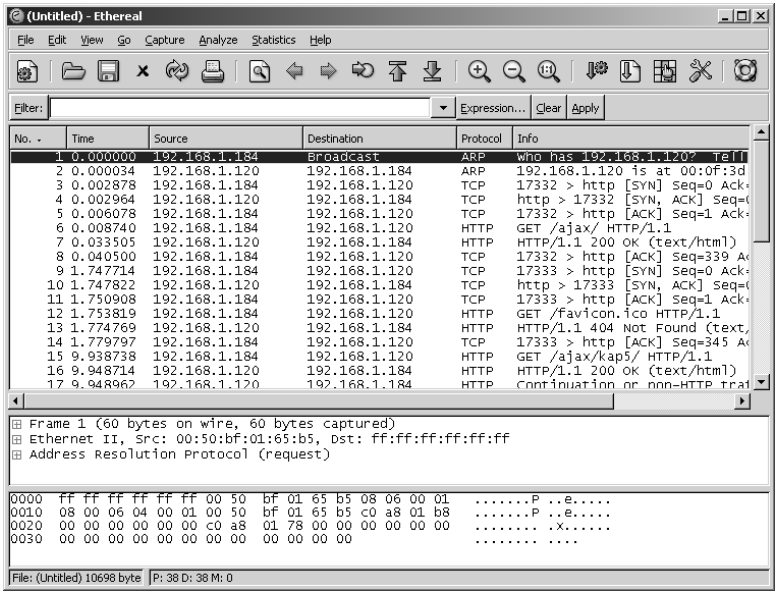


Abbildung 6.2: Mit einem Sniffer lässt sich der Netzwerkverkehr analysieren und viel über HTTP und andere Protokolle lernen

6.2 Details zum XMLHttpRequest-Objekt

Moderne Browser bieten mit dem Objekt `XMLHttpRequest` eine eingebaute Schnittstelle zur Kontrolle von HTTP-Transaktionen aus clientseitigen Programmiersprachen (hauptsächlich JavaScript). `XMLHttpRequest`-Objekte sind damit unmittelbar an dem internen Aufbau von HTTP orientiert und das Rückgrat jeder AJAX-Anfrage. Für die asynchrone Kommunikation zwischen Browser und Webserver erlaubt das Objekt zum einen die Registrierung von so genannten **Callback-Funktionen**, die bei jeder Änderung des Transaktionszustands ausgewertet werden, zum anderen kann auf alle Header-Felder von Anfrage und Antwort zugegriffen werden. Damit gehen die Möglichkeiten eines `XMLHttpRequest`-Objekts weit über die Funktionalität alternativer Lösungen wie die skriptbasierte Manipulation von Inline-Frames mit asynchroner Übertragung von Nutzdaten oder die Sitzungsverwaltung in Cookies hinaus.

Um mit einem `XMLHttpRequest`-Objekt arbeiten zu können, muss dieses entsprechend der Regel der eingesetzten Programmiersprache erzeugt werden. Wir haben dies in Kapitel 5 genauer angesehen. Ein Objekt vom Typ `XMLHttpRequest` stellt dann alle relevanten Methoden und Eigenschaften zur asynchronen Kommunikation bereit. Dabei können auf Basis der bekannten Übertragungsmethoden (`POST`, `GET`, `PUT` und `HEAD`) Daten verschickt und Antworten empfangen sowie der Request-Header gesetzt werden.

Achtung

Allgemein dürfen durch AJAX-Anfragen aus Sicherheitsgründen nur Daten von der gleichen Domain angefordert werden, von der die anfordernde Webseite stammt. Das ist ein oft zu findendes **Sandkastenprinzip** – Sandbox, wie wir es ja auch bei Java haben. Der URL muss also identisch mit dem der aktuellen Webseite sein (relativ oder absolut). Wenn also die Webseite den URL *http://rjs.de/ajax.html* hat, wäre *http://rjs.de/ajax.php* als Anforderung der Zusatzdaten erlaubt. Die Datennachforderung *http://webscripting.de/ajax.php* wäre verboten. Allerdings können Anwender verschiedener Browser – etwa eines Mozilla-Browsers wie Firefox – diese Sandbox deaktivieren und die Nachforderung von Daten von anderen Domains gestatten. Das nennt man dann einen **Cross-Domain-Zugriff**. Wenn Anwender bei einem Mozilla-Browser in der Adresszeile des Browsers zum Beispiel `about:config` eingeben, in der folgenden Liste den Eintrag `signed.applets.codebase_principal_support` suchen und dort den Wert auf `true` ändern (einfach einen Doppelklick ausführen), können auch nichtsignierte Skripte erhöhte Zugriffsrechte erhalten und damit Daten von fremden Domains nachfordern.

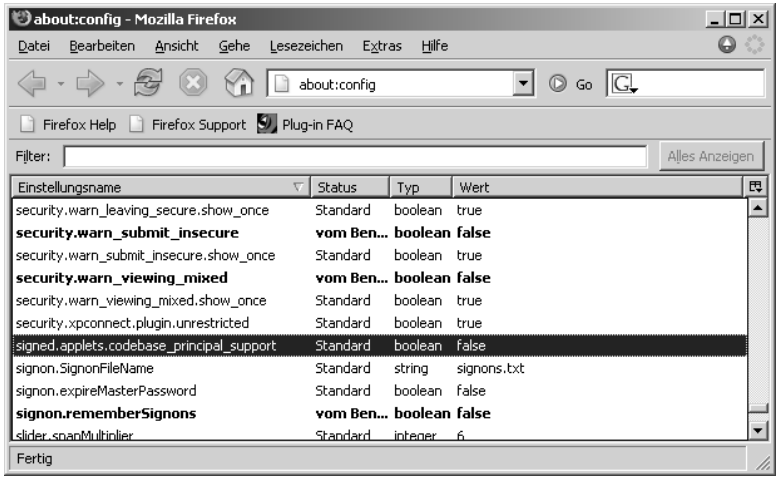


Abbildung 6.3: Änderung der Einstellungen bei einem Mozilla-Browser über about:config

Die Freischaltung ist allerdings aus Sicherheitsgründen hochkritisch und sollte in der Regel unterbleiben. Und für die Praxis von AJAX-Anwendungen ist die Möglichkeit von Cross-Domain-Zugriffen ohnehin vollkommen uninteressant. Da nur eine verschwindend kleine Anzahl an Besuchern diese Möglichkeit aktiviert haben wird, benötigen Sie für alle Anwender ohne diese freigeschalteten Cross-Domain-Zugriffe sowieso eine andere Lösung. Die Freischaltung bietet sich höchstens für eine Testumgebung an.

Abbildung 6.4: Die Einstellung zur Beschränkung von nichtsignierten Skripten wurde aufgehoben.

6.2.1 Methoden eines XMLHttpRequest-Objekts

Nachfolgend finden Sie die Methoden eines `XMLHttpRequest`-Objekts:

Methode	Beschreibung
`abort()`	Stopp der aktuellen Server-Anfrage
`getAllResponseHeaders()`	Rückgabe der vom Server gesendeten Header-Felder als String
`getResponseHeader ("headerLabel")`	Die Methode gibt das als Parameter benannte Header-Feld als String zurück.
`open("method", "URL" [, asyncFlag[, "user-Name"[, "password"]]])`	Die `open()`-Methode ist eine der wichtigsten Methoden eines `XMLHttp-Request`-Objekts. Sie öffnet eine Verbindung zu einem Webserver. Dabei werden mindestens zwei Parameter angegeben. Diese ersten beiden Argumente sind mittlerweile klar. Für `method` kann `GET`, `POST`, `PUT` oder `HEAD` verwendet werden, wobei in der Praxis `GET` oder `POST` dominieren. `POST` kommt insbesondere dann zum Einsatz, wenn die gesendeten Daten größer als 500 Byte sind. `HEAD` wird verwendet, wenn nur Response-Header und keine Daten angefordert werden. Dies kann z. B. verwendet werden, wenn für eine Datei auf dem Server das Datum der letzten Änderung abgefragt werden soll. `PUT` wird zum Kopieren von Dateien auf den Server eingesetzt.

Tabelle 6.3: Die Methoden eines XMLHttpRequest-Objekts

Methode	Beschreibung
	Der URL ist der relative oder absolute Pfad zum Serverskript, wobei gegebenenfalls (bei `GET`) der Querystring angefügt und ein Pseudo-URL verwendet wird. Das dritte Argument gibt an, ob eine Anfrage synchron (`false`) oder asynchron (`true`) verarbeitet wird. Sofern eine synchrone Verarbeitung festgelegt wird, wird der folgende Versand der Daten mit der `send()`-Methode die Ausführung des Skripts so lange blockieren, bis die Antwort des Servers vollständig empfangen wurde. In diesem Fall kann die Antwort des Servers im unmittelbar folgenden Schritt des Skripts verarbeitet werden. Beim asynchronen Anfordern von Daten wird das blockierende Warten des Browsers auf die Antwort vermieden. Das Skript läuft dann nach dem Absenden des Request einfach weiter. Wenn Sie so eine Anfrageform wählen, kann die Antwort des Webservers nicht im unmittelbar nächsten Schritt des Skripts verarbeitet werden, da die Antwort wegen der Antwortzeiten der Transaktion so gut wie nie rechtzeitig da sein kann. Stattdessen wird eine so genannte **Callback-Funktion** definiert, die immer dann aufgerufen wird, wenn sich der Bearbeitungszustand der Transaktion ändert. Dies kann mit einer speziellen Eigenschaft eines `XMLHttpRequest`-Objekts – `onreadystatechange` – verfolgt werden. Der optionale Parameter `userName` ist ein gegebenenfalls benötigter Benutzername für eine Ressource auf dem Server und `password` entsprechend das Passwort.
`send(content)`	Die `send()`-Methode wird zum Abschicken einer Anfrage verwendet. Sie wird nach dem Aufruf der `open()`-Methode aufgerufen. Der Parameter `content` ist entweder `null` (bei `GET`) oder ein Querystring bei `POST`.
`setRequestHeader ("label", "value")`	Mit dieser Methode können individuelle Header-Felder gesetzt werden.
`setMimeType ("mime-type")`	Setzen des MIME-Typs der angeforderten Daten (Response). Der Parameter `mimetype` ist die übliche Angabe eines MimeType als String (z.B. `"text/xml"` oder `"text/html"`). Achtung: Die Methode wird von einigen Browsern nicht unterstützt (etwa dem Internet Explorer). In der Praxis benötigen Sie beim Anfordern von XML-Daten auch oft keine explizite Festlegung von XML. Selbst wenn die XML-Daten ohne Festlegung vom Server gesendet werden, können Sie auf dem XML-Baum navigieren. Die XML-Elemente stehen Ihnen auch dann zur Verfügung, wenn reiner Text oder HTML als MIME-Typ gesendet wird.

Tabelle 6.3: Die Methoden eines XMLHttpRequest-Objekts (Forts.)

6.2.2 Eigenschaften

Ein `XMLHttpRequest`-Objekt besitzt neben den Methoden eine Reihe von Eigenschaften, die Sie für eine AJAX-Applikation benötigen:

<table>
<tr><th>Eigenschaft</th><th>Kurzbeschreibung</th></tr>
<tr><td>onreadystate-change</td><td>Der Eventhandler wird jedes Mal aufgerufen, wenn sich der Verbindungsstatus (<code>readyState</code>) eines <code>XMLHttpRequest</code>-Objekts ändert. Man registriert bei diesem Eventhandler in der Regel eine Funktionsreferenz auf eine Callback-Funktion.</td></tr>
<tr><td>readyState</td><td>Die Eigenschaft enthält den aktuellen Verbindungsstatus einer Transaktion. Mögliche Werte sind folgende:
<table>
<tr><td>0</td><td>UNINITIALIZED</td><td>Das Objekt wurde noch nicht initialisiert. Das bedeutet, die Verbindung wurde noch nicht geöffnet und <code>open()</code> noch nicht aufgerufen.</td></tr>
<tr><td>1</td><td>LOADING</td><td>Das Request-Objekt wurde erstellt und initialisiert, aber es wurde noch keine Anfrage mit <code>send()</code> gesendet.</td></tr>
<tr><td>2</td><td>LOADED</td><td>Die Anfrage wurde gesendet und der Antwort-Header sowie der Antwortstatus können ausgewertet werden.</td></tr>
<tr><td>3</td><td>INTERACTIVE</td><td>Die Daten vom Server treffen gerade ein. Die Eigenschaften <code>responseText</code> bzw. <code>responseXML</code> enthalten die bereits empfangenen Daten.</td></tr>
<tr><td>4</td><td>COMPLETED</td><td>Die Kommunikation mit dem Server ist abgeschlossen und alle Daten sind angekommen, wenn kein Fehler aufgetreten ist. Dies ist der wichtigste Fall bei der Erstellung von AJAX-Applikationen.</td></tr>
</table>
Sie können beispielsweise in einer <code>switch-case</code>-Entscheidung die Statuscodes auswerten und entsprechend Reaktionen implementieren. Aber Achtung: Die Statuscodes außer 4 (<code>COMPLETED</code>) sind in verschiedenen Browsern nicht wirklich einheitlich implementiert. Deshalb macht in der Praxis fast nur die Abfrage auf <code>readyState == 4</code> Sinn.</td></tr>
<tr><td>responseText</td><td>Die Eigenschaft enthält die vom Server gesendeten Daten als Text.</td></tr>
<tr><td>responseXML</td><td>Die Eigenschaft enthält die vom Server gesendeten Daten als XML-Daten. Wenn die Daten nicht in XML-Form gesendet worden sind, enthält <code>responseXML</code> den Wert <code>null</code>.</td></tr>
<tr><td>status</td><td>Die Eigenschaft enthält den HTTP-Status der Verbindung als Zahl.</td></tr>
<tr><td>statusText</td><td>Die Eigenschaft enthält den HTTP-Status als Textmeldung, sofern eine solche übermittelt wurde.</td></tr>
</table>

Tabelle 6.4: Die Eigenschaften eines XMLHttpRequest-Objekts

6.3 Exemplarischer Ablauf einer AJAX-Anfrage

Eine AJAX-Anfrage kann sich zwar in diversen Details unterscheiden, folgt aber in der Regel immer dem gleichen Schema:

1. Zuerst wird ein `XMLHttpRequest`-Objekt erzeugt.
2. Eine Callback-Funktion wird beim `XMLHttpRequest`-Objekt als Funktionsreferenz registriert. Diese wird dann bei jeder Zustandsänderung der Transaktion aufgerufen. Beispiel: `resObjekt.onreadystatechange = handleResponse;`. Die angegebene Funktion wird fortan für jede Statusänderung des `XMLHttpRequest`-Objekts aufgerufen. Das Feld `readyState` des `XMLHttpRequest`-Objekts gibt Aufschluss über den aktuellen Status der Transaktion beim Aufruf dieser Callback-Funktion. So lassen sich einzelne Phasen der Datenübertragung unterscheiden. Der wichtigste Fall ist hier das Erreichen des Status `COMPLETED` mit dem Statuscode 4. Nur dieser Status wird in den verschiedenen Browsern einheitlich gehandhabt. Deshalb werden fast alle AJAX-Anwendungen darauf prüfen.
3. Die Verbindung wird geöffnet, indem die `open()`-Methode des `XMLHttpRequest`-Objekts aufgerufen wird. Beispiel: `resObjekt.open('get', 'http1.jsp',true);`. Das ist aber noch nicht die konkrete Anfrage. Deshalb ist es auch unerheblich, ob Schritt 2 und 3 vertauscht werden.
4. Die Anfrage wird mit der `send()`-Methode abgeschickt. Beispiel: `resObjekt.send (null);`.
5. Die Antwort wird verwertet. Dazu kann die Statusänderung eines `XMLHttpRequest`-Objekts explizit genutzt werden.

6.4 Ein praktisches AJAX-Beispiel mit Auswertung der HTTP-Header

Schauen wir uns nun ein praktisches AJAX-Beispiel an, das die Eigenschaften und Methoden eines `XMLHttpRequest`-Objekts nutzt. Die Webseite implementiert nur einen Eventhandler, der die Funktion `sndReq()` bei einem Tastendruck durch den Anwender aufruft. Zudem ist wieder ein `<span>`-Container für die Antwort vorgesehen:

Listing 6.5: Die Webseite mit der Anforderung

```
01 <html>
02 <script language="JavaScript" src="http1.js"></script>
03 <script language="JavaScript">
04  window.onkeypress = sndReq;
05 </script>
06 <body>
07 <h1>Drücken Sie eine Taste!</h1>
08 <span id="antwort"></span>
09 </body>
10 </html>
```

Die externe JavaScript-Datei enthält zwei interessante Funktionen, die wir ansehen sollten:

Listing 6.6: Der interessante Ausschnitt der externen JavaScript-Datei

```
22 function sndReq() {
23     resObjekt.open('get', 'http1.jsp',true);
24     resObjekt.onreadystatechange = handleResponse;
25     resObjekt.send(null);
26 }
27 function handleResponse() {
28   if(resObjekt.readyState == 4){
29     document.getElementById("antwort").innerHTML =
30     "JSP-Antwort: " + resObjekt.responseText +
31     "<br />Übertragungsstatus: " + resObjekt.readyState +
32     "<br />Status: " + resObjekt.status +
33     "<br />Statustext: " + resObjekt.statusText +
34     "<hr />Die vom Server gesendeten Header-Felder: " +
35     resObjekt.getAllResponseHeaders();
36   }
37 }
```

Abbildung 6.5: Die Antwort vom Server – hier im Konqueror

In der `sndReq()`-Methode erfolgt dieses Mal nur der Aufruf eines URL und es werden keinerlei weitere Daten an das Serverskript gesendet. Das ist zwar unüblich, aber keinesfalls verboten. Das Serverskript wird auch nur eine hartkodierte Antwort liefern[5]. Interessant ist in diesem Beispiel die Funktion `handleResponse()`, in der die oben beschriebenen Eigenschaften bzw. der Rückgabewert der Methode `getAllResponseHeaders()` ausgewertet und in der Webseite angezeigt werden.

6.5 Zusammenfassung

Wir haben in diesem Kapitel die wichtigen Fakten zu HTTP und den Interna eines `XMLHttpRequest`-Objekts erarbeitet. HTTP ist ein schnelles, einfaches und zuverlässiges Dienstprotokoll, das man seit dem Auftreten von AJAX respektive der `XMLHttpRequest`-Schnittstelle auch aus Skriptsprachen im Client gezielt nutzen kann, um damit asynchron Daten nachzufordern. Ein `XMLHttpRequest`-Objekt stellt dazu Eigenschaften und Methoden bereit, um auf die Bestandteile eines HTTP-Headers wie auch die Nutzdaten einer Response gezielt zugreifen zu können. Damit wird das Rückgrat jeder AJAX-Applikation gebildet.

5 *Auf den Code verzichten wir hier. Der Befehl zur Ausgabe wird aus dem Screenshot deutlich – es ist reines HTML.*

7 DHTML für AJAX-Entwickler

Die wesentlichen Vorteile einer AJAX-Anwendung gegenüber einer klassischen Webanwendung basieren auf dem meist unbemerkten Nachladen und Austauschen von Informationen in einer bereits geladenen Seite. Diese neuen Informationen müssen dann aber smart in die Webseite integriert werden. Der Schlüssel dazu ist ein Verfahren, das unter **DHTML** (**Dynamic HTML**) bekannt geworden ist und das wir bisher auch schon in Grundzügen angewendet haben. Aber auch für den Fall, dass eine AJAX-Applikation Daten nachlädt und dem Anwender diesen Ladevorgang deutlich machen muss, ist DHTML das Mittel der Wahl. Dabei bezeichnet DHTML keine neue oder eigenständige Technik, sondern die Verbindung von (X)HTML mit JavaScript und Style Sheets.

7.1 Grundlagen von DHTML

Dynamic HTML oder DHTML ist ein Begriff, der eng mit dem DOM zusammenhängt. Der Begriff ist jedoch weder eindeutig noch standardisiert. Insbesondere legen verschiedene Interessengruppen DHTML unterschiedlich aus.

Hinweis

Zentraler Part von DHTML der Firma Netscape war die Layer-Technik, die ja mittlerweile von keinem aktuellen Browser mehr unterstützt wird. Dabei sollte jedoch nicht übersehen werden, dass der Begriff **Layer** auch heute noch häufiger verwendet wird. Damit sind dann aber andere Dinge (nicht mehr `<layer>`-Elemente) gemeint. Oft wird Layer als Synonym für einen `<div>`- oder `<span>`-Container verwendet.

Bei aller Differenz in der konkreten Auslegung – die wichtigsten Protagonisten im Internet verstehen und verstanden unter dynamischen HTML die Veränderungen einer Webseite, **nachdem** die Seite respektive Daten bereits beim Client (also im Browser) angelangt sind. Die konkrete Technik, wie die Veränderung dabei realisiert wird, ist absolut irrelevant, wobei DHTML sehr oft als Verbindung von (X)HTML, JavaScript und Style Sheets oder zumindest (X)HTML und JavaScript zur optischen Veränderung der Webseite bezeichnet wird. In jedem Fall bildet das DOM-Konzept

den zentralen Part, denn über dessen Objekte manipuliert man mit JavaScript bestehende Parts der Webseite. Allgemein zählt die dynamische Veränderung einer Webseite zu den eindrucksvollsten Anwendungen von JavaScript, aber rein von der Programmierung her auch zu den einfacheren. Die einzige Krux ist die hohe Inkompatibilität verschiedener Browser-Modelle, sofern man explizit bestimmte Möglichkeiten ausreizen will. Insbesondere sollte auf proprietäre Ansätze wie die Verwendung des `all`-Objekts verzichtet werden (mit Ausnahme der Eigenschaft `innerHTML`), wenn man seine Webpräsenz nicht nur ausschließlich für Besucher mit dem Internet Explorer erstellen will. Dieses `all`-Objekt, das in der JavaScript-Objekthierarchie von Microsoft direkt dem `document`-Objekt untergeordnet ist, bietet im Grunde vollständigen Zugang auf alle einzelnen Elemente und Inhalte einer HTML-Datei. Aber obwohl fast alle Browser-Hersteller durch die reine Marktdominanz von Microsoft eine gewisse Unterstützung für `all` mittlerweile bereitstellen, ist diese nie vollständig und deshalb eine Quelle hoher Ungewissheit und Ärgers. Zudem löst das offizielle `node`-Objekt des DOM-Konzepts sukzessive rein auf einen Browser oder auch nur HTML beschränkte Objekte ab. Das `node`-Objekt, das ebenso wie `all` ein Unterobjekt von `document` ist, ist nicht auf eine einzelne Sprache wie HTML beschränkt, sondern funktioniert im Zusammenhang mit allen Auszeichnungssprachen, die ein Dokument wie einen Baum aus so genannten Knoten auffassen lassen. Dort stellen jedes Element, jedes Attribut und alle Inhalte eigene Knoten dar. Das `node`-Objekt stellt Eigenschaften und Methoden für alle Knoten in so einem Baum bereit. Rein praktisch können Sie bei DHTML jedoch sowohl mit HTML-Elementobjekten als auch mit dem `node`-Objekt arbeiten. Sie müssen sich der Details oft nicht einmal bewusst sein.

7.2 Der Zugriff auf Elemente einer Webseite

Um DHTML-Effekte in einer Webseite zu realisieren, müssen Sie deren einzelne Bestandteile ansprechen. Wie wir in den bisherigen Kapiteln bereits gesehen haben, gibt es verschiedene Wege, um auf die Elemente einer Webseite zuzugreifen. Diese funktionieren zum Teil nur in bestimmten Browsern, teilweise aber auch in allen modernen Browsern.

7.2.1 Zugriff über Objektfelder

Eine Möglichkeit für DHTML-Effekte basiert auf der automatischen Speicherung von Teilen einer Webseite durch den Browser. Es ist im Rahmen des DOM-Konzepts ja so, dass für bestimmte Elemente einer Webseite beim Laden durch den Browser Objektfelder angelegt werden, etwa Links, Java-Applets, Formulare, Formularfelder oder Grafiken. In dem Fall ist der Zugang zu einem spezifischen Element der Webseite über `window.document` und dann den Namen des Objektfelds samt Index möglich. Eventuell auch über mehrere Ebenen und auch mit verkürztem Zugang unter Weglassung von `window`. Beispiele:

Listing 7.1: Zugriff auf Objektfelder

```
window.document.images[9]
document.forms[2]
document.forms[0].elements[6]
```

Die jeweiligen Objekte stellen natürlich spezifische Eigenschaften und Methoden bereit. Wir werden gleich verschiedene praktische Beispiele sehen.

7.2.2 Zugriff über Namen

Objektfelder sind zwar ein bequemer Weg, um auf Elemente in einer Webseite zuzugreifen, insbesondere in Zusammenhang mit Schleifen. Aber sie unterliegen ein paar Einschränkungen. So werden beileibe nicht alle Elemente in einer Webseite über Objektfelder zur Verfügung gestellt. Und wenn sich der Aufbau einer Webseite ändert[1], muss auch der Zugriff unter Umständen angepasst werden, da die Indizierung nicht mehr stimmt. Neben Objektfeldern steht Ihnen aber auch der Weg zu Elementen einer Webseite zur Verfügung, bei denen das HTML-Attribut `name` gesetzt wurde. Dieses muss dazu natürlich in der Webseite eindeutig sein oder zumindest indiziert. In diesem Fall ist der Zugang über `window.document` und dann den Namen des Elements möglich – eventuell auch wieder über mehrere Ebenen und auch mit verkürztem Zugang unter Weglassung von `window`. Wenn eine Grafik zum Beispiel wie folgt in der Webseite referenziert wird:

Listing 7.2: Ein Grafikelement mit einem Namen

```
<img src="leer.gif" height="50" name="status" />
```

Dann ist der Zugriff wie folgt möglich:

Listing 7.3: Zugriff auf das Element mit Namen status

```
document.status
```

Oder wenn Sie ein Webformular mit einem Eingabefeld nehmen:

Listing 7.4: Ein Webformular mit einem einzeiligen Eingabefeld

```
<form name="f1"><input name="a" /></form>
```

Dann ist der Zugriff wie folgt möglich:

Listing 7.5: Zugriff auf das Formular und dann das Eingabefeld mit dem jeweiligen Namen

```
document.f1.a
```

1 *Zum Beispiel wird am Anfang der Webseite eine neue Grafik eingefügt.*

Natürlich können Sie auch den Zugriff über einen Namen und ein Objektfeld mischen. Beispiel:

Listing 7.6: Zugriff auf das Formular über das Objektfeld und dann das Eingabefeld mit dem Namen

```
document.forms[0].a
```

Zugriff über getElementsByName()

Eine Alternative zum Zugriff auf Elemente einer Webseite beruht auf einer Methode des `document`-Objekts mit Namen `getElementsByName()`. Als Parameter geben Sie den Wert an, der mit dem `name`-Attribut unter HTML gesetzt wurde. Beispiel:

Listing 7.7: Zugriff auf ein Element mit getElementsByName()

```
document.getElementsByName("zustand")[0]
```

Achtung

Beachten Sie, dass auch dann mit Array-Syntax auf die Elemente zugegriffen werden muss, wenn ein Elementname nur einmal im Dokument vorkommt. Die Methode liefert in jedem Fall eine Knotenmenge als Rückgabewert und Sie wählen über den Index einen der Knoten aus. Ebenso sollten Sie auf den kleinen, aber feinen Unterschied zur Schreibweise der Methode `getElementById()` achten. Es heißt `getElementsByName()` (Mehrzahl) und nicht `getElementByName()` (Einzahl).

7.2.3 Zugriff über eine ID

Wenn Sie einem Element in einer Webseite eine Style Sheets-Klasse zuweisen, erfolgt das über das Attribut `class`. Über JavaScript kann man diese Klasse ansprechen und auch austauschen – zumindest im Internet Explorer, aber auch in einigen anderen neueren Browsern. Dazu ist zusätzlich das Attribut `id` (eindeutig) notwendig. Dessen Wert steht als Referenz auf das Objekt zur Verfügung, dem die Klasse zugeordnet ist. In einem Skript können Sie auf die Eigenschaft `className` des Objekts zugreifen und damit dann das Element formatieren. Wenn zum Beispiel ein `<span>`-Container wie folgt definiert ist:

Listing 7.8: Ein HTML-Tag mit class- und id-Parameter

```
<span class="still" id="P1">
```

Dann ist ein Zugriff so möglich:

Listing 7.9: Zugriff auf die Stilklasse per id

```
P1.className
```

Dahinter verbirgt sich ein indirekter Zugriff über `document.all`. Genau genommen müssten Sie also Folgendes notieren:

Listing 7.10: Zugriff auf die Stilklasse per id

```
document.all.P1.className
```

Achtung

Diese Art des Zugriffs funktioniert nicht in allen Browsern.

Zugriff über getElementById()

Der direkte Zugriff auf die ID eines HTML-Layoutelements ist durch die Kopplung an den Internet Explorer und einige wenige weitere Browser(-Versionen) nicht universell einsetzbar. Aber es gibt eine universelle Lösung, über die ID auf alle Elemente in Tag-Containern per JavaScript zuzugreifen. Allerdings wird dabei beim Zugriff über JavaScript nicht die Eigenschaft `className` des `all`-Objekts verwendet, sondern die Methode `getElementById()` des `document`-Objekts[2]. Dieser Methode wird als Parameter der Wert des ID-Attributs des entsprechenden Tags übergeben. Der Rückgabewert der Methode ist eine Referenz auf das entsprechende Element (ein Knoten des Elementbaums). Darüber können Sie dann auf alle Eigenschaften des Elements zugreifen und die Unterstützung ist in allen modernen Browsern gewährleistet. Mit dem folgenden Code macht man beispielsweise den mit der ID `a` gekennzeichneten Container unsichtbar:

Listing 7.11: Zugriff auf ein Element mit getElementById()

```
document.getElementById("a").style.visibility = "hidden"
```

7.2.4 Zugriff über den Typ des HTML-Elements

Das DOM-Konzept sieht auch vor, dass Sie auf Elemente einer Webseite nach dem Typ zugreifen können. Mit der Methode `getElementsByTagName()` des `document`-Objekts greifen Sie auf ein beliebiges Element in der DOM-Objektabbildung der Webseite zu, indem Sie als Parameter den Elementnamen angeben. Allerdings benötigen Sie auch einen Index, der die Position in dem Array angibt. Dieses ergibt sich sequenziell. Beispiele:

2 *Dies haben wir bisher schon oft gemacht.*

Listing 7.12: Zugriff über getElementsByTagName()

```
document.getElementsByTagName("h2")[0]
document.getElementsByTagName("p")[3]
document.getElementsByTagName("div")[0]
```

Hinweis

Die Methode `getElementsByTagName()` ist nicht Bestandteil des HTML-spezifischen DOM, sondern des allgemeinen Kern-DOM für beliebige XML- bzw. SGML-basierte Dokumente. Sie gehört also eigentlich zum Objekttyp `node` und liefert eine Knotenmenge im Sinne von XML, auf der Sie operieren.

Die verschiedenen Formen für den Zugriff auf Webseitenelemente demonstrieren die nachfolgenden Abschnitte mit praktischen Beispielen.

7.3 Manipulation von Bildobjekten

Ein sehr wichtiger Objekttyp bei DHTML-Effekten ist das Bildobjekt. Damit können Sie zum Beispiel eine Rückmeldung für einen Besucher einer Webseite aufbauen. Ein solche visuelle Anzeige wird insbesondere ein Schlüssel sein, um bei Datennachforderungen per AJAX dem Anwender das Laden von Daten deutlich zu machen (sofern es von der Benutzerführung her sinnvoll ist).

7.3.1 Image und images

Wenn ein Browser beim Laden einer Webseite eine Grafik vorfindet, legt er ein Datenfeld mit Namen `images` an, in dem die Grafik einsortiert wird. Dieses Datenfeld ist über `window.document.images` zugänglich. Ebenso ist es möglich, von Hand mit der Anweisung `new Image();` in JavaScript ein Bildobjekt zu erzeugen. Darüber kann beispielsweise per Skript bereits ein Bild geladen werden, das nicht sofort in der Webseite angezeigt wird. Jedes Bildobjekt stellt folgende Eigenschaften bereit:

Eigenschaften	Beschreibung
`border`	Die korrespondierende Eigenschaft zum gleichnamigen HTML-Parameter für den Rahmen um die Grafik. Die Eigenschaft wird nicht in allen Browsern unterstützt.
`complete`	Die Eigenschaft zeigt an, ob eine Grafik vollständig geladen ist (`true`) oder nicht (`false`).

Tabelle 7.1: Die Eigenschaften eines Bildobjekts

Eigenschaften	Beschreibung
`height`	Die korrespondierende Eigenschaft zum gleichnamigen HTML-Parameter für die Höhe der Grafik. Allerdings muss der HTML-Parameter nicht spezifiziert sein, damit diese Eigenschaft die Höhe beinhaltet. Dann ergibt sich der Wert aus der natürlichen Höhe des Bilds.
`hspace`	Der horizontale Abstand zwischen einer Grafik und ihren benachbarten Elementen. Korrespondierende Eigenschaft zum gleichnamigen HTML-Parameter. Wenn die Angabe im HTML-Tag nicht gesetzt ist, hat `hspace` den Wert 0.
`length`	Diese Eigenschaft ist eine Besonderheit, da sie nicht zu einem Bildobjekt, sondern zum Objektfeld gehört. Die Eigenschaft `document.images.length` enthält die Anzahl der Grafiken in der Webseite.
`lowsrc`	Der URL der Vorschaugrafik. Korrespondierende Eigenschaft zum gleichnamigen HTML-Parameter.
`name`	Der Name der Grafik. Korrespondierende Eigenschaft zum gleichnamigen HTML-Parameter.
`src`	Der URL der Grafikdatei. Korrespondierende Eigenschaft zum gleichnamigen HTML-Parameter.
`vspace`	Der vertikale Abstand zwischen einer Grafik und ihren benachbarten Elementen. Korrespondierende Eigenschaft zum gleichnamigen HTML-Parameter. Wenn die Angabe im HTML-Tag nicht gesetzt ist, hat `vspace` den Wert 0.
`width`	Die Breite der Grafik. Korrespondierende Eigenschaft zum gleichnamigen HTML-Parameter. Allerdings muss der HTML-Parameter nicht spezifiziert sein, damit diese Eigenschaft die Breite beinhaltet. Dann ergibt sich der Wert aus der natürlichen Breite des Bilds.

Tabelle 7.1: Die Eigenschaften eines Bildobjekts (Forts.)

Tipp

Grundsätzlich ist es beim dynamischen Verändern von Bildeigenschaften sinnvoll, mit Platzhaltern für Bilder zu arbeiten. Damit halten Sie sich Probleme mit älteren Browsern vom Hals, die mit rein dynamisch per JavaScript erzeugten Grafiken gelegentlich Schwierigkeiten haben. Das bedeutet, wenn Sie in einer Webseite eine Grafik anzeigen wollen, legen Sie auf jeden Fall einen `<img>`-Tag dafür an. Auch wenn dieser am Anfang nur eine unsichtbare Grafik referenziert. Eine solche unsichtbare Grafik kann eine vollkommen transparente Grafik sein oder eine Grafik, die nur ein Pixel groß ist. Allerdings werden moderne Browser diesen Trick kaum noch benötigen. Dort ist es sogar möglich, in eine bestehende Webseite beliebige Elemente einzufügen. Das erfolgt über das Objekt `node`.

Mit den entsprechenden Eigenschaften eines Bildobjekts können Sie nun die verschiedensten DHTML-Effekte rein aus JavaScript heraus programmieren. Sie können die Höhe und die Breite eines Bilds dynamisch verändern oder ein Bild dynamisch austauschen. Selbst Animationen sind damit möglich. Die nachfolgenden Beispiele zeigen dazu einige Effekte.

7.3.2 Anzeige des Ladevorgangs

Schauen wir uns zuerst ein vollständiges Beispiel an, in dem der Besucher mittels einer Grafik über einen Ladevorgang bei einer AJAX-Anfrage informiert wird. Sie sollten in der Praxis zwar genau überlegen, ob und wann Sie dem Anwender so eine Rückmeldung geben. Aber es gibt diverse Situationen, in denen das notwendig sein kann. Hier ist die HTML-Seite (*image1.html*):

Listing 7.13: Die Webseite mit dem Nachladen der Informationen

```
01 <html>
02 <body>
03 <h1>Drücken Sie eine Taste!</h1>
04 <span id="antwort"></span>
05 <img src="leer.gif" height="50"
06    style="position:absolute;top:100px;left:500px;" />
07 </body>
08 <script language="JavaScript" src="image1.js"></script>
09 <script language="JavaScript">
10   window.onkeypress = sndReq;
11 </script>
12 </html>
```

Beachten Sie, dass sowohl die Einbindung der JavaScript-Datei als auch die Referenz auf `sndReq()` hinter dem `<body>`-Tag stehen. Das ist leider wegen der Reaktion einiger Browser notwendig, denn wir greifen in der Folge auf `document.images[0]` zu. Das Objekt wird in manchen Browsern erst angelegt, wenn der Browser den kompletten `<body>`-Bereich geparst hat. Wird vorher bereits in einem Skript darauf verwiesen, kommt ein Fehler.

Gegenüber den bisherigen Beispielen ist nur die Grafik neu, die über die Zeilen 9 und 10 in der Webseite platziert wird. Sie können dort am Anfang zum Beispiel eine leere, transparente Grafik platzieren. In Zeile 10 sehen Sie, dass in dem Beispiel die Grafik mittels eines internen Style Sheet auf eine feste Position in der Webseite geschoben wird, in unserem Fall auf 100 Pixel von oben und 500 Pixel vom linken Rand aus gesehen.

Achtung

Beachten Sie wieder, dass die Funktion `sndReq()` als Funktionsreferenz an den Eventhandler `onkeypress` des `window`-Objekts gebunden wird. Damit kommen – wie schon mehrfach erwähnt – einige Browser nicht zurecht. Für diese veralteten Browser können Sie z. B. einen HTML-Eventhandler im `<body>`-Tag verwenden.

Zugriff über Objektfelder

Auf JavaScript-Seite (Datei *image1.js*) werden wir nun mit einem rekursiven Aufruf einer Funktion arbeiten. So eine rekursive Vorgehensweise wird bei vielen Fortschrittsanzeigen der Schlüssel zum Erfolg sein. In dieser Funktion nutzen wir einen Selbstaufruf so lange, bis der Ladezustand des AJAX-Aufrufs `COMPLETED` aufweist. Solange zeigen wir in der Webseite ein animiertes GIF an. Dies macht dem Besucher deutlich, dass aktuell Daten geladen werden.

Listing 7.14: Die externe JavaScript-Datei

```
22 function sndReq() {
23     resObjekt.open('get', 'image1.jsp',true);
24     resObjekt.onreadystatechange = nachladen;
25     resObjekt.send(null);
26 }
27 function nachladen() {
28  if(resObjekt.readyState != 4){
29    document.images[0].src="laden.gif";
30 setTimeout('nachladen()','2');  }
31 else {
32 document.images[0].src="leer.gif";
33 handleResponse();  }
34 }
35 function handleResponse() {
36   if(resObjekt.readyState == 4) {
37     document.getElementById("antwort").innerHTML = resObjekt.responseText;
38   }
39 }
```

In der Funktion `sndReq()` wird die rekursive Funktion `nachladen()` als Eventhandler registriert (Zeile 24 – `resObjekt.onreadystatechange = nachladen;`). In dieser Funktion überwachen wir in Zeile 28 den Status der Ladeoperation (`if(resObjekt.readyState != 4)`). Ist dieser noch nicht `COMPLETED`, wird in Zeile 29 über das Objektfeld das erste Bild der Webseite durch ein animiertes GIF ausgetauscht (`document.images[0].src="laden.gif";`) und in Zeile 30 die Funktion selbst wieder aufgerufen. Ist der Ladezustand `COMPLETED` erreicht, wird der Selbstaufruf abgebrochen und das Originalbild wieder angezeigt (Zeile 32 – `document.images[0].src="leer.gif";`). Danach erfolgt der

Aufruf der Funktion `handleResponse()`, über die wie üblich die Antwort des Servers in die Webseite befördert wird.

Hinweis

Das Skript, das für dieses Beispiel die Antwort generiert, ist für unseren konkreten Fall vollkommen uninteressant. Es muss nur lang genug arbeiten, damit eine Verzögerung bei der Antwort erreicht werden kann (was in der Praxis aber natürlich ausdrücklich nicht gewünscht ist). Eine Möglichkeit, für den Test des Beispiels ein solches verzögertes Antwortverhalten zu erreichen, ist ein etwas zeitaufwändigerer mathematischer Algorithmus, etwa die Multiplikation aller Zahlen von 0 bis 99.000.

Abbildung 7.1: Eine Eieruhr symbolisiert das Nachladen von Daten.

Tipp

Es ist natürlich auch leicht möglich, statt einer einfachen Eieruhr in der Grafik eine komplexere Information unterzubringen, etwa eine Fortschrittsanzeige, wobei dies aufgrund des unvorhersehbaren Datendurchsatzes bei einer Übertragung via Internet viel »Kaffeesatzleserei« darstellt. Aber ebenso können Sie auch die ersten Teile der Antwort des Servers so gestalten, dass dieser »Bitte warten« oder so sendet. Sie haben ja beim Ladezustand `INTERACTIVE` die ersten Daten vom Server bereits zur Verfügung und können mit den Eigenschaften `responseText` bzw. `responseXML`[3] darauf zugreifen (sofern ein Browser das auch unterstützt – das ist wie gesagt derzeit noch nicht einheitlich gewährleistet). Die erste Zeile könnte zum Beispiel in einem speziellen `<span>`-Element angezeigt werden (mit `innerHTML`), wie es auch für die tatsächliche Antwort verwendet wird. Noch einfacher wäre es

3 *Dazu siehe Seite 194.*

hier aber, einen festen Text im Skript unterzubringen und mit `innerHTML` an einer bestimmten Stelle anzuzeigen. Rein optisch ist jedoch eine animierte Grafik sicher eine sehr guteLösung.

Hinweis

Das Prinzip der Überwachung des Status basiert auf Listenern, wie man sie auch in Java findet. Man kann sich das wie einen Radioempfänger vorstellen, der auf einen bestimmten Kanal eingestellt ist. Sobald ein entsprechendes Signal auf diesem Kanal kommt, kann man darauf reagieren.

Nun ist es aber so, dass man diesen Ratioempfänger genau einmal eingeschaltet und dann auf ein ganz bestimmtes Signal wartet (z. B. die Beendigung der Übertragung). Wenn Sie während dieser Zeit allerdings gezielt bestimmte Maßnahmen ausführen wollen (Anzeige eines Fortschrittsbalkens, der mit JavaScript erzeugt wird oder Mitzählen der Millisekunden oder Ähnliches), müssen Sie in der Zeit weitere JavaScript-Aktionen ausführen. Deshalb bietet sich hier ein rekursiver Aufruf an, während Sie auf die Antwort warten. Über den Status der Datenübertragung kontrollieren Sie, wie lange der Selbstaufruf laufen soll. Das Beispiel kann selbstverständlich auch für viele andere Dinge eingesetzt werden und es ist nicht darauf beschränkt, die Verzögerung beim Nachladen der Daten anzuzeigen.

Zugriff über Namen

Zum Umformulieren des Beispiels zur Fortschrittsanzeige des Ladevorgangs von oben müssen in der externen JavaScript-Datei bloß die Zeilen 29 und 32 wie folgt ausgetauscht werden, wenn Sie über den Namen zugreifen wollen:

Listing 7.15: Direkter Zugriff über einen Namen

```
29 document.zustand.src="laden.gif";
...
32  document.zustand.src="leer.gif";
```

Alternativ geht auch das:

Listing 7.16: Zugriff über einen Namen über getElementsByName()

```
29  document.getElementsByName("zustand")[0].src="laden.gif";
...
32  document.getElementsByName("zustand")[0].src="leer.gif";
```

7.4 Verändern von Stilinformationen

Die Veränderung von Stilinformationen für Teile einer Webseite aus JavaScript heraus ermöglicht einen weiteren Weg, um Besuchern das Nachladen von Informationen zu verdeutlichen oder aber auch Informationen elegant in eine Webseite zu platzieren.

7.4.1 Anzeige des Ladevorgangs mit Zugriff über eine ID und die Eigenschaft className

Schauen wir uns für eine weitere Variante einer Anzeige des Ladevorgangs den Zugriff auf die Stilinformationen mit der ID und `className` an. In dieser Version wird der Besucher mit einer veränderten Style Sheet-Klasse über den aktuellen Ladevorgang informiert. Hier ist die HTML-Seite (*classname1.html*):

Listing 7.17: Die Webseite

```
01 <html>
02 <link rel="stylesheet" href="stil.css" type="text/css">
03 <script language="JavaScript" src="classname.js"></script>
04 <body onClick="sndReq()">
05 <h1>Klicken Sie auf die Seite!</h1>
06 <span id="antwort"></span>
07 <span class="stil1" id="P1">
08   Bitte Warten - Es werden Daten nachgeladen</span>
09 </body>
10 </html>
```

In Zeile 2 wird mit `<link rel="stylesheet" href="stil.css" type="text/css">` eine externe CSS-Datei referenziert, die wir uns gleich ansehen wollen. In Zeile 4 finden Sie dieses Mal beim `<body>`-Tag einen HTML-Eventhandler. Bei einem Klick auf die Webseite werden Daten nachgeladen (`onClick="sndReq()"`>). Das Beispiel verwendet eine Stilklasse `stil1` für den `<span>`-Container in Zeile 7 und 8. Dem Container wird über den `class`-Parameter diese Klasse als Defaultwert zugewiesen. Der Container besitzt zusätzlich eine ID `P1`. Schauen wir uns nun die externe CSS-Datei an:

Listing 7.18: Die externe CSS-Datei

```
01 .stil1 {
02   color : white;
03   background-color : white;
04   position : absolute;
05   top : 100px;
06   left : 500px;
07   }
08 .stil2 {
```

```
09   color : white;
10   background-color : blue;
11   position : absolute;
12   top : 100px;
13   left : 500px;
14  }
```

Die CSS-Datei definiert zwei Klassen. Die Default-Klasse `stil1` positioniert das Element, dem die Klasse zugewiesen wird, und hisst dabei die Ostfriesische Kriegsflagge[4]. Damit ist der Text in dem -Container so lange unsichtbar, wie die Klasse `stil1` zugewiesen ist. Die Klasse `stil2` wird den Text sichtbar machen.

In der externen JavaScript-Datei ist nur folgende Stelle wieder interessant:

Listing 7.19: Die interessanten Änderungen in der externen JavaScript-Datei

```
27 function nachladen() {
28  if(resObjekt.readyState != 4){
29    P1.className="stil2";
30 setTimeout('nachladen()','2'); }
31 else {
32 P1.className="stil1";
33 handleResponse(); }
```

In der Funktion `sndReq()` wird in Abhängigkeit vom Status der Ladeoperation die Stilklasse des -Containers ausgetauscht (Zeile 29: `P1.className="stil2";`) und in Zeile 30 die Funktion selbst wieder aufgerufen. Ist der Ladezustand `COMPLETED` erreicht, wird der Selbstaufruf abgebrochen und die Originalstilklasse wieder zugewiesen (Zeile 32 – `P1.className="stil1";`). Danach erfolgt erneut der Aufruf der Funktion `handleResponse()`.

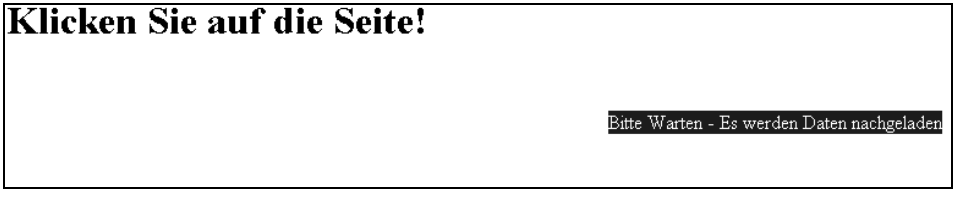


Abbildung 7.2: Die Stilklasse wird zur Laufzeit dynamisch ausgetauscht.

4 *Weißer Adler auf weißem Grund ;-). Hier ist es weiße Schrift auf weißem Hintergrund.*

Achtung

Diese Art des Zugriffs funktioniert im Grunde nur im Internet Explorer. Sie ergibt als allgemeine Lösung keinen Sinn. Dazu sollten Sie im allgemeinen Fall `getElementById()` oder `getElementsByName()` verwenden.

Tipp

Die Effekte, die Sie mit dynamischem Verändern der Stilklasse erreichen können, sind vielfältig. Sie können zum Beispiel Daten bereits im Offscreen-Bereich[5] einer Webseite halten (etwa mit `position : absolute; top : 0px; left : -500px`) und dann durch Zuweisung einer Klasse mit Positionsangaben im sichtbaren Bereich ohne Nachladen anzeigen – oder animierte CSS-Effekte wie Blinken zuweisen. Selbst Animationen sind möglich. Was auch immer Sie machen – der Trick beruht auf dem Austausch der Stilklassen oder aber den Grafiken.

7.5 Stilveränderungen über das style-Objekt

Ein sehr zuverlässiger Weg, um aus JavaScript auf die Formatierungen von Layoutelementen der Webseite zuzugreifen, führt über das `style`-Objekt. Die Layoutelemente aller wichtigen modernen Browser stellen dieses Objekt als Eigenschaft zur Verfügung. Das Objekt selbst stellt die üblichen CSS-Formatierungseigenschaften als Eigenschaften bereit.

7.5.1 Temporär Informationen unsichtbar machen

Mit Style Sheets können Sie Teile einer Webseite temporär unsichtbar oder sichtbar machen. Diese Möglichkeit sowie die Vorhaltung im Offscreen-Bereich kann man hervorragend dazu einsetzen, um das Nachladen von Informationen deutlich zu machen oder aber unabhängig vom physikalischen Laden von Informationen hochperformant Daten in einer Webseite anzuzeigen. Das schließt nicht aus, dass zusätzlich mit asynchroner Datennachforderung gearbeitet wird. Es ist nur ein zusätzliches Mittel und die jeweils beste Möglichkeit zu nutzen.

5 *Der Offscreen-Bereich hat negative Koordinaten bezüglich oben oder links (bei zu großen positiven Werten nach unten und rechts werden unerwünschte Bildlaufleisten angezeigt). Damit wird darin enthaltener Inhalt nicht angezeigt, solange der Inhalt nicht so groß ist, dass er wieder in den sichtbaren Bereich der Webseite hineinreicht. Sie können den Offscreen-Bereich verwenden wie den normalen sichtbaren Bereich der Webseite.*

Die nachfolgende CSS-Datei spezifiziert eine Klasse mit positioniertem, aber unsichtbarem Inhalt:

Listing 7.20: Eine CSS-Datei mit einer Klasse

```
01 .stil1 {
02   color : white;
03   background-color : blue;
04   visibility : hidden;
05   position : absolute;
06   top : 100px;
07   left : 500px;
08   }
```

Die HTML-Datei ist mittlerweile gewohnte Hausmannskost:

Listing 7.21: Der ‹span›-Container in Zeile 7 ist zuerst unsichtbar.

```
01 <html>
02 <link rel="stylesheet" href="stil1.css" type="text/css">
03 <script language="JavaScript" src="stil1.js"></script>
04 <body  onClick="sndReq()">
05 <h1>Klicken Sie auf die Seite!</h1>
06 <span id="antwort"></span>
07 <span class="stil1" id="zustand">Bitte Warten - Es werden Daten nachgeladen</span>
08 </body>
09 </html>
```

Mit dem folgenden Code macht man beispielsweise den mit der ID `zustand` gekennzeichneten Container temporär (während des Ladens von Daten) sichtbar und anschließend wieder unsichtbar. Wir verwenden hier `getElementById()`, um auf den `<span>`-Container zuzugreifen:

Listing 7.22: Zugriff auf ein Element mit getElementById()

```
27 function nachladen() {
28 if(resObjekt.readyState != 4){
29   document.getElementById("zustand").style.visibility = "visible";
30  setTimeout('nachladen()','2'); }
31 else {
32   document.getElementById("zustand").style.visibility = "hidden";
33 handleResponse(); }
34 }
```

7.6 Das Einfügen von Informationen in eine Webseite

Die bisher in diesem Kapitel gezeigten DHTML-Effekte dienten vor allem dazu, einem Anwender bestimmte Verhaltensweisen der Applikation (Nachladen) zu verdeutlichen. Aber auch das Hinzufügen von neuen Informationen in die Webseite fällt unter DHTML.

7.6.1 Zugriff auf Inhalte von Elementen in der Webseite

Wir haben ja auch bisher schon mit `innerHTML` gearbeitet. Als Alternative steht `innerText` zur Verfügung. Beide Eigenschaften gehören zum Objekt `document.all` und damit nicht zum offiziellen JavaScript-Sprachstandard. Dennoch funktioniert der Zugriff über `innerHTML` zumindest in allen neuen Browser-Generationen. Der Unterschied zwischen `innerHTML` und `innerText` ist recht gering. Sowohl über `innerHTML` als auch `innerText` haben Sie Zugang zum Inhalt eines HTML-Elements. Wenn Sie beim dynamischen Ändern des gespeicherten Inhalts bei `innerHTML` jedoch HTML-Tags notieren, werden diese bei der Aktualisierung des Elementinhalts interpretiert. Das ist offiziell bei `innerText` nicht der Fall. Allerdings wirkt sich das bei Datenanforderungen mit AJAX nicht aus. Wenn Sie HTML-Tags senden, werden sie bei beiden Eigenschaften interpretiert.

Die ebenfalls verfügbare Eigenschaft `outerText` gibt Ihnen Zugang zum gleichen Wert wie `innerText`. Nur werden beim Ändern umgebende HTML-Tags entfernt und durch Text ersetzt. Als weitere Alternativen zum Inhalt eines Elements gibt es `outerHTML`. Mit `outerHTML` erhalten Sie Zugang zum Inhalt eines HTML-Tags samt der umgebenden Tags mit allen Angaben. Bei beiden Eigenschaften, aber auch `innerText`, gibt es aber in einigen Browsern Probleme. Wenn nichts dagegen spricht, genügt `innerHTML` zum Aktualisieren von Inhalt mit AJAX.

Achtung

Alle hier genannten Eigenschaften sollten nicht direkt beim Laden der HTML-Datei zum Einsatz kommen, sondern erst nach dem vollständigen Laden der Seite. Beim Einsatz während des Ladevorgangs melden einige Browser Fehler.

7.6.2 Nutzen des node-Objekts zum Datenaustausch

In den meisten Fällen kommen Sie mit den bisher beschriebenen Eigenschaften und Methoden aus, wenn Sie Teile einer Webseite austauschen wollen. Allerdings stellt das `node`-Objekt weitere Methoden bereit, die an der XML-Denkweise orientiert sind. Sie können in einer Webseite sowohl die bisher beschriebenen HTML-Objektmetho-

den und -eigenschaften als auch `node` zuzuordnende Techniken verwenden. Letzteres wird insbesondere dann Nutzen bringen, wenn vom Server XML-Daten geschickt werden. Um nun die Eigenschaften und Methoden des `node`-Objekts nutzen zu können, benötigen Sie einen Knoten. Diesen liefern die behandelten Methoden des `document`-Objekts (`getElementById()` und `getElementsByName()` sowie `getElementsByTagName()`, was es sowieso noch einmal beim `node`-Objekt gibt). Ausgehend davon können Sie die Attributknoten, Textknoten und weitere Element-Kindknoten eines Knotens ansprechen. Dazu nutzen Sie die spezifischen Eigenschaften und Methoden eines `node`-Objekts, die insbesondere über die XPath-Syntax genauer spezifiziert werden. Die Details führen ohne XML zu weit, aber wir werden im folgenden Kapitel zu XML Hintergründe klären.

7.6.3 responseText und responseXML

Die neuen Inhalte von Elementen in der Webseite werden aus der Antwort des `XMLHttpRequest`-Objekts entnommen. Wenn Sie sich unsere bisherigen Beispiele ansehen, werden Sie beim Auswerten des `XMLHttpRequest`-Objekts immer die Eigenschaft `responseText` gesehen haben. Alternativ gibt es noch `responseXML`. Und wie der Name eindeutig suggeriert, ist diese Eigenschaft dann interessant, wenn Sie Daten im XML-Format vom Server geschickt bekommen. Oder wenn Sie im Client auf den gesendeten Informationen baumorientiert navigieren wollen (das können Sie auch dann, wenn der Server Ihnen HTML geschickt hat). Wir werden auf diese Eigenschaft zurückgreifen, wenn wie XML-Daten verwerten (im nächsten Kapitel). Und dann kommt auch das `node`-Objekt zum Einsatz. Allerdings soll schon jetzt die Erwartung gedämpft werden, denn die Unterstützung von `responesXML` ist in vielen Browsern noch recht mangelhaft implementiert.

7.7 Zusammenfassung

Sie haben in diesem Kapitel zentrale DHTML-Techniken kennen gelernt. Mit diesen Techniken können Sie dynamisch auf Teile der Webseite zugreifen, nachdem diese bereits in den Browser geladen wurden, und die Webseite ohne Neuladen verändern. Damit lassen sich sowohl Effekte zur Rückmeldung an den Besucher bei AJAX-Datentransfers realisieren als auch die neuen Informationen in die Webseite einfügen.

8 XML für AJAX-Entwickler

Wie Ihnen die Abkürzung AJAX schon sagt, stellen XML sowie damit verbundene Techniken wesentliche Bestandteile des Umfelds dar. Auch diese Verfahren müssen Sie als AJAX-Entwickler – zumindest in Grundzügen – beherrschen. Dabei kann XML auf dem Server zum Bereitstellen einer Antwort eine Rolle spielen, aber natürlich müssen Sie dann auch auf Seiten des Clients damit umgehen können. Bisher haben wir zwar auf den Einsatz von XML verzichtet und die Antwort des Servers auf eine AJAX-Anfrage als reinen Text bzw. HTML zurückgeschickt. In vielen AJAX-Applikationen wird dies sogar genügen. Aber Sie werden viel anspruchsvollere Applikationen erstellen können, wenn Sie nun die Antwort des Servers auf das XML-Format erweitern. Die Strukturierung von Information lässt Ihnen weit mehr Möglichkeiten, um Geschäftsprozesse bereits auf dem Server ablaufen zu lassen und dem Client speziell aufbereitete Ergebnisse zu übermitteln, die dieser nur noch anzeigen muss. Aber nicht nur die Prozesslogik auf dem Server kann viel granularer erstellt werden. Ebenso kann auf dem Client Logik bereitgestellt werden, da Sie viel Metainformation in das Übertragungsformat einbinden können. Da für die Zielgruppe des Buchs keine XML-Kenntnisse vorausgesetzt werden, besprechen wir in diesem Kapitel die elementaren Grundlagen. Dies umfasst XML selbst sowie einen kurzen Einblick in die Validierung mit DTDs/Schemata. Und einen ganz wesentlichen Part wird XPath spielen, denn dessen Konzepte lassen sich auf dem Client zur Navigation in dem Dokument nutzen. Und sogar neue Elemente lassen sich in einer bestehenden Webseite erzeugen, wenn Sie Elemente darin über XPath-Ausdrücke ansprechen. Dazu kommen wir noch einmal auf DTHML zurück, denn der Umgang mit dem DOM-Objekt `node` ist explizit an XML respektive XPath gekoppelt. Und am Ende des Kapitels wollen wir ganz kurz auf die XML-Verarbeitung mit Java eingehen, denn für unsere serverseitige Generierung einer Antwort mittels JSP oder Servlets ist dies eine ziemlich wichtige Grundvoraussetzung.

8.1 XML-Grundlagen

XML ist die Abkürzung für **eXtensible Markup Language.** Dies beschreibt einen plattformneutralen Klartextstandard auf Unicode-Basis zur Erstellung maschinen- und menschenlesbarer Dokumente, um darüber beliebige Informationen auszutauschen. XML-Dokumente liegen in Form einer Baumstruktur vor, die eine Navigation zu den einzelnen Zweigen des Baums gestattet. Dabei ist XML wie HTML eine **Aus-**

zeichnungssprache (**Markup Language**), um über die Textinformation hinaus eine Struktur der Information zu bieten. Die in einem Dokument enthaltenen Informationen werden dazu durch Tags strukturiert, die sowohl in HTML als auch in XML in spitzen Klammern notiert werden. Die Elemente in XML sind im Gegensatz zu HTML aber nicht vorgegeben. Es gibt keinen vorgefertigten, beschränkten Sprachschatz an Elementen. Mit anderen Worten – es gibt keine vorgegebenen XML-Tags in dem Sinne, wie die meisten Anwender HTML-Tags kennen, also Tags wie `<br>` oder `<i>` mit einer festen Bedeutung. Im Gegensatz zu HTML, das auch über eine solche Semantik verfügt und neben jedem Tag auch dessen Bedeutung beschreibt, besteht XML lediglich aus einer Beschreibung der Syntax für Elemente und Strukturen. Damit ist XML freilich beliebig erweiterbar. Die XML-Spezifikation beschreibt lediglich, nach welchen Regeln Sie Tags zu definieren haben. Die Festlegung von Tags machen Sie selbst[1]. Daher kann ein Leser oder Parser (zum Beispiel ein Webbrowser) die Bedeutung eines Elements aber auch nicht aus dem Kontext erschließen. Für einen konkreten Anwendungsfall (die Interpretation eines XML-Dokuments) müssen sämtliche relevanten Details spezifiziert werden. Dies betrifft insbesondere die Festlegung der Strukturelemente und ihre Anordnung innerhalb des Dokumentenbaums[2].

Im Gegensatz zu HTML ist XML syntaktisch eine sehr strenge Sprache, bei der es kein Prinzip der Fehlertoleranz gibt. Die XML-Spezifikation ist streng formal und lässt keine Ausnahmen und unklaren Strukturen zu. Dadurch ist XML jedoch einfach und automatisiert zu validieren. XML beschreibt nur wenige, einfache, aber eben sehr strenge und absolut eindeutige Regeln, nach denen ein Dokument zusammengesetzt sein kann.

Auch die Zielrichtung von XML und HTML sind unterschiedlich. HTML ist auf das Design von Webseiten hin optimiert, während ein zentraler Aspekt von XML die Trennung von Daten und ihrer Darstellung ist. XML dient nur zum Strukturieren von Daten (was eine optische Aufbereitung nicht ausschließt) und ist datenzentriert. Die Anordnung und Reihenfolge von Elementen in einem Dokument sind sekundär. Dennoch ist die Struktur eines XML-Dokuments streng hierarchisch und damit ist der Kontext jedes Elements im Dokument eindeutig festgelegt.

Hinweis

Im Umfeld von XML tauchen eine ganze Reihe von Abkürzungen und Bezeichnungen auf, die oftmals verwirren. Meist bezeichnen diese Begriffe Sprachen, die mit XML generiert wurden, darauf aufbauen oder als Alternativen oder Ergänzungen das Umfeld bereichern. *Im Anhang finden Sie dazu einige Erläuterungen*.

1 *Das bedeutet aber nicht, dass Sie in XML nicht Elemente definieren können, die denen von HTML entsprechen. Im Gegenteil – genau das werden Sie im Fall von AJAX wahrscheinlich öfter machen, um bei der Interpretation im Client ein Verhalten wie bei HTML-Elementen zu gewährleisten.*

2 *XML kann auf diese Weise zur Definition beliebiger eigener Sprachen verwendet werden. Es ist eine so genannte **Metasprache** zur Definition von beliebigen, in ihrer Grundstruktur jedoch stark verwandten Auszeichnungssprachen.*

Tipp

Zur Erstellung von XML-Dokumenten gibt es zahlreiche Tools (XML-Editoren). Im Grunde genügt aber ein Klartexteditor zum Schreiben von XML-Dokumenten. Für die strukturelle Anzeige von XML-Dokumenten können Sie jeden neueren Webbrowser verwenden. Die meisten modernen Browser stellen die Struktur einer XML-Datei in Form eines Baums dar. Dabei erfolgt die Darstellung der verschiedenen Ebenen im Elementbaum entweder kollabiert (mit einem +-Zeichen, das dem Element vorangestellt ist) oder expandiert (mit einem -Zeichen). Das Plus- und das Minuszeichen sind sensitiv. Das bedeutet, wenn Sie es anklicken, wird der Zustand umgeschaltet. Ein kollabiertes Element wird expandiert und ein expandiertes Element kollabiert.

Abbildung 8.1: Ein XML-Baum ist vollständig kollabiert – es wird nur ein Element angezeigt.

Sofern eine Style Sheet-Datei mit dem XML-Dokument verknüpft ist, kann auch eine andere Darstellung der XML-Datei erfolgen. Dann wird von den meisten Browsern keine Baumdarstellung mehr gewählt. Das sehen Sie etwas weiter unten bei PIs.

8.2 XML-Elemente

So genannte **Komponenten** bilden die Bausteine eines XML-Dokuments. Die Grundstruktur eines XML-Dokuments besteht dabei aus **Elementen**, die – sofern sie nicht eingeschränkt werden – selbst Unterelemente enthalten können und die wichtigste Form von Komponenten darstellen. Elemente selbst sind so aufgebaut, wie man es im Wesentlichen von HTML kennt. Es gibt einen Anfangs-Tag, der beispielsweise so aussieht:

Listing 8.1: Ein Anfangs-Tag eines XML-Elements

```
<tag>
```

Der Anfangs-Tag muss in XML **immer** mit einem Ende-Tag abgeschlossen werden, der hinter einem Slash den Bezeichner wiederholt – es sei denn, ein Tag wird als leeres Element gekennzeichnet. Eventuell im Anfangs-Tag notierte Attribute werden auf keinen Fall im Ende-Tag wiederholt. In unserem Fall sieht der Ende-Tag so aus:

Listing 8.2: Der Ende-Tag zum vorher definierten Anfangs-Tag eines XML-Elements

```
</tag>
```

Im Inneren des Elements kann – sofern nicht besondere Regeln vorgegeben sind – beliebiger Inhalt notiert werden. Das können weitere Elemente oder Text sein. Auch gemischte Formen sind erlaubt.

8.2.1 Leere Elemente

XML erlaubt die Deklaration von **leeren Elementen**. Diese werden meist in Verbindung mit Attributen verwendet. Ein leeres Element wird wie folgt gekennzeichnet:

Listing 8.3: Ein leeres XML-Element

```
<tag />
```

Zwischen dem Slash und dem Bezeichner darf ein Leerzeichen notiert werden. Diese Schreibweise wird sogar vom W3C empfohlen. Alternativ geht auch diese Form:

Listing 8.4: Eine alternative Schreibweise für ein leeres XML-Element

```
<tag></tag>
```

Dabei darf nicht einmal ein Leerzeichen in dem Container notiert werden. Leerraum wird in XML als echter Inhalt angesehen.

8.2.2 Elementbezeichner

Der Name eines Elements muss mit einem Buchstaben, dem Unterstrich oder einem Doppelpunkt[3] beginnen und darf danach aus beliebigen Buchstaben, Zahlen, Punkten, Doppelpunkten, Bindestrichen und Unterstrichen bestehen. Ein Bezeichner darf jedoch nicht mit `xml` beginnen. Dieser Token hat in XML eine spezifische Bedeutung.

3 *Der Doppelpunkt als erstes Zeichen eines Elements macht in einigen XML-Tools Probleme, weil dieser Token alternativ auch zur Trennung von so genannten Namensräumen verwendet wird. Die Probleme sind aber auf Programmierfehler in einigen Tools zurückzuführen. Dennoch vermeiden Sie besser Bezeichner, die mit einem Doppelpunkt beginnen.*

8.3 Die Syntax eines XML-Dokuments

Ein XML-Dokument muss wenige, aber strenge syntaktische Regeln einhalten. Wenn ein Dokument diese Regeln einhält, nennt man es **wohlgeformt**! Dieser Begriff ist zentral für ein XML-Dokument. Entweder ein Dokument ist wohlgeformt oder aber es ist kein XML-Dokument. Schauen wir uns die Regeln an, die ein XML-Dokument einhalten muss. Die Nummerierung der Regeln ist dabei willkürlich.

8.3.1 Regel 1: Unicode

XML-Dokumente bestehen aus Unicode-Zeichen, wobei Sie eine spezielle Kodierung (encoding) angeben können. Allerdings können Sie bei der Erstellung Ihres XML-Dokuments in einem Editor ANSI-Code speichern. Dann interpretiert der Parser das als Unicode.

8.3.2 Regel 2: XML ist casesensitive

In XML wird grundsätzlich zwischen Groß- und Kleinschreibung streng unterschieden. Element- und Attributnamen werden in XML in der Regel klein geschrieben, aber das ist nicht zwingend.

8.3.3 Regel 3: Ein Prolog ist zwingend

Ein XML-Dokument beginnt **immer** mit einem so genannten **Prolog**. Der Prolog muss am Beginn eines XML-Dokuments stehen und sieht derzeit in der einfachsten Form so aus:

Listing 8.5: Die einfachste Form eines XML-Prologs

```
<?xml version="1.0" ?>
```

Diese Syntax bezeichnet eine **XML-Deklaration**, die mit dem reservierten Token `version` die XML-Version angibt und zwingend mit dem ersten Zeichen eines XML-Dokuments zu beginnen hat (nicht einmal ein Leerzeichen zur Einrückung ist gestattet). Derzeit ist `"1.0"` die einzig mögliche Wertzuweisung. Dennoch ist die Versionsangabe für die Wohlgeformtheit eines XML-Dokuments verpflichtend. Das ist auch in Hinsicht auf zukünftige Entwicklungen unabdingbar, um zu gewährleisten, dass ältere XML-Dokumente bei der Einführung neuer Varianten grundsätzlich gekennzeichnet sind. Optional kann die XML-Deklaration um Angaben wie die Textkodierung erweitert werden. Wenn diese Angabe gemacht wird, muss sie zwingend als zweites Attribut der XML-Deklaration folgen. Das Attribut heißt `encoding` und erhält als Wertzuweisung eine der international genormten Kennungen für einen Zeichensatz. Die nachfolgende Tabelle zeigt einige Beispiele:

Textkodierung	Zeichensatz
`encoding="ISO-8859-1"`	Der Zeichensatz für Westeuropa
`encoding="UTF-16"`	Unicode mit 16 Bit Zeichenbreite
`encoding="UTF-8"`	Unicode mit 8 Bit Zeichenbreite. Dieser Zeichensatz wird in der Regel als Vorgabe verwendet, wenn das Attribut nicht angegeben wird.

Tabelle 8.1: Beispiele für die Angabe von Textkodierungen

Eine weitere Angabe in der XML-Deklaration legt fest, ob eine externe Grammatik (ein Schema oder eine DTD) verwendet werden soll (`standalone="no"`) oder nicht (`standalone="yes"`). Dazu kann ein Prolog um PIs (Processing Instructions) ergänzt werden. Auf beide Punkte gehen wir gleich noch ein.

8.3.4 Regel 4: Eindeutigkeit des Wurzelelements

Ein XML-Dokument darf nur genau ein **Wurzelelement** (auch **Root-Tag** oder **Dokumentelement** genannt) besitzen, das alle andere Elemente umschließt[4]. So ein Wurzelelement muss auch auf jeden Fall vorhanden sein und folgt, abgesehen von Kommentaren, unmittelbar dem Prolog.

8.3.5 Regel 5: Saubere Verschachtelung aller Elemente

Elemente müssen sauber verschachtelt werden. Das bedeutet, zwei öffnende und einander verschachtelnde Tags müssen in umgekehrter Reihenfolge geschlossen werden.

8.3.6 Regel 6: Jedes Element muss einen Ende-Tag haben oder als leeres Element notiert werden

In XML muss jeder Tag einen Ende-Tag haben oder aber als leeres Element notiert werden. Ein alleinstehender Tag wie `<br>` unter HTML ist nicht erlaubt.

8.3.7 Regel 7: Attributen muss immer ein Wert zugewiesen werden

In XML können Sie wie in HTML Elemente mit Attributen genauer spezifizieren. Allerdings muss Attributen in XML **immer** ein Wert zugewiesen werden. Und der Attributwert muss zwingend in Anführungszeichen stehen. Ein XML-Element kann auch mehrere Attribute besitzen, die in beliebiger Reihenfolge zum Anfangs-Tag hinzugefügt und einfach durch Leerzeichen getrennt werden. Beispiel:

4 *Bei einer HTML-Datei ist das Wurzelelement* `<html>`.

Listing 8.6: Ein XML-Element mit mehreren Attributen

```
<name titel="Dr" anrede="Herr">
```

Hinweis

Es gibt in XML einige reservierte Attribute mit fester Bedeutung. Diese beginnen alle mit dem Token `xml`, gefolgt von einem Doppelpunkt.

Eine wohlgeformte XML-Datei könnte also so aussehen:

Listing 8.7: Eine wohlgeformte XML-Datei

```
01 <?xml version="1.0"?>
02 <musik>
03  <rock>
04   <titel interpret="Deep Purple">Smoke on the water</titel>
05   <titel interpret="Led Zep">Stairway to heaven</titel>
06   <titel interpret="Safety First">History</titel>
07  </rock>
08  <pop>
09   <titel interpret="Mandona">Like a virgin</titel>
10  </pop>
11  <reggae>
12   <titel interpret="Bob Marley">No woman no cry</titel>
13  </reggae>
14 </musik>
```

8.4 Weitere Komponenten von XML

Neben Elementen gibt es in XML weitere Komponenten, die die Struktur eines XML-Dokuments zusätzlich bestimmen.

8.4.1 Kommentare

Kommentare bezeichnen Abschnitte in einem XML-Dokument, die vom Parser ignoriert werden. Sie werden wie in HTML mit der Zeichenfolge `<!--` eingeleitet und mit `-->` beendet und können sich über mehrere Zeilen erstrecken.

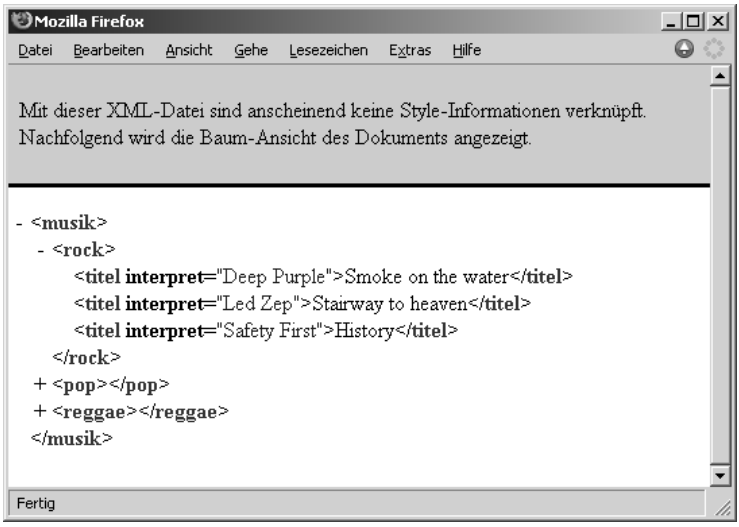


Abbildung 8.2: Die Baumdarstellung der XML-Datei in teilweise expandierter Form in einem Webbrowser

8.4.2 Prozessinstruktionen

Eine **Prozessinstruktion** bzw. **Processing Instruction** (kurz **PI**) übermittelt dem verarbeitenden Programm eines XML-Dokuments spezifische Informationen. Eine PI muss immer im Prolog einer XML-Datei hinter der XML-Deklaration notiert werden. Dort können auch mehrere PIs nacheinander stehen. Eine PI beginnt mit `<?`, gefolgt von einem beliebigen Bezeichner, der in der verarbeitenden Anwendung zum Identifizieren der PI verwendet wird. Dieser wird auch **Target-Angabe** genannt. Optional folgen XML-Attribute. Die Zeichenfolge `?>` beendet eine PI. Beispiel:

Listing 8.8: Eine PI

```
<?ausgabe drucken="yes" ?>
```

Sie können sowohl eigene PIs definieren als auch Standard-PIs mit festgelegter Bedeutung verwenden.

Die Verwendung von Style Sheets mit einer PI festlegen

Eine häufig genutzte PI ist die Angabe des zu verwendenden Style Sheet, um die XML-Daten optisch aufzubereiten. Webbrowser unterstützen die Verwendung in der Regel. Beispiel:

Listing 8.9: Eine Standard-PI zum Festlegen des verwendeten Style Sheet in der XML-Datei

```
<?xml-stylesheet href="stil.css" type="text/css" ?>
```

Wenn Sie eine geeignet aufgebaute CSS- oder XSL-Datei erstellen und einer XML-Datei zuweisen, werden moderne Webbrowser keine Baumstruktur mehr anzeigen, sondern die XML-Datei rein optisch wie eine Webseite interpretieren. Dabei werden die Tags selbst (wie bei HTML) nicht mehr angezeigt, aber die Inhalte entsprechend der Style Sheets-Regeln formatiert.

Abbildung 8.3: Eine mit CSS optisch aufbereitete XML-Datei im Webbrowser – Sie sehen keine Baumstruktur mehr.

Sofern eine Style Sheet-Datei mit einer XML-Datei verknüpft ist und diese aber Elemente enthält, die in der XML-Datei nicht formatiert werden, wird für diese Elemente ausschließlich der Inhalt der Elemente hintereinander angezeigt.

Abbildung 8.4: Für die Elemente der XML-Datei fehlen in der CSS-Datei Regeln

8.4.3 Referenzen

Einen weiteren Komponententyp in XML stellen **Referenzen** dar. Wie in HTML gibt es in XML einige Zeichen (zum Beispiel < oder >), die Sie nicht direkt in Textinhalt eines Elements schreiben können. Diese müssen maskiert[5] werden.

5 *Das bedeutet, über unkritische Zeichen abgebildet werden.*

Entity-Referenzen

Zur Maskierung gibt es in XML einige vorgegebene **Entity-Referenzen**, die analog in HTML vorhanden sind (aber XML kennt nicht alle Entity-Referenzen von HTML):

Entity-Referenz	Zeichen
`&`	&
`&apos;`	'
`>`	>
`<`	<
`&quot;`	"

Tabelle 8.2: Die vordefinierten Entity-Referenzen in XML

Hinweis

XML bietet auch die Möglichkeit, in Form von selbst definierten Entitäten eine Art Makros einzuführen, die im XML-Dokument ähnlich wie Variablen in einer Programmiersprache einen Wert besitzen.

Zeichenreferenzen

In XML gibt es alternativ auch **Zeichenreferenzen**. Diese Referenzen beginnen mit dem Token `&#`, dem eine Dezimalzahl oder Hexadezimalzahl (dann mit vorangestelltem `x`) mit dem Nummerncode (Unicode) des gewünschten Zeichens folgt. Ein Semikolon beschließt die Zeichenreferenz. Beispiele:

Referenz	Zeichen
`&#252;`	ü
`&#xFC;`	ü
`&#228;`	ä

Tabelle 8.3: Beispiele für Zeichenreferenzen

8.4.4 Unkontrollierte Bereiche – CDATA-Abschnitte

Einen weiteren Komponententyp stellt ein unkontrollierter Bereich dar. Wenn ein XML-Parser ein XML-Dokument analysiert, wird jeder Inhalt, der gegen XML-Regeln verstößt und nicht in einem Kommentarbereich notiert ist, einen Fehler auslösen. Das kann zum Beispiel Text sein, der Sonderzeichen nicht maskiert. Wenn solcher Inhalt

in einem XML-Dokument vorkommen soll, kann man ihn in einen `CDATA`-Abschnitt einschließen. Ein solcher unkontrollierter Bereich wird vom Parser beim Analysieren ignoriert. Er sieht schematisch so aus:

Listing 8.10: CDATA-Bereiche in einem XML-Dokument werden vom Parser ignoriert.

```
<![CDATA[ beliebiger Inhalt ]]>
```

8.4.5 Namensräume

Da Sie über XML in der Lage sind, ihre eigenen Tags frei zu definieren, kann es zu Überschneidungen kommen. Tags können gleiche Namen, aber verschiedene Bedeutungen haben. Mit Namensräumen wird in XML versucht sicherzustellen, dass Elemente in bestimmten Bereichen eindeutig sind. In der XML-Spezifikation ist ein Namensraum nichts anderes als ein per Doppelpunkt abgetrenntes, definiertes Präfix für einen spezifischen Elementnamen. So weit sind Namensräume nicht sonderlich schwierig. Aber um Namensräume von verschiedenen Gruppen zu trennen, wird dem Präfix eine Struktur zugeordnet, für die das Präfix nur eine Abkürzung darstellt. Das Verfahren sprengt aber unseren Rahmen.

8.5 Gültige Dokumente

Die minimale Forderung an ein XML-Dokument ist die Wohlgeformtheit. Darüber hinaus kann man von einem XML-Dokument die **Gültigkeit** fordern. Dies bedeutet, einem XML-Dokument wird über die syntaktische Wohlgeformtheit hinaus eine Grammatik zugeordnet, die ein validierter Parser berücksichtigen kann. Diese Grammatik wird in Form von Informationen abgelegt, die nicht zum Inhalt des XML-Dokuments zählen. Die Grammatik wird in Form einer **DTD** (**Document Type Definition**) oder eines **Schematas** formuliert. Ein XML-Dokument ist **gültig** oder **valid**, wenn es eine DTD oder ein Schemata besitzt und sämtliche Elemente samt Attribute usw. gemäß dieser Vorgabe angeordnet sind und allen sonstigen Vorgaben darin (Anzahl, Art des Inhalts etc.) entsprechen. Gültigkeit setzt zwingend die Wohlgeformtheit voraus!

Hinweis

Validierende Parser werden bei der Verarbeitung von XML-Dateien die Gültigkeit von XML-Daten überprüfen und gegebenenfalls mit Fehlermeldungen reagieren, etwa wenn ein Datenbankprogramm XML-Daten einliest. Aus Sicht von AJAX ist die Gültigkeit eines XML-Dokuments jedoch zweitrangig. Auch ein ungültiges XML-Dokument ist ja rein syntaktisch nicht unbedingt falsch. Das Geschäft von AJAX beginnt an der Stelle, wo die XML-Daten bereits fertig sind und Sie auf die Korrektheit der Daten vertrauen müssen. Oder anders

ausgedrückt – bei AJAX geht es darum, Daten vom Server nachzufordern. Natürlich müssen diese Daten korrekt sein, aber die Gewährleistung kann nicht Aufgabe des Mechanismus zur Datenübertragung sein. Das bedeutet nicht, dass die serverseitige Geschäftslogik bei einer AJAX-Applikation sich nicht darum zu kümmern hat. Aber die muss sich auch um viele andere Dinge kümmern, die nicht den unmittelbaren Datenaustauschprozess betreffen (Benutzerverwaltung, Erstellen regelmäßiger Logdateien, Datenbankzugriffe etc.). Dennoch sollten Sie in Grundzügen Bescheid wissen, was es mit der Gültigkeit von XML-Dokumenten auf sich hat.

8.6 XPath

Für den praktischen Umgang mit XML und vor allem der XML-Verarbeitung mit Java oder auch XSL ist eine Technologie namens **XPath** (XML Path Language) elementar. Dies ist eine vom W3C entwickelte **Lokalisierungssprache**, um Bestandteile in einem XML-Dokument oder allgemein in einem baumartig aufgebauten Dokument ansprechen zu können. Derzeit ist in der Praxis XPath 1.0 aktuell und die Entwicklung der Nachfolgeversion 2.0 schon weit gediehen. Ein XPath-Ausdruck versteht ein strukturiertes Dokument als Baum, der aus **Knoten** (**nodes**) und **Achsen** (**axis**) aufgebaut ist. Als Knoten des Baums werden alle Komponenten wie Elemente, Attribute, Kommentare oder Inhalte gesehen. Allerdings werden nur ineinander enthaltene Elemente als untergeordnete **Kindelemente** (**child-Elemente**) gesehen. Alle anderen Knotentypen werden in die gleiche Hierarchiestufe des Baums wie der Knoten einsortiert, der sie enthält. Ein Attribut befindet sich also in der gleichen Hierarchieebene wie das Element, dem das Attribut zugeordnet ist. Achsen werden in XPath wie der Stamm bzw. die Äste des Baums verstanden und berücksichtigen auch die Reihenfolge der Elementdeklarationen (aus sequentieller Sicht). Achsen und Knoten können in XPath mit einer Reihe eigener Token oder Funktionen angesprochen werden.

Tipp

Ein sehr gutes und kostenloses Tool, um XPath-Ausdrücke zu testen (aber auch um XML- und XSL-Dateien zu erstellen und zu testen), ist **Cooktop** (*http://xmlcooktop.com*). Besonders interessant ist dabei, dass das Programm eine eigene XPath-Konsole bereitstellt, in der Sie XPath-Ausdrücke einfach eingeben und das Resultat Ihrer Eingabe direkt angezeigt bekommen.

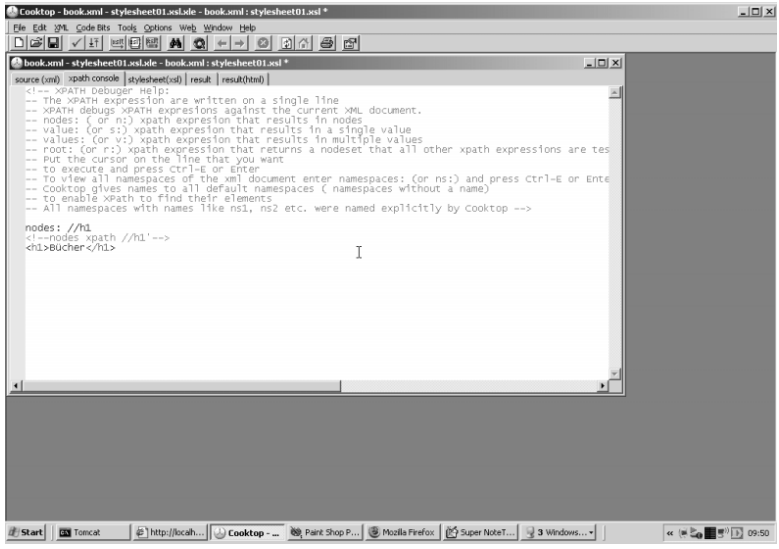


Abbildung 8.5: Die XPath-Konsole in Cooktop ist eine einfache Möglichkeit, um XPath-Ausdrücke zu testen.

8.6.1 Die Lokalisierung auf Achsen

Wie in einem Verzeichnispfad können Sie unter XPath nun einen Knoten oder eine Achse lokalisieren. Bei der Angabe wird immer der aktuelle Knoten in der Baumstruktur des XML-Dokuments als Basis verwendet. Dies ist vollkommen analog zur Art und Weise, wie Sie Pfadangaben in einem Verzeichnisdienst machen. Auch dort stehen Sie in einem aktuellen Verzeichnis und geben von da den Weg zu einem anderen Ort (einem anderen Verzeichnis oder einer Datei) an. Dabei sind sowohl relative als auch absolute Pfadangaben möglich.

Unter XPath verwenden Sie zur Lokalisierung einen oder mehrere **Lokalisierungsschritte** (Location Steps), die gegebenenfalls wie unter Unix mit dem Token / getrennt werden. Ein Lokalisierungsschritt besteht immer aus der Angabe einer Achse und einem so genannten **Knotentest** (**node-Test**), dem optional **Prädikate** (**predicates**) folgen können. Dabei lassen sich beliebig viele XPath-Ausdrücke mit dem Oder-Operator | verbinden und Achsen können gelegentlich mit zusammengesetzten Bezeichnern angesprochen werden, wenn dies einen XPath-Ausdruck verkürzt. Ebenso gibt es für Pfadangaben auf Achsen einige Abkürzungen, die der Notation in Verzeichnisangaben sehr ähnlich sind.

8.6.2 Knoten selektieren

Wollen Sie die Wurzel des XML-Dokuments ansprechen, wird ein XPath-Ausdruck (wie unter Unix) direkt mit dem Token / begonnen[6]. Der Punkt steht für den aktuellen Knoten innerhalb des XML-Dokuments. Pfadangaben ab hier sind relativ. Meist verzichtet man auf die Angabe eines Punkts und gibt direkt den Unterknoten an. Das ist äquivalent. Zwei Punkte stehen für das Elternelement. Pfadangaben ab hier sind relativ. Der doppelte Slash // steht für einen beliebigen Pfad im XML-Dokument.

Schauen wir uns ein paar Beispiele zur Lokalisierung mit und ohne Knotentest an. Dazu verwenden wir den jeweiligen Ausdruck auf die nachfolgende angegebene XML-Datei:

Listing 8.11: Eine XML-Datei

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet href="../stylesheets/simpleStyle.xsl" type="text/xsl"?>
<html>
 <head>
 <title>Meine Bücher</title>
 </head>
 <body>
 <h1>Bücher</h1>
 <h2>Die aktuellen Bücher von Ralph Steyer</h2>
 <buch>
 <titel>
 <a href='http://www.addison-wesley.de/'>
 <b>AJAX</b>
 </a>
 </titel>
 <autor>Ralph Steyer</autor>
 <verlag>Addison-Wesley</verlag>
 <isbn>3827323584</isbn>
 <erscheindat>03.2006</erscheindat>
 <seiten>310</seiten>
 <preis>24.95</preis>
 </buch>
 <buch>
 <titel>
 <a href='http://www.addison-wesley.de/'>
 <b>AJAX mit Java-Servlets und JSP</b>
 </a>
 </titel>
 <autor>Ralph Steyer</autor>
 <verlag>Addison-Wesley</verlag>
```

6 *Grundsätzlich beginnen alle Ausdrücke, die mit einem Slash als ersten Token anfangen, auf der Wurzel als Referenzelement.*

```
 <isbn>3-8273-2418-1</isbn>
 <erscheindat>06.2006</erscheindat>
 <seiten>350</seiten>
 <preis>24.95</preis>
 </buch>
 </body>
</html>
```

XPath	Wirkung
`//h1`	Alle Elemente vom Typ `h1` im gesamten Dokument
Resultat	`<h1>Bücher</h1>`
`/html/body/h1`	Alle Elemente vom Typ `h1`, die im Element `body` enthalten sind. Das Element `body` muss im Wurzelelement `html` enthalten sein.
Resultat	`<h1>Bücher</h1>`
`/html/h1`	Alle Elemente vom Typ `h1`, die im Wurzelelement `html` enthalten sind. In dem Beispiel gibt es den Fall nicht.
`//body`	Alle Elemente vom Typ `body`, die sich irgendwo in dem XML-Dokument befinden.

Resultat	`<body>`
	`<h1>Bücher</h1>`
	`<h2>Die aktuellen Bücher von Ralph Steyer</h2>`
	`<buch>`
	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX</b>`
	`</a>`
	`</titel>`
	`<autor>Ralph Steyer</autor>`
	`<verlag>Addison-Wesley</verlag>`
	`<isbn>3827323584</isbn>`
	`<erscheindat>03.2006</erscheindat>`
	`<seiten>310</seiten>`
	`<preis>24.95</preis>`
	`</buch>`
	`<buch>`
	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX mit Java-Servlets und JSP</b>`
	`</a>`
	`</titel>`
	`<autor>Ralph Steyer</autor>`
	`<verlag>Addison-Wesley</verlag>`
	`<isbn>3-8273-2418-1</isbn>`
	`<erscheindat>06.2006</erscheindat>`
	`<seiten>350</seiten>`
	`<preis>24.95</preis>`
	`</buch>`
	`</body>`

`//a`	Alle Knoten vom Typ `a`

Tabelle 8.4: Lokalisierung auf Knoten

XPath	Wirkung

Resultat	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX</b>`
	`</a>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX mit Java-Servlets und JSP</b>`
	`</a>`

`/html/body/buch/titel/a`	Alle Knoten vom Typ `a`, die sich im Element `titel` befinden

Resultat	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX</b>`
	`</a>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX mit Java-Servlets und JSP</b>`
	`</a>`

Tabelle 8.4: Lokalisierung auf Knoten (Forts.)

Den x-ten Knoten einer Liste selektieren

Mit einer Form von Arrays können Sie in einer Liste bzw. einem Node-Set Knoten selektieren. Der Index wird in eckige Klammern notiert und die Indizierung beginnt mit 1. Beispiel:

XPath	Wirkung
`/html/body/buch[1]`	Das erste Element vom Typ `buch` in `/html/body`

Resultat	`<buch>`
	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX</b>`
	`</a>`
	`</titel>`
	`<autor>Ralph Steyer</autor>`
	`<verlag>Addison-Wesley</verlag>`
	`<isbn>3827323584</isbn>`
	`<erscheindat>03.2006</erscheindat>`
	`<seiten>310</seiten>`
	`<preis>24,95</preis>`
	`</buch>`

Tabelle 8.5: Einen gezielten Knoten auswählen

Die Ergebnismenge einschränken

Die Ergebnismenge eines XPath kann man durch Angabe von weiteren Eigenschaften eingrenzen. So können Sie die Werte von Elementen testen. Beispiele:

XPath	Wirkung
`/html/body/buch[preis < '24.95']`	Alle Elemente vom Typ `buch` unter `/html/body`, bei denen der Wert des Elements `preis` kleiner als `24.95` ist. In dem Beispiel gibt es den Fall nicht. Beachten Sie die Maskierung von <.
`/html/body/buch[verlag='Irgendeiner']`	Alle Elemente vom Typ `buch` unter `/html/body`, bei denen der Wert des Elements `verlag` `Irgendeiner` ist. In dem Beispiel gibt es den Fall nicht.
`//buch[preis > 20]/titel`	Alle Elemente vom Typ `titel`, die in dem Element `buch` enthalten sind und bei denen der Wert von `preis > 20` ist.

Resultat	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX</b>`
	`</a>`
	`</titel>`
	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX mit Java-Servlets und JSP</b>`
	`</a>`
	`</titel>`

Tabelle 8.6: Eingrenzen der Ergebnismenge

Ebenso kann man die Attribute eines Elements abfragen. Dies erfolgt durch ein dem Namen vorangestelltes `@`. Prinzipiell verhalten sich Attribute wie Elemente und können wie diese beispielsweise zur Eingrenzung von Mengen verwendet werden. Beispiele:

XPath	Wirkung
`//a[@href='down/Beispiel.zip']`	Alle Elemente vom Typ `a`, bei denen der Wert des Attributs `href` `down/Beispiel.zip` ist. In dem Beispiel gibt es den Fall nicht.

Tabelle 8.7: Zugriff auf Attribute

8.6.3 Alternative XPath-Syntax

Statt der aus der Betriebssystemumgebung bekannten Syntax kann man unter XPath eine alternative bzw. erweiterte Syntax anwenden. Jeder darüber durchgeführte Test hat folgende Syntax:

Listing 8.12: Schema eines Knotentests

```
<Knotentyp>::<Testknoten>[<Bedingungen>]
```

Ein Knotentest besteht also immer aus einem vorangestellten Achsenausdruck und – mit zwei Doppelpunkten abgetrennt – einem Testknoten dahinter. Damit werden alle Knoten bezeichnet, die auf der vorangestellten Achse vorkommen. Folgende Knotentypen stehen dabei zur Verfügung:

Achse	Angesprochene Knoten
`ancestor`	Alle übergeordneten Knoten. Dies umfasst alle Vorfahren des aktuellen Knotens im Baum bis zur Wurzel.
`ancestor-or-self`	Alle übergeordneten übergeordnete Knoten sowie der aktuelle Knoten selbst
`attribute`	Alle Attribute des angesprochenen Elements
`child`	Alle Knoten, die dem aktuellen Knoten direkt untergeordnet sind
`descendant`	Alle Knoten, die dem aktuellen Knoten direkt oder indirekt untergeordnet sind. `descendant` steht allgemein für Nachfolger oder Nachfahre.
`descendant-or-self`	Alle Knoten, die dem aktuellen Knoten direkt oder indirekt untergeordnet sind sowie der aktuelle Knoten selbst
`following`	Alle Knoten, die in der Reihenfolge des Baums dem aktuellen Knoten nachfolgen
`following-sibling`	Alle Knoten, die auf der gleichen Hierarchieebene der Baumstruktur zu finden sind (so genannte Geschwister, die vom gleichen `parent`-Knoten stammen) und in der Reihenfolge des Baums dem aktuellen Knoten nachfolgen
`namespace`	Zugriff auf den Namensraum eines Knotens
`parent`	Der direkt übergeordnete Knoten des aktuellen Knotens
`preceding`	Alle Knoten, die in der Reihenfolge des Baums dem aktuellen Knoten vorhergehen
`preceding-sibling`	Alle Knoten, die auf der gleichen Hierarchieebene der Baumstruktur zu finden sind (so genannte Geschwister, die vom gleichen `parent`-Knoten stammen) und in der Reihenfolge des Baums dem aktuellen Knoten vorangehen.
`self`	Der aktuelle Knoten selbst

Tabelle 8.8: Alternative Angaben von Achsen unter XPath

Beispiele:

XPath	Wirkung
`/html/body/buch/child::titel`	Alle Kinder des Elements `/html/body/buch`, die dem Knoten `titel` entsprechen

Resultat	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX</b>`
	`</a>`
	`</titel>`
	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX mit Java-Servlets und JSP</b>`
	`</a>`
	`</titel>`

`/html/body/buch/parent::body`	Das Elternelement von `buch`, das mit `body` übereinstimmt

Resultat	`<body>`
	`<h1>Bücher</h1>`
	`<h2>Die aktuellen Bücher von Ralph Steyer</h2>`
	`<buch>`
	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX</b>`
	`</a>`
	`</titel>`
	`<autor>Ralph Steyer</autor>`
	`<verlag>Addison-Wesley</verlag>`
	`<isbn>3827323584</isbn>`
	`<erscheindat>03.2006</erscheindat>`
	`<seiten>310</seiten>`
	`<preis>24.95</preis>`
	`</buch>`
	`<buch>`
	`<titel>`
	`<a href="http://www.addison-wesley.de/">`
	`<b>AJAX mit Java-Servlets und JSP</b>`
	`</a>`
	`</titel>`
	`<autor>Ralph Steyer</autor>`
	`<verlag>Addison-Wesley</verlag>`
	`<isbn>3-8273-2418-1</isbn>`
	`<erscheindat>06.2006</erscheindat>`
	`<seiten>350</seiten>`
	`<preis>24.95</preis>`
	`</buch>`
	`</body>`

Tabelle 8.9: Beispiele für die alternative Syntax

Zu beachten ist, dass Attribute und Namensräume nicht in Knotentests angesprochen werden, sondern als eigene Achse. Ebenso unterstützen manche Parser nicht die Verwendung aller Token.

8.6.4 XPath-Prädikate: Operatoren, Relationen und Funktionen

XPath zieht viele Möglichkeiten aus der Tatsache, dass Sie statische Adressangaben nicht nur nutzen, sondern auch programmieren können. Es gibt sowohl Operationen und Relationen als auch Funktionen, mit denen Sie Adressangaben flexibel formulieren können. Eine solche Angabe von Prädikaten wird in eckige Klammern eingeschlossen und stellt einen Ausdruck dar. Dabei können Sie Prädikate in beliebiger Zahl hintereinander schreiben, wobei die Reihenfolge wesentlich ist. Im einfachsten Fall schreiben Sie – wie bereits gesehen – eine Zahl in die eckigen Klammern und geben damit genau das Element an. Zum Beispiel `hd2[2]`, um das zweite Element vom Typ `hd2` anzusprechen. Beachten Sie, dass die Nummerierung der Elemente in XPath mit 1 und nicht mit 0 beginnt. Auch boolesche Ausdrücke können ausgewertet werden. Ergibt der Ausdruck ein »wahr«, dann wird der Knoten verwertet, anderenfalls nicht. Allgemein werden Ausdrücke mithilfe von Operatoren und Funktionen formuliert. Nachfolgend finden Sie die erlaubten Operatoren, wobei wir nicht zwischen arithmetisch, boolesch und vergleichend trennen.

Token	Beschreibung
*	Multiplikation
-	Subtraktion
!=	Ungleichheit
|	Oder
+	Addition
<	Kleiner
<=	Kleiner oder gleich
=	Gleichheit
>	Größer
>=	Größer oder gleich
and	Logisches Und
div	Division (ganzzahlig)
mod	Modulo
or	Logisches Oder

Tabelle 8.10: Operatoren (arithmetisch, boolesch und vergleichend)

8.6.5 Wichtige Funktionen

XPath stellt Ihnen sowohl zur Formulierung von Prädikaten als auch zur direkten Verwendung in der Lokalisierung von Achsen und Elementen eine große Anzahl von Funktionen bereit, von denen hier nur ein paar wichtige beschrieben werden sollen.

Funktion	Beschreibung	Beispiel	Ergebnis
`boolean()`	Umwandlung eines Strings in einen Wahrheitswert	`boolean('true')`	
`ceiling()`	Nächstgrößter Integerwert	`ceiling(sum(//preis))`	Aufgerundeter Wert des Parameters
`concat()`	Verknüpfung zweier Strings	`concat (/html/body/buch[1]/titel, '- ', /html/body/buch[2]/titel)`	Der Gesamtstring
`contains()`	Test, ob übergebene Zeichenkette in einem String enthalten ist	`contains(//titel, 'AJAX')`	`true` oder `false`
`count()`	Anzahl der Knoten in einer Knotenmenge	`//Artikel[count (Anzahl)<=100 and count(Anzahl)>=10]`	Alle Knoten vom Typ `Artikel`, die mindestens zehn, aber höchstens 100 Kindelemente vom Typ `Anzahl` haben
`false()`	Konstanter Wert `false`		
`floor()`	Abschneiden des Nachkommateils	`floor(sum(//preis))`	Abgerundeter Wert des Parameters
`last()`	Letzte Position eines Elements in einer Liste mit Knoten	`last()`	Anzahl aller Knoten in einer Liste
`name()`	Qualifizierter Name des Elements	`name(/html/body)`	`body`
`normalize-space()`	Entfernt alle führenden Leerzeichen	`normalize-space ($alletitel)`	Eventuelle führende Leerzeichen des Knotens werden entfernt.
`not()`	Umkehrung des Wahrheitswerts		
`number()`	Casting einer Zeichenkette in eine Zahl	`number(/html/body/buch[1]/preis) + number(/html/body/buch[2]/preis)`	Summe der beiden Elementinhalte
`position()`	Die Position des aktuellen Knotens im gesamten gefundenen Knotensatz	`child::*[position()=1]`	Der erste Knoten im gefundenen Knotensatz. Die Kurzform davon ist `child::*[1]`.

Tabelle 8.11: Wichtige XPath-Funktionen

Funktion	Beschreibung	Beispiel	Ergebnis
`round()`	Runden auf einen Integer (kaufmännisch)	`round(sum(//preis))`	Kaufmännisch gerundeter Wert des Parameters
`string-length()`	Anzahl der Zeichen in einem String	`string-length(/html/body/buch[2]/titel)`	Anzahl der Zeichen in dem Knoten
`substring()`	Extrahiert von einer bestimmten Position an eine gewisse Anzahl an Zeichen. Wird diese Anzahl nicht angegeben, werden alle Zeichen bis zum Ende extrahiert.	`substring(//autor, 1, 5)`	Die ersten fünf Zeichen des Knotens
`sum()`	Summe der Knotenelemente	`sum(//preis)`	Summe der Elementinhalte
`text()`	Auswahl eines Textknotens	`following::text()`	Alle folgenden Textknoten
`translate()`	Ersetzt die im ersten Argument enthaltenen Zeichen durch die Zeichen im dritten Argument. Im zweiten Argument stehen die damit zu ersetzenden Zeichen.	`translate($alletitel, ' -','*#')`	Ersetzt alle Leerzeichen durch * und alle – durch #
`true()`	Konstanter Wert `true`		

Tabelle 8.11: Wichtige XPath-Funktionen (Forts.)

8.7 XML-Daten senden und der Nutzen des node-Objekts

Wenn Sie nun bei einer AJAX-Anfrage dem Client XML-Daten senden wollen, ist die rein auf die AJAX-Anfrage bezogene Arbeit auf Seiten des Servers trivial. Zumindest was die reine Versendung der Daten angeht – die Erstellung von den XML-Daten kann komplexeste Prozesse erfordern[7]. Es gibt kaum einen Unterschied zu dem Fall, dass Sie HTML oder einen Text senden wollen. Sie generieren unmittelbar mit einem serverseitigen Skript oder Programm ein XML-Dokument (das sogar nur im Hauptspeicher des Servers existieren muss) und schicken es zum Client. Oder Sie versenden direkt eine XML-Datei, die bereits anderweitig erstellt wurde. Betrachten Sie die nachfolgenden Abwandlungen unserer Standardfunktionen:

7 *Was aber rein auf AJAX beschränkt wieder vollkommen uninteressant ist.*

Listing 8.13: Anfordern von XML-Daten

```
22 function sndReq() {
23     resObjekt.open('get', 'xmlfile1.xml',true);
 ...
26 }
```

In Zeile 23 wird einfach direkt eine XML-Datei angefordert, die dann mit `document.getElementById("antwort").innerHTML = "Antwort:<br>" + resObjekt.responseText;` in die Webseite übernommen wird. Hier ist rein gar kein Unterschied zu bisherigen Verhaltensweisen zu sehen. Die Vorteile liegen rein beim Server, auf dem Sie sämtliche Möglichkeiten von XML zur Abbildung von Geschäftslogik ausreizen können.

Abbildung 8.6: Die XML-Daten wurden wie gehabt in die Webseite integriert.

8.7.1 Die Verwendung von node und responseXML

Gerade wenn Sie vom Server Daten im XML-Format erhalten, ist die Möglichkeit der differenzierten Auswertung der Antwort gegeben. Statt pauschal einen Bereich der Webseite mit der kompletten Antwort zu füttern, suchen Sie auf dem Client den Teil einer XML-Antwort heraus, der an eine ganz bestimmte Stelle gehört. Aber Sie können auch allgemein auf einem strukturierten Antwortdokument im Client navigieren (auch im HTML- oder XHTML-Format). Um diese Differenzierung der Antwortdaten auf dem Client[8] durchführen zu können, benötigen Sie zwei Dinge, sofern Sie sich nicht unnötig Mühe durch Handarbeit machen wollen[9]:

1. Die Eigenschaft `responseXML` zur Auswertung der Antwort eines `XMLHttpRequest`-Objekts
2. Das `node`-Objekt mit seinen diversen Eigenschaften und Methoden, die an der XML-Denkweise und den XPath-Angaben orientiert sind, um darüber auf den Elementen in der Webseite navigieren zu können

8 Was im Grunde die Übertragung eines Teils der Geschäftslogik zum Client bedeutet.

9 Natürlich können Sie den Antwort-String des Servers mit geeigneten JavaScript-Techniken zur String-Verarbeitung durchforsten, aufspalten etc. – sogar bei reinem Klartext oder HTML. Aber das ist wie gesagt unnötige Handarbeit.

Tipp

Insbesondere kann und wird man es sich bei AJAX zunutze machen, dass über `node` auch Elemente in eine Webseite ***ergänzt*** werden können.

Hinweis

Sie sollten bei den nachfolgend vorgestellten Techniken beachten, dass diese in mehreren derzeit aktuellen Browsern noch nicht oder nur eingeschränkt unterstützt werden. Dies wird sich aber mit neuen Browser-Versionen nach und nach geben. Wichtig sind diese Techniken allemal, denn sie gestatten in zukünftigen AJAX-Applikationen die Verlagerung von Logik auf den Client. Allerdings sollten Sie die Techniken derzeit in der Praxis nur mit großer Vorsicht und ausführlichen Tests in allen für Sie relevanten Browsern einsetzen.

Navigation auf den Elementen einer Webseite

Um nun die Eigenschaften und Methoden des `node`-Objekts zur Navigation auf den Elementen einer Webseite nutzen zu können, benötigen Sie einen Knoten. Und diesen liefern die bereits behandelten Methoden des `document`-Objekts `getElementById()` und `getElementsByName()` sowie `getElementsByTagName()`. Damit können Sie bereits in jeder Webseite die Strukturen abfragen und gezielt verwenden. Dies gestattet aber zusätzlich in Hinsicht auf die Datennachforderung per AJAX eine sehr qualifizierte Ansteuerung beliebiger Teile der Webseite und den Austausch verschiedenster Teile.

Navigation auf den Elementen der Server-Antwort

Aus Sicht von AJAX ist es jedoch mindestens ebenso interessant, dass das `XMLHttpRequest`-Objekt selbst einen Knoten liefert, wenn Sie die `responseXML`-Eigenschaft betrachten. Dies ist der Wurzelknoten des XML-Dokuments, das als Antwort vom Server geliefert wird. Ausgehend davon können Sie die Attributknoten, Textknoten und weitere Element-Kindknoten eines Knotens in der Antwort des Servers ansprechen.

Hinweis

Selbst wenn der Server HTML oder XHTML als Antwort schickt, können Sie diese in der Regel der `responseXML`-Eigenschaft zuweisen und dann auf dem Knoten die nachfolgenden Techniken einsetzen.

Eigenschaften von node

Das `node`-Objekt stellt folgende Eigenschaften bereit:

Eigenschaft	Beschreibung
`attributes`	Die Eigenschaft repräsentiert ein Array mit den verfügbaren Attributen eines Elements. Natürlich stehen damit alle Eigenschaften und Methoden zur Verfügung, die auf jedes JavaScript-Array angewendet werden können, insbesondere die Eigenschaft `length` für die Anzahl der Elemente.
`childNodes`	Die Eigenschaft repräsentiert ein Array mit den verfügbaren Kindknoten eines Elements. Natürlich stehen damit alle Eigenschaften und Methoden zur Verfügung, die auf jedes JavaScript-Array angewendet werden können, insbesondere die Eigenschaft `length` für die Anzahl der Elemente.
`data`	Eine interessante Eigenschaft eines `node`-Objekts ist `data`. Darüber steht der Inhalt eines **Textknotens** in Form von Zeichendaten zur Verfügung.
`firstChild`	Der erste Kindknoten eines Knotens
`lastChild`	Der letzte Kindknoten eines Knotens
`nextSibling`	Der nächste Knoten eines Knotens auf derselben Ebene des Dokumentenbaums
`nodeName`	Der Name des Knotens
`nodeType`	Der Knotentyp in numerischer Form. Die Zahlenwerte sind vom W3C standardisiert worden und zum Teil explizit nur in einem XML-Dokument verfügbar. (X)HTML-Dokumente stellen also eine Teilmenge bereit:

Nummer	Knotentyp
1	Elementknoten
2	Attributknoten
3	Textknoten
4	CDATA-Bereich
5	Entity-Referenz
6	Entity
7	PI
8	Kommentar
9	Dokument
10	Dokumenttyp
11	Dokumentfragment
12	Notation

Konkret für AJAX sind die Typen 1 bis 3 am wichtigsten.

Tabelle 8.12: Eigenschaften von node

Eigenschaft	Beschreibung
nodeValue	Die Eigenschaft enthält den Wert oder Inhalt eines Knotens. Bei Textknoten ist dies der Text, bei Attributknoten der zugewiesene Attributwert. Bei Elementknoten steht in der Eigenschaft der Wert null.
parentNode	Der Elternknoten
previousSibling	Der vorherige Knoten auf derselben Ebene

Tabelle 8.12: Eigenschaften von node (Forts.)

Methoden von node

Folgende Methoden stehen Ihnen bei einem Objekt vom Typ node zur Verfügung:

Methode	Beschreibung
appendChild()	Mit dieser Methode hängen Sie einen zuvor neu erzeugten Knoten als letztes Kindelement des vorangestellten Knotens an (Kindknoten).
appendData()	Die Methode fügt einem Textknoten oder dem Wert eines Attributknotens am Ende neue Zeichendaten hinzu. Die bereits vorhandenen Daten bleiben erhalten.
cloneNode()	Mit dieser Methode ist die Erstellung einer Kopie eines Knotens möglich.
deleteData()	Mit dieser Methode können die Zeichendaten eines Textknotens oder der Wert eines Attributknotens gelöscht werden.
getAttribute()	Rückgabe des Werts eines Attributknotens.
getAttributeNode()	Rückgabe des Attributknotens.
getElementsByTagName()	Mit dieser Methode ist der Zugriff auf Kindelemente eines Knotens über den Elementnamen möglich. Die Methode liefert ein Array mit den Knoten und kann im Fall von XML auf beliebige Elementknoten angewendet werden. Im Fall der gleichnamigen Methode des document-Objekts ist immer die gesamte Webseite gemeint.
hasChildNodes()	Mit der Methode können Sie auf die Existenz von Kindknoten prüfen. Wenn ein Knoten Kindknoten hat, liefert sie den booleschen Wert true, andernfalls false.
insertBefore()	Mit dieser Methode fügen Sie im aktuellen Knoten einen Kindknoten vor einem anderen Kindknoten ein.
insertData()	Mit dieser Methode fügen Sie Zeichendaten in einem Textknoten ein. Dazu geben Sie als ersten Parameter die Zeichenposition an.
removeAttribute()	Über den Aufruf dieser Methode erfolgt das Löschen eines benannten Attributs des aktuellen Knotens.
removeAttributeNode()	Mit dieser Methode löschen Sie den genannten Attributknoten.

Tabelle 8.13: Die Methoden von node

Methode	Beschreibung
`removeChild()`	Löschen eines benannten Kindknotens
`replaceChild()`	Über den Aufruf dieser Methode erfolgt das Ersetzen eines Kindknotens durch einen anderen Knoten. Der neue Kindknoten wird als erster Parameter und der zu ersetzende Knoten als zweiter Parameter angegeben.
`replaceData()`	Ersetzen der Zeichendaten eines Textknotens durch einen anderen Text. Als Parameter geben Sie die Startposition in der Zeichenkette an, ab der ersetzt werden soll. Der zweite Parameter gibt die Anzahl der zu ersetzenden Zeichen und eine Zeichenkette an, mit der die Zeichenkette im Textknoten ersetzt werden soll.
`setAttribute()`	Über diese Methode können Sie in einem Element einen Attributwert neu zuweisen. Ist das Attribut bereits vorhanden, wird der alte Wert durch den neuen Wert ersetzt. Andernfalls wird das Attribut erzeugt und mit dem Wert belegt. Beachten Sie, dass die Methode nicht in allen Browsern korrekt unterstützt wird.
`setAttributeNode()`	Über diese Methode können Sie in einem Element einen neuen Attributknoten einfügen. Ist der Attributknoten bereits vorhanden, wird der alte Knoten durch den neuen Knoten ersetzt. Andernfalls wird er neu angelegt. Beachten Sie, dass die Methode nicht in allen Browsern korrekt unterstützt wird.

Tabelle 8.13: Die Methoden von node (Forts.)

8.7.2 Zerlegen einer XML-Antwort

Nutzen wir die Eigenschaften und Methoden von `node` und `responseXML` nun in einer praktischen Anwendung, die auch mit einem Beispiel untermauert werden soll. Wir zerlegen die etwas komplexere Antwort des Servers wieder auf dem Client und nutzen Teile davon individuell.

Hinweis

Es ist keinesfalls trivial zu entscheiden, ob Sie überhaupt Geschäftslogik bei einer AJAX-Applikation auf den Client verlagern sollten. Wenn Sie eine komplexere XML-Datenstruktur zum Client schicken und diese dann erst dort aufgespalten und in verschiedener Form verarbeitet wird, erzeugen Sie eine möglicherweise recht komplexe Multi-Tier[10]-Applikation. Die Wartbarkeit kann verzwickter werden. Auch die Trennung von Darstellung und Geschäftslogik wird unter Umständen wieder aufgehoben. Mit anderen Worten – was technisch mit dem `node`-Objekt und `responseXML` machbar ist, muss in der Praxis keineswegs sinnvoll sein. Es ist eine Frage der Konzeption, ob Sie es

10 Mehr-Schichten

anwenden sollten oder nicht. Oftmals ist es sinnvoller, bereits auf dem Server eine einfache Antwort zu generieren. Dennoch – mit dieser Möglichkeit eröffnen sich viele interessante Perspektiven für die Erstellung von AJAX-Applikationen.

Betrachten wir als XML-Dokument, das vom Server aufgrund einer AJAX-Anfrage übermittelt wird, die XML-Struktur aus Listing 8.7 als Beispiel:

Listing 8.14: Die uns schon bekannte XML-Datei wird vom Server als Antwort geliefert.

```
<?xml version="1.0"?>
<musik>
 <rock>
  <titel interpret="Deep Purple">Smoke on the water</titel>
  <titel interpret="Led Zep">Stairway to heaven</titel>
  <titel interpret="Safety First">History</titel>
 </rock>
 <pop>
  <titel interpret="Mandona">Like a virgin</titel>
 </pop>
 <reggae>
  <titel interpret="Bob Marley">No woman no cry</titel>
 </reggae>
</musik>
```

Sie können nun mittels XPath auf dem XML-Baum des Dokuments navigieren. Oder für unseren Zweck die daran angelehnten Methoden und Eigenschaften von `node` verwenden, um auf der Antwort im Client Elemente anzusteuern. Nun liefert `resObjekt.responseXML` ein `node`-Objekt, das dem Antwortdokument selbst entspricht. Mit `firstChild` als auch `lastChild` erhalten Sie die Wurzel `<musik>`[11]. Die Eigenschaft `childNodes` von `firstChild` ist ein Array, in dem sich dann die direkten Kindelemente von `<musik>` befinden. Als da wären `<rock>`, `<pop>` und `<reggae>`. Wenn wir uns das Element `<rock>` vornehmen, ist der Zugang dahin also über `resObjekt.responseXML.firstChild.childNodes[0]` möglich. Mit `resObjekt.responseXML.firstChild.childNodes[0].childNodes[0]` sprechen wir das erste `<titel>`-Element darin an. Und `resObjekt.responseXML.firstChild.childNodes[0].childNodes[0].childNodes[0]` ist wiederum der erste Textknoten darin. Damit finden Sie in `resObjekt.responseXML.firstChild.childNodes[0].childNodes[0].childNodes[0].data` den Wert `Smoke on the water`.

Zugang zu den Attributen erhalten Sie natürlich auch. Mit `resObjekt.responseXML.firstChild.childNodes[0].childNodes[0].getAttributeNode('interpret').nodeValue` erhalten Sie den Wert `Deep Purple`. Beachten Sie, dass das Attribut bei dem Elementknoten vom Typ `<titel>` gesetzt ist.

11 *Beide Eigenschaften referenzieren das gleiche Element, da es nur genau ein Wurzelelement gibt.*

Natürlich brauchen Sie nicht mit `firstChild`, `lastChild` oder `childNodes` zu arbeiten. Sie können jede der oben genannten Lokalisierungseigenschaften oder -methoden anwenden, sofern Sie damit auf dem Baum einen Knoten vernünftig lokalisieren. Zum Beispiel können Sie mit den bereits mehrfach verwendeten Methoden `getElementById()`, `getElementsByName()` sowie `getElementsByTagName()`ebenso arbeiten. Natürlich lassen sich die Angaben auch mischen. Dabei ist aber zu beachten, auf welchen Knoten Sie die Methode anwenden. Mit `resObjekt.responseXML.getElementsByTagName("titel")[0].childNodes[0].data` erhalten Sie zum Beispiel wieder den Wert `Smoke on the water`. Mit `resObjekt.responseXML.getElementsByTagName("titel")[0]` sprechen Sie das erste `<titel>`-Element an und dessen erstes Kindelement ist ein Textknoten mit dem besagten Inhalt.

Um nun die Eigenschaften und Methoden von `node` in Verbindung mit `responseXML` in vollständiger Form und einem etwas komplexeren Zusammenhang zu sehen, betrachten wir eine neue Variante unserer Methode `handleResponse()`. Darin nutzen wir die Pfadangaben entsprechend der XPath-Notation und bauen eine relativ komplexe Tabelle mit der Analyse der XML-Datei auf. Beachten Sie, dass die Funktion nicht auf beliebige XML-Dokumente anzuwenden ist. Die Funktion ist schon recht neutral formuliert und eine vollständige Verallgemeinerung wäre zwar nicht schwierig, aber auch so ist der Quelltext schon etwas aufwändiger:

Listing 8.15: Eine etwas aufwändigere Variante der Methode handleResponse()

```
01 function handleResponse() {
02   xmlDok = resObjekt.responseXML;
03   text="<table border='1'>" +
04   "<tr><th>XPath</th><th>Wert</th><th>Beschreibung</th></tr>";
05   if(resObjekt.readyState == 4){
06     text += "<tr><td>xmlDok.nodeType</td><td>" +
07       xmlDok.nodeType + "</td><td>Dokument</td></tr>";
08     text += "<tr><td>xmlDok.hasChildNodes()</td><td>" +
09       xmlDok.hasChildNodes() +
10       "</td><td>Gibt es Kindelemente?</td></tr>";
11     text += "<tr><td>xmlDok.childNodes.length</td><td>" +
12       xmlDok.childNodes.length +
13       "</td><td>Ein Wurzelelement</td></tr>";
14     text += "<tr><td>xmlDok.firstChild.nodeType</td><td>" +
15       xmlDok.firstChild.nodeType +
16       "</td><td>Typ Element - das Wurzelelement</td></tr>";
17     text += "<tr><td>xmlDok.firstChild.hasChildNodes()</td><td>"
18       + xmlDok.firstChild.hasChildNodes() +
19       "</td><td>Wurzelelement hat Kindelemente?</td></tr>";
20     text += "<tr><td>xmlDok.firstChild.childNodes.length</td><td>"
21       + xmlDok.firstChild.childNodes.length +
22       "</td><td>Anzahl Kindelemente in Wurzelelement</td></tr>";
23 // Alle direkten Kindelemente des Wurzelelements
24     for(i = 0;i < xmlDok.firstChild.childNodes.length; i++) {
```

```
25      text += "<tr><td>xmlDok.firstChild.childNodes[" + i +
26        "].nodeType</td><td>" +
27        xmlDok.firstChild.childNodes[i].nodeType +
28        "</td><td>Direktes Kindelement des Wurzelelements - Knotentyp</td></tr>";
29      text += "<tr><td>xmlDok.firstChild.childNodes[" + i +
30        "].hasChildNodes()</td><td>" +
31        xmlDok.firstChild.childNodes[i].hasChildNodes() +
32        "</td><td>Direktes Kindelement des Wurzelelements - Kindknoten ↵
          vorhanden?</td></tr>";
33      text += "<tr><td>xmlDok.firstChild.childNodes[" + i +
34        "].childNodes.length</td><td>" +
35        xmlDok.firstChild.childNodes[i].childNodes.length +
36        "</td><td>Direktes Kindelement des Wurzelelements - Anzahl ↵
          Kindknoten</td></tr>";
37  // Nächste Ebene im Baum - die titel-Elemente
38      for(j = 0; j < xmlDok.firstChild.childNodes[i].
39      childNodes.length; j++ ){
40        text +=
41          "<tr><td>xmlDok.firstChild.childNodes[" + i +
42          "].childNodes[" + j + "].nodeType</td><td>" +
43          xmlDok.firstChild.childNodes[i].childNodes[j].nodeType
44          + "</td><td>Knotentyp Ebene titel</td></tr>";
45        text += "<tr><td>xmlDok.firstChild.childNodes[" + i +
46          "].childNodes[" + j +
47          "].getAttributeNode('interpret')</td><td>" +
48           xmlDok.firstChild.childNodes[i].childNodes[j].
49           getAttributeNode('interpret').nodeValue +
50           "</td><td>Zugriff auf das Attribut in der Ebene titel</td></tr>";
51  // Nächste Ebene im Baum - Textknoten
52        for(k = 0; k < xmlDok.firstChild.childNodes[i].
53        childNodes[j].childNodes.length; k++ ){
54          text += "<tr><td>xmlDok.firstChild.childNodes[" + i +
55            "].childNodes[" + j + "].childNodes[" + k +
56             "].nodeType </td><td>" +
57             xmlDok.firstChild.childNodes[i].childNodes[j].
58             childNodes[k].nodeType +
59             "</td><td>Knotentyp unterhalb von titel</td></tr>";
60          text +=
61            "<tr><td>xmlDok.firstChild.childNodes[" + i +
62            "].childNodes[" + j +
63            "].childNodes[" + k + "].data </td><td>" +
64            xmlDok.firstChild.childNodes[i].
65            childNodes[j].childNodes[k].data +
66            "</td><td>Inhalt Textknoten unterhalb von titel</td></tr>";
67        } // Ende innere for-Schleife
68      } // Ende mittlere for-Schleife
69    } // Ende äußere for-Schleife
70 text += "</table>";
```

```
71 document.getElementById("antwort").innerHTML = text;
72    }  // Ende if
73 }
```

Achtung

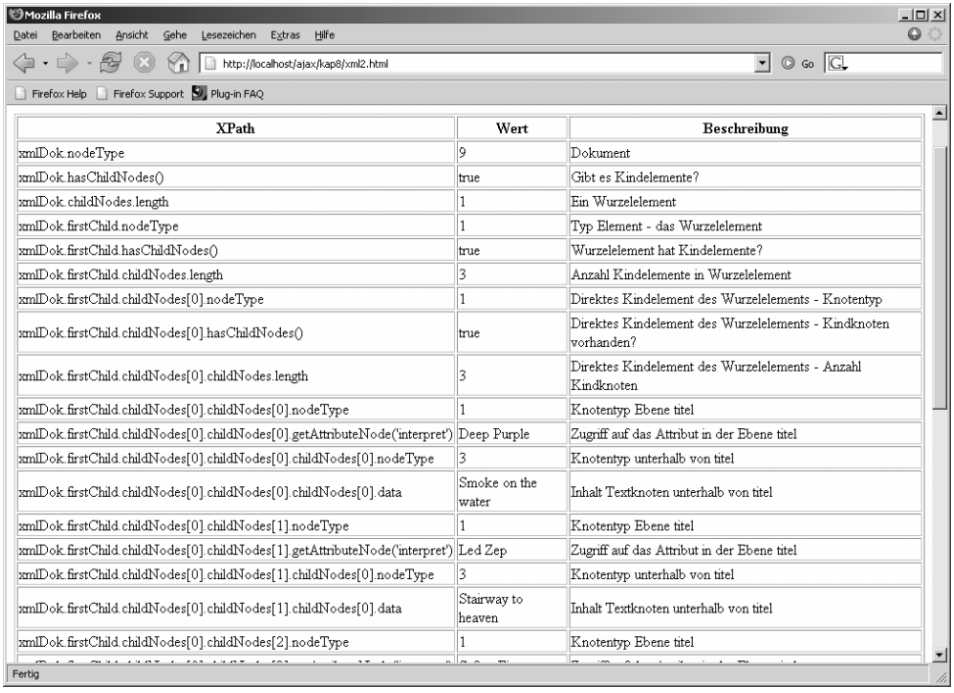
Beachten Sie, dass einige Browser mit der clientseitigen Auswertung von XML-Strukturen auf diese Art nicht zurechtkommen. Das Beispiel läuft in neueren Versionen von Opera, Firefox und Konqueror (dort aber eingeschränkt). Der Internet Explorer 6 unterstützt es nicht.

In Zeile 2 wird mit `xmlDok = resObjekt.responseXML;` zuerst eine Variable angelegt, um nicht jedes Mal die ausgeschriebene Version für den Zugriff auf das XML-Dokument verwenden zu müssen. Die Variable `text`, die in Zeile 3 eingeführt wird, wird in der Funktion sukzessive zusammengesetzt und erst in Zeile 67 mit `document.getElementById("antwort").innerHTML = text;` dem `<span>`-Container zugewiesen.

In Zeile 3 und 4 zeigen die Anweisungen `text="<table border='1'>" + "<tr><th>XPath</th><th>Wert</th><th>Beschreibung</th></tr>";` bereits, dass die Ausgabe in dem `<span>`-Container als Tabelle erfolgen wird. Diese besteht aus drei Spalten. In der ersten Spalte wird der genaue Zugriffsweg auf das XML-Dokument (mit dem Alias `xmlDok` für `resObjekt.responseXML`), in der zweiten Spalte der damit ermittelte Wert und in Spalte 3 eine kurze Beschreibung des Zugriffs zu finden sein. In den folgenden Zeilen des Quelltextes arbeiten wir uns sukzessive von der Wurzel über die Kindelemente vor und geben dabei verschiedene Eigenschaften sowie Rückgabewerte von `node`-Methoden aus.

Ab Zeile 24 nähern wir uns den eigentlichen Inhalten in dem XML-Dokument. Und da wir hier konsequent mit dem Array `childNotes` arbeiten, bieten sich Schleifen zum Durchlauf geradezu an, um alle Zeicheninhalte des XML-Dokuments zu erhalten. Wir benötigen hier in unserem Beispiel eine dreifach verschachtelte Schleifenstruktur, da wir vom Wurzelelement aus drei Ebenen betrachten. Die äußere Schleife von Zeile 24 bis 69 durchläuft alle direkten Kindelemente des Wurzelelements. Dafür werden dann der jeweilige Knotentyp, ob Kindelemente vorhanden sind und die Anzahl der Kindknoten ausgegeben. Ab Zeile 38 kommen wir zur nächsten Ebene im Baum – die `<titel>`-Elemente. Die Schleife von Zeile 38 bis 68 durchläuft diese für jeden übergeordneten Elementcontainer. Beachten Sie die Zeilen 45 bis 49 (`text += "<tr><td> xmlDok.firstChild.childNodes[" + i + "].childNodes[" + j + "].getAttributeNode('interpret')</td><td>" + xmlDok.firstChild.childNodes[i].childNodes[j].getAttributeNode ('interpret').nodeValue + "</td><td>Zugriff auf das Attribut in der Ebene titel</td></tr>";`). Dies ist der Zugriff auf die jeweiligen Attribute.

Ab Zeile 52 erreichen wir die nächste Ebene im Baum – die eigentlichen Textknoten mit den gewünschten Zeicheninhalten. Die Schleife von Zeile 52 bis 67 gibt zu diesen Knoten verschiedene Informationen aus, insbesondere in den Zeilen 60 bis 66 den Zeicheninhalt (`text += "<tr><td>xmlDok.firstChild.childNodes[" + i + "].childNodes[" + j + "].childNodes[" + k + "].data </td><td>" + xmlDok.firstChild.childNodes[i].childNodes[j].childNodes[k].data + "</td><td>Inhalt Textknoten unterhalb von titel</td></tr>";`).



XPath	Wert	Beschreibung
xmlDok.nodeType	9	Dokument
xmlDok.hasChildNodes()	true	Gibt es Kindelemente?
xmlDok.childNodes.length	1	Ein Wurzelelement
xmlDok.firstChild.nodeType	1	Typ Element - das Wurzelelement
xmlDok.firstChild.hasChildNodes()	true	Wurzelelement hat Kindelemente?
xmlDok.firstChild.childNodes.length	3	Anzahl Kindelemente in Wurzelelement
xmlDok.firstChild.childNodes[0].nodeType	1	Direktes Kindelement des Wurzelelements - Knotentyp
xmlDok.firstChild.childNodes[0].hasChildNodes()	true	Direktes Kindelement des Wurzelelements - Kindknoten vorhanden?
xmlDok.firstChild.childNodes[0].childNodes.length	3	Direktes Kindelement des Wurzelelements - Anzahl Kindknoten
xmlDok.firstChild.childNodes[0].childNodes[0].nodeType	1	Knotentyp Ebene titel
xmlDok.firstChild.childNodes[0].childNodes[0].getAttributeNode('interpret')	Deep Purple	Zugriff auf das Attribut in der Ebene titel
xmlDok.firstChild.childNodes[0].childNodes[0].childNodes[0].nodeType	3	Knotentyp unterhalb von titel
xmlDok.firstChild.childNodes[0].childNodes[0].childNodes[0].data	Smoke on the water	Inhalt Textknoten unterhalb von titel
xmlDok.firstChild.childNodes[0].childNodes[1].nodeType	1	Knotentyp Ebene titel
xmlDok.firstChild.childNodes[0].childNodes[1].getAttributeNode('interpret')	Led Zep	Zugriff auf das Attribut in der Ebene titel
xmlDok.firstChild.childNodes[0].childNodes[1].childNodes[0].nodeType	3	Knotentyp unterhalb von titel
xmlDok.firstChild.childNodes[0].childNodes[1].childNodes[0].data	Stairway to heaven	Inhalt Textknoten unterhalb von titel
xmlDok.firstChild.childNodes[0].childNodes[2].nodeType	1	Knotentyp Ebene titel

Abbildung 8.7: Die aufbereitete Ausgabe der zurückgelieferten XML-Antwort

Um nur die reinen Inhalte und die Werte des jeweiligen Attributs von dem Elternelement `<titel>` zu erhalten, können Sie folgende stark vereinfachte Variante der Funktion verwenden:

Listing 8.16: Der Zugriff auf den Wert des Textknotens und den Attributwert des Elternelements

```
01 function handleResponse() {
02   xmlDok = resObjekt.responseXML;
03   text="";
04   if(resObjekt.readyState == 4){
05   // Alle direkten Kindelemente des Wurzelelements
06     for(i = 0;i < xmlDok.firstChild.childNodes.length; i++) {
```

```
07        text += xmlDok.getElementsByTagName("titel")[i].childNodes[0].data + " - ";
08        text += xmlDok.getElementsByTagName("titel")[i].getAttributeNode ↵
          ('interpret').nodeValue + "<br>";
09      }
10      document.getElementById("antwort").innerHTML = text;
11    }
12 }
```

Beachten Sie, dass wir hier nur den ersten direkten Kindcontainer von `<musik>` durchlaufen.

Abbildung 8.8: Der Wert des Textknotens, gefolgt von dem Attributwert

Achtung

Beachten Sie, dass auch bei diesem Beispiel einige Browser mit der clientseitigen Auswertung von XML-Strukturen auf diese Art nicht zurechtkommen.

8.7.3 Daten per node in der Webseite bereitstellen

Es ist wie gesagt oft nicht sinnvoll, die Aufbereitung einer komplexen XML-Antwort im Client vorzunehmen. Aber das `node`-Objekt hat noch eine andere sinnvolle Anwendung für AJAX zu bieten. Statt `innerHTML` können Sie die Eigenschaften und Methoden von `node` nutzen, um Daten in der Webseite anzuzeigen. Insbesondere können Sie damit ja auch neue Knoten erzeugen, die sich unter Umständen sogar erst dynamisch zur Laufzeit ergeben. Stellen Sie sich den Fall vor, dass die Inhalte einer Auswahlliste, also deren Einträge, sich dynamisch zur Laufzeit ergeben. Das ist genau das, was bei

Google suggest passiert. Zwar ist es im Grunde keine große Aktion, die gesamte HTML-Struktur der Auswahlliste zu schicken und dann mit `innerHTML` in der Webseite anzuzeigen. Aber es geht auch anders. Lassen Sie uns das Beispiel angehen.

Erzeugen einer Auswahlliste aus XML-Daten

In unserer bisher verwendeten Webseite wird nicht viel geändert. Nur soll nun statt eines `<span>`-Containers ein leerer `<select>`-Container notiert werden:

Listing 8.17: Austausch des ‹span›-Elements durch ein ‹select›-Element

```
<form><select id="antwort"></select></form>
```

Das führt zu einer leeren Auswahlliste.

Abbildung 8.9: Eine leere Auswahlliste

Diese Auswahlliste soll nun gefüllt werden, sobald der Anwender eine Taste drückt. Es ist klar, dass Sie auf Serverseite damit eine komplexe Geschäftslogik verbinden und damit die Listeneinträge angepasst generieren können. Für uns soll diese Geschäftslogik aber nur als Black-Box vorhanden sein und wir erhalten einfach Daten im XML-Format.

Betrachten wir die neue Variante von `handleResponse()`. Dabei erzeugen wir dynamisch neue Knoten in der Webseite. Die Daten der AJAX-Anfrage werden verwendet, um in dem leeren Listenfeld Einträge zu erzeugen:

Listing 8.18: Dynamisches Erzeugen von Knoten

```
01 function handleResponse() {
02   xmlDok = resObjekt.responseXML;
03    if(resObjekt.readyState == 4){
04    // Alle direkten Kindelemente des Wurzelelements
05     for(i = 0;i < xmlDok.firstChild.childNodes.length; i++)  {
06       newOption = document.createElement("option");
07       newOText = document.createTextNode(xmlDok.getElementsByTagName("titel") ↵
         [i].childNodes[0].data);
```

```
08        document.getElementById("antwort").appendChild(newOption);
09        document.getElementsByTagName("option")[i].appendChild(newOText);
10      }
11    }
12 }
```

Wir erhalten als Antwort wieder ein XML-Dokument, von dem wir die Elemente des ersten Kindelements durchlaufen. In Zeile 6 kommt eine Methode des `document`-Objekts zum Einsatz, um ein neues Element vom Typ `option` zu erzeugen (`newOption = document.createElement("option");`). In Zeile 7 erzeugen wir mit `newOText = document.createTextNode(xmlDok.getElementsByTagName("titel")[i].childNodes[0].data);` einen neuen Textknoten. Als Wert verwenden wir den Wert des Textknotens aus dem XML-File des aktuellen Durchlaufs der Schleife. In Zeile 8 wird dann mit `appendChild()` an den bestehenden Elementknoten mit der ID `antwort` (dem `<select>`-Element) ein neues `<option>`-Element als Kindelement angefügt (`document.getElementById("antwort").appendChild(newOption);`). In Zeile 9 fügen wir schließlich mit `document.getElementsByTagName("option")[i].appendChild(newOText);` dem aktuellen `<option>`-Element einen Textknoten als Kindelement hinzu. Dieser hat den Wert aus der XML-Datei.

Abbildung 8.10: Die dynamisch erweiterte Liste

Achtung

Beachten Sie, dass auch bei diesem Beispiel einige Browser mit der clientseitigen Auswertung von XML-Strukturen auf diese Art nicht zurechtkommen.

Sie können also mit Methoden wie `appendChild()`, `appendData()`, `insertBefore()`, `insertData()`, `setAttribute()` und `setAttributeNote()` in einem Baum neue Knoten dynamisch zur Laufzeit ergänzen. Diese können vorher auf verschiedenste Weise erzeugt werden (kopiert oder mit Methoden des `document`-Objekts erstellt). Natürlich funktioniert das Beseitigen oder Ersetzen von Knoten analog. Damit stehen Ihnen alle Möglichkeiten offen, in einer Webseite dynamisch neue Elemente zu ergänzen, wegzunehmen oder auszutauschen – in Verbindung mit XML, AJAX allgemein oder auch nur reinem DHTML.

8.8 XML-Verarbeitung mit Java

Zum Abschluss dieses Kapitels soll noch ein kurzer Einblick in die XML-Verarbeitung mit Java gegeben werden. Natürlich kann das kein vollständiger Einblick werden. Und ebenso müssen für diesen Abschnitt elementare Java-Kenntnisse zwingend vorhanden sein. Dennoch – um die folgenden Ausführungen im folgenden Kapitel zu serverseitigem Java anwenden zu können (wenn dort bereits XML aufbereitet wird), sollten Sie sich mit XML-Verarbeitung unter Java etwas beschäftigen.

8.8.1 Standard-APIs zur XML-Verarbeitung

Im Grunde ist die Verarbeitung von XML mit Java (oder einer anderen Programmiersprache) eine recht einfache Angelegenheit. Vor allem das Schreiben einer XML-Datei ist sehr einfach. Sie erstellen nicht mehr und nicht weniger als eine Klartextdatei. Es ist allerdings Ihre Aufgabe, den zu schreibenden Text vorher nach XML-Regeln zusammenzusetzen. Wenn ein XML-Dokument in Form einer Klartextdatei vorliegt, können Sie es auch wie jede gewöhnliche Klartextdatei einlesen. Dazu nutzen Sie unter Java Standardklassen, die Sie in `java.io` finden und auch zum Einlesen gewöhnlicher Dateien verwenden können. Was Sie dann aber tun müssen, ist die Identifikation von XML-Elementen und -Strukturen sowie die Überprüfung der Wohlgeformtheit eines Dokuments. Die Regeln dafür sind klar, aber es ist eine Menge Aufwand, dies alles von Hand zu programmieren. Wenn Sie XML-Dateien mit Java verarbeiten, greifen Sie deshalb in der Praxis auf vorgefertigte APIs zurück, die in verschiedenen Projekten vorangetrieben werden. In diesen APIs werden Sie weit entwickelte XML-Parser und zahlreiche Methoden zur XML-Verarbeitung finden, die Sie »nur« anwenden müssen.

8.8.2 XML-Verarbeitung mit SAX

Die erste verbreitete und standardisierte Möglichkeit, XML-Dokumente zu verarbeiten, bot das **Simple API for XML** (**SAX**). SAX ist aber weder ein Standard von Sun noch des W3C und eignet sich vor allem für die Verarbeitung großer XML-Dokumente sowie bei Fällen, in denen es auf performante, ressourcenschonende Verarbeitung ankommt. Dabei betrachtet man unter SAX beim Parsen eines XML-Dokuments

stets nur das aktuelle Element und hat keine Möglichkeit, auf vorhergehende Elemente zurückzugreifen. Ebenso kann man zum Zeitpunkt der Verarbeitung noch keine Aussage über zukünftige Inhalte des XML-Dokuments machen. Dies beeinflusst massiv die Art, wie und wann Inhalte verarbeitet werden. Ein XML-Parser durchläuft beim Lesen das XML-Dokument grundsätzlich sequenziell und ruft beim Vorfinden bestimmter XML-Strukturen eine Reihe von Standardmethoden auf (so genannte **Callback-Methoden**). Es gibt insgesamt elf solcher Callback-Methoden, die in der Schnittstelle `org.xml.sax.ContentHandler` definiert sind. Diese Schnittstelle wird in eine allgemeine Klasse zur XML-Verarbeitung mit SAX implementiert.

8.8.3 XML-Verarbeitung und DOM

SAX hat als Verarbeitungskonzept von XML-Daten insbesondere bei großen XML-Dateien seine Berechtigung, aber die unmittelbare Verwertung von Daten zum Zeitpunkt des Parsens ist nicht immer ideal. Aber man kann XML-Daten natürlich auch als DOM-Baum sehen und diese baumartige Struktur erst nach dem Beenden des Parsens verwerten. Parsen von XML-Dateien mit DOM baut im Hintergrund auf einem SAX-Parser auf. Nur werden die von diesem Parser zurückgegebenen Daten in einem Objektmodell, das den DOM-Baum abbildet, zwischengeparkt.

Nun stellt Sun in seinem Java-API zur XML-Verarbeitung explizite Techniken zum Umgang mit DOM bereit. Allerdings entspricht das DOM-Konzept nicht so ganz der Denkweise von Java. Im DOM-Konzept wird jede Struktur innerhalb eines XML-Dokuments als Knoten verstanden – vom XML-Dokument über Elemente (allerdings nicht Endeelemente) bis hin zu den Attributen und Elementinhalten und selbst den Kommentaren und ignorierbarem Text. DOM kennt insgesamt zwölf verschiedene Knotenformen. Alle Knotenformen werden gleichberechtigt in einem Baum abgebildet. Das macht DOM einerseits mächtig und universell, andererseits aber auch komplex und unhandlich. DOM ist explizit sprachunabhängig, was aber auch bedeutet, dass es auf eine Java-Optimierung verzichtet. Alle Techniken von DOM sind so konzipiert, dass sie von allen denkbaren Programmiertechniken aus verwendet werden können. Spezielle Java-Philosophien und -Techniken können damit jedoch nicht unterstützt werden. Auf Java optimierte Erweiterungen von DOM sind – wie auch DOM selbst – nun dafür da, sowohl aus XML-Daten beim Parsen im Speicher des Rechners eine baumartige Struktur aufzubauen, die nach dem Beenden des Parsens verarbeitet werden kann, als auch den umgekehrten Weg zu unterstützen. Aber dabei wird im Gegensatz zu DOM auf bewährte Java-Techniken gesetzt, was einem Java-Programmierer sehr viele Vorteile gegenüber dem allgemeinen DOM-Konzept bietet. Ein solches Konzept ist **JDOM** (Java Document Object Model). JDOM ist zwar keine Entwicklung von Sun, sondern eine freie API des JDOM OpenSource-Projekts. Dennoch unterstützt Sun dieses API. Allerdings müssen Sie – solange JDOM nicht zum Standard-API von Java gezählt wird – das JDOM-API nachträglich installieren und in der Java-Laufzeitumgebung verfügbar machen. Unter *http://jdom.org/* können Sie das API samt Dokumentation laden. Das Paket `org.jdom` ist die Basis der XML-Verarbeitung mit JDOM. Dieses stellt nahezu alle Klassen bereit, die bei der Erstellung und

Verarbeitung von XML notwendig sind. Beim Umgang mit JDOM ist vor allem interessant, dass Sie nicht eine XML-Datei im Mittelpunkt aller Aktivitäten haben müssen, sondern Ihr Hauptaugenmerk der Repräsentation der XML-Daten im Hauptspeicher gilt. Hier agieren Sie mit Java und es spielt keine Rolle, ob die XML-Daten aus einer Datei oder anderen Quellen stammen. Neben JDOM existiert mit dem **DOM4J** (*http://www.dom4j.org*) ein weiteres populäres API zur Verarbeitung von XML mit Java. Sie müssten auch dieses bei einem Einsatz nachinstallieren und den Klassenpfad gegebenenfalls anpassen.

8.9 Zusammenfassung

Sie haben in diesem Kapitel die elementaren Grundlagen von XML samt einem kurzen Einblick in die Validierung von XML-Dokumenten sowie XPath kennen gelernt. XML ist eine strenge datenzentrierte Metasprache, deren Dokumente als minimale Forderung wohlgeformt sein müssen. Das bedeutet, sie befolgen alle Grundregeln der Syntax. Wenn ein XML-Dokument jedoch gültig sein soll, muss es Regeln einer weiteren Grammatik einhalten. Diese werden in Form einer DTD oder eines Schemas festgelegt. Ein XML-Dokument selbst ist – wie auch ein (X)HTML-Dokument – als Baum zu verstehen, der aus Knoten und Achsen besteht. Auf diesem können Sie navigieren. Grundlage ist die Abfragesprache XPath, die in den Eigenschaften und Methoden des `node`-Objekts eine praktische Anwendung findet. Damit sowie mit der Eigenschaft `responseXML` des `XMLHttpRequest`-Objekts stehen Ihnen verschiedene Möglichkeiten zur Verfügung, beim Datenaustausch mit AJAX XML hervorragend einzusetzen.

9 Serverseitige Java-Programmierung für AJAX

Die Datenanforderung von AJAX-Applikationen erzwingt die Verarbeitung auf dem Webserver, sofern Sie nicht die gesamte Geschäftslogik auf dem Client halten und einfach nur vollständige Dateien als neue Daten nachladen wollen. Dort haben Sie allerdings die freie Auswahl, welche Technologie Sie einsetzen wollen. Auf der einen Seite wird es in der Praxis sinnvoll sein, die meiste oder gar sämtliche Geschäftslogik einer AJAX-Applikation auf dem Server zu programmieren. Auf der anderen Seite stellt AJAX aber keine konkrete Anforderung an die dort verwendete Technologie. Es stehen Ihnen alle Möglichkeiten offen. Bei Programmiersprachen können Sie etwa **Perl** verwenden, was hauptsächlich in älteren Anwendungen Standard ist und heutzutage vor allem noch in sehr professionellen und großen Server-Applikationen verwendet wird, oder **ASP** (**Active Server Pages**) und dessen Nachfolger **ASP.NET**. Beide letztgenannten Techniken gehen auf Microsoft zurück. Insbesondere ist es bei ASP interessant, dass Sie auf Serverseite bei der Erstellung von ASP-Skripts neben VBScript ebenso JavaScript verwenden können. Eine weitere serverseitig eingesetzte Technologie nennt sich **PHP** (eine OpenSource-Entwicklung). Ursprünglich war es die Abkürzung für **Personal Home Page Tools**, aber mittlerweile wird es als so genanntes rekursives Akronym[1] verwendet und bedeutet **PHP Hypertext Preprocessor**. PHP ist eine Skriptsprache, deren Syntax wie bei JavaScript stark an C angelehnt ist. Dennoch gilt PHP von allen serverseitigen Programmiersprachen als am einfachsten zu erlernen, denn gerade in Hinsicht auf Datentypen und den Umgang mit Variablen läuft bei PHP nahezu alles automatisch im Hintergrund. Dazu bietet PHP eine sehr einfache und recht weitreichende Datenbankunterstützung. Der Umgang mit Formulardaten bzw. Datenübermittlung per AJAX ist ein Kinderspiel (was bei vielen Konkurrenztechniken nicht direkt gesagt werden kann) und es gibt mittlerweile unzählige Funktionsbibliotheken für die verschiedensten Anwendungen.

Im professionellen Umfeld wird jedoch bei serverseitigen Techniken sehr oft auf Java gesetzt, etwa in Form von **JavaServer Pages** (**JSP**), **Java Servlets** oder auch **JavaServer Faces**, **Struts** oder **Java Enterprise Beans** (**JEB**). Wir werden uns in diesem Buch auf JSP und Java-Servlets konzentrieren. Java bietet sich immer dann an, wenn sehr

1 *Eine Selbstwiederholung. Das bedeutet, in der ausgeschriebenen Variante der Abkürzung wiederholt sich die Abkürzung. Das haben Sie etwa auch bei GNU – das steht für GNU is not Unix.*

stabile, sichere und hoch komplexe Anwendungen geschrieben werden sollen und wenn sich Webanwendungen bzw. Ajax-Anwendungen in bestehende Infrastrukturen auf Java-Basis integrieren sollen. Das bedeutet jedoch nicht, dass Sie nicht auch einfache und kleinere AJAX-Projekte auf Basis von Java aufsetzen können. Java zeichnet sich nicht nur durch hohe Sicherheit und Stabilität aus, sondern auch durch einen reichen Fundus an Standardpaketen, mit denen Sie z.B. Datenanwendungen sehr komfortabel realisieren können. Dabei deutet das Stichwort »Datenbankunterstützung« an, dass Sie auf Serverseite natürlich auch mit Datenbanken umgehen können – und bei etwas aufwändigeren AJAX-Applikationen auch müssen, denn bei großen Datenbeständen lassen sich Datennachforderungen nur damit handhaben. Auch bei Datenbanken oder weiteren ergänzenden Programmen auf dem Server macht AJAX keine explizite Vorgabe. Wie PHP arbeitet auch Java beispielsweise hervorragend mit **MySQL** zusammen, eine oft verwendete OpenSource-Datenbank im Webumfeld. Wir werden deshalb auch MySQL als Vertreter der Datenbankzunft hier einsetzen.

9.1 Grundlagen serverseitiger Programmierung

Wer sich (bisher noch) nicht mit serverseitiger Programmierung auskennt, vermutet dahinter oft etwas sehr Kompliziertes und Geheimnisvolles. Aber dem ist eigentlich nicht so. Es gibt nur ein paar Feinheiten zu beachten.

9.1.1 Unterschiede und Gemeinsamkeiten zwischen serverseitiger und clientseitiger Programmierung

Im Grunde unterscheidet sich die serverseitige Programmierung kaum von der Programmierung auf einem Client. Es gibt Skripte und Interpreter oder auch eigenständige Programme, die vom Server aus aufgerufen werden. Zumindest zwei wesentliche Unterschiede sind jedoch zu beachten.

Authentifizierung und Sicherheit

Sie müssen bei der Ausführung serverseitiger Aktionen bedenken, dass die serverseitigen Skripte und Programme in der Regel nicht vom gleichen Rechner aus aufgerufen werden, auf dem sie ausgeführt werden. Ebenso können sie meist von mehreren Anwendern verwendet werden und laufen auf einem möglicherweise fremden Rechner (denken Sie an einen Internet Provider). Dementsprechend muss ein Authentifizierungs- und Sicherheitssystem gewährleisten, dass nur berechtigte Anwender Skripte und Programme ausführen dürfen und ein Server auch nicht kompromitiert werden kann. Wie so ein Authentifizierungs- und Sicherheitssystem konkret implementiert wird, hängt vom Server, der Serverplattform, der Programmiertechnik, der Art des Netzwerks und des Zugriffs darauf sowie diversen weiteren Faktoren ab.

Im Fall eines Webservers oder auch Java-Containers wie Tomcat wird im Rahmen eines solchen Konzepts immer nur eine bestimmte Verzeichnisstruktur bereitgestellt, die öffentlich zugänglich ist. Dabei bedeutet »öffentlich zugänglich«, dass nur ganz

bestimmte Verzeichnisse der Verzeichnisstruktur eines Hosts zum Lesen, Schreiben oder auch beidem freigegeben sind. Oft findet man in so einem Konzept den Fall, dass eine Verzeichnisansicht verboten ist, aber gezielt Dateien abgerufen werden können (etwa eine HTML-Seite). Ebenso können bestimmte Verzeichnisse zur Ausführung von aktiven Operationen auf der Server-Plattform (etwa PHP- oder ASP-Skripts) freigegeben oder auch gesperrt werden.

Verfügbare Objekte und Funktionen

Wenn Sie an serverseitige Skripte oder Programme denken, die Objekte verwenden, werden auf dem Server natürlich andere Objekte bereitstehen als auf dem Client. Auf dem Server sind zum Beispiel Objekte wenig sinnvoll, die das Browser-Fenster oder die Browser-Version repräsentieren. Dies sind Objekte, die nur auf dem aufrufenden Client-Rechner Sinn ergeben. Dafür wird es andere Objekte geben, die speziell auf dem Server sinnvoll sind. Das gilt auch für Funktionen bzw. Methoden. Wenn auf dem Client eine Methode beispielsweise unmittelbar etwas auf dem Bildschirm des Clients ausgibt, wird eine analoge Methode auf dem Server in der Regel die Ausgabe flexibler gestalten und meist als eine von mehreren Möglichkeiten die Ausgabe auf einem entfernten Client gestatten, der eine Anforderung gestellt hat.

Gemeinsamkeiten zwischen Server und Client

Wenn Sie von den verfügbaren Objekten und Funktionen sowie dem Authentifizierungs- und Sicherheitssystem absehen, unterscheidet sich zum Beispiel ein JavaScript auf dem Client oder Server (das kann wie gesagt etwa bei ASP verwendet werden) nicht viel. Im Fall von Java ist Ihnen sicherlich klar, dass Sie Java-Applikationen sowohl auf dem Client als auch auf dem Server ausführen können. Aber auch PHP, das von vielen als reine Server-Sprache verstanden wird, kann direkt auf einem lokalen Rechner ausgeführt werden. Die reine Programmierung ist identisch. Nur wird der hauptsächliche Nutzen im Rahmen von serverseitiger Programmierung entfaltet. Das gilt auch für Perl oder diverse weitere Sprachen, die hauptsächlich auf dem Server eingesetzt werden.

Abbildung 9.1: Aufruf eines PHP-Skripts in der Konsole auf dem Client – der HTML-Tag wird dort natürlich nicht interpretiert, sondern als Text angezeigt.

9.1.2 Voraussetzungen zur Ausführung von serverseitigen Skripten und Programmen

Wenn nun Applikationen auf einem Server laufen sollen, muss selbstredend ein Server mit Zugang zu eventuell geforderten ergänzenden Modulen oder externen Programmen verfügbar sein. Sie haben ja auch schon bisher Zugang zu einem Webserver mit entsprechender Unterstützung benötigt, um AJAX-Anfragen entgegennehmen und verarbeiten zu können – selbst für den einfachsten Fall, dass aufgrund einer AJAX-Anfrage einfach nur eine neue Datei als Antwort geschickt wurde. Wenn Sie eine bestimmte Programmiersprache einsetzen wollen, muss der Webserver Unterstützung dafür direkt implementiert haben oder es muss eine externe Laufzeitumgebung, ein Interpreter oder ein Programm verfügbar sein. Die vorgeschlagene XAMPP-Lösung bietet Ihnen zum Beispiel PHP-Unterstützung samt MySQL als Datenbank und Tomcat ist unsere Laufzeitumgebung für Java-Servlets oder JSP, die oft ein XAMPP-System ergänzt.

In der Praxis hängt die Wahl des konkreten Webservers von Ihrem Internet Provider ab, wenn Sie keinen eigenen Server betreiben. Und da stellt sich die Frage, ob und welche der serverseitigen Techniken der Webserver bei Ihrem Provider unterstützt? Viele Provider setzen aus Kosten- und Sicherheitsgründen mittlerweile auf Apache als Webserver in Verbindung mit PHP und MySQL. Oft wird diese Kombination durch **Jakarta Tomcat** erweitert, um auch Unterstützung für JSP und Java Servlets anzubieten. Dies ist aber in der Regel mit etwas teureren Verträgen gekoppelt. Aber auch die anderen angesprochenen Techniken werden häufig – insbesondere bei höherpreisigen Angeboten – zur Verfügung gestellt. Wenn Sie ASP oder ASP.NET als serverseitige Technik einsetzen wollen, bleibt Ihnen im Grunde nur der Einsatz eines Microsoft-Webservers (wobei auch einige andere Webserver diese Techniken unterstützen – jedoch meist eingeschränkt). Für die Entwicklungsphase sind Sie als Windows-Anwender aber in einer guten Situation. Windows bringt bereits – in den meisten Versionen zumindest – einen Webserver automatisch mit. Oder er lässt sich nachinstallieren – entweder eine ältere Lightvariante namens **PWS** (**Personal Web Server**) für ältere Windows-Versionen oder den in der gleichen Liga wie Apache angesiedelten **IIS** (**Internet Information Server**).

9.1.3 Serverseitige Skripts und Programme zum Laufen bringen

Bei einem Webserver ist es ein zentrales Faktum, dass ein Skript oder ergänzendes Programm vom Webbrowser immer indirekt über den Webserver bzw. ein Containerprogramm wie Tomcat aufgerufen wird. Eine Anfrage durch einen Client wird vom Webserver dann an einen entsprechenden Interpreter bzw. eine Laufzeitumgebung weitergereicht oder ein nachgeschaltetes Programm wie ein Datenbankserver wird aufgerufen. Wenn Sie nun einen passenden Webserver zur Verfügung haben, laden Sie Ihre Skripte oder serverseitigen Progamme in der Regel in ein entsprechendes Verzeichnis auf dem Server. Welches Verzeichnis für die Ausführung von Skripten oder Programmen freigegeben ist, unterscheidet sich je nach Server und der kon-

kreten Konfiguration. In der Praxis werden Sie diese Informationen von Ihrem Internet Provider oder Administrator erhalten oder selbst einstellen[2]. Wir hatten bei der Wahl vom XAMPP mit Apache als Webserver bisher ja das Verzeichnis *htdocs* oder ein Unterverzeichnis davon verwendet und bei Tomcat das Verzeichnis *webapps*. Über den Server rufen wir dann das Skript oder Programm so auf, wie wir es in allen Beispielen in dem Buch gemacht haben[3] – wobei Sie für die Ausführung von Java-Servlets noch einige weitere Details beachten müssen, die wir in diesem Kapitel behandeln, wenn wir Servlets besprechen.

9.2 Java auf einem Webserver

Wir wollen zur Vertiefung unserer bisherigen AJAX-Skripte als kleinen Einblick nun etwas genauer Java anschauen und wie es auf dem Server verwendet werden kann. Dabei können wir im Rahmen dieses AJAX-Buchs (natürlich) keine ausführlichen Details zu allen Feinheiten von Java oder gar den Details serverseitiger Java-Programmierung besprechen. Bezüglich Java sollten Ihnen auf jeden Fall bereits die wesentlichen Syntaxstrukturen wie Blöcke, Schleifen, Entscheidungsanweisungen, Funktionen oder Kommentare bekannt sein. Ebenso möchte ich voraussetzen, dass Sie das objektorientierte Konzept von Java in Grundzügen beherrschen. Sie sollten also wissen, was in Java eine Klasse ist und wie man sie schreibt, wie man ein Objekt erzeugt, wie man Klassen und Objekte benutzen kann, was Vererbung darstellt und wie man mit dem JDK oder einer Java-IDE vom Quelltext zur fertigen Java-Applikation kommt[4]. Das bedeutet nicht, dass Sie Java-Profi sein müssen. Sie sollten nur in der Lage sein, die folgenden Quelltexte in diesem Kapitel zu verstehen oder in weiterführenden Java-Quellen die entsprechenden Details nachzulesen.

9.2.1 Was sind JSP und Servlets?

Eine JSP ist ein Skript und damit reiner Klartext. Es umfasst im Allgemeinen folgende Bestandteile, die jedoch nicht alle zwingend vorhanden sein müssen:

- Reiner Text. Das kann auch (X)HTML- oder XML-Code sein, der aber auf Serverseite nur als reiner Text verstanden wird.
- Skript-Begrenzer, die sich von denen in (X)HTML- oder XML unterscheiden. Darin befindet sich der eigentliche JSP-Skriptcode. Man redet hier von so genannten **JSP-Tags**. Dies umfasst Ausdrücke, Deklarationen, Aktionen, Direktiven und so genannte Scriptlets.

2 *Gegebenenfalls unter Zuhilfenahme der Dokumentation Ihres konkreten Server-Programms.*
3 *Für Details beachten Sie die Ausführungen in Kapitel 2 (bei der Behandlung unseres ersten Beispiels).*
4 *Letzteres benötigen Sie allerdings nur für Servlets – bei JSP läuft die Übersetzung im Hintergrund und Sie müssen nur den Quellcode speichern.*

- In der JSP stehen Ihnen auch so genannte **implizite Objekte** (Objekte, die Ihnen direkt in jedem JSP zur Verfügung stehen) bereit, die Sie direkt nutzen können, z. B. das `request`-Objekt für Anfragen, das `response`-Objekt für Antworten, das `session`-Objekt zum Verfolgen von Sitzungen oder das `config`-Objekt zur Konfiguration.
- Des Weiteren können Sie **Java Beans** für die Implementierung Ihrer Geschäftslogik verwenden (das führt aber im Rahmen des Buchs zu weit).

Grundsätzlich ist ein JSP-Skript eine Textdatei, die meist die Erweiterung *.jsp* hat. In einer JSP-Datei befinden sich die verschiedenen Skriptbefehle, welche auf dem Java-Container ausgeführt werden sollen. Diese Skriptpassagen können auf mehrere Skriptcontainer aufgeteilt werden, die immer wieder von reinen Textpassagen respektive HTML-Code durchzogen sein können.

Aus dem gesamten Skript wird letztendlich dynamisch (X)HTML- oder XML-Inhalt generiert und zum Client geschickt. JSP-Skripts werden dazu auf dem Server im Hintergrund mit einem **JSP-Compiler** in echten Java-Quellcode umgewandelt. Dieser Quellcode wird im Anschluss wie gewöhnlicher Java-Quellcode durch den **Java-Compiler** in Bytecode umgewandelt. Auch das läuft im Hintergrund ab und erfordert keine aktive Mitarbeit des Programmierers. Bei dieser Übersetzung des JSP-Quellcodes entsteht ein **Java Servlet** oder auch eine ganze Gruppe verschiedener Java Servlets. Servlets selbst wiederum stellen eigenständige Java-Applikationen dar, die im Rahmen des Java-Containers auf Server-Seite ausgeführt werden. Ein Servlet ist physikalisch eine *.class*-Datei – ganz wie gewöhnliche Java-Applikationen, die auf dem Client ausgeführt werden.

Servlets müssen natürlich nicht indirekt über JSP generiert werden. Sie können ebenso direkt erzeugt werden, indem Sie einfach Java-Quellcode schreiben und dieser dann mit einem Java-Compiler kompiliert wird. Natürlich müssen Sie sich dabei an gewisse Vorgaben halten, nach denen ein Servlet-Quellcode geschrieben werden muss. In jedem Fall werden jedoch die erzeugten Java-Klassen dann vom Server an seine zugeordnete JVM weitergereicht und darüber ausgeführt.

Hinweis

Viele Webseitenersteller, die eher aus dem Designer-Lager kommen, sehen den entscheidenden Vorteil des JSP-Konzepts gegenüber dem Servlet-Konzept in der Trennung der grafischen Seite und der Programmierung. Das erleichtert die Verteilung von Aufgaben, so dass eine JSP auch teilweise durch einen Web-Designer ohne Java-Kenntnisse erstellt werden kann.

9.3 JavaServer Pages

Wir betrachten nun zuerst genauere Details zu JSP. Damit sollten viele Dinge klar werden, die wir bereits im Rahmen dieses Buchs mehrfach verwendet haben.

9.3.1 Wichtige JSP-Strukturen

Wenn Sie eine JSP-Datei erzeugen, schreiben Sie wie gewöhnlich eine (X)HTML- oder XML-Datei mit entsprechender Grundstruktur, die von Skriptbegrenzern und Skriptcode durchzogen sein kann. Dazu brauchen Sie nicht mehr als einen gewöhnlichen Texteditor.

Skript-Begrenzer und JSP-Tag-Elemente

Ein JSP-Skript unterteilt sich wie gesagt in reine Textpassagen, die für einen Java-Parser uninteressant sind und nicht interpretiert werden, sowie Skript-Bereiche, die er beachten und verarbeiten wird. Diese Skript-Bereiche werden in Begrenzer eingeschlossen, die in der Regel in der Form `<% ... %>` oder `<jsp:...> </jsp:... >` notiert werden. Die erste Variante wird in der Praxis sehr oft verwendet und sehr weitreichend unterstützt, die zweite Variante ist XML-konform und damit sehr flexibel (aber noch nicht flächendeckend unterstützt). In jedem Fall ist zu beachten, dass es verschiedene Facetten von JSP gibt. Insbesondere die Version 2.0 hat einige Veränderungen und vor allem Erweiterungen gebracht, deren Details wir hier nicht besprechen werden. Für unsere Ausführungen und vor allem weiterführenden Beispiele verzichten wir weitgehend auf die XML-konformen Tags und beschränken uns auf grundlegende JSP-Techniken.

Der JSP-Tag
In der Form `<% ... %>` (also ohne weitere Zeichen hinter dem Prozentzeichen am Beginn) liegt ein so genannter **JSP-Tag** vor, in dem Sie Ihre Programmlogik in Form einer beliebigen Anzahl von Java-Anweisungen einbinden und damit dynamische Seiten generieren. Beispiel:

Listing 9.1: Ein JSP-Tag

```
<%
java.util.Random zufall = new java.util.Random();
%>
```

Sie können im Grunde in einem JSP-Tag genauso programmieren, wie Sie es von der Erstellung einer normalen Java-Klasse her gewohnt sind.

Der JSP-Deklarations-Tag
Zum Definieren von Variablen und Methoden, die in einem JSP-Quellcodes verwendet werden sollen, gibt es eine eigene Deklarationsanweisung. Diese Deklarationsanweisung ist ein Spezialfall eines JSP-Tags und wird mit folgender JSP-Syntax erstellt:

Listing 9.2: Schema eines JSP-Deklarations-Tags

```
<%!
[Deklarationen]
%>
```

Beispiel:

Listing 9.3: Deklaration einer Variablen

```
<%!
java.util.Random zufall = new java.util.Random();
 %>
```

Alternativ geht das mit dem XML-konformen Äquivalent so:

Listing 9.4: Schema der XML-konformen Variante eines JSP-Deklarations-Tags

```
<jsp:declaration>
[Deklarationen]
</jsp:declaration>
```

JSP-Ausdrucks-Tags und die Ausgabe im Webbrowser
Wenn Sie auf dem Server mit JSP eine dynamische Antwort an den Client generieren wollen, schreiben Sie entweder außerhalb der Scriptlets reinen Text, HTML- oder XML-Code oder aber Sie verwenden einen JSP-Ausdrucks-Tag für den Fall, dass Sie das Ergebnis eines Ausdrucks dem Client liefern wollen. Auch damit erzeugen Sie eine Ausgabe im Anzeigebereich des aufrufenden Webbrowsers. Ihnen dürfte bekannt sein, dass **Ausdrücke** Kombinationen aus Konstanten, Zuweisungen von Variablen, Rückgabewerten von Methoden, Operatoren etc. sind. Die Elemente in einem Ausdruck werden ausgewertet und kombiniert und können dann als Wert verwendet werden. Ausdrücke können innerhalb eines JSP-Skriptlet oder aber selbstständig in Form eines so genannten **JSP-Ausdrucks-Tags** vorkommen. Die Syntax sieht wie folgt aus:

Listing 9.5: Ein JSP-Ausdrucks-Tag

```
<%= [Ausdruck] %>
```

Wie das Gleichheitszeichen andeutet, wird der Wert des Ausdrucks zugewiesen. Dies bedeutet im Fall von JSP, dass der Wert des Ausdrucks an den aufrufenden Client geschickt wird. Beispiel:

Listing 9.6: Mit diesem Ausdrucks-Tag wird der Wert an den Client geschickt und von diesem in der Webseite angezeigt.

```
<%= zufall.nextInt(49) %>
```

Hinweis

Ein JSP-Ausdrucks-Tag wird für **genau eine** einzelne Java-Anweisung verwendet.

Achtung

Die Anweisung im Inneren eines JSP-Ausdrucks-Tags wird **niemals** mit einem ; beendet. Dies würde einen Fehler erzeugen.

Eine JSP-Ausdrucksanweisung ist vergleichbar mit der normalen Java-Ausgabeanweisung `System.out.print()`. In einem der JSP-Ausdruck können nur Anweisungen aufgerufen werden, die eine Zeichenkette oder ein Objekt zurückgeben. Bei Letzterem wird implizit die Methode `toString()` aufgerufen.

Hinweis

Alternativ geht das mit dem XML-konformen Äquivalent so:

Listing 9.7: Die XML-konforme Version eines JSP-Ausdruck-Tags

```
<jsp:expression> [Ausdruck] </jsp:expression>
```

Achtung

Im Fall von AJAX wird die Antwort an den Client in der Regel kein (X)HTML-Grundgerüst haben. Sie senden ja meist nur ein Fragment, das zum Austausch von Daten in einer bestehenden (X)HTML-Seite verwendet wird. Ein vollständiges Grundgerüst mit `<html>`- oder `<body>`-Tag stört da nur und macht unnötig Probleme.

JSP-Direktiven-Tags

Direktiven-Tags sind Anweisungen in einer JSP, bestimmte Methoden auszuführen oder bestimmte Dinge durchzuführen. Das kann beispielsweise der Import verschiedener Klassen sein. In JSP kennt man drei Formen von Direktiven. Eine Direktive erkennen Sie daran, dass nach dem Prozentzeichen des einleitenden Skriptbegrenzers ein `@` steht. Schema:

Listing 9.8: Beginn einer JSP-Direktive

```
<%@...
```

Die include-Direktive
Mit der **include-Direktive** binden Sie eine Datei als statisches Objekt genau an der Stelle in den Code ein, an der diese Referenz notiert wird. Dabei wird der eingebundene Code zur Übersetzungszeit hinzugefügt. Der eingebundene Code kann z. B. eine HTML-Seite oder eine andere JSP sein. Diese wird dann in den erzeugten Servlet-Code eingebunden, übersetzt, kompiliert und dann ausgeführt. Beispiel:

Listing 9.9: Einbinden einer Webseite in das JSP

```
<%@ include file="webseite.html" %>
```

Die page-Direktive
Mit der **page-Direktive** können Sie Java-Klassen und -Pakete in Ihre JSP **importieren**. Dies entspricht der `import`-Anweisung in normalem Java-Quellcode. Beispiel:

Listing 9.10: Eine import-Direktive zum Bekanntmachen des Pakets java.net

```
<%@ page import="java.net.*" %>
```

Achtung

Beachten Sie, dass die `import`-Anweisung in Java keinen echten Import darstellt, sondern nur dafür da ist, dass Sie im folgenden Quellcode mit verkürzten Namen auf die Elemente eines Pakets zugreifen können.

Schauen wir uns ein vollständiges Beispiel für eine JSP an. In dem Beispiel wollen wir die Lottozahlen von nächster Woche berechnen.

Listing 9.11: Eine JSP, die die Lottozahlen der nächsten Woche berechnet

```
01 <%@ page import="java.util.*" %>
02 <%!
03 Random zufall = new Random();
04  %>
05 <html>
06 <body bgcolor="red" text="white">
07 Und hier sind die Lottozahlen von nächster Woche:
08 <hr />
09 <ul>
10 <% for(int i=0; i < 7; i++) { %>
11 <li>
12 <%= 1 + zufall.nextInt(48) %>
13 </li>
14 <% }%>
15 </ul>
16 </body>
17 </html>
```

Wenn Sie das Beispiel laufen lassen, erhalten Sie in Form einer Aufzählungsliste sieben zufällige Zahlen zwischen 1 und 49. Die letzte Zahl soll dabei die Zusatzzahl darstellen.

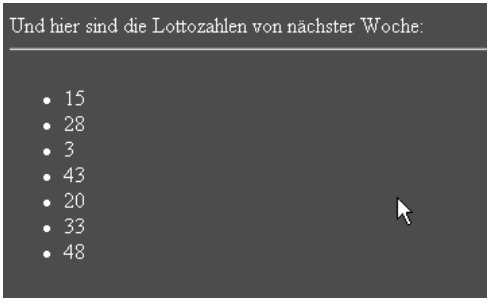


Abbildung 9.2: Die Lottozahlen von nächster Woche – die letzte Zahl ist die Zusatzzahl.

Hinweis

Beachten Sie, dass die Logik dieses Beispiels nicht voll identisch mit der einer Lottoziehung ist. Genau genommen wird hier »Ziehen mit Zurücklegen« umgesetzt. Eine Zahl kann also mehrfach gezogen werden. Damit das Beispiel aber nicht zu kompliziert wird, soll das genügen.[5]

5 *Und Sie glauben doch nicht im Ernst, dass ich weiter Bücher schreiben würde, wenn das Beispiel in der Realität funktionieren würde. ;-)).*

In Zeile 1 stellen wir über ein `import` sicher, dass im folgenden Quelltext die Klasse `Random` über einen verkürzten Namen angesprochen werden kann. Die Klasse `Random` wird zur Erzeugung eines Zufallsmechanismus (genau genommen eines **Zufallsgenerators**) verwendet. Genau das machen wir in Zeile 3, wo der Konstruktor der `Random`-Klasse aufgerufen wird. Die Referenz auf den Zufallsgenerator wird in der Variablen `zufall` gespeichert. Zeile 2 bis 4 ist dabei ein JSP-Deklarations-Tag. In den Zeilen 5 bis 9 finden Sie reinen HTML-Code. Die Zeilen 10 bis 14 definieren eine `for`-Schleife. Allerdings nicht nur in Form eines einzelnen JSP-Tags, sondern verteilt auf drei JSP-Tags, die von reinen HTML-Passagen getrennt werden.

Tag-Bibliotheken
Eine **Tag-Bibliothek** ist eine Sammlung vorgefertigter Funktionalitäten, die Sie im Rahmen eines JSP verwenden können. Diese Bibliotheken werden mit einer speziellen Direktive eingebunden. Schema:

Listing 9.12: Beginn der Referenz auf eine Taglib-Bibliothek

```
<%@ taglib ...
```

Beispiel:

Listing 9.13: Referenz auf eine Taglib-Bibliothek

```
<%@ tablib uri="tablib.tld" prefix="irgendwas" %>
```

JSP-Aktions-Tags

Mithilfe so genannter **Aktions-Tags** erweitert man die Funktionalität von JSP-Skripts. Sie können darüber in der JSP z.B. in Java erzeugte Objekte verwenden sowie andere JSPs oder Java Beans einbinden. Das gesamte Verfahren sprengt zwar bei weitem unseren Rahmen, aber ein paar kleinere Beispiele (ohne genauere Erklärung allerdings) sollen die Verwendung demonstrieren. Beispiele:

Listing 9.14: Einfügen einer anderen JSP zur Laufzeit

```
<jsp:include page="andereJSP.jsp" />
```

Listing 9.15: Bekanntmachen einer Java Bean. Bei class muss der komplette Pfad inklusive eventueller Pakete angegeben werden.

```
<jsp:useBean id="lokalerNameinJSP" class=NameBean"/>
```

Listing 9.16: Abfragen der Eigenschaft einer JavaBean

```
<jsp:getProperty name="lokalerNameinJSP" property="meineEigenschaft"/>
```

Listing 9.17: Ändern von Eigenschaften einer JavaBean

```
<jsp:setProperty name="myName" property="someProperty" value="someValue"/>
```

Listing 9.18: Weiterleiten auf andere Seite

```
<jsp:forward page="fehlerReport.jsp"/>
```

Die Datenübergabe an JSP – das request-Objekt

Die Entgegennahme von Formulareingaben oder sonst per HTTP übermittelter Daten an das JSP-Skript ist ziemlich einfach. Die Verarbeitung von solchen Anfragen ist ja insbesondere für AJAX-Applikationen eine der wichtigsten Situationen. Fast jede AJAX-Anfrage in der Praxis wird Daten an den Server schicken, die dort zur Steuerung der Antwort verwendet werden – ebenso eine Übergabe mit der `open()`-Methode oder das Anfügen eines Wertepaars an den URL bei einer Adresseingabe in der Adresszeile im Webbrowser. Dafür gibt es in jeder JSP das implizite Objekt `request`. Dieses stellt eine ganze Reihe von Methoden bereit, um die Anfrage qualifiziert zu verarbeiten. Nachfolgend finden Sie ein paar der wichtigsten Methoden:

Methode	Beschreibung
`getHeader()`	Über die Methode haben Sie die Möglichkeit, die übermittelten Header-Werte auszulesen. Als Übergabewert an die Methode wird der Bezeichner des Header-Felds angegeben. Sie erhalten als Rückgabewert einen String mit dem Wert, der für diesen Header-Eintrag übermittelt wurde.
`getHeaderNames()`	Mit dieser Methode erhalten Sie eine Aufzählung (`java.util.Enumeration`) der Namen aller Felder im Header.
`getParameter()`	Die Methode `getParameter()` ist sicherlich eine der wichtigsten Methoden überhaupt. Darüber haben Sie die Möglichkeit, die übermittelten Parameter an die JSP auszulesen. Als Übergabewert an die Methode wird der Bezeichner angegeben, der von dem aufrufenden Webbrowser als Parametername per GET oder POST übermittelt wird. Sie erhalten als Rückgabewert einen String mit dem Wert, der für diesen Parameter übermittelt wurde.
`getParameterNames()`	Mit dieser Methode erhalten Sie eine Aufzählung (`java.util.Enumeration`) aller übermittelten Parameter. Diese Methode können Sie dann verwenden, wenn Sie die übermittelten Parameter an ein JSP nicht kennen. In der Praxis wird es aber in der Regel so sein, dass Sie die Parameter schon kennen.
`getProtocol()`	Das verwendete Protokoll bei der Datenkommunikation. Dies wird zwar so gut wie immer HTTP sein, aber es gibt verschiedene Typen.
`getQueryString()`	Über diese Methode erhalten Sie den vollständigen Abfrage-String
`getRemoteAddr()`	Die Adresse vom anfragenden Client

Tabelle 9.1: Die wichtigsten Methoden des impliziten request-Objekts

Methode	Beschreibung
`getRemoteHost()`	Der anfragende Rechner
`getRemoteUser()`	Der anfragende User. Diese Information ist in vielen Fällen nicht belegt (das bedeutet, der Wert `null` wird darin zu finden sein). Allerdings gibt es Fälle, in denen einige Browser diese Information mitsenden. Dies ist in der Regel vom Anwender aber nicht gewünscht.
`getRequestURI()`	Der relative Pfad der angeforderten Ressource
`getServerName()`	Der Name des Servers
`getServerPort()`	Der Port des Servers
`getServletPath()`	Der Name des JSP oder Servlet

Tabelle 9.1: Die wichtigsten Methoden des impliziten request-Objekts (Forts.)

Das nachfolgende Beispiel zeigt in einer Ausgabe im Webbrowser die Daten, die über einige wichtige Methoden des `request`-Objekts zur Verfügung stehen. Dabei werden auch verschiedene JSP-Tags im praktischen Einsatz gezeigt:

Listing 9.19: Verwendung verschiedener Methoden des impliziten request-Objekts

```
<%@ page import="java.util.*" %>
<html>
<body bgcolor="red" text="white">
<h3 align="center">Informationen zum Server</h3>
Serverport:
<%= request.getServerPort() %>
<br />
Servername:
<%= request.getServerName() %>
<br />
Protokoll:
<%= request.getProtocol() %>
<hr />
<h3 align="center">Informationen zum Client</h3>
Remotehost:
<%= request.getRemoteHost() %>
<br />
Remoteuser:
<%= request.getRemoteUser() %>
<br />
Remoteadresse:
<%= request.getRemoteAddr() %>
<hr />
<h3 align="center">Informationen zum angeforderten URI</h3>
URI:
<%= request.getRequestURI() %>
```

```
27 <br />
28 Pfad JSP/Servlet:
29 <%= request.getServletPath() %>
30 <hr />
31 <h3 align="center">Parameterliste</h3>
32 <%
33 Enumeration params = request.getParameterNames();
34 while(params.hasMoreElements()) {
35   String temp = params.nextElement().toString();
36 %>
37 <%= temp %>
38 :
39 <%= request.getParameter(temp) %>
40 <br />
41 <%
42 }
43 %>
44 <hr />
45 <h3 align="center">Abfrage-String</h3>
46 Querystring:
47 <%= request.getQueryString() %>
48 <hr />
49 <h3 align="center">Headerinformationen</h3>
50 <%
51 params = request.getHeaderNames();
52 while(params.hasMoreElements()) {
53   String temp = params.nextElement().toString();
54 %>
55 <%= temp %>
56 :
57 <%= request.getHeader(temp) %>
58 <hr />
59 <%
60 }
61 %>
62 </body>
63 </html>
```

In Zeile 1 wird mit dem JSP-Direktiven-Tag `<%@ page import="java.util.*" %>` das Paket `java.util` importiert. Zeile 2 bis 5 ist reines HTML. Mit dem JSP-Ausdruck in Zeile 6 (`<%= request.getServerPort() %>`) wird der verwendete Port an den Client geschickt. In Zeile 9 folgt der Servername (`<%= request.getServerName() %>`), in Zeile 12 das Protokoll (`<%= request.getProtocol() %>`), in Zeile 16 der Remotehost (`<%= request.getRemoteHost() %>`), in Zeile 19 der Remoteuser (`<%= request.getRemoteUser() %>`) und in Zeile 22 die Remoteadresse (`<%= request.getRemoteAddr() %>`).

Die Informationen zum angeforderten URI sind einmal der komplette relative Pfad (Zeile 26: `<%= request.getRequestURI() %>`) und einmal der reine Name des JSP (Zeile 29: `<%= request.getServletPath() %>`). In den folgenden Schritten wird die Liste der überge-

benen Parameter an das JSP abgearbeitet. Dazu kommt von Zeile 32 bis 36 ein JSP-Tag zum Einsatz, in dem in Zeile 33 mit `Enumeration params = request.getParameterNames();` eine Aufzählung aller Parameter, die an das JSP übergeben wurden, in der Variablen `params` gespeichert wird. Ein Objekt vom Typ `Enumeration` besitzt nun eine sehr interessante Methode mit Namen `hasMoreElements()`. Wie der Name schon andeutet, liefert diese Methode die Information, ob eine Aufzählungsliste weitere Elemente enthält. Darüber können Sie dann mit der Methode `nextElement()` iterieren. Die Methode liefert so lange den booleschen Wert `true`, wie es weitere Elemente in der Aufzählungsliste gibt. Gibt es keine Elemente mehr, liefert die Methode den booleschen Wert `false`. Das kann man in der Formulierung von Schleifenbedingungen hervorragend verwenden.

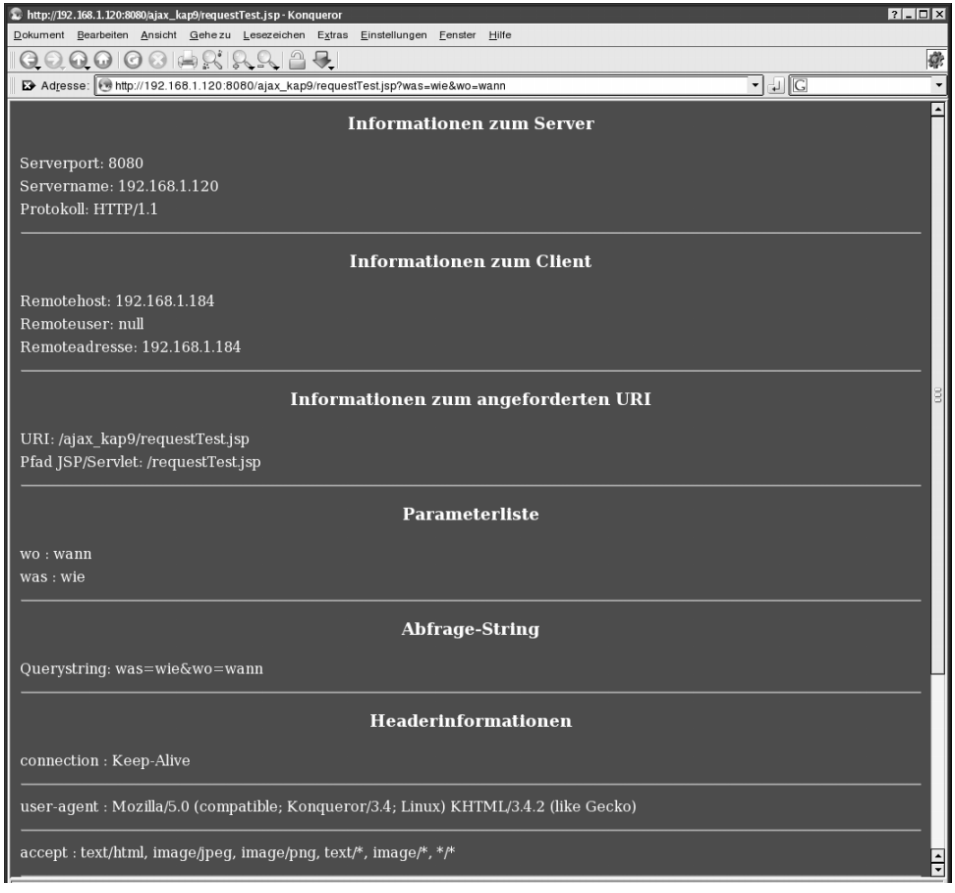


Abbildung 9.3: Verwendung des impliziten request-Objekts

In Zeile 34 beginnt die Schleife (`while(params.hasMoreElements()) {`). In Zeile 35 wird mit `String temp = params.nextElement().toString();` die String-Repräsentation des Parameternamens in einer temporären String-Variablen mit Namen `temp` gespeichert. Diese wird in Zeile 37 mit dem JSP-Ausdrucks-Tag `<%= temp %>` für jeden Schleifendurchlauf ausgegeben. In Zeile 39 greifen wir mit `<%= request.getParameter(temp) %>` den Wert des jeweiligen Parameters ab. In Zeile 47 sehen Sie den vollständigen Querystring (`<%= request.getQueryString() %>`).

Alle Header-Informationen, die zwischen Client und Server ausgetauscht werden, zeigen wir mit der Schleife von Zeile 52 bis 60 an. Diese hat vollkommen die gleiche Struktur wie eben. Nur verwenden wir in Zeile 51 mit `params = request.getHeaderNames();` eine andere Methode des `requests`-Objekts, mit der die Namen der Header-Felder ausgelesen werden können. Auch diese liefert wieder eine Aufzählungsliste zurück, über die wir iterieren. In Zeile 57 geben wir mit `<%= request.getHeader(temp) %>` den Wert des jeweiligen Header-Felds aus.

Die Antwort der JSP – das response-Objekt

Das `response`-Objekt stellt diverse Methoden bereit, mit denen Sie die Ausgabe an den aufrufenden Client steuern – etwa zur Umleitung der Ausgabe (`sendRedirect()`), zum Setzen von Header-Details, zum Kodieren, zum Setzen von Fehler- und Statuscodes etc. Dafür sei aber auf spezielle JSP-Literatur verwiesen.

9.3.2 AJAX-Praxisbeispiel – Nachfordern eines Bilds

Um nun ein vollständiges AJAX-Praxisbeispiel mit JSP zu zeigen, erstellen wir ein kleines Demonstrationsprogramm. Dieses zeigt, wie bei einem Klick auf eine Grafik in einem anderen Bereich der Webseite eine andere Grafik nachgefordert und mit DHTML angezeigt werden kann. Dies erfolgt, ohne die Webseite neu zu laden, was ja ein wesentliches Kriterium von AJAX ist. In der Praxis ist dies eine sehr häufig zu findende Anwendung von AJAX, denn Applikationen, die zum Beispiel Karten in einem sehr großen Maßstab zeigen, können so eine Ausschnittsvergrößerung anzeigen. Um die Sache einfach zu halten, werden wir nur eine HTML-**Imagemap** mit vier Bereichen einsetzen und bei einem Klick einen vergrößerten Ausschnitt des Bereichs anzeigen. Diese Anzeige wird beim Loslassen der Maustaste wieder gelöscht. Das Beispiel verwendet HTML, JavaScript und JSP sowie zum Formatieren der Webseite interne Style Sheets.

Hier ist die HTML-Seite *bildernachfordern.html*:

Listing 9.20: Eine Webseite mit einer Imagemap

```
01 <html>
02 <script language="JavaScript" src="bildernachfordern.js"></script>
03 <body>
```

```
04   <map name="Landkarte">
05     <area shape="rect" coords="0,0,100,100" href="#" onMouseDown="sndReq(1)" ↩
       onMouseUp="loesch()" />
06     <area shape="rect" coords="101,0,200,100" href="#" onMouseDown="sndReq(2)" ↩
       onMouseUp="loesch()" />
07     <area shape="rect" coords="0,101,100,200" href="#" onMouseDown="sndReq(3)" ↩
       onMouseUp="loesch()" />
08     <area shape="rect" coords="101,101,200,200" href="#" onMouseDown="sndReq(4)" ↩
       onMouseUp="loesch()" />
09   </map>
10 <h3>Größeres Bild nachfordern</h3>
11 <h3>Um das um das Beispiel auszuprobieren, klicken Sie die angezeigte Grafik an.
12 Neben der Grafik wird eine gr&ouml;&szlig;ere Version des angeklickten ↩
   Ausschnitts der Grafik zu sehen sein.
13 Sobald Sie die Maustaste wieder loslassen,
14 verschwindet die gr&ouml;&szlig;ere Grafik wieder.</h3>
15 <img src="karte.png" width="200" height="200" alt="Karte" usemap="#Landkarte" ↩
   style="position:absolute;top:150px;left:50px;"/>
16 <div style="position:absolute;top:150px;left: 300px;" id="b1" />
17 </body>
18 </html>
```

In den Zeilen 4 bis 9 wird eine Imagemap definiert, die in vier Bereiche unterteilt ist. In jedem Bereich wird beim Herunterdrücken der Maustaste die JavaScript-Funktion `sndReq()` mit einem spezifischen Parameter aufgerufen. Darüber wird dann gesteuert, welcher vergrößerte Bildausschnitt geladen werden soll. Beim Loslassen der Maustaste wird die JavaScript-Funktion `loesch()` aufgerufen. Diese löscht den Anzeigebereich für den vergrößerten Bildausschnitt.

Die Zeilen 15 und 16 sind dann wieder interessant. In Zeile 15 wird die Originalgrafik angezeigt, die mittels `usemap="#Landkarte"` mit der Imagemap überzogen wird. Mit `style="position:absolute;top:150px;left:50px;"` wird die Grafik positioniert. In Zeile 16 befindet sich der `<div>`-Container, in dem die Antwort des Servers angezeigt wird. Auch dieser wird mit Style Sheets positioniert (`<div style="position:absolute;top:150px;left: 300px;" id="b1" />`).

Hier ist der interessante Ausschnitt der verwendeten JavaScript-Datei *bildernachfordern.js*:

Listing 9.21: Der interessante Teil der JavaScript-Datei

```
02 var was = null;
...
23 function sndReq(klick) {
24     was = klick;
```

```
25     resObjekt.open('get', 'bildernachfordern.jsp?was='+klick,true);
26     resObjekt.onreadystatechange = handleResponse;
27     resObjekt.send(null);
28 }
29 function handleResponse() {
30   if(resObjekt.readyState == 4){
31       document.getElementById("b1").innerHTML = "<img
src='"+resObjekt.responseText+ "' />";
32   }
33 }
34 function loesch() {
35  document.getElementById("b1").innerHTML = "";
36 }
37 resObjekt=erzXMLHttpRequestObject();
```

Mit der globalen Variablen `klick` in Zeile 2 merkt man sich, welcher Übergabewert an die Methode `sndReq()` beim Aufruf übergeben wurde. In Zeile 25 wird dieser Wert an das JSP-Skript übergeben. In der Funktion `handleResponse()` wird der Rückgabewert dafür verwendet, eine Bildreferenz zusammenzusetzen (Zeile 31: `document.getElement-ById("b1").innerHTML = "<img src='"+resObjekt.responseText+ "' />";`). Die Funktion `loesch()` setzt einfach den Wert von `innerHTML` auf leer (Zeile 35: `document.getElement-ById("b1").innerHTML = "";`).

Hier ist die JSP-Datei *bildernachfordern.jsp*, die die neuen Grafiken schickt:

Listing 9.22: Das JSP-Skript

```
01 <%
02   switch(new Integer(request.getParameter("was")).intValue()) {
03     case 1: %> k1.png<%break;
04     case 2: %> k2.png<%break;
05     case 3: %> k3.png<%break;
06     default: %> k4.png
07 <% } %>
```

In dem Skript wird eine `switch-case`-Anweisung verwendet. Der Testwert muss aus dem Übergabewert extrahiert werden. Da dieses in jedem Fall ein String ist, erfolgt dies mit einem Wrapper, aus dem ein primitiver Datentyp extrahiert wird. Dies wird in Zeile 2 erledigt. Die einzelnen Testfälle schicken die Bildnamen als reinen Klartext zum Client.

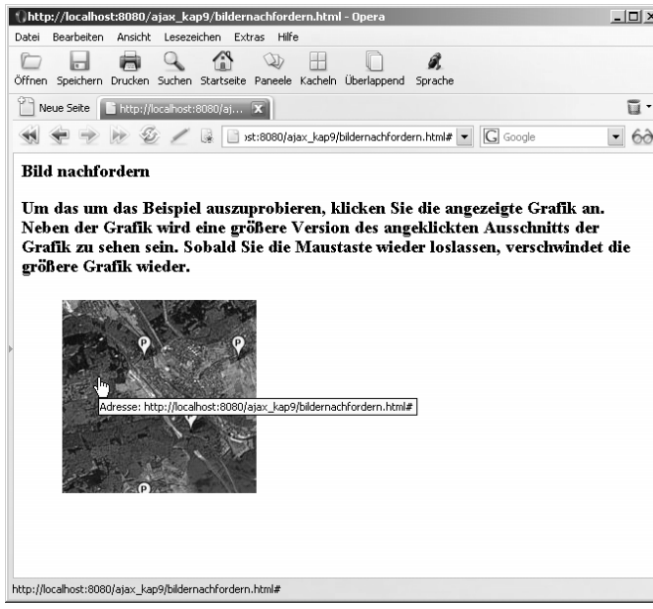


Abbildung 9.4: Ein kleine Version einer Grafik oder eine Version mit einem großen Maßstab wird angezeigt.

Abbildung 9.5: Bei einer Mausaktion wie dem Drücken der Maustaste wird eine größere Version oder eine Version in einem besseren Maßstab per AJAX nachgeladen und angezeigt.

Tipp

Es ist natürlich ein Leichtes, wenn Sie die vergrößerte Grafik in dem Bereich anzeigen wollen, in dem sich die Originalgrafik befindet. Sie müssen nur die ID der Originalgrafik bzw. des Bereichs der Originalgrafik als Ziel des Austauschs angeben.

9.4 Bei AJAX auf Webdatenbanken zugreifen

Mit allen relevanten serverseitigen Sprachen können Sie auf Datenbanken zugreifen. Und gerade Java bietet dazu eine umfangreiche Sammlung relativ einfach anzuwendender Techniken. Im Fall von Webdatenbanken kommt fast ausschließlich ein Datenbankverwaltungssystem[6] bzw. ein Datenbankserver zum Einsatz, über den ein indirekter Zugriff auf eine Datenbank erfolgt[7]. Einen der derzeit wohl populärsten Vertreter schuf ein Programmierer namens Michael Widenius mit dem OpenSource-Datenbankverwaltungssystem **MySQL**. MySQL gibt es für die meisten wichtigen Betriebssysteme und wird auch aus Java heraus hervorragend unterstützt. Natürlich gibt es zahlreiche Alternativen wie MS SQL-Server, DB2, Oracle oder Interbase.

9.4.1 Grundsätzlicher Ablauf einer Datenbankabfrage

In Java wird eine Datenbankabfrage in der Regel aus immer den gleichen Schritten bestehen:

1. Zuerst wird eine Verbindung zum Datenbankserver aufgebaut.
2. Im nächsten Schritt wird eine Datenbankabfrage formuliert und abgeschickt. Dazu kommt fast immer eine Sprache mit Namen **SQL** zum Einsatz.
3. Abschließend werten Sie das Ergebnis der Abfrage aus.
4. Zuletzt wird die Datenbankverbindung geschlossen.

Verständlich wird diese Abfolge bei vollständigen Beispielen, die wir gleich durchspielen. Vorher soll aber kurz SQL erklärt werden, das zum Formulieren einer Datenabfrage verwendet wird.

9.4.2 Kurzüberblick SQL

Wie nahezu alle Webdatenbanken können Sie auch MySQL-Datenbanken[8] mit einer speziellen Sprache abfragen – **SQL** (ursprünglich die Abkürzung für **Structured**

6 *DBMS – Datenbank Management System*
7 *Mit Zugriffsverwaltung.*
8 *Was der Name ja bereits andeutet.*

Query Language – mittlerweile steht **S** für **Standard**[9]). SQL unterscheidet vier Sprachschichten: die Data Query Language, die Data Control Language, die Data Manipulation Language und die Data Definition Language.

Unter **Data Query Language** fasst man SQL-Befehle zur Datenabfrage zusammen. Diese Art der SQL-Befehle werden bei AJAX vom größten Interesse sein. Genau genommen handelt es sich nur um den Befehl `select`, der aber mit zahlreichen Klauseln sehr genau spezifiziert werden kann und den Inhalt von Datenbanktabellen auslesen lässt. Beispiel:

Listing 9.23: Auflistung aller Werte in der Tabelle adressen

```
select * from adressen;
```

Die **Data Manipulation Language** umfasst Befehle zum Einfügen (`insert`), Aktualisieren (`update`) und Löschen (`delete`) von Daten. Auch diese Befehle lassen sich mit Attributen sehr weitreichend konfigurieren. Beispiel:

Listing 9.24: Einfügen einer Zeile mit den angegebenen Werten in die Spalten Name und Vorname in der Tabelle adressen

```
insert into adressen (Name, Vorname) values ('Steyer', 'Ralph');
```

Die **Data Definition Language** umfasst drei SQL-Befehle zum Erstellen einer Datenbank bzw. Tabelle (`create`), zum Löschen einer Datenbank bzw. Tabelle (`drop`) und zum Definieren einer Spalte in einer Tabelle (`alter`). Auch diese Befehle lassen sich mit Attributen sehr weitreichend konfigurieren. Beispiel:

Listing 9.25: Eine neue Tabelle namens neu erzeugen

```
create table neu;
```

Mit der **Data Control Language** können Sie Rechte in der Datenbank manipulieren. Der Befehl `grant` legt die Art des Zugriffs fest und `revoke` entzieht Rechte. Wie bei allen SQL-Anweisungen bestehen umfangreiche Konfigurationsmöglichkeiten.

9 *Bei Kennern löst **Standard** immer großes Gelächter aus, denn es gibt zahlreiche Dialekte von SQL, so dass es fast lächerlich anmutet, von einem Standard zu reden. Aber im Groben stimmen die Dialekte überein.*

Hinweis

Bei AJAX-Applikationen werden Sie hauptsächlich Datenbankabfragen ausführen und kaum Aktionen durchführen, die in einer Datenbank Rechte verändern – zumindest nicht unmittelbar in Verbindung mit einer Nachforderung von Daten. Allerdings sind natürlich auch AJAX-Applikationen sinnvoll, die Daten in einer Datenbank eintragen und auch Tabellen dynamisch erzeugen, zum Beispiel um eine Sitzung (also die Kommunikation eines Benutzers mit dem Webserver über verschiedene Schritte hinweg) über Einträge in einer Datenbank zu verfolgen. Nehmen Sie beispielsweise einen Onlineshop, bei dem ein Besucher verschiedene Produkte auswählen möchte. Wenn der Besucher ein Produkt ausgewählt hat, wird er es in einem typischen Onlineshop in einen so genannten Warenkorb ablegen und weitere Aktionen durchführen, zum Beispiel weitere Produkte auswählen. Mit AJAX und SQL können Sie diesen Warenkorb natürlich in einer Datenbank pflegen und nach dem endgültigen Abschicken der Bestellung an ein Transaktionssystem weitergeben. Dies bedeutet zwar einigen Netzwerkverkehr und Belastung für den Webserver respektive die Datenbank, ist aber zuverlässiger als das lokale Vorhalten von Daten mittels Cookies. Und da im Fall von AJAX die Webseite nicht bei jeder Datenspeicherung neu geladen werden muss, haben Sie eine hervorragende Performance der Seite. Wir werden im Laufe des Kapitels die Verfolgung einer Sitzung mit einer Datenbank in einfacher Form praktisch durchspielen.

9.4.3 Datenbankzugriff mit Java

Für den Zugang zu Datenbanken bietet Java eine einfache Möglichkeit – das **Java DataBase Connectivity-API** (**JDBC**). JDBC ist nicht als Produkt zu verstehen, sondern es handelt sich um die abstrakte Spezifikation einer Schnittstelle zwischen einer Client-Anwendung und einer SQL-Schnittstelle. Damit lassen sich datenbankunabhängige Java-Clients schreiben, die miitels dieses Layers auf zahlreiche verbreitete relationale Datenbanken zugreifen können. Die JDBC-Treiber übernehmen dabei die gesamte Datenbankanbindung. Sofern die Datenbankentwickler SQL-Syntax unterstützen, sollte jedes Datenbankprodukt mit einem JDBC-kompatiblen Treiber verwendet werden können.

Hinweis

Einer der ersten JDBC-kompatiblen Treiber war der JDBC-Treiber für **ODBC**-kompatible (**Open Database Connectivity**) Datenbanken, mit dem Java-Programmierer leicht auf eine beliebige ODBC-kompatible Datenbank zugreifen können. Deshalb ist es auch heute noch sehr einfach, einen ODBC-Treiber aus Java heraus zu verwenden. Dennoch ist diese Art des Zugriffs auf eine Datenbank funktional sehr beschränkt und wird meist nur auf einfache Datenbanken wie Access angewendet.

Die JDBC-Treiber existieren in verschiedenen Ausprägungen, deren Details hier natürlich nicht behandelt werden sollen. JDBC arbeitet aber grundsätzlich auf zwei Stufen. Die erste Stufe ist der Verbindungsaufbau zwischen der Java-Anwendung und dem JDBC-Treibermanager mittels JDBC-API. Über diesen JDBC-Treibermanager kann ein Java-Programm dann mehrere JDBC-Treiber verwalten und mit ihnen Informationen und Daten austauschen. Jeder Treiber wiederum kann aus Java direkt auf lokale Daten zugreifen, ODBC als Zwischenebene dazwischen schalten oder einen Netzwerkzugriff auf eine Datenbank auslösen. Die Treiber registrieren sich beim JDBC-Manager während der Initialisierung, so dass der Manager einen Überblick über alle verfügbaren Treiber hat.

Während des Versuchs, sich mit einer Datenbank zu verbinden, gibt das Java-Programm einen Datenbank-URL an den JDBC-Manager weiter. In einem Java-Programm kann man explizit angeben, welcher Treiber verwendet werden soll. Die URLs von JDBC haben immer die folgende Form:

Listing 9.26: Schema eines JDBC-URL

```
jdbc:subprotocol:subname
```

Das Subprotokoll ist der Name des Verbindungsprotokolls und der Subname ist der Name der jeweiligen Datenbank innerhalb der Domäne des Protokolls. Sofern über den Subnamen Informationen über Host und Port enthalten sind (was meist der Fall ist), sollte dieser den Host und den Port in der URL-Standard-Notation angeben. Das sieht schematisch so aus:

Listing 9.27: Schema für den Hostnamen mit Port

```
//hostname:port/
```

Beispielsweisse kann eine MySQL-Datenbank auf dem gleichen Rechner über den folgenden URL referenzieren:

Listing 9.28: Zugriff auf eine MySQL-Datenbank auf localhost mit Port 3306

```
jdbc:mysql://localhost:3306/
```

Im allgemeinen Fall braucht man zum Aufbau der konkreten Verbindung zur Datenbank folgende Informationen:

- Treiberalias
- Host
- Port
- Name der Datenbank
- User
- Passwort

Die einzelnen Angaben werden durch entsprechende Trennzeichen getrennt. Beispiel:

Listing 9.29: Alle notwendigen Informationen, um eine Verbindung zu einer Datenbank aufzubauen

```
jdbc:mysql://localhost:3306/eins?user=root&pwd=geheim
```

In unserem konkreten Fall werden der MySQL-Treiber, `localhost` als Host, der Port `3306` (Standardport des MySQL-Servers), die Datenbank mit Namen `eins`, der User `root` und das Passwort `geheim` angeben.

9.4.4 Schematischer Ablauf einer Datenbankverbindung in Java mit JDBC

Um mit JDBC unter Java eine Datenbank verwenden zu können, benötigen Sie für die konkrete Wahl eines jeden Datenbankservers den passenden JDBC-Treiber. Und dieser muss für Ihre Java-Applikation verfügbar gemacht werden.

Der MySQL-Connector

Ein besonderer Charme von MySQL ist es nun, dass der passende JDBC-Treiber (**MySQL-Connenctor** genannt) kostenlos über die Homepage des MySQL-Projekts zur Verfügung steht und aus dem Internet geladen werden kann.

Das Treiberpaket muss nur entpackt und die entsprechende *.jar*-Datei für den MySQL-Connector[10] nach dem Speichern auf dem Rechner noch dem Klassenpfad hinzugefügt werden. Dieses bedeutet konkret in Hinblick auf Tomcat, dass die *.jar*-Datei in der Regel in das Unterverzeichnis *common/lib* im Installationsverzeichnis von

10 Der Name kann zum Beispiel mysql-connector-java-3.1.7-bin.jar lauten. Der genaue Name hängt hauptsächlich von der Version ab.

Tomcat (genauer – unter *$CATALINA_HOME/common/lib*) kopiert wird. Dies sollte genügen, dass Tomcat den Datenbanktreiber auch findet, wenn er in einer Java-Applikation geladen wird. Und dies machen Sie dann über `Class.forName()`. Damit wird in einer Java-Applikation die Klasse für den Treiber verfügbar gemacht. So sieht das dann zum Beispiel für den MySQL-Treiber aus:

Listing 9.30: Den MySQL-Treiber in einer Java-Applikation verfügbar machen

```
Class.forName("com.mysql.jdbc.Driver");
```

Die Schritte zum Erfolg

Allgemein geht man bei der Programmierung konkreter Datenbankzugriffe aus einer Java-Applikation (egal ob JSP, Servlet oder auch eigenständiges Java-Client-Programm) in der Regel ähnlich vor. Zuerst wird man die notwendigen Klassen bzw. Packages einbinden. Das sind oft `java.net.URL` bzw. `java.net.*` und immer `java.sql.*`. Letzteres wird die zentrale Funktionalität für Datenbankzugriffe beinhalten.

Im nächsten Schritt laden Sie wie oben beschrieben die gewünschten JDBC-Treiber. Allerdings muss die Geschichte mit einem `try-catch`-Konstrukt abgesichert werden, denn eine zu ladende Klasse kann natürlich unter Umständen nicht bereitstehen (die Methode `forName()` wirft dann eine `ClassNotFoundException` aus). Dies kann mit folgender Syntax geschehen:

Listing 9.31: Laden einer Treiberklasse

```
try {
// Laden der gewünschten Treiber
Class.forName(drivername);
// ... weitere sinnvolle Aktionen
}
catch (ClassNotFoundException e) {
// etwas Sinnvolles tun
}
```

Im nächsten Schritt können wir bereits eine Datenbankverbindung aufbauen. Etwa so:

Listing 9.32: Aufbau einer Verbindung zur Datenbank

```
String connStr = "jdbc:mysql://localhost:3306/" + "ajax?user=root&pwd=";
try {
  Class.forName("com.mysql.jdbc.Driver");
   Connection conn = DriverManager.getConnection(connStr);
...
```

Die Methode `getConnection()` der Klasse `DriverManager` liefert eine Verbindung zur Datenbank (ein Objekt vom Typ `java.sql.Connection`). Sie gibt es in verschiedenen Varianten.

Der nächste Schritt dient dazu, ein Abfrageobjekt zu erstellen und darauf ein (im Grunde beliebiges) SQL-Kommando auszuführen. In unserem Beispiel wollen wir sämtliche Datensätze aus einer Tabelle namens `adressen` ausgeben und diese dabei sortieren.

Listing 9.33: Erstellen eines Statements und Ausführung

```
String sql = "SELECT * FROM `adressen` ORDER BY `id` ASC";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
```

Die Methode `createStatement()` eines `Connection`-Objekts erstellt ein Objekt vom Typ `java.sql.Statement` und dieses stellt die Methode `executeQuery()` bereit, mit der eine SQL-Abfrage ausgeführt werden kann. Als Ergebnis erhalten Sie ein Objekt vom Typ `java.sql.ResultSet`.

Wenn wir mittels einer SQL-Abfrage die Datensätze selektiert haben, sollten wir damit auch etwas Sinnvolles tun, z.B. auf dem Bildschirm des Clients nach gewissen Kriterien anzeigen. Zum Bewegen des Datensatzzeigers auf dem Resultset sind insbesondere die folgenden Methoden von Interesse:

Methode	Beschreibung
`boolean next()`	Nächsten Datensatz auswählen
`boolean previous()`	Vorherigen Datensatz auswählen
`void afterLast()`	Bewegt den Datensatzzeiger auf das Ende des ResultSet-Objekts (direkt hinter die letzte Zeile)
`void beforeFirst()`	Bewegt den Datensatzzeiger vor den Beginn des ResultSet-Objekts (direkt vor die erste Zeile)
`boolean first()`	Bewegt den Datensatzzeiger auf die erste Zeile des ResultSet-Objekts
`boolean last()`	Bewegt den Datensatzzeiger auf die letzte Zeile des ResultSet-Objekts
`void moveToCurrentRow()`	Bewegt den Datensatzzeiger auf die aktuelle Zeile

Tabelle 9.2: Methoden für die Navigation auf einem ResultSet-Objekt

Im letzten Schritt erfolgen die üblichen Aufräumarbeiten und mit der `close()`-Methode werden offene Verbindungen geschlossen.

Um die Sache noch einmal komprimiert zusammenzufassen: Zuerst laden Sie die Klasse mit dem Datenbanktreiber. Anschließend können Sie die Verbindung zum

Datenbankserver aufbauen. Das erledigt man in der Regel über `DriverManager.getConnection()` mit dem vollständigen Verbindungs-URL als String-Argument. Danach erzeugen Sie ein SQL-Statement und führen es aus. Auf dem Resultat navigieren Sie dann und wählen den Teil aus, den Sie benötigen. Die Sache hört sich etwas abstrakt an, wird aber an einem konkreten Beispiel sicher klar.

9.4.5 Praktische AJAX-Beispiele mit Datenbankanbindung

Spielen wir nun praktische AJAX-Beispiele mit Datenbankanbindung durch. Wir bilden bei den ersten beiden Beispielen die Funktionsweise von **Google suggest** nach. Wie bereits bei der Vorstellung von AJAX besprochen, bezeichnet Google suggest eine Erweiterung des konventionellen Suchdienstes dieser Suchmaschine. Während der Anwender in einem Eingabefeld im Google-Webformular einen Suchbegriff eingibt, werden ihm unter dem Eingabefeld Zeichen für Zeichen verbesserte Vorschläge in einem Listenfeld präsentiert. Dabei werden ihm solche Begriffe vorgeschlagen, die mit den bisher eingegebenen Zeichen beginnen und deshalb in Verbindung mit dem beabsichtigten Suchbegriff des Anwenders stehen könnten.

Dieses Verhalten werden wir hier im ersten Beispiel analog realisieren. Bei einer Eingabe in einem einzeiligen Listenfeld in einer Webseite sollen in unserem Beispiel dem Anwender aus einer Datenbank Vorschläge gemacht werden, die mit den eingegebenen Zeichen beginnen. Diese Vorschläge werden unter dem Eingabefeld in einem Listenfeld angezeigt, aus dem der Anwender mit einem Klick den genauen Datensatz auswählen kann. Dieser wird dann in das Eingabefeld übernommen[11]. Dabei werden die Daten aus der Datenbank bereits im JSP-Skript (also auf dem Server) so aufbereitet, dass die vollständige HTML-Struktur für das `<select>`-Element samt enthaltener `<option>`-Elemente als Antwort geschickt wird. Die Antwort muss nur noch in einem `<span>`-Container angezeigt werden. Unser zweites Beispiel hingegen wird Daten im XML-Format zusammensetzen. Daraus erstellen wir in der Webseite mithilfe des `node`-Objekts dynamisch eine Aufzählungsliste.

Bereitstellen der Datenbank

Zuerst brauchen wir eine Datenbank mit Informationen, aus denen eine Auswahl nach einer AJAX-Anfrage dem Anwender angezeigt werden soll. Sie haben, wenn Sie XAMPP installiert haben, bereits einen MySQL-Datenbankserver zur Verfügung, den Sie über das Adinistrationstool starten und einrichten können.

Eine weit verbreitete und sehr bequeme grafische Administrationsoberfläche für MySQL ist die OpenSource-Anwendung **phpMyAdmin**. Auch dieses Tool wird bei XAMPP mitgeliefert. Wir werden damit in dem Beispiel arbeiten.

11 *Weiter werden wir die Logik der Applikation nicht entwickeln, da damit alles Wesentliche in Hinsicht auf AJAX gezeigt ist.*

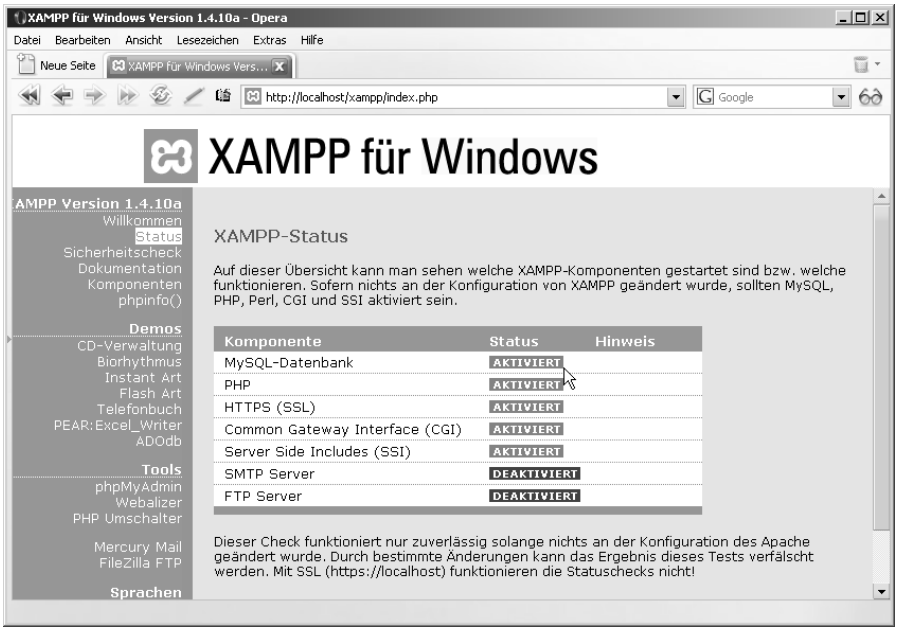


Abbildung 9.6: XAMPP stellt auch einen MySQL-Datenbankserver bereit – hier sehen Sie, dass er läuft.

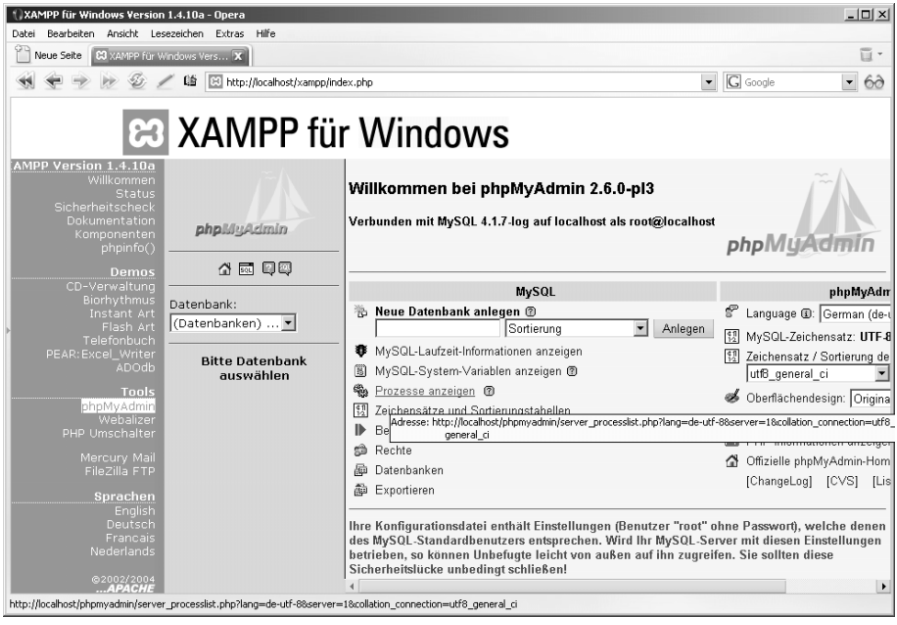


Abbildung 9.7: Mit phpMyAdmin lässt sich MySQL sehr einfach administrieren.

Erstellen wir nun das vollständige Beispiel. Im ersten Schritt brauchen wir eine Datenbank. Diese legen Sie am einfachsten in phpMyAdmin an, indem Sie dort unter der Überschrift ***Neue Datenbank anlegen*** einen Namen eintragen – wir werden `ajax`[12] nehmen – und auf die Schaltfläche ANLEGEN klicken.

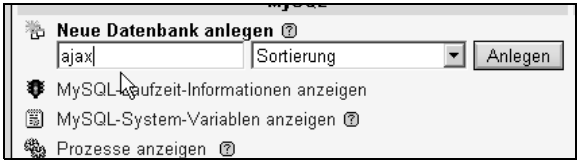


Abbildung 9.8: Anlegen einer neuen MySQL-Datenbank

Auf der linken Seite von phpMyAdmin unter der Rubrik ***Datenbank:*** wird bei erfolgreicher Ausführung die neu angelegte Datenbank angezeigt. Alternativ können Sie dort weitere Datenbanken auswählen, die von dem Datenbankmanagementsystem verwaltet werden. Unter anderem gibt es da immer einige Vorgabedatenbanken, die MySQL zum Betrieb braucht.

Abbildung 9.9: Die Datenbank wurde angelegt und kann jetzt weiterbearbeitet werden.

Im nächsten Schritt müssen wir eine Tabelle anlegen. Das Web-Interface dazu sehen Sie auf der rechten Seite von phpMyAdmin. Tragen Sie dort unterhalb von ***Neue Tabelle in Datenbank ajax erstellen:*** bei ***Name:*** `adressen` und bei ***Felder:*** den Wert `5` ein.

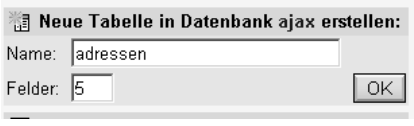


Abbildung 9.10: Name der Tabelle und Anzahl der Felder

12 *Warum nur ;-)).*

Wenn die Tabelle angelegt ist, müssen Sie Namen und Typ der Felder darin festlegen. Sie können dabei zahlreiche Details spezifizieren, aber für unsere Zwecke genügen folgende Angaben:

- Das erste Feld bekommt den Namen `id` und den Typ `int`. Die Länge lassen Sie offen.
- Das zweite Feld ist `name` vom Typ `varchar` mit der Länge `40`.
- Das dritte Feld ist `vname` vom Typ `varchar` mit der Länge `40`.
- Das vierte Feld ist `vorwahl` vom Typ `varchar` mit der Länge `10`.
- Das letzte Feld heißt `nr` vom Typ `varchar` mit der Länge `15`.

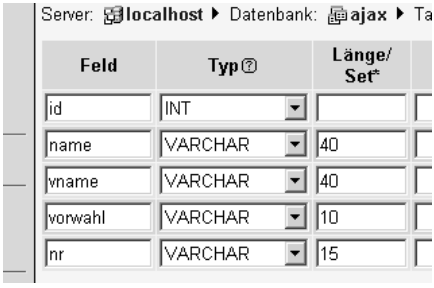


Abbildung 9.11: Das sind die Angaben für die Felder in der Datenbank.

Beim Feld `id` legen Sie noch fest, dass es sich um einen Primärschlüssel handelt. Das können Sie mit einem Optionsfeld machen, das sich ziemlich weit rechts im Webinterface befindet (Sie müssen wahrscheinlich scrollen).

Abbildung 9.12: Das Feld id wird als Primärschlüssel festgelegt.

Mit einem Klick auf die Schaltfläche Speichern ist die Datenbanktabelle fertig. Sie sollten die Struktur nun angezeigt bekommen.

Über den Link *Einfügen* oberhalb der Strukturanzeige können Sie die Datenbanktabelle nun mit Werten füllen. Wie Sie sicher erkannt haben, haben wir eine Adressdatenbank mit Name, Vorname und einer Telefonnummer mit Vorwahl und eigentlicher Nummer angelegt. Das Feld `id` ist der Schlüssel, der aus einer eindeutigen Nummer bestehen muss. Legen Sie nun die Tabelle mit einigen Datensätzen an (etwa 15 bis 20 genügen). Achten Sie dabei jedoch darauf, dass Sie Nachnamen **mehrfach** vergeben.

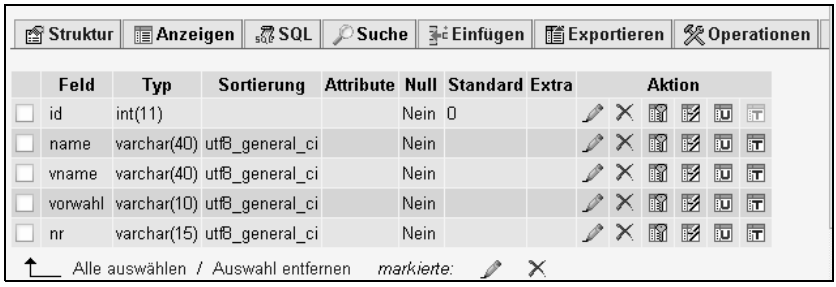


Abbildung 9.13: Die fertige Datenbankstruktur

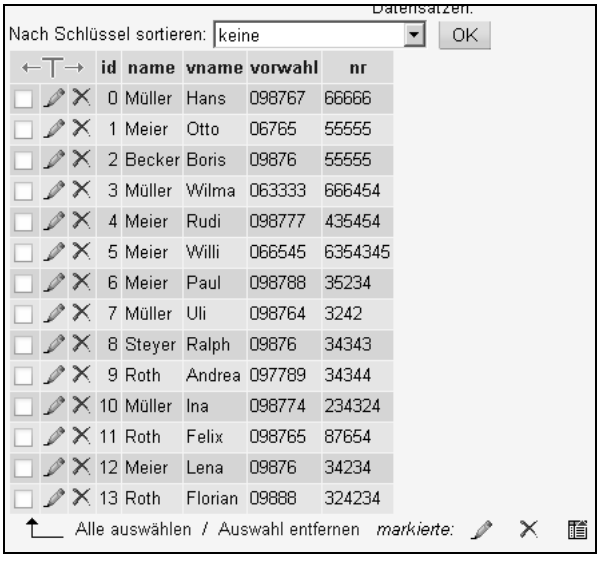


Abbildung 9.14: Etwa 15 Datensätze genügen für unser Beispiel.

Die Datenbank ist nun fertig. Für den Zugriff aus PHP heraus brauchen Sie nun noch Zugangsdaten, die Ihnen der Adminstrator der Datenbank oder Ihr Internet Provider zukommen lassen muss. Wenn Sie selbst MySQL administrieren, kennen Sie die Zugangsdaten natürlich. Im Fall von XAMPP können Sie in der Grundkonfiguration mit dem User `root` ohne Passwort auf die Datenbank zugreifen, wenn MySQL auf dem gleichen Rechner wie Apache und PHP ausgeführt wird. Das ist zwar aus Sicherheitsgründen wenig sinnvoll, aber wie schon erwähnt kommt XAMPP in der Grundeinstellung mit minimalen Sicherheitseinstellungen daher.

Tipp

In der Datenbank `mysql`, die eine Standardverwaltungsdatenbank von MySQL ist, können Sie sich in der Tabelle `users` anzeigen lassen, welche Zugangsdaten zu Ihrem Datenbankserver eingetragen sind. Natürlich können Sie dort neue User anlegen und bestehende Zugänge auch bearbeiten.

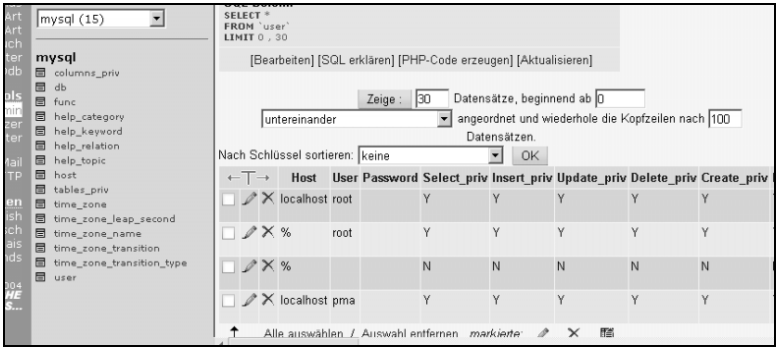


Abbildung 9.15: Die Zugangsverwaltung von MySQL erfolgt über die Datenbank mysql und die Tabelle users.

Suggest 1 – Senden einer dynamischen Auswahlliste als HTML-Fragment

Betrachten wir jetzt die Webseite (*suggest.html*) für die erste Variante unseres Beispiels, in der laufend mit AJAX-Anfragen neue Informationen angezeigt und zur Auswahl durch den Anwender bereitgestellt werden sollen:

Listing 9.34: Die Webseite

```
01 <html>
02 <link rel="stylesheet" href="stil.css" type="text/css">
03 <script language="JavaScript" src="suggest.js"></script>
04 <script language="JavaScript">
05  window.onkeyup = sndReq;
06 </script>
07 <body>
08 <br />
09 <form name="f1"><span id="name">
10 Geben Sie den Nachnamen ein:<br />
11 <input name="name" type="text" size="30" /></span>
12 <span id="antwort"></span>
13 </form>
14 </body>
15 </html>
```

Achtung

Im Internet Explorer 6.0 funktioniert dieses Beispiel so nicht (siehe dazu auch die Ausführungen in Kapitel 5). Statt der Anweisung `window.onkeyup = sndReq;` in Zeile 5 (Sie können hier auf den gesamten `<script>`-Container verzichten) müssen Sie für den Internet Explorer in Zeile 11 Folgendes notieren:

Listing 9.35: Die Variante für den Internet Explorer

```
<input name="name" type="text" size="30" onKeyUp="sndReq()">
```

Es ist aber zu hoffen, dass neuere Versionen des Internet Explores diesen Mangel beheben. Die Trennung der Ereignisebene vom HTML-Tag ist eine sehr sinnvolle Strukturierungsmaßnahme.

In Zeile 2 verlinken wir die Webseite mit einer externen Style Sheet-Datei (`<link rel="stylesheet" href="stil.css" type="text/css">`). In Zeile 5 verbinden wir das Eventhandling zum Nachfordern der Daten mit dem Eventhandler `onkeyup` (`window.onkeyup = sndReq;`). Es ist wichtig, die Datennachforderung ***nicht*** mit `onkeypress` oder `onkeydown` auszulösen, denn das vom Anwender eingegebene Zeichen steht erst dann zur Verfügung, wenn die Taste losgelassen wird. In der Webseite finden Sie ein Formular mit einem einzeiligen Eingabefeld in Zeile 11 `<input name="name" type="text" size="30" />`. Dort soll der Besucher den Nachnamen eingeben, den er sucht. Der `<span>`-Container in Zeile 12 wird die Antwort des Servers aufnehmen (`<span id="antwort"></span>`). Wie Ihnen auffallen sollte, befindet sich der `<span>`-Container immer noch innerhalb des Formulars. In den `<span>`-Container wird bei der Nachforderung der Daten ein vollständiges Listenfeld platziert.

Betrachten wir kurz die CSS-Datei *stil.css*:

Listing 9.36: Die CSS-Datei

```
01 #antwort{
02 position:absolute;top:100px;left:190px;
03 visibility : hidden;
04 }
05 #name{
06 position:absolute;top:60px;left:190px;
07 }
```

Die Stilregel in den Zeilen 1 bis 4 legt für den `<span>`-Container eine Position fest, die sich in unserem Beispiel auf das Listenfeld bezieht. Zusätzlich wird der Container damit erst einmal unsichtbar gemacht. Die zweite Regel positioniert den `<span>`-Container mit der ID `name`. Das ist das einzeilige Eingabefeld mit dem vorangestellten Text.

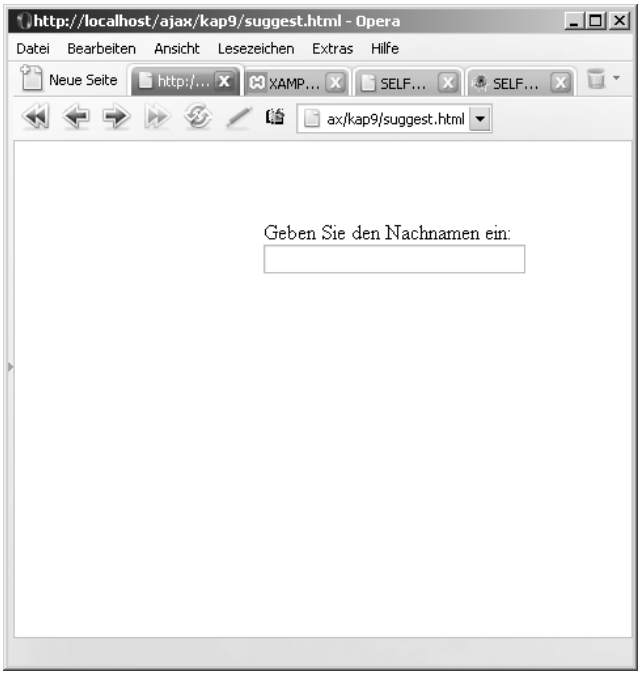


Abbildung 9.16: Solange in dem Eingabefeld nichts eingegeben wird, ist das Listenfeld unsichtbar.

Die externe JavaScript-Datei `suggest.js` hat nun ein paar interessante Feinheiten zu bieten:

Listing 9.37: Die externe JavaScript-Datei

```
22 function sndReq() {
23   if(document.f1.name.value !=""){
24     resObjekt.open('get', 'suggest.jsp?name=' + escape(document.f1.name.value. ↵
       toLowerCase()) ,true);
25     resObjekt.onreadystatechange = handleResponse;
26     resObjekt.send(null);
27   }
28   else {
29     document.getElementById("antwort").style.visibility = "hidden";
30   }
31 }
32 function handleResponse() {
33   document.getElementById("antwort").style.visibility = "visible";
34   if(resObjekt.readyState == 4){
35     document.getElementById("antwort").innerHTML = resObjekt.responseText;
36   }
37 }
```

```
38 resObjekt=erzXMLHttpRequestObject();
39 function uebernehme(){
40   document.f1.name.value = document.f1.vorschlag.value;
41 }
```

Die Funktion `sndReq()` enthält in dem Beispiel mehr Logik, als es bisher bei unseren Beispielen der Fall war. Es ist in der Praxis sehr sinnvoll, bei permanent abgefangenen Ereignissen zu entscheiden, ob Sie tatsächlich jedes Mal eine Datenanfrage an den Server schicken sollen. In unserem Beispiel reagieren wir mit dem Eventhandler `onkeyup` auf jedes Loslassen einer Taste. Aber das muss nicht in jedem Fall mit der Eingabe eines Zeichens in dem Eingabefeld übereinstimmen. Sie können nun mit JavaScript zum Beispiel vor einer expliziten Datenanforderung überprüfen, ob überhaupt eine Taste für ein Zeichen oder eine Zahl gedrückt wurde[13] oder ob sich der Inhalt des Eingabefelds geändert hat. Damit lässt sich viel unnötiger Datenverkehr bei AJAX-Anwendungen vermeiden. Für unser Beispiel entscheiden wir uns für ein Überprüfung, ob das Eingabefeld überhaupt einen Inhalt hat, wenn eine Taste losgelassen wird (Zeile 23 – `if(document.f1.name.value !="")`). Diese Prüfung schließt auch die Logik mit ein, dass im Grunde die Registrierung bei `window` jeden Tastendruck im Browser-Fenster beobachtet und wir nur die Änderung in dem Eingabefeld mit einer AJAX-Datennachforderung koppeln wollen. Nur dann wird eine Datenanfrage an den Server geschickt.

Wenn das Eingabefeld keine Zeichen enthält, blenden wir in Zeile 29 mit `document.getElementById("antwort").style.visibility = "hidden";` das Listenfeld wieder aus. Das bewirkt auch das Verschwinden des Listenfelds, wenn der Besucher Zeichen eingegeben hat und anschließend wieder löscht.

In Zeile 24 wird die Anfrage an den Server abgeschickt. Dabei übergeben wir das Feld `name`, dessen Wert aus dem Formular ausgelesen wird. Beachten Sie, dass wir die Zeichen in dem Eingabefeld mit der JavaScript-Methode `toLowerCase()` des `String`-Objekts in Kleinbuchstaben konvertieren (`resObjekt.open('get', 'suggest.jsp?name=' + escape(document.f1.name.value.toLowerCase()) ,true);`). Wir unterscheiden in diesem Beispiel also explizit nicht zwischen Groß- und Kleinschreibung[14]. Was Ihnen noch auffallen sollte, ist die `escape()`-Funktion. Damit maskieren wir alle Sonderzeichen, die ein Anwender im Eingabefeld eingibt und die sonst bei der Übertragung mit HTTP Probleme bereiten würden[15]. Diese Aktion ist bei der `open()`-Methode notwendig, wohingegen die Kodierung bei einer Versendung von Formulardaten in einem Webformular unterbleiben kann. In diesem Fall kodiert der Webbrowser automatisch.

13 Und nicht nur eine Sondertaste.

14 Das können Sie aber natürlich machen, wenn Ihnen das sinnvoll erscheint.

15 In der von mir verwendeten Beispieldatenbank ist beispielsweise der Name Müller vorhanden. Das ü würde bei der Eingabe in dem Eingabefeld ohne die Maskierung nicht richtig zum Server übertragen.

Hinweis

Eine schöne Eigenschaft von JSP ist, dass wir maskierte Zeichen dort nicht mehr manuell dekodieren müssen, wenn sie auf dem Server angelangt sind. Sie können sie einfach direkt verwenden. Die Dekodierung erledigt Java im Hintergrund.

In der Funktion `handleResponse()` machen wir in Zeile 33 mit `document.getElementById("antwort").style.visibility = "visible";` das Listenfeld sichtbar. Sonst ist die Funktion wie gewohnt aufgebaut.

In den Zeilen 39 bis 41 finden Sie eine neue Funktion (`function uebernehme(){ document.f1.name.value = document.f1.vorschlag.value; }`). Diese Funktion übernimmt den Wert aus dem Formularfeld mit Namen `vorschlag` und setzt ihn in das einzeilige Eingabefeld. Sie werden sich nun fragen, wo das Formularfeld `vorschlag` definiert ist? Das Feld wird dynamisch mit JSP generiert. Betrachten wir die JSP-Datei `suggest.jsp`:

Listing 9.38: Die JSP-Datei

```
01 <%@ page import="java.sql.*;" %>
02 <%
03 String text = "<select name=\"vorschlag\" size=\"10\" onClick=\"uebernehme()\">";
04 String connStr = "jdbc:mysql://localhost:3306/" + "ajax?user=root&pwd=";
05 String name = request.getParameter("name");
06 String vergleich = "";
07 String sql = "SELECT * FROM `adressen` ORDER BY `id` ASC";
08 try {
09   Class.forName("com.mysql.jdbc.Driver");
10   Connection conn = DriverManager.getConnection(connStr);
11   Statement stmt = conn.createStatement();
12   ResultSet rs = stmt.executeQuery(sql);
13   while (rs.next()) {
14     vergleich = rs.getString(2).substring(0, Math.min(name.length(),rs. ↵
       getString(2).length()).toLowerCase();
15     if(name.equals(vergleich))
16       text += "<option>" + rs.getString(2) + ", " + rs.getString(3) +
17        ": " + rs.getString(4) + "/" + rs.getString(5) + "</option>";
18   }
19   rs.close();
20   conn.close();
21 } catch (ClassNotFoundException e) {
22   e.printStackTrace();
23 } catch (SQLException e) {
24   e.printStackTrace();
25 } catch (Exception e) {
```

```
26   e.printStackTrace();
27 }
28 text += "</select>";
29 %>
30 <%= text %>
```

In der JSP-Datei wird der Datenbankzugriff ausgeführt. In Zeile 1 importieren wir das Paket `java.sql` (`<%@ page import="java.sql.*;" %>`), das die zentrale Funktionalität für den Umgang mit Datenbanken unter Java beinhaltet. Die Zeilen 2 bis 7 definieren im Wesentlichen entspechende Variablen für den Zugang zum Datenbankserver und die Auswahl sowohl der konkreten Datenbank als auch der Tabelle. Diese Werte ergeben sich aufgrund unserer Wahl beim Anlegen der MySQL-Datenbank und der Tabelle darin sowie den Zugangsdaten. In Zeile 3 wird eine Variable `text` eingeführt, die für das Zusammensetzen der Antwort verwendet wird. Der erste Teil der Antwort ist der Anfangs-Tag eines HTML-Listenfelds. Dieses umfasst zehn Zeilen und bekommt den Namen `vorschlag`, der oben bereits angesprochen wurde. Ebenso definieren wir einen Eventhandler `onClick`, der die neue JavaScript-Funktion `uebernehme()` aufruft, die den Wert aus diesem Formularfeld nimmt und in das einzeilige Eingabefeld setzt (`String text = "<select name=\"vorschlag\" size=\"10\" onClick=\"uebernehme()\">";`).

Zeile 4 definiert den Verbindungsstring, über den der Datenbanktreiber ausgewählt wird, sowie den Host, den Port, die Datenbank, den User und das Passwort (`String connStr = "jdbc:mysql://localhost:3306/" + "ajax?user=root&pwd="`). In Zeile 5 wird der Übergabewert an das JSP in der String-Variablen `name` gespeichert (`String name = request.getParameter("name");`) und in Zeile 6 der String `vergleich` eingeführt. Diese Variable verwenden wir, um die eingegebenen Zeichen mit den ersten Zeichen in einem Feld des selektierten Datensatzes zu vergleichen.

In Zeile 7 wird eine SQL-Abfrage formuliert (`sql = "SELECT * from `adressen` ORDER BY `id` ASC";`). Die SQL-Abfrage selektiert alle Werte in der vorgegebenen Tabelle und sortiert sie absteigend nach unserem Index. Der Block von Zeile 8 bis Zeile 27 ist nun das `try-catch`-Statement, in dem die eigentliche Datenbankaktivität stattfindet. In Zeile 9 wird zuerst der Datenbanktreiber geladen (`Class.forName("com.mysql.jdbc.Driver");`). In Zeile 10 bauen wir dann eine Verbindung zum Datenbankserver auf (`Connection conn = DriverManager.getConnection(connStr);`) und in Zeile 11 wird ein `Statement`-Objekt erzeugt (`Statement stmt = conn.createStatement();`). Zeile 12 ist das konkrete Ausführen des Statements. Als Ergebnis erhalten Sie ein Objekt vom Typ `ResultSet` (`ResultSet rs = stmt.executeQuery(sql);`).

In der Schleife von Zeile 13 bis 18 iterieren wir über die Ergebnismenge. Dabei kommt mit `next()` eine Java-Datenbankstandardmethode zum Einsatz, die pro Durchlauf der Schleife einen Datensatz als Ergebnis liefert (`rs.next()`). Genau genommen versucht diese Methode einen Zeiger auf den nächsten Datensatz zu positionieren und liefert im Erfolgsfall den booleschen Wert `true`. Wenn es keinen Datensatz mehr gibt, liefert die Methode den booleschen Wert `false`.

Die Variable `vergleich` dient nun als eine Art Testballon, um eine Übereinstimmung der Benutzereingabe in den ersten Zeichen des Felds `name` in den selektiven Datensatz zu überprüfen. Beachten Sie die Feinheiten, die bei der Wertzuweisung für `vergleich` verwendet werden. Mit `toLowerCase()` werden alle Zeichen in Kleinbuchstaben konvertiert[16] und mit `substring()` nur die ersten Zeichen des Datenbankfelds zum Vergleich herangezogen. Die Anzahl der Zeichen ergibt sich aus der Anzahl der eingegebenen Zeichen durch den Anwender. Diese Anzahl kann man mit der Methode `length()` abfragen (`vergleich = rs.getString(2).substring(0, Math.min(name.length(),rs.getString(2).length())).toLowerCase();`). Allerdings dürfen Sie dabei in jedem Datensatz nicht über die Anzahl der Zeichen hinausgreifen, die für den Namen dort gespeichert sind. Deshalb wird das Minimum zwischen der Benutzereingabe und dem gespeicherten Wert hier herangezogen.

Wenn nun die eingegebenen Zeichen im Formularfeld mit den ersten Zeichen in dem Feld `name` des selektierten Datensatzes übereinstimmen, wird die Variable `text` um einen `<option>`-Tag mit allen relevanten Informationen des Datensatzes erweitert. Dazu werden die `<option>`-Elemente des Listenfelds aus den Datenbankeinträgen zusammengesetzt. Dabei fügen wir auch Trennzeichen ein (`text += "<option>" + rs.getString(2) + ", " +rs.getString(3) + ": " + rs.getString(4) + "/" + rs.getString(5) + "</option>";`). In Zeile 19 und 20 werden nach Beendigung der Schleife das Statement und die Verbindung zur Datenbank geschlossen.

Die nachfolgenden `catch`-Strukturen können Sie selbstverständlich mit geeigneten Gegenmaßnahmen für einen Fehlerfall erweitern. Für unser Beispiel sind sie nur ganz einfach gehalten und geben in der Konsole die Standardmeldung der jeweiligen Exception aus.

In Zeile 28 vollenden wir die Variable `text` mit dem schließenden `</select>`-Tag. In Zeile 30 finden Sie einen JSP-Ausdrucks-Tag, mit dem die Antwort zurück zum Client geschickt wird.

Wenn Sie das Beispiel nun testen, werden Sie sehen, dass eine Auswahlliste auftaucht, sobald Sie das erste Zeichen in dem Eingabefeld eingeben. Diese ist mit allen Einträgen in der Datenbank gefüllt, deren erster Buchstabe im Feld `name` mit der Eingabe übereinstimmt.

16 *Bei den übergebenen Werten an das Skript konvertieren wir ja bewusst alle Benutzereingaben in Kleinbuchstaben und so macht es Sinn, die Testwerte aus der Datenbank auch in Kleinbuchstaben zu konvertieren.*

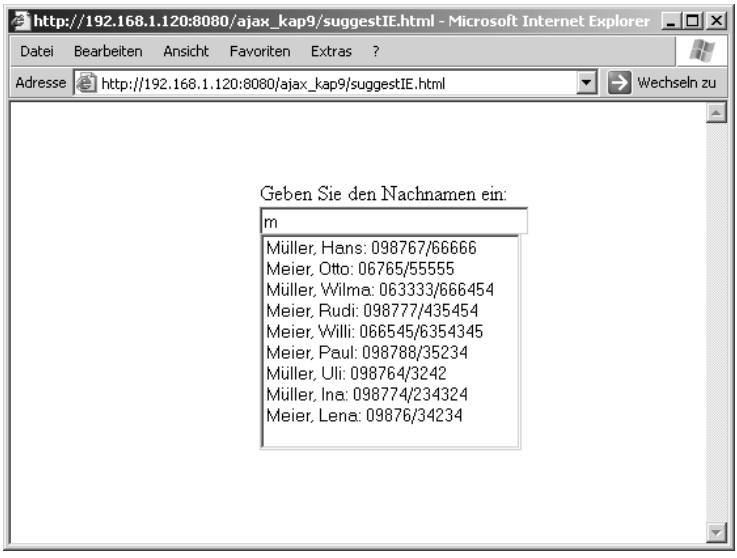


Abbildung 9.17: Das Listenfeld bietet alle Einträge der Datenbank an, bei denen der Name mit »m« beginnt.

Sobald Sie die Eingabe konkretisieren, reduziert sich die Menge der angezeigten Einträge in der Auswahlliste, sofern die neuen Zeichen vorher noch angezeigte Einträge ausschließen. Löschen Sie wieder ein Zeichen, vergrößert sich die Menge der angezeigten Einträge wieder.

Hinweis

Diese Aktualisierung der Einträge in der Auswahlliste erfolgt beeindruckend schnell. Die Datennachforderung passiert – sofern die Verbindung nicht hängt – nahezu unbemerkt.

Wenn Sie auf einen Eintrag in der Auswahlliste klicken, wird der Eintrag in das Listenfeld übernommen, zumindest in den meisten Browsern. Der Internet Explorer unterstützt das in der Version 6.0 nicht.

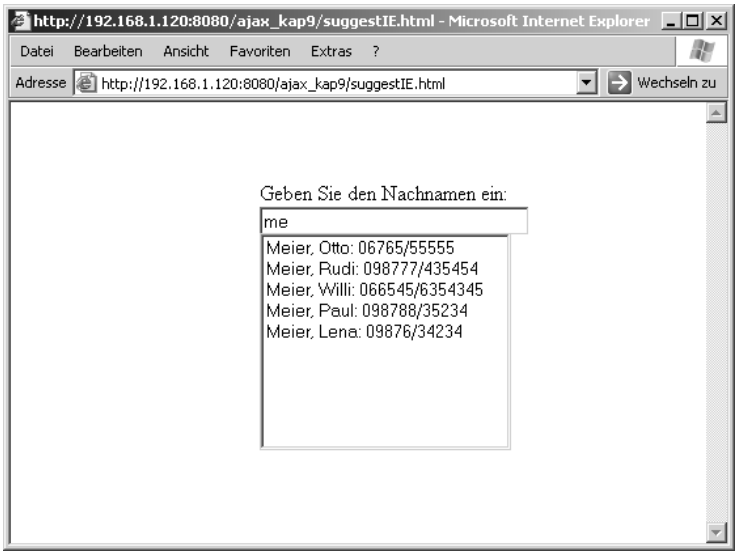


Abbildung 9.18: Wenn die Eingabe im Eingabefeld genauer wird, werden auch weniger Einträge angezeigt.

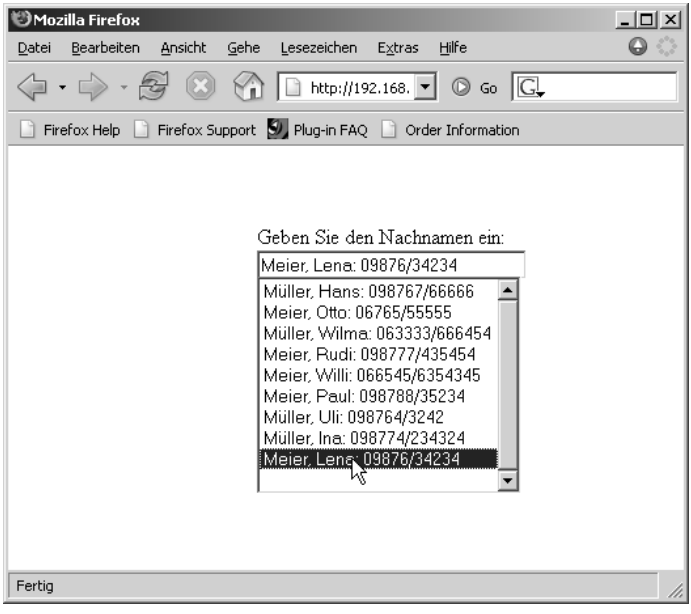


Abbildung 9.19: Ein Klick auf einen Eintrag in der Auswahlliste übernimmt den Wert in das Eingabefeld.

9.4.6 Aufbereiten einer XML-Antwort mit Java

Im letzten Kapitel haben wir gesehen, dass Sie bei AJAX-Anfragen XML-Daten senden können, die dann erst im Client aufbereitet werden. Bisher haben wir dazu statische XML-Seiten angefordert. Im Beispiel werden wir eine XML-Struktur mit Java generieren.

Suggest 2 – Senden einer Aufzählungsliste in XML-Form

Das Erzeugen von XML-Informationen auf dem Server können wir uns im Grunde ganz leicht machen, indem wir mit grundlegenden Java-Techniken dort einfach einen Text zusammensetzen, der mit XML-Elementen durchsetzt ist.

Wenn wir nun XML für unser konkretes Beispiel auf dem Server erzeugt und zum Client geschickt haben, generieren wir auf Client-Seite mithilfe des `node`-Objekts eine Aufzählungsliste, die unterhalb des Eingabefelds angezeigt wird. Diese wird rein zur Anzeige von Informationen dienen und in dieser Variante nicht eine Auswahl per Klick unterstützen. Um die Liste zu erzeugen, werden wir sowohl dynamisch die Listenpunkte erzeugen als auch bestehende Listenpunkte löschen.

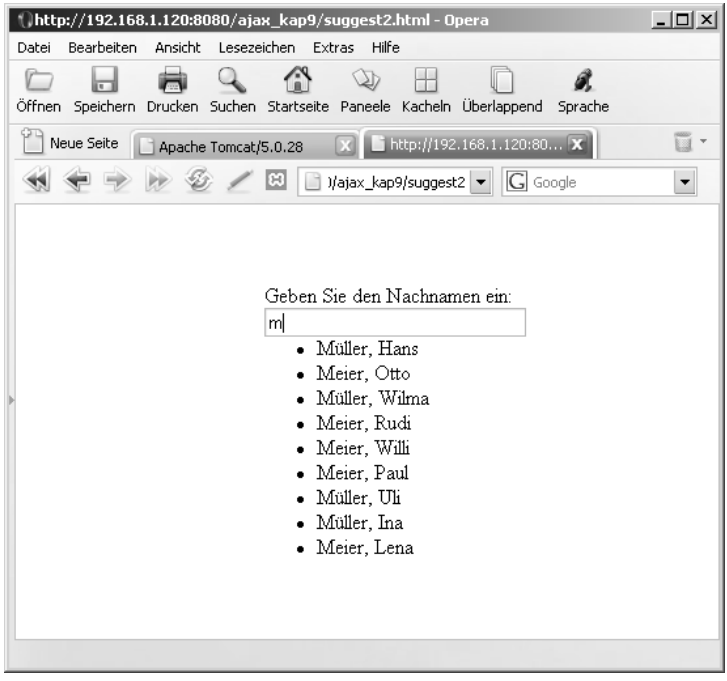


Abbildung 9.20: In dieser Variante wird aus XML-Daten im Client eine Aufzählungsliste generiert.

In der Webseite hat sich nicht viel getan. Nur steht dort statt eines -Containers der -Container für eine Aufzählungsliste:

Listing 9.39: In der Webseite muss nur der ‹span›-Container ausgetauscht werden.

```
12 <ul id="antwort"></ul>
```

Falls Sie Ihre externe JavaScript-Datei umbenennen, müssen Sie natürlich die Referenz darauf anpassen. In der JSP-Datei müssen wir die Antwort überarbeiten. Vom Server wird nun ein strukturiertes Dokument mit selbst definierten XML-Tags gesendet. Das sind die geänderten Teile:

Listing 9.40: Die Änderungen in der JSP-Datei

```
03 String text = "<adressen>";
...
16   text += "<datensatz><name>"+rs.getString(2)+"</name>" + ", " +
17        "<vname>"+rs.getString(3)+"</vname>" + ": " +
18        "<vorwahl>"+rs.getString(4)+"</vorwahl>" + "/" +
19        "<nr>"+rs.getString(5)+"</nr></datensatz>";
...
31 text += "</adressen>";
32 %>
33 <%= text %>
```

Die JSP-Datei erzeugt eine Antwort im XML-Format. Dabei wird in unserem Beispiel der Einfachheit halber auf einen Prolog verzichtet. Damit ist die XML-Antwort zwar nicht wohlgeformt, aber da wir darauf auf Client-Seite sowieso nicht achten werden, ist das vollkommen belanglos[17]. In Zeile 7 wird der Anfangs-Tag des Wurzelelements erstellt. Die Zeilen 14 bis 17 fügen ein Kindelement <datensatz> zusammen, das wiederum vier Kindelemente hat (<name>, <vname>, <vorwahl> und <nr>). Diese enthalten jeweils einen Textknoten, der mit dem passenden Wert aus der Datenbank gefüllt wird. Die Anzahl der <datensatz>-Elemente ergibt sich aufgrund der Anzahl der Datensätze in der Datenbank. In Zeile 21 wird dann das Wurzelelement geschlossen.

Auf JavaScript-Seite tut sich das Meiste. Genau genommen aber nur in der Funktion `handleResponse()`:

Listing 9.41: Die geänderte Funktion handleResponse()

```
32 function handleResponse() {
33   if(resObjekt.readyState == 4){
34     document.getElementById("antwort").style.visibility
35       = "visible";
36     xmlDok = resObjekt.responseXML;
```

17 *Im Gegenteil – Sie umgehen so einige Probleme bei der Navigation auf dem Baum im Client, die bei Verwendung eines Prologs auftreten.*

```
37      while(document.getElementById("antwort").hasChildNodes()) {
38        kntn = document.getElementById("antwort").firstChild;
39        document.getElementById("antwort").removeChild(kntn);
40      }
41      for(i = 0;i < xmlDok.childNodes[0].childNodes.length; i++)  {
42        newLi = document.createElement("li");
43        newOText = document.createTextNode(
44        xmlDok.getElementsByTagName("name")[i].childNodes[0].data
45        + ", " +
46        xmlDok.getElementsByTagName("vname")[i].childNodes[0].data
47        );
48        document.getElementById("antwort").appendChild(newOption);
49        document.getElementsByTagName("li")[i].appendChild(newLi);
50      }
51    }
52 }
```

In Zeile 36 erstellen wir eine Variable, die die XML-Antwort des Servers enthält (`xmlDok = resObjekt.responseXML;`).

Achtung

Der Umgang mit der Eigenschaft `responseXML` ist leider im Zusammenhang mit AJAX in einigen Browsern nicht zuverlässig implementiert. Hier haben Sie ein Beispiel, das in mehreren Browsern nicht funktionieren wird, obwohl im Grunde keine großen Unterschiede zu der angeforderten statischen XML-Datei des letzten Kapitels vorhanden sind. Wir besprechen dieses Beispiel aus dem Grund, weil hier sehr viel über die Verwendung von Baumstrukturen und `node` im Client zu sehen ist. Getestet ist es im Opera 8.5 und dort funktioniert es einwandfrei. Aber leider kommen diverse andere Browser damit nicht zurecht. Es ist zu hoffen, dass neuere Versionen dieser Browser die Eigenschaft `responseXML` irgendwann konsistent unterstützen, damit diese doch sehr mächtige Technik im Client gefahrlos verwendet werden kann.

In den Zeilen 37 bis 40 löschen wir eventuell bereits vorhandene Listenpunkte. Dies ist bei dieser Variante notwendig, denn sonst würde bei jedem nachfolgenden Hinzufügen von Kindelementen neuer Inhalt an bereits bestehende Textknoten angefügt. Ein solches Hinzufügen erfolgt jedes Mal, wenn ein Besucher im Eingabefeld ein Zeichen eingibt oder auch löscht.

In Zeile 37 testen wir, ob das Element mit der ID `antwort` (der `<ul>`-Container) bereits Elemente enthält. Ist das der Fall, wird die nachfolgende Schleife durchlaufen (`while(document.getElementById("antwort").hasChildNodes())`). In Zeile 38 nehmen wir das erste Kindelement und legen es in der Variablen `kntn` ab (`kntn = document.getElementById("antwort").firstChild;`). Diese Referenz auf den Knoten nutzen wir in Zeile

39, um den Knoten mit `removeChild()` zu löschen (`document.getElementById("antwort").removeChild(kntn);`). Wir führen die Löschaktion in einer Schleife so lange durch, bis keine Kindelemente mehr da sind. Daher können wir anschließend sicher sein, dass das `<ul>`-Element keine Kindelemente mehr enthält.

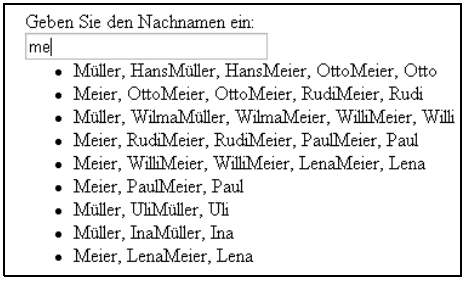


Abbildung 9.21: Wenn Sie vor dem Hinzufügen von Inhalten die bestehenden Elemente nicht löschen, passiert das hier Gezeigte.

Die folgende `for`-Schleife kennen Sie im Prinzip aus dem letzten Kapitel. Die XML-Antwort des Servers wird sukzessive durchlaufen. Die Anzahl der Durchläufe entspricht der Anzahl der Elemente vom Typ `<datensatz>` (Zeile 41 – `for(i = 0;i < xmlDok.childNodes[0].childNodes.length; i++)`). Bei jedem Durchlauf wird ein `<li>`-Element erzeugt (Zeile 42 – `newLi = document.createElement("li");`) und aus den Inhalten der Textknoten der Elemente `<name>`, `<vname>`, `<vorwahl>` und `<nr>` sowie einigen Trennzeichen der Inhalt eines jeden `<li>`-Elements generiert (Zeile 43 bis 47 – `newOText = document.createTextNode(xmlDok.getElementsByTagName("name")[i].childNodes[0].data + ", " + xmlDok.getElementsByTagName("vname")[i].childNodes[0].data);`).

In Zeile 48 fügen wir mit `document.getElementById("antwort").appendChild(newLi);` das `<li>`-Element dem `<ul>`-Element hinzu. Und in Zeile 49 wird mit `document.getElementsByTagName("li")[i].appendChild(newOText);` dem aktuellen `<li>`-Element der zusammengesetzte Text als Inhalt beigefügt.

XML-Daten mit Java dynamisch generieren

Im letzten Beispiel haben wir im Grunde gänzlich die Tatsache ignoriert, dass wir dabei XML erzeugen wollen. Der Antwortstring an den Client wurde einfach zusammengesetzt und »rein zufällig« mit XML-Elementen durchzogen. Wenn man in der Praxis so arbeiten würde, würde dies den Nutzen unzähliger bereits vorgefertigter Java-Technologien zur XML-Verarbeitung ignorieren. Java arbeitet aber mithilfe zahlreicher spezieller Klassen und Pakete hervorragend mit XML zusammen. Dies ist sicher auch aufgrund der Tatsache zu sehen, dass die Firma Sun massiv die Entwicklung von XML beeinflusst hat. Wie wir im Kapitel zu XML schon gesehen haben, gibt es dabei unter Java mehrere Möglichkeiten, wie Sie XML verarbeiten können. Dies können Sie selbstverständlich auch dazu nutzen, um auf Server-Seite mittels JSP oder Java Servlets die Antwort an einen Client aufzubereiten. Das wird in vielen komplexen und hochprofessionellen Webapplikationen auch genauso gemacht. Die aus-

schließliche Vorratshaltung von Geschäftslogik auf dem Server ist aus mehreren Gründen sehr oft sinnvoll – seien es Sicherheitsüberlegungen, unbedingt notwendige Voraussetzungen, die man im Client nicht garantieren kann, Zugriffsbeschränkungen des Clients, zentrale Wartbarkeit elementarer Prozesse der Geschäftslogik oder noch vieles mehr.

Sie können nun mittels Sax, DOM oder JDOM leicht XML-Dokumente lesen und mittels DOM und vor allem JDOM auch sehr komfortabel erstellen. So aufbereitete dynamische XML-Informationen können dann mittels AJAX abgefragt und in einer Webseite eingebaut werden. Dazu werden wir jetzt ein Beispiel durchspielen, das allerdings insbesondere zur XML-Verarbeitung mit Java nur rudimentär erklärt werden kann. Und leider gilt auch hier wieder festzuhalten, dass die Auswertung einer XML-Antwort im Client, die mittels AJAX angefordert wird, nur recht mühsam manuell erfolgt (wenn die Antwort mit `responseText` entgegengenommen wird) oder aber sehr uneinheitlich in den verschiedenen Browsern gehandhabt wird (wenn die Antwort mit `responseXML` entgegengenommen wird). Allerdings ist wie gesagt zu hoffen, dass zukünftige Browser-Versionen die Auswertung im Client verbessern. Deshalb wollen wir uns die dynamische Aufbereitung eines XML-Dokuments mit einem JSP ansehen, denn über kurz oder lang wird das der Weg sein, um mit AJAX-Applikationen sinnvoll auf dem Server XML-Daten zu generieren.

Die HTML-Datei können Sie so gut wie unverändert übernehmen (`suggest3.html`). Sie müssen nur auf eine neue externe JavaScript-Datei (`suggest3.js`) referenzieren. In dieser ist für diese Demonstration die Funktion `handleResponse()` extrem vereinfacht:

Listing 9.42: Die Entgegennahme der Antwort des Servers

```
function handleResponse() {
   if(resObjekt.readyState == 4){
  document.getElementById("antwort").style.visibility = "visible";
  document.getElementById("antwort").innerHTML = resObjekt.responseText;
}
}
```

In dem Beispiel machen wir nichts anderes, als die Antwort des Servers in einem `<div>`-Container anzuzeigen.

Tipp

Sie können mit der Antwort des Servers natürlich viel mehr machen, als diese nur in einem HTML-Container anzeigen. Wenn Sie die Antwort des Servers einer Variablen zuweisen, können Sie darauf sämtliche Techniken zur String-Verarbeitung anwenden, die Sie sich so vorstellen können – beispielsweise bestimmte Zeichen ersetzen (das wäre für XML-Tags zum Beispiel interessant) oder gezielt gewisse Dinge im String suchen. Das bedeutet, Sie würden die Antwort des Servers manuell mit JavaScript-Techniken verarbeiten.

Betrachten Sie den nachfolgenden Code eines JSP-Skripts (*suggest3.jsp*):

Listing 9.43: Dynamisches Erzeugen von XML-Daten mittels JDOM

```
01 <%@ page import="java.sql.*" %>
02 <%@ page import="java.io.*" %>
03 <%@ page import="java.util.*" %>
04 <%@ page import="org.jdom.*" %>
05 <%@ page import="org.jdom.output.*" %>
06 <%
07 String connStr = "jdbc:mysql://localhost:3306/" + "ajax?user=root&pwd=";
08 String name = request.getParameter("name");
09 String vergleich = "";
10 String sql = "SELECT * FROM `adressen` ORDER BY `id` ASC";
11 String text1="", text2="";
12 Document doc = new Document();
13 doc.setDocType(new DocType("html",
14  "-//W3C//DTD XHTML 1.0 Transitional//EN",
15  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"));
16 Element root = new Element("adressen");
17 doc.setRootElement(root);
18 Iterator it;
19 try {
20  Class.forName("com.mysql.jdbc.Driver");
21  Connection conn = DriverManager.getConnection(connStr);
22  Statement stmt = conn.createStatement();
23  ResultSet rs = stmt.executeQuery(sql);
24  while (rs.next()) {
25  vergleich = rs.getString(2).substring(0,
Math.min(name.length(),rs.getString(2).length()).toLowerCase();
26  if(name.equals(vergleich)) {
27    // Werte der Datenbankspalten der aktuellen Zeile
28    String a = rs.getString(2);
29    String b = rs.getString(3);
30    String c = rs.getString(4);
31    String d = rs.getString(5);
32    // Aufbau des Resultats als JDOM
33    // Element datensatz repräsentiert den Datensatz
34     Element datensatz = new Element("datensatz");
35    Element nname = new Element("name");
36    nname.addContent(a);
37    Element vname = new Element("vname");
38    vname.addContent(b);
39    Element vorwahl = new Element("vorwahl");
40    vorwahl.addContent(c);
41    Element nr = new Element("nr");
42    vorwahl.addContent(d);
43    // Hinzufügen der Elemente zum Element datensatz
44    datensatz.addContent(nname);
```

```
45    datensatz.addContent(vname);
46    datensatz.addContent(vorwahl);
47    datensatz.addContent(nr);
48    // Hinzufügen des Elements datensatz zum root-Element adressen
49    root.addContent(datensatz);
50    XMLOutputter xml_out = new XMLOutputter();
51    text1 = xml_out.outputString(doc);
52  }
53 }
54 rs.close();
55 conn.close();
56 } catch (ClassNotFoundException e) {
57  e.printStackTrace();
58 } catch (SQLException e) {
59  e.printStackTrace();
60 } catch (Exception e) {
61  e.printStackTrace();
62 }
63 %>
64 <%= text1 + "<hr />" %>
65 <%
66  it = doc.getDescendants();
67  while (it.hasNext()) {
68  text2 +=it.next().toString();
69  }
70 %>
71 <%= text2 %>
```

In dem JSP werden zuerst verschiedene Standardpakete von Java importiert. Die Pakete `java.sql`, `java.io` und `java.util` dürften Ihnen bekannt sein. Die Pakete `org.jdom` und `org.jdom.output` sind hingegen speziell für die XML-Verarbeitung mit JDOM gedacht. Von Zeile 6 bis 11 basiert nichts Neues. Es sollte Ihnen auffallen, dass es zwei String-Variablen gibt (`String text1="", text2="";`).

In Zeile 12 treffen wir die Vorbereitung zum Erzeugen einer XML-Struktur im Speicher des Rechners. Man nennt so eine XML-Struktur dann einen JDOM. Ein Erzeugen eines JDOM als Repräsentation einer XML-Struktur ist mit Java einfach. Sie müssen bloß ein Objekt vom Typ `org.jdom.Document` erzeugen und ihm mindestens ein Element (das Wurzelelement) hinzufügen. Damit wird implizit bereits eine XML-Deklaration erzeugt und für die Wohlgeformtheit der erzeugten Struktur gesorgt. Die Zeilen 12, 16 und 17 zeigen, wie leicht das geht:

```
12 Document doc = new Document();
16 Element root = new Element("adressen");
17 doc.setRootElement(root);
```

Damit wird im Speicher des Rechners bereits eine vollständige und wohl geformte XML-Struktur abgebildet.

Die Zeilen 13 bis 15 legen den Dokumententyp fest. Dies wäre nicht notwendig, aber hier wird explizit HTML als Dokumententyp angegeben (`Doc.setDocType(new DocType("html", "-//W3C//DTD XHTML 1.0 Transitional//EN", "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"));`).

Die bisherige XML-Struktur enthält außer dem Prolog und einem Wurzelelement noch keinen Inhalt. Wir müssen uns nun noch um das Hinzufügen von Elementen und gegebenenfalls Attributen sowie Inhalten zum JDOM kümmern, sofern wir diese Informationen in der XML-Struktur abbilden wollen.

Bis Zeile 26 ist das JSP wieder unverändert gegenüber der Vorgängerversion (mit Ausnahme der Einführung einer Variablen vom Typ `java.util.Iterator`, die wir später verwenden wollen). Mit einem Iterator sind wir in der Lage, eine Liste sukzessive zu durchlaufen. In den Zeilen 28 bis 31 werden die Werte der Felder aus dem aktuell selektierten Datensatz dieses Mal Stringvariablen zugewiesen.

Interessant wird es wieder ab Zeile 34. Über das Wurzelelement sind Sie nun in der Lage, beliebige XML-Strukturen durch das Hinzufügen von Elementen und Attributen zu erzeugen. Wie Sie sehen, erzeugen wir nach Belieben mit `java.jdom.Element` neue Elemente und fügen diesen über den Aufruf `addContent()` weitere Elemente oder Texte hinzu. Attribute erzeugt bei Bedarf die Methode `setAttribute()`, die eventuell vorhandene Attribute überschreibt. Abschließend bauen wir in Zeile 49 das Root-Element zusammen.

Nun stellt sich ganz zum Abschluss die Frage, was wir mit dem so aufgebauten JDOM machen wollen? An dieser Stelle liefert das Beispiel zwei verschiedene Möglichkeiten. In der Praxis werden Sie sich natürlich für eine Lösung entscheiden. Aber um zu sehen, wie und was man zum Client schicken kann, werden hier beide Varianten demonstriert.

In dem Listing verwenden wir ein Objekt vom Typ `org.jdom.output.XMLOutputter`, um den zuvor erzeugten JDOM in einer aufbereiteten Form zum Client zu schicken. So ein `XMLOutputter` ist mehrfach überladen und kann nahezu mit jeder Form eines Stream oder Writer zurechtkommen. Das nutzen wir hier allerdings nicht aus – der erzeugte JDOM wird einfach einer String-Variablen `text1` zugewiesen. Diese wird in Zeile 64 an den Client geschickt.

Da es bei jeglicher Ausgabe über Streams und Writer zu einer `IOException` kommen kann, muss diese abgefangen werden.

Die zweite Variante des JDOM, die wir zum Client schicken wollen, setzen wir darüber zusammen, dass zunächst alle Elemente des JDOM in der gleichen Reihenfolge durchlaufen (traversiert) werden, wie sie erstellt wurden. Dieses Vorgehen entspricht

dem Durchlesen der XML-Datei von oben nach unten mit gleichzeitiger Verarbeitung. An dieser Stelle kommt unser Iterator zum Einsatz. Die Methode `getDescendants()` (Nachkommen) des `Documents`-Objekts gibt einen entsprechenden `java.util.Iterator` zurück, den man anschließend auswerten kann (Zeile 66 bis 69). In Zeile 71 wird das Ergebnis dieses Traversierens an den Client geschickt.

Nun stellt sich noch die Frage, was beim Client tatsächlich ankommt? Der JDOM, der über die Variable `text1` zurückgegeben wird, ist ein normales XML-Dokument. Wenn Sie dieses im Anzeigebereich des Browsers anzeigen (so wie wir das tun), werden Sie die XML-Elemente nicht sehen, sondern nur den Inhalt von nichtleeren Elementen. Dieses Verhalten haben wir im Kapitel zu XML behandelt. Sie könnten nun mithilfe von Style Sheets oder manueller JavaScript-Verarbeitung dieses XML-Dokument so aufbereiten, wie Sie es benötigen.

Den Inhalt in der Variablen `text2` können Sie samt Struktur (allerdings immer noch ohne die Darstellung der tatsächlichen Tags) im Rahmen der Webseite sehen. Die Informationen werden in eckigen Klammern aufbereitet und sequenziell nach Typ unterteilt zur Verfügung gestellt. Dies können Sie mittels JavaScript sehr bequem verwerten.

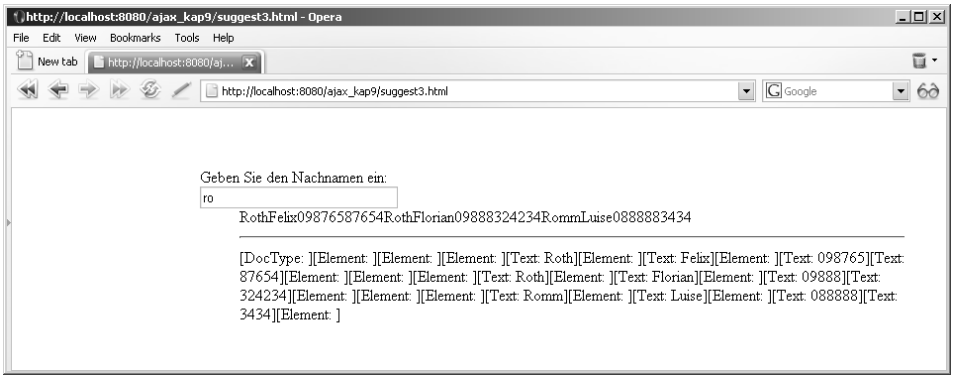


Abbildung 9.22: Die Rückgabe des Servers ohne die Darstellung der XML-Tags

Tipp

Wenn Sie die gesamte XML-Struktur einschließlich der Tags auf einfache Weise sehen wollen, notieren Sie im Java-Skript in der Funktion `handleResponse()` einfach ein `alert()` der folgenden Art:

Listing 9.44: Ausgabe des vollständigen Inhalts der Antwort

```
alert(resObjekt.responseText);
```

Hier erfolgt die Darstellung des Inhalts rein mittels JavaScript und da werden XML-Tags nicht interpretiert und damit ausgeblendet.

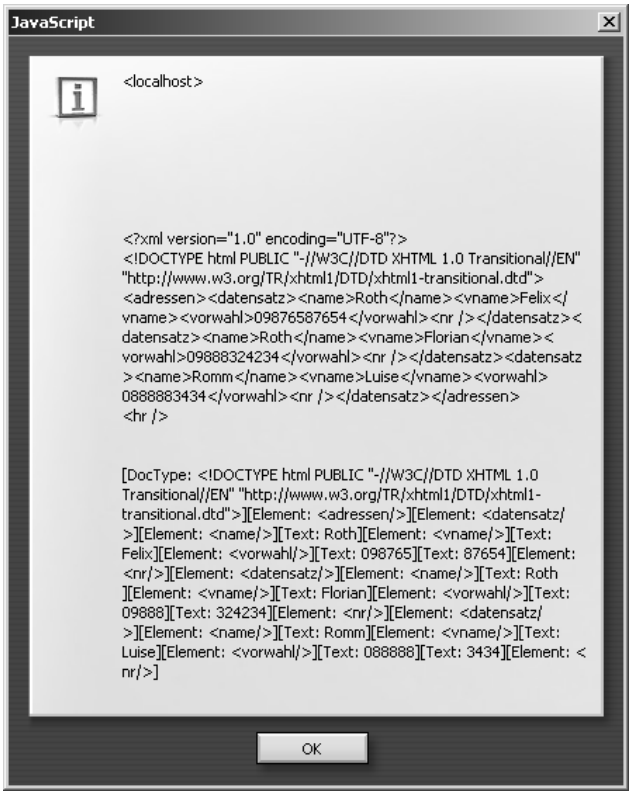


Abbildung 9.23: So sehen Sie auch die XML-Tags.

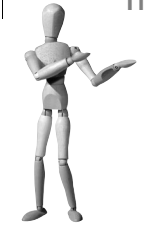
Hinweis

Sie sollten durch die uneinheitliche Verwendung von `responseXML` in den Browsern und anderen recht mühsamen Vorgängen zum Verwerten einer baumartigen Struktur im Client ein Gefühl dafür bekommen haben, dass man die angeforderten Informationen so weit wie möglich auf dem Server vorbereiten und möglichst geschlossene HTML- oder XHTML-Fragmente schicken sollte. Es ist auch in AJAX mehr als nur sinnvoll, die Geschäftslogik einer Webapplikation so weit wie möglich (wenn nicht gar ausschließlich) auf dem Server zu realisieren. Damit umgehen Sie vor allem Komplikationen im Client, die Sie sonst mit zahlreichen Fallunterscheidungen kompensieren müssen. Allerdings muss man auch festhalten, dass AJAX noch diverse Entwicklungen und Standardisierungen durchlaufen muss und wird, bevor die Techniken auf Client-Seite allesamt zuverlässig eingesetzt werden können.

9.5 Java-Servlets

Sie haben im Laufe dieses Buchs gesehen, dass Sie mit JSP und einem Java-Container wie Tomcat eine Antwort des Servers an den aufrufenden Client dynamisch generieren können. Wozu nun Java Servlets? Die Aufgaben von JSP und Java Servlets überschneiden sich in der Tat. Rein technisch gibt es elementare Unterschiede, aber mit Grunde können Sie mit einem JSP genau das Gleiche tun, was Sie auch mit Servlets machen können. Und da JSP vor ihrer Ausführung sowieso in Servlets übersetzt werden, scheinen die Unterschiede kaum erklärbar. Dennoch – wenn Sie typische Webapplikationen ansehen, die mit Java auf Server-Seite arbeiten, wird Ihnen auffallen, dass bei komplexeren Anwendungen meist Servlets (oder noch spezialisiertere Java-Dinge) statt JSP zum Einsatz kommen. Also kann man festhalten, dass JSP wohl für einfachere Webapplikationen sinnvoll sind, während Servlets zunehmend an Bedeutung gewinnen, wenn die Applikation komplexer wird. Der Grund ist, dass bei JSP Gestaltungslogik und Funktionalität durchmischt sind. In einem JSP haben Sie – wie Sie bisher schon mehrfach gesehen haben – reines (X)HTML oder XML durchmischt mit Java-Anweisungen. Dies kann von Vorteil sein, wenn man die optischen Teile einer JSP-generierten Seite von einem Webdesigner ohne Java-Kenntnisse und die logischen Teile von einem Programmierer erstellen lassen möchte.

Je komplexer die Applikation wird, desto mehr wünscht man sich jedoch, dass es eine klare Trennung zwischen Struktur, Funktionalität bzw. Geschäftslogik und Layout gibt. Servlets kapseln nun Java-Funktionalität vollständig in einem eigenen Bereich. Bei Servlets arbeiten sie ausschließlich mit Java-Quellcode. Auch dies ist ein zweischneidiges Schwert, denn alle HTML-Ausgaben erfolgen über Ausgabemethoden im Java-Quellcode. Das halten manche für nicht vernünftig lesbar, wohingegen andere[18] genau diese Kapselung für ideal halten.



Tipp

Spätestens beim Erstellen von Servlets sollten Sie auf jeden Fall eine Java-IDE verwenden. Diese bietet in der Regel eine erhebliche Unterstützung für die Erstellung oder aber Sie können solche nachrüsten. Für Eclipse gibt es zum Beispiel ein Plug-in namen **Lomboz** mit Unterstützung von Servlets, aber auch JSP.

18 *Ich zähle mich zum Beispiel dazu – ich finde Ausgabe von HTML-Code über Java-Methoden aus einem Servlet heraus perfekt und sehr gut lesbar. Dafür kann ich mit dem Mischmasch in einem JSP wenig anfangen.*

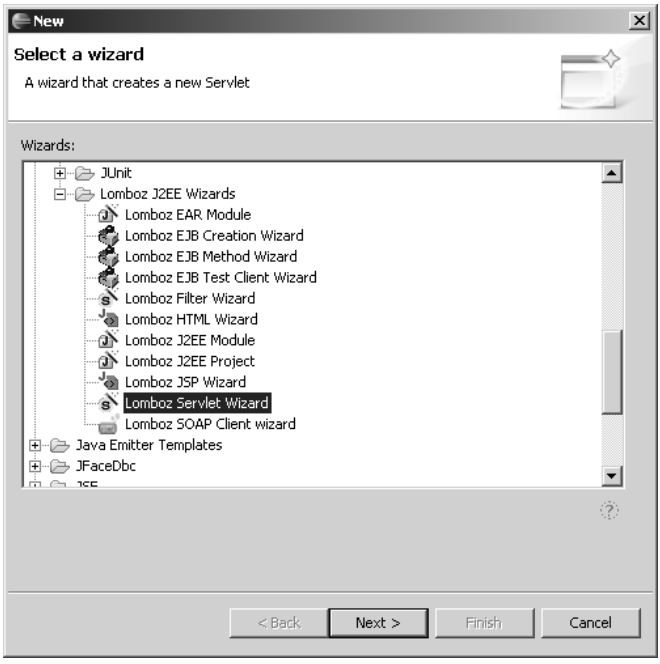


Abbildung 9.24: Unterstützung bei der Erstellung von Servlets und JSP

9.5.1 Was sind Servlets?

Java-Servlets werden von Sun selbst wie folgt definiert: »A servlet can almost be thought of as an applet that runs on the server side – without a face. « Sun vergleicht also Servlets explizit mit Java-Applets – nur ohne ein sichtbares Benutzer-Interface und die Tatsache, dass Servlets nicht im Client, sondern im Server-Bereich ausgeführt werden. Der Bezeichner **Servlet** ist eine Wortkreation aus den Begriffen **Server** und **Applet**. Diese offizielle Argumentation von Sun muss man vor allem historisch sehen, denn als Servlets eingeführt wurden, waren Java-Applets ein Synonym für Java überhaupt. Und Sun versuchte zu diesem Zeitpunkt, Servlets als Alternative zu dem damals noch sehr populären CGI zu etablieren. Sehr vereinfacht sind Servlets also eine Technologie, mit der in Java geschriebene vollständige und geschlossene Programme (nicht mit HTML gemischte Skripte wie bei JSP) auf dem Server Anfragen von Clients über HTTP empfangen, verarbeiten und beantworten können.

9.5.2 Die technischen Hintergründe

Servlets sind vollständige Java-Klassen, die zwingend die Schnittstelle `javax.servlet.Servlet` oder eine davon abgeleitete Schnittstelle implementieren. Die Implementierung einer solchen Schnittstelle gewährleistet, dass ein jedes Servlet Methoden

implementiert, mit denen HTTP-Anfragen vom Client verarbeitet werden können und Methoden bereitstehen, über die Servlets eine Antwort zum Client schicken können.

Entgegennahme einer Client-Anfrage

Die beiden Methoden `doGet()` und `doPost()` werden beispielsweise überschrieben, um auf die beiden wichtigsten HTTP-Anfragearten GET und POST individuell reagieren zu können.

Achtung

Die Namensgebung von `doGet()` und `doPost()` ist möglicherweise ein wenig missverständlich. Auf den ersten Blick könnten Java-Laien vermuten, dass `doGet()` für die Entgegennahme einer Client-Anfrage zuständig ist und `doPost()` für das »Posten« (also eine Antwort an den Client) gebraucht wird. Dies ist aber explizit nicht so! Beide Methoden bearbeiten Anfragen vom Client und die Namensgebung ergibt sich ausschließlich aus der Art der Datenübertragung per HTTP.

Wenn Sie unabhängig von der Art der Anfrage auf eine HTTP-Anfrage reagieren wollen, können Sie die `service()`-Methode verwenden. Sie wird in jedem Fall ausgeführt, wenn an ein Servlet eine Anfrage gestellt wird. Alle drei Methoden können Parameter vom Typ `javax.servlet.http.HttpServletRequest` und `javax.servlet.http.HttpServletResponse` als Parameter übergeben bekommen (siehe unten)[19]. Sie können auch deren Superklassen `ServletRequest` und `ServletResponse` aus dem Paket `javax.servlet` verwenden.

Alle drei Methoden werfen Exceptions vom Typ `javax.servlet.ServletException` und `java.io.IOException` aus.

Die Superklasse HttpServlet

Nun ist es in der Praxis so, dass eigene Servlets in der Regel Servlet-Schnittstellen nicht unmittelbar implementieren, sondern standardisierte Basisklassen erweitern (als Superklasse verwenden), die bereits diese Schnittstellen implementieren und vervollständigt haben. Die wohl wichtigste Basisklasse ist `javax.servlet.http.HttpServlet`.

Die Übergabewerte an die Methoden zur Entgegennahme einer Anfrage

Die Methoden `doGet()`, `doPost()` und `service()` verwenden nun als Parameter Objekte, über die zahlreiche Informationen über den anfragenden Client und Informationen für den angefragten Server bzw. die darüber angeforderte Ressource bereitstehen. Diese Objekte stellen ziemlich genau das dar, was über die impliziten Objekte `request` und `response` im Fall von JSP verfügbar ist. Im Fall von Servlets sind dies das `HttpServletRequest`- und das `HttpServletResponse`-Objekt.

19 *Daneben gibt es diverse weitere Methoden wie* `doPut()`, *die wir aber nicht weiter besprechen.*

HttpServletRequest
Ein `HttpServletRequest`-Objekt kapselt die Anfrage eines Browsers und stellt Methoden bereit, mit denen sich Informationen aus der Anfrage auslesen lassen. Zu den Informationen gehören die übergebenen Parameterwerte oder Informationen zu Cookies. Die entsprechenden Methoden sind im Fall von JSP extrem verwandt bzw. sogar identisch.

HttpServletResponse
Ein `HttpServletResponse`-Objekt ist das Gegenstück zum `HttpServletRequest`-Objekt und kümmert sich um das Versenden der Antwort zum Client. Die wesentlichen Methoden des `HttpServletResponse`-Objekts kümmern sich um das Setzen neuer Cookies, das Setzen von Header-Feldern, das Setzen des Status der Antwort und das Senden von Fehlermeldungen. Dazu stellt ein `HttpServletResponse`-Objekt Methoden bereit, um verschiedenartig aufbereiteten Code an den Client zu senden. Hauptsächlich wird über die Methode `getWriter()` ein Objekt vom Typ `PrintWriter` besorgt, um HTML-Code an den Browser zu senden. Der MIME-Type der gesendeten Daten lässt sich über die Methode `setContentType()` festlegen.

9.5.3 Eintragen von Metainformationen und das Deployen von Servlets

Ein Servlet lässt sich nun nicht einfach so unter Tomcat oder einem anderen Webserver mit Java-Unterstützung kopieren und ausführen. Der Server braucht verschiedene **Metainformationen**, um das Servlet ausführen zu können. Diese Metainformationen stellen auf der einen Seite ein Sicherheitsfeature dar, auf der anderen Seite sind Sie eine Möglichkeit, Pfadangaben und Zugriffsregeln für ein Servlet sehr flexibel zu handhaben. Die gesamte Thematik mit all ihren Facetten sprengt bei weitem unseren Rahmen, aber wir kommen nicht umhin, zumindestens die elementarsten Regeln durchzusprechen. Andernfalls könnten Sie unsere Servlet-Beispiele überhaupt nicht ausführen.

Metainformationen zu jedem Servlet werden bei Tomcat in einer XML-Datei namens `web.xml` hinterlegt, dem so genannten **Deployment Descriptor**. Diese XML-Datei wird zusammen mit der kompilierten Klasse des Servlets sowie gegebenenfalls weiteren benötigten Klassen auf dem Server in einer vorgegebenen Struktur bereitgestellt. Die XML-Datei *web.xml* muss sich für jedes Servlet-Projekt im Unterverzeichnis *WEB-INF* befinden. Darin befinden sich auch die Servlet-Klassen (in einem parallel zu *web.xml* befindlichen Unterverzeichnis mit Namen *classes*).

Die Beschreibung des vollständigen Aufbaus der Datei `web.xml` und aller Möglichkeiten, die Sie darin haben, sprengt natürlich unseren Rahmen. Wichtig zum reinen Ausführen eines Servlet sind allerdings hauptsächlich zwei Stellen – genau genommen zwei XML-Elemente, mit denen Sie Metainformationen zum Servlet spezifizieren müssen.

Es ist einmal das Element `servlet`. Darin geben Sie ein Element `servlet-name` an, mit dem Sie den Namen des Servlet festlegen. Dieser Name ist eine eindeutige Kennung für das Servlet in `web.xml`. Als weiteres Element müssen Sie `servlet-class` angeben. Das ist der Name der Servlet-Klasse (mit Angabe einer eventuell verwendeten Paketstruktur). Beispiel:

Listing 9.45: Teil 1 der Informationen in web.xml

```
<servlet>
     <servlet-name>RalphsReader</servlet-name>
     <servlet-class>de.rjs.xmlutils.RalphsReader</servlet-class>
 </servlet>
```

Das zweite Element, das sie zwingend in der Datei `web.xml` spezifizieren müssen, heißt `servlet-mapping`. Damit legen Sie fest, wie das Servlet von außen zugänglich ist. Tomcat kann einem Aufrufer einen virtuellen Pfad zur Verfügung stellen, der vollkommen unabhängig von der physikalischen Struktur des *webapps*-Verzeichnis ist. Und hier legen Sie diese Zugangswege fest. Im Inneren von `servlet-mapping` müssen Sie wieder das Element `servlet-name` angeben. Dies ist die Referenz auf die eigentliche Servlet-Klasse, wie sie im Element `servlet` spezifiziert wurde. Das Element `url-pattern` legt nun den relativen Pfad fest, über den das Servlet ab der Adresse des Tomcat-Servers erreichbar ist.

Beispiel:

Listing 9.46: Angabe eines relativen Pfads für das Servlet

```
<servlet-mapping>
    <servlet-name>RalphsReader</servlet-name>
    <url-pattern>/meinkunde/RalphsReader</url-pattern>
</servlet-mapping>
```

Für das oben beschriebene Beispiel wäre das Servlet zum Beispiel nun so zu erreichen:

Listing 9.47: Der vollständige URL für das Servlet

```
http://localhost:8080/meinkunde/RalphsReader
```

Achtung

Beachten Sie, dass Tomcat normalerweise bei jeder Änderung in der Datei `web.xml` neu gestartet werden muss oder Sie über den Administrationsbereich von Tomcat die Datei `web.xml` neu einlesen müssen. Andernfalls ist Tomcat das Servlet nicht bekannt und es gibt beim Aufruf eine Fehlermeldung. Das kann man zwar mithilfe einer so genannten **überwachten Ressource** anpassen, aber das sprengt unseren Rahmen.

Nachfolgend finden Sie den Aufbau der `web.xml`, wie sie für unsere nachfolgenden drei Servlet-Beispiele aufgebaut sein muss:

Listing 9.48: Die Datei web.xml für die nachfolgenden drei Servlet-Beispiele

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <servlet>
        <servlet-name>Zustand</servlet-name>
        <servlet-class>Zustand</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>LoginTest</servlet-name>
        <servlet-class>LoginTest</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>Altzustand</servlet-name>
        <servlet-class>Altzustand</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Zustand</servlet-name>
        <url-pattern>/Zustand</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>LoginTest</servlet-name>
        <url-pattern>/LoginTest</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Altzustand</servlet-name>
        <url-pattern>/Altzustand</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
   <welcome-file>
            index.jsp
        </welcome-file>
    </welcome-file-list>
</web-app>
```

Hinweis

In der Praxis werden sämtliche Dateien oft in einer einzigen Archivdatei zusammengeführt (das Webarchiv vom Typ *.war*, das uns bereits begegnet ist). Dieses wiederum wird Tomcat über eine von ihm bereitgestellte Funktionalität zur Verfügung gestellt (das so genannte Deployment). Damit wird die Bereitstellung eines Servlet natürlich viel einfacher.

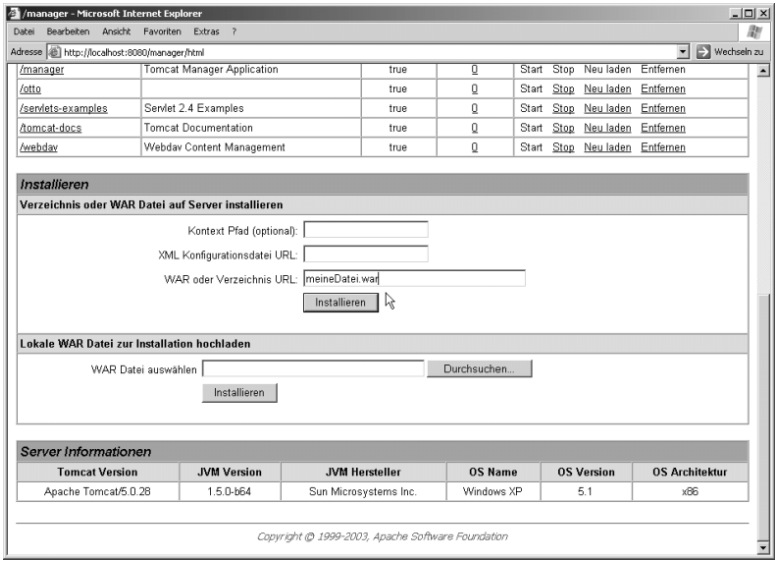


Abbildung 9.25: Über den Administrationsdialog von Tomcat können Sie eine .war-Datei installieren.

9.5.4 Beispiel 3 – Verfolgen einer Sitzung und Wiederherstellen eines vorherigen Zustands

Wir werden nun die Erstellung eines Servlet in Verbindung mit AJAX in der Praxis durchspielen. Dies ist für dieses Kapitel das dritte vollständige Beispiel und wird das Anwenderverhalten während einer Sitzung protokollieren. Dabei werden die relevanten Informationen in der Datenbank abgelegt und diese Information wird mittels einer AJAX-Anfrage dorthin befördert. Mit dieser Verwaltung des Anwenderverhaltens können wir dann auch die Funktion der ZURÜCK-Schaltfläche im Browser in einer AJAX-Applikation nachbilden, da diese ja bei AJAX-Applikationen nicht in der gewohnten Weise zur Verfügung steht.

Aber auch unabhängig von der Nachbildung der Funktionalität der ZURÜCK-Schaltfläche – bei vielen Webapplikationen ist es wichtig, eine Sitzung zu verfolgen. Das bedeutet, Sie verfolgen Aktionen eines Anwenders sowie möglicherweise auch die

daraus resultierenden Antworten des Servers. Denken Sie an einen Onlineshop. Diese Art Webapplikation hat ja schon mehrfach als Beispiel herhalten müssen. Ein Anwender wird während einer typischen Sitzung mehrere Schritte durchführen (Produktauswahl mit eventuell mehreren Produkten, Registrierung der Personendaten und Zahlungsmodalitäten, zur Kasse gehen und bezahlen etc.), die bei der endgültigen Bestellung (dem letztendlichen Ende der Sitzung) alle noch präsent sein müssen.

Sie können nun die Daten in Form von **Cookies** auf dem Client-Rechner speichern, aber dieses Verfahren hat sehr viele Schwächen. Darauf gehen wir im nächsten Kapitel noch ein. Eine sehr sinnvolle Alternative bei AJAX-Applikationen ist das Speichern von Sitzungsdaten in JavaScript-Variablen respektive Arrays. Das ist zwar bei konventionellen Webapplikationen selten eine vernünftige Lösung. Denn mit jeder Neuanforderung von Daten wird die bestehende Webseite aus dem Browser entfernt und damit sind auch die darüber gespeicherten Daten in JavaScript-Variablen und -Arrays nicht mehr verfügbar. Zwar gibt es Möglichkeiten, in JavaScript gespeicherte Werte an die neue Seite zu übergeben (zum Beispiel mit versteckten Formularfeldern, die beim Aufruf der neuen Seite an diese weitergereicht werden), aber das ist mühsam und wenig elegant. Jetzt gestattet eine AJAX-Applikation es jedoch, dass Sie im Extremfall auch bei einer komplexen Webpräsenz mit nur einer (X)HTML-Seite auskommen. Wenn alle Änderungen der Seite über `XMLHttpRequest`-Objekte angefordert werden, braucht diese Seite niemals erneuert zu werden. Und damit stehen auch alle JavaScript-Variablen und -Arrays während einer vollständigen Sitzung zur Verfügung.

Dennoch – auch bei einer AJAX-Applikation ist diese Art der Sitzungsverfolgung nur eingeschränkt zu verwenden. Sie ist beispielsweise dann nicht sinnvoll, wenn mehrere Webseiten benötigt werden. Oder auch dann nicht, wenn gewährleistet werden soll, dass Sitzungsdaten nicht manipuliert werden. Problematisch ist es ebenfalls, wenn Sitzungsdaten längerfristig gespeichert oder allgemein auf dem Server protokolliert werden müssen. In dem Fall ist eine Speicherung in einer Datenbank sehr vernünftig.

Wir könnten nun ohne große Probleme die beiden letzten JSP-Beispiele nehmen und jede Tastatureingabe eines Anwenders in dem Formlularfeld in einem eigenen Datensatz protokollieren. Aber das ist sicher kaum sinnvoll. Zum einen wird die Menge der Daten einfach zu groß.

Zwar könnte man bei der Protokollierung jedes einzelnen Tastendrucks mit einer gewissen Logik nur Schlüsselsituationen als entscheidende Zustände der Sitzung identifizieren, aber das ist sehr mühsam und auch dann werden die gespeicherten Daten wahrscheinlich viel zu granulat. Aber zum anderen gilt ebenso, dass das Wiederherstellen einer vorherigen Situation einer Webapplikation ja bei konventionellen Webanwendungen explizit an eine Anwenderaktion wie die Betätigung der ZURÜCK-Schaltfläche des Browsers oder einen Zugriff auf die History des Browsers gekoppelt ist. Aus dem Grund wandeln wir unser Beispiel wieder etwas ab. Wie die beiden Beispiele zuvor soll ein Anwender in einem Formulareingabefeld einen Suchbegriff eingeben können. Und wie im ersten Beispiel wird dem Anwender in einer Auswahlliste

das Ergebnis der Abfrage angezeigt. Aus dieser kann er per Klick einen Eintrag in das Eingabefeld übernehmen. Aber die Datenbankabfrage per AJAX soll nicht mit `onkeyup` nach jedem eingegebenen Zeichen ausgelöst werden, sondern mit einem Klick auf eine Schaltfläche. Und die bis dahin im Eingabefeld eingegebenen Zeichen protokollieren wir als entscheidenden Zustand der Sitzung als vollständigen String in einer Tabelle in der Datenbank als eigenständigen Datensatz.[20]

Abbildung 9.26: Wenn jedes Zeichen eines Anwenders in einem eigenen Datensatz abgelegt wird, erhalten Sie eine Unmenge an Daten.

Diese Speicherung entscheidender Zustände der Webapplikation lässt uns einen beliebigen vorherigen Zustand wiederherstellen, wobei wir nur die Wiederherstellung des unmittelbar letzten Zustands realisieren wollen. Das bedeutet, der Anwender kann mit einer gezielten Aktion (Klick auf eine Schaltfläche) die Eingaben reproduzieren, die er vor der letzten Abfrage eingegeben hatte.

20 *Sie können selbstverständlich auch die Abfrage wie gehabt lassen und die Speicherung des Sitzungszustands zum Beispiel mit der Auswahl aus dem Listenfeld koppeln (einem Klick in die Auswahlliste, wie es im ersten Beispiel realisiert ist). Sie haben alle Möglichkeiten, die eine sinnvolle Verfolgung der Sitzung gewährleisten.*

Achtung

Das nachfolgende Beispiel berücksichtigt nur den Fall, dass genau eine Sitzung auf dem Server protokolliert wird. Wenn Sie mehrere Browser öffnen und parallel Daten vom Server anfordern oder auch mehrere Anwender gleichzeitig mit dem Server verbunden sind, werden diese Anfragen nicht getrennt. Das Beispiel demonstriert, um nicht unnötig kompliziert zu werden, also keine echte Session-Verwaltung verschiedener Sitzungen.

Wenn Sie das umsetzen wollen, können Sie z.B. eine Tabelle zum Protokollieren einer Sitzung auch aus dem Servlet selbst mit SQL dynamisch erzeugen. Die Tabelle `verlauf` aus unserem Beispiel würde mit folgendem SQL-Befehl generiert:

Listing 9.49: Erstellen einer Tabelle mit SQL

```
CREATE TABLE `verlauf` (`id` int(11) NOT NULL auto_increment,
`eingabe` varchar(120) NOT NULL default '', PRIMARY KEY (`id`))
ENGINE=MyISAM DEFAULT CHARSET=utf8 AUTO_INCREMENT=3
```

Sie müssen diese SQL-Syntax nur mit einem entsprechenden Aufruf vor dem ersten Schreiben von Werten in die Tabelle abschicken.

Und diese dynamische Erzeugung mit dem Servlet selbst eröffnet eine Menge Möglichkeiten, insbesondere die angepasste Wahl eines Tabellennamens. Eine solche dynamische Erzeugung einer Protokolltabelle erlaubt z.B. eben eine echte Verwaltung verschiedener Sessions, wenn mehrere Besucher synchron protokolliert werden müssen, also zeitgleich mehrere Sitzungen laufen. Sie können für jeden Besucher eine eigene Tabelle anlegen, deren Name sich eindeutig ergibt, zum Beispiel über einen Timestamp (Zeitstempel). Solange die Sitzung läuft, verwalten Sie in der Tabelle alle temporären Informationen, und wenn die Sitzung beendet ist, beseitigen Sie die Tabelle wieder, sobald Sie sie nicht mehr benötigen[21]. Der SQL-Befehl zum Löschen der Tabelle würde übrigens so aussehen:

Listing 9.50: Löschen einer Tabelle mit SQL

```
DROP TABLE `verlauf`
```

21 *Zum Beispiel, wenn die Bestellung abgeschlossen ist und die Daten an ein Transaktionssystem weitergereicht wurden.*

Betrachten Sie zuerst die neue Webseite *zustand.html*:

Listing 9.51: Die neue Webseite mit einem Eingabefeld und zwei Schaltflächen

```
01 <html>
02 <link rel="stylesheet" href="stil2.css" type="text/css">
03 <script language="JavaScript" src="zustand.js"></script>
04 <body>
05 <br />
06 <form name="f1"><span id="name">
07 Geben Sie den Nachnamen ein:<br />
08 <input name="name" type="text" size="30" /></span>
09 <span id="antwort"></span>
10 </span>
11 <input type="button" size="30" onClick="sndReq()" value="Suche"  id="suche" />
12 <input type="button" size="30" onClick="sndReq2()" value="Zur&uuml;ck" ↩
   id="retur" />
13 </form>
14 </body>
15 </html>
```

In Zeile 2 wird die Webseite wieder mit einem externen Style Sheet verlinkt (`<link rel="stylesheet" href="stil2.css" type="text/css">`) und in Zeile 3 mit einer externen JavaScript-Datei (`<script language="JavaScript" src="zustand.js"></script>`). Wie Sie sehen, enthält die Seite aber nachfolgend nun keine Registrierung des `onkeyup`-Ereignisses beim `window`-Objekt mehr. Stattdessen finden Sie dieses Mal zwei Schaltflächen in dem Formular vor, die mit dem Eventhandler `onClick` versehen sind (Zeile 11 und 12)[22]. Diese stellen eine Schaltfläche bereit, über die die Suche in der Datenbank gestartet wird (über den Aufruf der Funktion `sndReq()`), und eine weitere Schaltfläche, mit der die Funktionalität der ZURÜCK-Schaltfläche des Browsers nachgebildet wird (über den Aufruf der Funktion `sndReq2()`). Wie schon erwähnt, ist es ein grundsätzliches Problem von AJAX-Applikationen, den Zustand vor einer Datenanforderung wiederherzustellen, wenn der Besucher die `Zurück`-Schaltfläche des Browsers betätigt oder sonst die History des Browsers verwenden möchte. In dem Beispiel werden wir im Ansatz zeigen, wie Sie dieses Verhalten in AJAX-Applikationen nachbilden können.

22 *Das ist keine zwingende Wahl. Das Beispiel soll nur noch einmal eine alternative Ereignisbehandlung verwenden.*

Hinweis

Für die Nachbildung der ZURÜCK-Schaltfläche bzw. der History des Browsers eignet sich ebenso hervorragend die Speicherung der entscheidenden Sitzungssituationen in JavaScript-Variablen bzw. -Arrays. Allerdings gelten dann die gleichen oben genannten Einschränkungen.

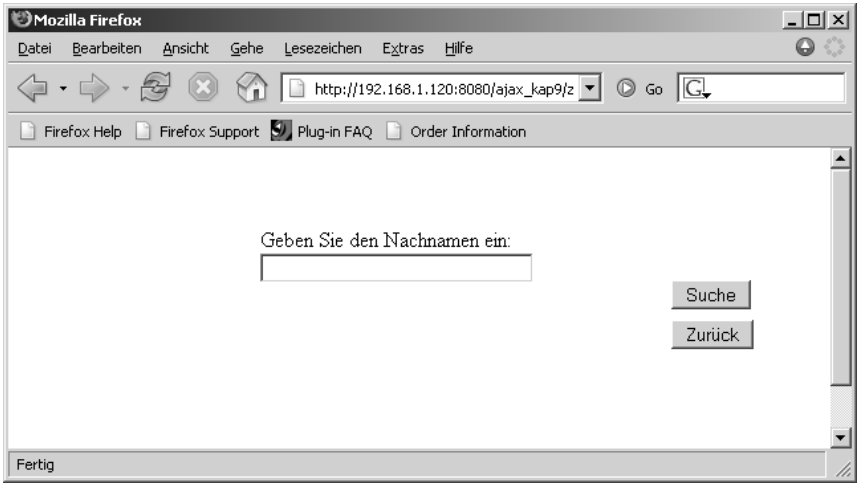


Abbildung 9.27: Das geänderte Formular mit den zwei Schaltflächen ohne Suche

Die neue Style Sheet-Datei *stil2.css* positioniert zu den bisherigen Regeln die beiden neuen Schaltflächen. Diesen wird ja jeweils eine eindeutige ID zugewiesen (`retur` und `suche`):

Listing 9.52: Die Erweiterungen der CSS-Datei

```
08 #retur{
09 position:absolute;top:130px;left:500px;
10 }
11 #suche{
12 position:absolute;top:100px;left:500px;
13 }
```

Wenn ein Besucher im Eingabefeld Zeichen eingibt, wird das keine unmittelbare Anzeige einer Zusatzinformation bewirken, wie es in den letzten beiden Beispielen der Fall war.

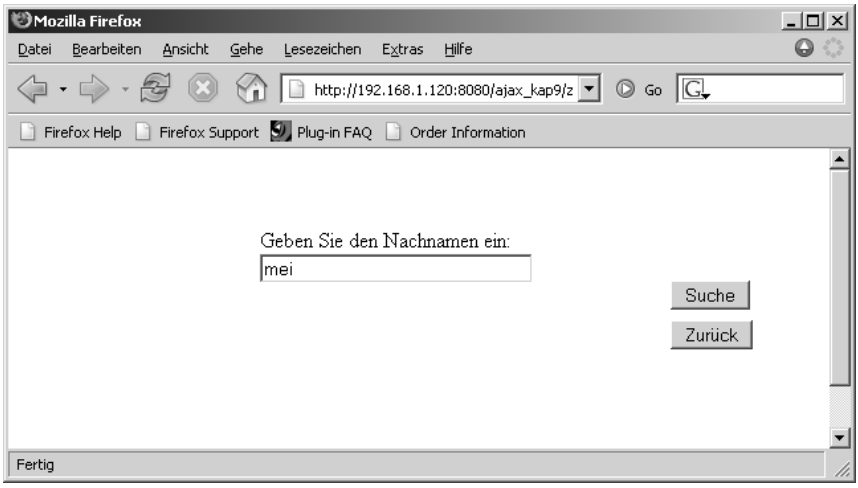


Abbildung 9.28: Obwohl in dem Eingabefeld Zeichen eingegeben wurden, wird keine Zusatzinformation angezeigt.

Wenn ein Anwender aber auf die SUCHE-Schaltfläche klickt, erhält er wieder die Hilfe in Form einer Auswahlliste, wie es in der Variante 1 des Beispiels bereits auf jede Tastatureingabe erfolgte.

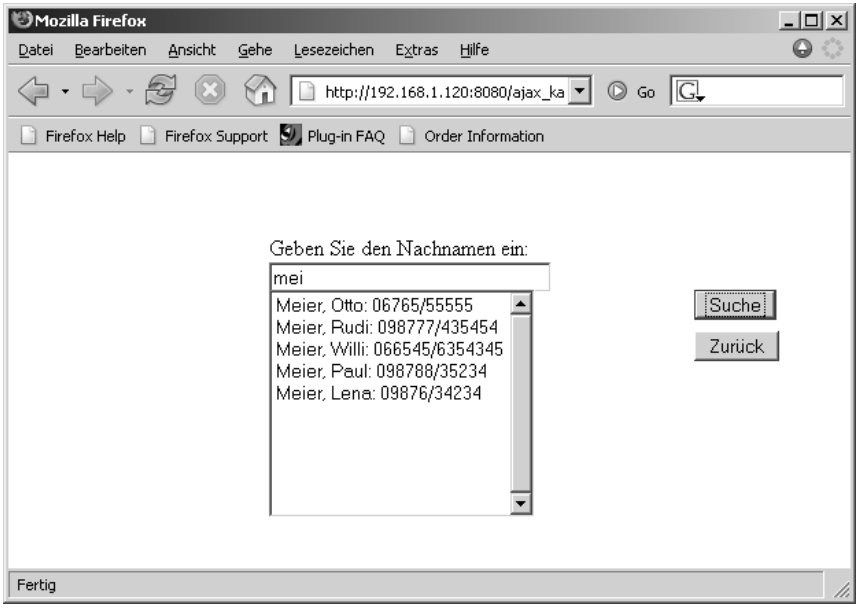


Abbildung 9.29: Nach dem Klick auf die Suche-Schaltfläche wird das Ergebnis der AJAX-Anfrage in dem Listenfeld angezeigt.

So weit hat sich das Beispiel nur unwesentlich verändert. Aber es gibt ja noch die ZURÜCK-Schaltfläche. Damit soll ja ein Anwender den Zustand vor der letzten Datenanforderung wiederherstellen können. Und dazu muss noch einiger Aufwand betrieben werden. Eingaben müssen gespeichert und wieder ausgelesen werden.

In der neuen JavaScript-Datei `zustand.js` hat sich zum Beispiel eine Menge getan. Die Erzeugung des `XMLHttpRequest`-Objekts erfolgt natürlich wie gehabt. Auch in der Funktion `sndReq()` hat sich nicht viel geändert. Es wird nur ein **Java-Servlet** mit Namen `Zustand` aufgerufen. Die Funktion `handleResponse()` bleibt im Vergleich zu der Variante in unserem ersten Beispiel vollkommen unverändert. Aber dann passiert einiges. Schauen wir uns das Listing an, so weit Änderungen bzw. Erweiterungen durchgeführt werden:

Listing 9.53: Die geänderten Parts in der externen JavaScript-Datei zustand.js

```
22 function sndReq() {
23   if(document.f1.name.value !=""){
24     resObjekt.open('get', 'Zustand?name=' +
escape(document.f1.name.value.toLowerCase()) ,true);
...
31 }
...
38 function sndReq2() {
39     resObjekt.open('get', 'Altzustand?zufall=' + Math.random() ,true);
40     resObjekt.onreadystatechange = handleResponse2;
41     resObjekt.send(null);
42 }
43 function handleResponse2() {
44   if(resObjekt.readyState == 4){
45     document.getElementById("antwort").style.visibility = "hidden";
46     document.f1.name.value = resObjekt.responseText;
47   }
48 }...
```

Es gibt in der externen JavaScript-Datei zwei neue Funktionen `sndReq2()` und `handleResponse2()`.

Die Funktion `sndReq2()` wird aufgerufen, wenn der Besucher auf die ZURÜCK-Schaltfläche in der Webseite klickt. Die Funktion sendet eine AJAX-Anfrage an ein Java-Servlet mit Namen `Altzustand`.

Tipp

Das Servlet könnte man aufrufen, ohne dem Servlet Parameter zu übergeben. Wir brauchen hier keine spezifischen Informationen. Wie Ihnen in Zeile 39 jedoch auffallen sollte, übergeben wir dem Servlet einen Übergabewert. Dieser ist jedoch ein Zufallswert (`Math.random()`), der im Servlet überhaupt nicht ausgewertet wird. Wozu wird dann aber ein solcher Wert übergeben? Es handelt sich um einen Trick, mit dem das **Cachen** (Zwischenspeichern) des Servlets im Browser verhindert werden kann. Wenn wir dies nicht tun würden, würde ein mehrfaches Aufrufen des Servlet dazu führen, dass sich der Browser beim wiederholten Aufruf aus dem Cache bedient. Das hier aufgerufene Servlet soll in der Datenbank die aktuellen Datensätze zweimal abfragen und dazwischen einen Datensatz löschen. Dies ist nicht gewährleistet, wenn der Browser auf den Cache zugreift.

Hinweis

Im Allgemeinen ist das Cachen von Informationen ein sehr nützlicher Vorgang. Sofern sich keine relevanten Dinge ändern, brauchen dann bereits vorhandene Informationen nicht neu ermittelt und/oder übertragen zu werden. Aber das Cachen kann auch dazu führen, dass man nicht mit den neuesten Informationen arbeitet. Interessanterweise arbeitet so ein kleiner Trick wie hier beschrieben sehr zuverlässig.

Die andere Erweiterung ist, dass die neue Funktion `handleResponse2()` an die Antwort gebunden wird. Die Funktion `handleResponse2()` setzt den Wert in dem Eingabefeld auf den Rückgabewert der AJAX-Anfrage und blendet den Bereich für das Listenfeld aus.

Kommen wir zu den Servlets selbst und betrachten zuerst den Quellcode von `Zustand.class`:

Listing 9.54: Die Datei Zustand.java

```
01 import java.io.*;
02 import java.sql.*;
03 import javax.servlet.*;
04 import javax.servlet.http.HttpServlet;
05 public class Zustand extends HttpServlet {
06 public void service(ServletRequest request, ServletResponse
07   response) throws ServletException, IOException {
08   response.setContentType("text/html");
09   PrintWriter out = response.getWriter();
```

```
10   String text = "<select name=\"vorschlag\" size=\"10\" onClick= ↩
     \"uebernehme()\">";
11   String connStr = "jdbc:mysql://localhost:3306/" +
12     "ajax?user=root&pwd=";
13   String name = request.getParameter("name");
14   String vergleich = "";
15   String sql1 = "SELECT * FROM `adressen` ORDER BY `id` ASC";
16   String sql2 = "INSERT INTO `verlauf` (`id`, `eingabe`) VALUES ↩
     ('', '" + name + "')";
17   Connection conn;
18   ResultSet rs;
19   try {
20     Class.forName("com.mysql.jdbc.Driver");
21     conn = DriverManager.getConnection(connStr);
22     Statement stmt1 = conn.createStatement();
23     rs = stmt1.executeQuery(sql1);
24     while (rs.next()) {
25       vergleich = rs.getString(2).substring(0, Math.min(
26         name.length(), rs.getString(2).length() )).toLowerCase();
27       if(name.equals(vergleich))
28         text += "<option>" + rs.getString(2) + ", " +
29              rs.getString(3) + ": " + rs.getString(4)
30              + "/" + rs.getString(5) + "</option>";
31     }
32     rs.close();
33     stmt1.executeUpdate(sql2);
34     conn.close();
35   } catch (ClassNotFoundException e) {
36     System.out.println("Klasse nicht da");
37   } catch (SQLException e) {
38     System.out.println("SQL-Fehler");
39   } catch (Exception e) {
40     System.out.println("Sonstiges");
41   }
42   text += "</select>";
43   out.print(text);
44   }
45 }
```

Der Quellcode zeigt Ihnen ein typisches Servlet, das auf der Klasse `HttpServlet` aufbaut. Die gesamte Funktionalität ist in die `service()`-Methode verlagert. In den ersten vier Zeilen finden Sie `import`-Anweisungen für Pakete zur Ein- und Ausgabe (`java.io`), Datenbankfunktionalität (`java.sql`), grundlegende Servlet-Funktionalitäten (`javax.servlet`) und die Klasse `HttpServlet` selbst, die sich unter `javax.servlet.http` befindet. Sie kennen die Parts des Quellcodes, die sich auf die Tabelle `adressen` beziehen. Das ist die bekannte Abfrage aufgrund der eingegebenen Zeichen im Eingabefeld des Formulars, wie wir es in den Beispielen mit den JSP durchgespielt haben. Zwar wird das Servlet nicht mehr bei jeder losgelassenen Taste, sondern nur bei

einem Klick auf die SUCHE-Schaltfläche aufgerufen. Und zudem ist die gesamte Logik in die `service()`-Methode des Servlet verlagert. Aber die Funktionalität dieses Teils des Java-Codes bleibt davon selbstverständlich unberührt.

In Zeile 8 wird der Datentyp für die Antwort auf HTML festgelegt. In Zeile 9 besorgen wir uns einen `PrintWriter`, über den die Antwort zum Client geschickt wird. In Zeile 15 sollte Ihnen der Quellcode dann bekannt sein. In Zeile 16 sehen Sie, dass wir eine zweite SQL-Abfrage formulieren, die auf einer neuen Tabelle operiert. Es handelt sich um eine Anweisung zum Einfügen[23] von Daten in die neue Tabelle (`String sql2 = "INSERT INTO `verlauf` (`id`, `eingabe`) VALUES ('', '" + name + "')";`). Wir schauen uns die Tabelle `verlauf` gleich an. Dann wird die Syntax klarer. Die Abfrage wird in Zeile 33 mit `stmt1.executeUpdate(sql2);` ausgeführt.

Augenscheinlich benötigen wir für das Beispiel eine neue Tabelle mit Namen `verlauf` in der Datenbank `ajax`. Diese können Sie wieder mit phpMyAdmin anlegen.

Die Tabelle muss für das Beispiel zwei Felder besitzen. Das erste Feld heißt `id`, ist vom Typ `int`, ein Primärschlüssel und hat als Extras den Wert `auto_increment` eingestellt. Letzteres bedeutet, der Wert wird bei jedem Schreiben eines Datensatzes automatisch hochgezählt. Sie schreiben selbst keinen Wert in dieses Feld. Und wenn Sie einen Datensatz löschen, wird der Wert dieser `auto_increment`-ID beim Einfügen neuer Datensätze nicht mehr verwendet. Um diese Eigenschaft in phpMyAdmin zu setzen, müssen Sie beim Anlegen des Felds etwas nach rechts scrollen. Unter der Rubrik ***Extra*** finden Sie die Einstellung.

Abbildung 9.30: Das Feld id wird beim Anlegen eines neuen Datensatzes automatisch hochgezählt und als Primärschlüssel festgelegt.

Das zweite Feld in der neuen Tabelle hat den Namen `eingabe` und ist für unser einfaches Beispiel vom Typ `varchar` mit der Länge 120. In diesem Feld speichern wir die Eingaben, die ein Anwender in dem Formulareingabefeld vornimmt, wenn auf die SUCHE-Schaltfläche geklickt wird[24]. Der `INSERT`-Befehl macht genau das. In das Feld `id` wird wie gesagt nichts geschrieben und in das Feld `eingabe` der an das Skript übergebene Parameterwert, der die Benutzereingabe enthält.

23 *Es klingt für Laien etwas widersprüchlich, wenn man von einer SQL-**Abfrage** spricht und damit Daten einfügt. Aber diese Sprachwahl ist wirklich üblich. Mit einer SQL-Abfrage können Sie Daten abfragen, einfügen oder löschen und sogar Datenbanken oder Tabellen anlegen oder löschen.*

24 *Die Eingabe des Besuchers wird also parallel in den Tabellen* `adressen` *und* `verlauf` *verwendet.*

Abbildung 9.31: Die Struktur der Tabelle verlauf

Mit dieser Tabelle protokollieren wir also alle relevanten Zustände der Sitzung. Damit ist im Grunde das reine Verfolgen der Sitzung auch erledigt. Sie können durch Auswertung der Tabelle `verlauf` jede Suchanfrage des Besuchers reproduzieren. Aber wir wollen die Protokollierung ja noch weiter nutzen.

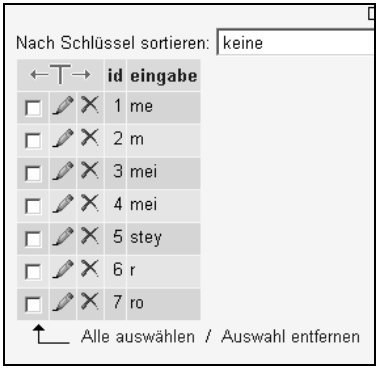


Abbildung 9.32: Jede Suchanfrage wird protokolliert.

Der gesamte Rest des Quellcodes ist Ihnen wie gesagt von unseren JSP-Beispielen her bekannt. Wir haben zwar in den `catch`-Abschnitten nun individuelle Fehlermeldungen eingebaut, aber das sollte Sie nicht irritieren. Eine Stelle sollten Sie jedoch noch beachten: die Zeile 43. Wo wir bisher mit einem JSP-Ausdrucks-Tag gearbeitet haben, finden Sie hier die Anwendung der `print()`-Methode bei unserem Objekt vom Typ `PrintWriter`.

Für unser drittes Beispiel fehlt noch das Servlet `Altzustand`. Wie der Name schon suggeriert, soll darüber der Zustand vor der letzten Sitzungsänderung (das ist eine Abfrage der Datenbank) wiederhergestellt werden. Dazu könnte man auf verschiedene Art und Weise vorgehen. Wir könnten zum Beispiel in der Tabelle `verlauf` den **vorletzten** Datensatz suchen, daraus die Benutzereingabe nehmen, diese in das Eingabefeld des Formulars befördern und den letzten Datensatz löschen. Das wäre eine absolute sinnvolle Vorgehensweise. Wir gehen hier jedoch einen alternativen Weg, der genauso sinnvoll ist. Zuerst ermitteln wir die ID des letzten Datensatzes in der Tabelle `verlauf`. Mithilfe dieser ID setzen wir eine SQL-Anweisung zum Löschen dieses Datensatzes zusammen. Danach löschen wir den Datensatz wirklich. Nun durch-

laufen wir erneut die Tabelle `verlauf` und suchen den letzten Wert der gespeicherten Benutzereingaben. In dem nun letzten Datensatz steht genau das, was der Anwender als vorletzte Eingabe getätigt hat.

Tipp

Löschen Sie den letzten Datensatz bei unserem Beispiel nicht, steht Ihnen eine ähnliche Funktionalität zur Verfügung, wie es ein Anwender bei einer konventionellen Webapplikation mit der VORWÄRTS-Schaltfläche des Browsers bzw. dem Vorwärtsgehen in der History kennt. Wenn Sie einmal die History zurückgegangen sind, können Sie den gerade verlassenen Zustand wiederherstellen. Aber das würde unser Beispiel komplexer machen, als es zum Zeigen der grundsätzlichen Funktionalität notwendig ist. Und zudem ist so eine Reproduktion auch nicht immer für das Verhalten der Webapplikation von Vorteil und zu guter Letzt soll hier auch das Löschen von Datensätzen demonstriert werden.

Hier ist das Listing der Java-Datei `Altzustand.java`:

Listing 9.55: Die Java-Datei zum Wiederherstellen des letzten Zustands

```
01 import java.io.*;
02 import java.sql.*;
03 import javax.servlet.*;
04 import javax.servlet.http.HttpServlet;
05 public class Altzustand extends HttpServlet {
06   public void service(ServletRequest request, ServletResponse
07     response) throws ServletException, IOException {
08   response.setContentType("text/html");
09   PrintWriter out = response.getWriter();
10   String text = "";
11   String connStr = "jdbc:mysql://localhost:3306/" +
12     "ajax?user=root&pwd=";
13   String loesch = "";
14   String sql1 = "SELECT * FROM `verlauf` ORDER BY `id` ASC";
15   Connection conn = null;
16   Statement stmt = null;
17   ResultSet rs = null;
18   try {
19     Class.forName("com.mysql.jdbc.Driver");
20     conn = DriverManager.getConnection(connStr);
21     stmt = conn.createStatement();
22     rs = stmt.executeQuery(sql1);
23     while (rs.next()) {
24       loesch = rs.getString(1);
25     }
```

```
26      String sql2 = "DELETE FROM `verlauf` WHERE `id`='"
27        + loesch + "' LIMIT 1";
28      stmt.executeUpdate(sql2);
29      rs = stmt.executeQuery(sql1);
30      while (rs.next()) {
31        text = rs.getString(2);
32      }
33    } catch (ClassNotFoundException e) {
34      System.out.println("Klasse nicht da");
35    } catch (SQLException e) {
36      System.out.println("SQL");
37    } catch (Exception e) {
38      System.out.println("Sonstiges");
39    } finally {
40      try {
41        rs.close();
42      } catch (SQLException e) {
43        e.printStackTrace();
44      }
45      try {
46        conn.close();
47      } catch (SQLException e) {
48        e.printStackTrace();
49      }
50    }
51    out.print(text);
52    }
53 }
```

Bis Zeile 12 finden Sie nichts Neues. In Zeile 13 wird eine Stringvariable `loesch` eingeführt. Zeile 14 setzt das SQL-Statement zusammen. Wir benötigen für dieses Servlet nur ein SQL-Statement, denn wir operieren hier natürlich nur auf der Tabelle `verlauf`.

Die Zeilen 15, 16 und 17 scheinen neu zu sein. Aber das täuscht. Denn in diesem Beispiel demonstrieren wir nur, wie man das Schließen der Datenbankverbindung und des `ResultSet` in den `finally`-Abschnitt verlagern kann (Zeile 39 bis 50). Dies ist funktional für das Beispiel nicht notwendig, soll aber eine alternative Praxis zeigen, wie man sie oft beim Umgang mit Datenbanken findet. Wenn wir solch eine Verlagerung in den `finally`-Abschnitt durchführen wollen, müssen die betroffenen Variablen auf jeden Fall initialisiert sein. Das ist nicht gewährleistet, wenn wir die Variablen im `try`-Abschnitt einführen und dort initialisieren.

Von Zeile 23 bis 29 läuft die erste Schleife. Deren Aufgabe ist die Ermittlung der ID des letzten Datensatzes. Beachten Sie Zeile 24. In der Schleife wird der Variablen `loesch` jeweils bei jedem Durchlauf der aktuelle Wert im Feld ID zugewiesen (`while (rs.next()) { loesch = rs.getString(1); }`). Mit diesem kleinen Trick ist gewährleistet, dass bei Beendigung der Schleife in dieser Variable die ID des letzten Datensatzes steht. Auf diese Weise kann vollständig auf eine Entscheidungsstruktur oder eine sonst komplexe Logik verzichtet werden.

In Zeile 26 und 27 wird mithilfe der ermittelten ID ein SQL-Befehl zum Löschen des letzten Datensatzes zusammengebaut. In Zeile 28 wird dieser Löschbefehl ausgeführt. Beachten Sie die Verwendung der Methode `executeUpdate()`. Wir könnten auch mit `executeQuery()` arbeiten, aber für Einfüge-, Update- oder Löschoperationen bietet sich die Verwendung dieser Methode schon rein vom Namen her an.

Hinweis

Beide Methoden liefern unterschiedliche Rückgabewerte. Bei `executeUpdate()` erhalten Sie die Anzahl der geänderten Datensätze und bei `executeQuery()` einen `ResultSet`. Sofern Sie das explizit verwenden wollen (was wir in dem Beispiel nicht machen), ergibt sich die Wahl der passenden Methode daraus.

In Zeile 29 fügen wir erneut die SQL-Anweisung aus, die den gesamten Inhalt der Tabelle `verlauf` ermittelt. Auf dem damit gewonnenen `ResultSet` operieren wir erneut. Die Schleife von Zeile 30 bis 32 interiert über diesen `ResultSet`. In Zeile 31 verwenden wir den gleichen Trick wie eben. Der Variablen `text` wird bei jedem Durchlauf der aktuelle Wert in der Spalte `eingabe` zugewiesen. Dies stellt sicher, dass nach dem letzten Durchlauf der Schleife der Wert des letzten Datensatzes in dieser Variablen steht. Da wir vorher den letzten Datensatz gelöscht haben, ist dies die vorletzte Benutzereingabe. Dieser Wert wird dann in Zeile 51 dem Client geschickt, damit dieser in dem Eingabefeld repoduziert werden kann. Damit wird also jedes Auslösen der ZURÜCK-Schaltfläche in der »AJAX-History« einen Zustand zurückgehen und so fort.

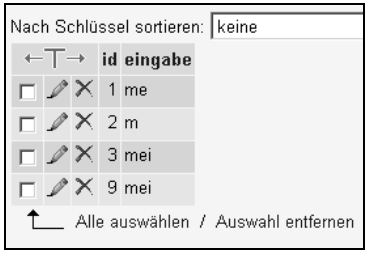


Abbildung 9.33: An den Lücken in der ID-Folge erkennen Sie, dass zwischenzeitlich Datensätze gelöscht wurden.

9.5.5 Beispiel – Login mit AJAX verifizieren

In Zusammenhang mit der Verwendung von Servlets bei AJAX-Applikationen soll noch ein zweites Beispiel gezeigt werden. Diese kleine Applikation zeigt, wie man AJAX zur Verifizierung einer Benutzeranmeldung verwenden kann. Auch hier besteht wieder die Besonderheit, dass die Verifizierung erfolgt, ohne die Webseite neu zu laden. Dies ist sicher eine der sinnvollsten Anwendungen von AJAX, denn

hier kann man die überflüssige mehrfache Versendung vieler Daten verhindern (alle Daten auf der Anmeldeseite). Ein Anwender gibt in dem Beispiel in einer Webseite seine Anmeldedaten ein und erhält in Abhängigkeit von der Eingabe ein Willkommen[25] oder aber eine Meldung, dass die Anmeldedaten falsch sind.

Hier ist zunächst die Webseite, die wenig Überraschungen bieten sollte:

Listing 9.56: Eine Webseite mit einem Login

```
01 <html>
02 <script language="JavaScript" src="logintest.js"></script>
03 <body>
04 <h4>Login</h4>
05 <form name="f1">
06  <table style="text-align: left; width: 300;" border="0"
07  cellpadding="2" cellspacing="2">
08  <tbody>
09  <tr>
10  <td>User:</td>
11  <td><input name="username" type="text" size="30" /></td>
12  </tr>
13  <tr>
14  <td>Passwort:</td>
15  <td><input name="password" type="password" size="30" />
16 </td>
17  </tr>
18  <tr>
19  <td> </td>
20  <td><input type="button" value="Login" onClick="sndReq()"/>
21 </td>
22  </tr>
23  </tbody>
24  </table>
25 </form>
26 <br />
27 <div id="antwort"></div>
28 </body>
29 </html>
```

Sie finden in der Webseite ein gewöhnliches Webformular, in dem der Anwender in einem normalen Eingabefeld seine User-ID und in einem Passwortfeld sein Passwort eingeben kann.

Die einzig relevante Besonderheit finden Sie in Zeile 20 (`<input type="button" value="Login" onClick="sndReq()"/>`). Statt eines Submit-Buttons, wie man ihn normalerweise in einem Webformular vorfindet, das Daten verschicken soll, sehen Sie hier

25 *Oder in der Praxis auch eine vollständig dynamisch aufgebaute Hauptseite.*

eine gewöhnliche Schaltfläche vom Typ `Button`. Die Verifizierung wird mit einem `onClick`-Eventhandler ausgeführt. Dies ist auch zwingend notwendig, denn wir wollen die Formulardaten ja nicht auf dem konventionellen Weg zum Server schicken. Wenn wir das machen würden, würden die Formulardaten verschickt und eine neue Webseite vom Server als Antwort geschickt. Damit wären die Vorteile von AJAX ausgehebelt.

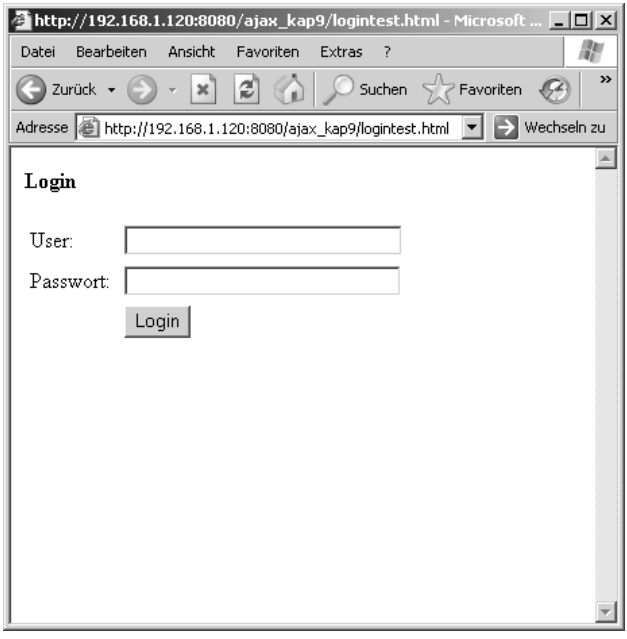


Abbildung 9.34: Das Anmeldeformular

In der JavaScript-Datei müssen wir uns nur die Funktion `sndReq()` betrachten:

Listing 9.57: Die interessanten Teile der externen JavaScript-Datei

```
22 function sndReq() {
23   if(document.f1.username.value !=""){
24     resObjekt.open('get', 'LoginTest?username=' +
escape(document.f1.username.value) + '&password='
25       + escape(document.f1.password.value) ,true);
26     resObjekt.onreadystatechange = handleResponse;
27     resObjekt.send(null);
28   }
29   else {
30     document.getElementById("antwort").style.visibility = "hidden";
31   }
32 }
```

Zur Absicherung, dass auf jeden Fall ein User-Name eingegeben wird, wird in Zeile 23 überprüft, dass dieses Feld im Webformular nicht leer ist (`if(document.f1.username.value !="")`{}). In Zeile 24 und 25 werden die Benutzereingaben zu einem Pseudo-URL zusammengefügt. Hier ist kein großer Unterschied zu bisherigen Beispielen zu sehen, nur werden dieses Mal die Eingaben von zwei Feldern verschickt.

Hinweis

Ihnen sollte bekannt sein, dass die Verwendung der GET-Methode beim Übermitteln von Passwortfeldern wenig intelligent ist. Bei einem Passwortfeld werden die Eingaben des Benutzers durch * verdeckt. Wenn Sie dann die GET-Methode verwenden, werden die Eingaben als Klartext in der Adresszeile des Browsers sichtbar. Preisfrage: Warum ist es für unser Beispiel dennoch nicht so dumm, wenn wir hier die GET-Methode verwenden? Antwort: Bei einer AJAX-Anfrage werden die Daten nicht in der Adresszeile des Browsers angezeigt. Dies sollte aber nicht darüber hinwegtäuschen, dass die Daten immer noch per HTTP verschickt und damit als Klartext übertragen werden. Ein Hacker kann diese unverschlüsselten Daten mit einem Sniffer jederzeit lesen.

Kommen wir nun zum Java-Quellcode unseres Servlet:

Listing 9.58: Der Quellcode des Servlet

```
01 import java.io.*;
02 import java.sql.*;
03 import javax.servlet.*;
04 import javax.servlet.http.*;
05 public class LoginTest extends HttpServlet {
06   protected void doGet(HttpServletRequest request,
07     HttpServletResponse response) throws
08       ServletException, IOException {
09     response.setContentType("text/html");
10     PrintWriter out = response.getWriter();
11     String connStr = "jdbc:mysql://localhost:3306/" + "ajax?user=root&pwd=";
12     String username = request.getParameter("username");
13     String password = request.getParameter("password");
14     String sql1 = "SELECT * FROM `users`";
15 // Boolesche Variable, die bei einem Treffer
16 // auf true gesetzt wird
17     boolean treffer = false;
18     Connection conn;
19     ResultSet rs;
20     try {
21       Class.forName("com.mysql.jdbc.Driver");
22       conn = DriverManager.getConnection(connStr);
```

```
23        Statement stmt1 = conn.createStatement();
24        rs = stmt1.executeQuery(sql1);
25        while (rs.next()) {
26          if ((rs.getString(1).equals(username)) // Userid
27                && (rs.getString(2).equals(password)) // Passwort
28              ) { // Übereinstimmung mit einer vorhandenen
29                  // Passwort-Userid-Kombination gefunden
30                  treffer = true;
31                  break;
32                }
33        }
34        rs.close();
35        conn.close();
36      } catch (ClassNotFoundException e) {
37        System.out.println("Klasse nicht da");
38      } catch (SQLException e) {
39        System.out.println("SQL");
40      } catch (Exception e) {
41        System.out.println("Sonstiges");
42      }
43      if (treffer) {
44        out.print("<h3>Willkommen im AJAX-NET.de-Portal</h3>");
45      } else {
46        out.print(
47          "<h3>Ihre Anmeldedaten sind leider nicht korrekt. Bitte melden Sie sich ↩
          im Portal an.</h3>");
48      }
49    }
50 }
```

Die meisten Teile des Quellcodes sollten Sie mittlerweile wieder erkennen. Dennoch dürfte Ihnen sofort auffallen, dass wir in diesem Beispiel mit der Methode `doGet()` arbeiten. Dies bedeutet, dass eine Anfrage mit der Post-Methode hier nicht behandelt werden würde. Rein funktional ist diese Tatsache im Grunde egal, aber in diesem Servlet soll einfach noch eine andere Methode zur Entgegennahme von Client-Anfragen in der Praxis gezeigt werden.

In Zeile 12 und 13 werden die übergebenen Werte aus dem Webformular entgegengenommen. In Zeile 17 wird eine Boolesche Variable `treffer` eingeführt und mit `false` initialisiert. Diese wird im Laufe des Servlets dann auf `true` gesetzt, wenn eine Übereinstimmung mit einer vorhandenen Kombination aus User-ID und Passwort gefunden wird. Die Bedingung in der `if`-Anweisung in Zeile 26 und 27 überprüft genau dieses (`if ((rs.getString(1).equals(username)) && (rs.getString(2).equals(password))) {`). Das `break` in Zeile 31 verhindert, dass die Schleife nach einem Treffer noch weiter durchlaufen wird. In Zeile 43 wird in Abhängigkeit von der Treffervariablen eine Antwort an den Client generiert.

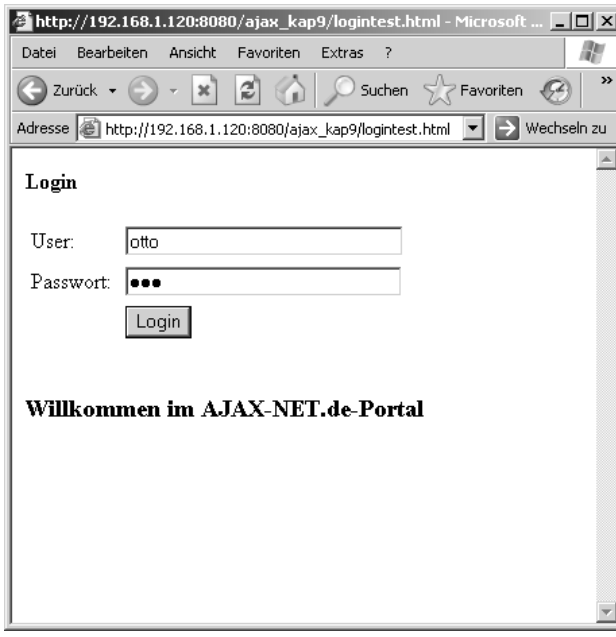


Abbildung 9.35: Korrekt angemeldet

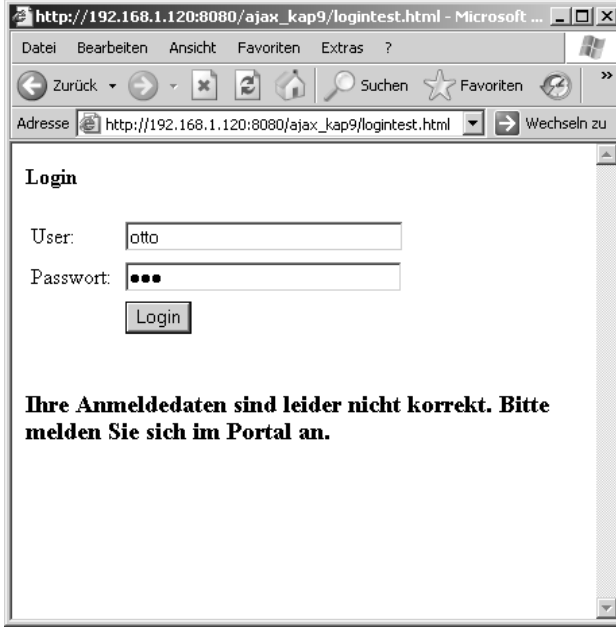


Abbildung 9.36: Die Zugangsdaten waren falsch.

9.6 Zusammenfassung

Sie haben in diesem Kapitel die Grundlagen von serverseitiger Programmierung und den Umgang mit Webdatenbanken gesehen. Sie werden bei AJAX auf Seiten des Servers auf keine bestimmten Technologien festgelegt. Aber für nahezu jede AJAX-Applikation müssen Sie auf Server-Seite programmieren und für die meisten etwas aufwändigeren AJAX-Applikationen auch Datenbanken nutzen. Auf Java und MySQL sind wir dabei als praktische Vertreter dieser Welt näher eingegangen. Sie können mit Java insbesondere aus Datenbanken Werte auslesen und im XML-Format an den Client senden. Dies eröffnet dort zahlreiche Möglichkeiten zum dynamischen Erzeugen von Elementen in einer Webseite. Aber auch das Verfolgen von Sitzungen und allgemein das Speichern von Daten lässt sich mit einer AJAX-Applikation genauso oder noch besser erledigen als mit einer konventionellen Webapplikation.

10 AJAX-Alternativen

Wie mehrfach erwähnt, beschreibt AJAX keine vollkommen neue Idee, die wie ein Blitz aus heiterem Himmel ins Web eingeschlagen hat. Insbesondere führten die Einschränkungen des WWW respektive HTTP über die Jahre hinweg zu verschiedenen Ansätzen, Daten asynchron vom expliziten Nutzerverhalten zu übertragen sowie den Zustand von Sitzungen zu verfolgen und kleinere Datenmengen auf dem Client zu speichern. Dies umfasst Techniken wie eine Vorratshaltung von Daten oder Elementen für eine Webseite (in der Regel mit JavaScript und DHTML implementiert), die Zustandspeicherung einer Sitzung in Cookies, das asynchrone Nachladen von Daten in Frames bzw. IFrames oder auch proprietäre Ansätze wie XML-Dateninseln (Microsoft), die –- wenn man ehrlich ist – bereits die wichtigsten Parts von AJAX vorweggenommen haben. Und bereits bevor AJAX als Modebegriff das Web stürmte, gab es so genannte Rich Internet Applications (RIA), die das Verhalten von Webapplikationen zu beschleunigen und zu optimieren versuchten – zum Beispiel Technologien wie **Flex** der Firma Macromedia (*http://www.macromedia.com/software/flex/*) oder **OpenLaszlo** (*http://www.openlaszlo.org/*). In diesem Kapitel schauen wir uns einige dieser Alternativen bzw. Ideen zu AJAX an. Diese haben zum Teil den Vorteil, dass sie die Implementation einer `XMLHttpRequest`-Schnittstelle in einem Browser nicht voraussetzen und auch in älteren Browsern funktionieren. Aber die Nachteile sind eine teilweise exorbitant reduzierte Leistungsfähigkeit gegenüber AJAX bzw. eine Spezialisierung auf eine Aufgabe wie die Zustandsverwaltung, teils eine massive Beschränkung auf einen bestimmten Browser oder proprietäre Standards bzw. Plugins sowie ein erheblich größerer Aufwand als bei AJAX.

10.1 Verfolgen einer Sitzung mit Cookies

Wie wir im Laufe des Buchs gesehen haben, ist die Verfolgung einer Sitzung mit AJAX kein echtes Problem mehr, vor allem nicht in Verbindung mit einer Datenbank. Da eine Webseite bei einer AJAX-Applikation so gemacht werden kann, dass sie nur einmal geladen werden muss und dann alle Änderungen über die `XMLHttpRequest`-Schnittstelle erfolgen, kann eine Sitzung sogar vollständig über eine einzelne Webseite realisiert werden, auch wenn es sich um komplexe Dinge wie den Zugang zu einem geschlossenen Bereich oder einen Onlineshop handelt. Und dann müssen Sie im Extremfall auch bei HTTP eine Sitzung überhaupt nicht mehr verfolgen. Aber was ist, wenn kein AJAX verfügbar ist? Auch vor AJAX musste eine solche Verfolgung

einer Sitzung gewährleistet werden. Schon recht lange gibt es Versuche, den Zustand von Websitzungen trotz des zustandslosen HTTP zu verfolgen. Ein Ansatz sieht vor, auf dem Rechner des Clients Informationen über eine Sitzung abzulegen. Dies erfolgt in Form kleiner Textdateien, so genannte **Cookies** (Kekse), die vom Browser in einem Schreibvorgang auf dem Client-Rechner angelegt werden. Was konkret in solchen Textdateien gespeichert wird, ist vollkommen frei. Aber es hat sich eine gewisse Standardisierung durchgesetzt, denn wenn etwas in einer Textdatei auf einem Client-Rechner abgelegt wurde, muss man es auch wiederfinden, auslesen und interpretieren können.

10.2 Die Erstellung von Cookies

Die Erstellung und die Auswertung von Cookies kann sowohl vom Client als auch dem Server[1] initiiert werden, wobei das konkrete Schreiben der Daten natürlich der Browser erledigen muss. Auf der einen Seite muss man beachten, dass die verschiedenen Browser zwar einen etwas unterschiedlichen Weg bezüglich der konkret angelegten Dateien gehen[2]. Aber wie auch immer Cookies tatsächlich durch den Webbrowser auf dem Rechner eines Besuchers gespeichert werden – das reale Handling aus einem Skript ist davon nicht betroffen. Das Verfahren ist durch die Programmierumgebung so weit abstrahiert, dass man sich keinerlei Gedanken darum machen muss. Das gilt sowohl für client- als auch für serverseitige Programmierung.

10.2.1 Mit JavaScript Cookies anlegen und auslesen

Wir schauen uns hier in der Praxis nur die Variante näher an, wie Sie mit JavaScript auf dem Client mit Cookies umgehen können. Grundsätzlich kann ein Cookie via JavaScript mit der Eigenschaft `document.cookie` gesetzt werden. Zur Erstellung und Analyse des eigentlichen Inhalts helfen Ihnen ergänzende Methoden des `String`-Objekts wie `substring()`, `charAt()`, `indexOf()` und `lastIndexOf()`, um die konkreten Werte in dem Cookie genauer zu handeln. Generell besteht ein Cookie mindestens aus der Angabe eines Namens als Bezeichner des Cookies und einer Wertzuweisung. Über den Bezeichner wird beim Auslesen ein Cookie wieder identifiziert. Der zugewiesene Wert ist die Information, welche man tatsächlich speichern möchte. Eine wichtige optionale Eigenschaft ist die `expires`-Eigenschaft, mit der ein Gültigkeitsdatum eines Cookies gesetzt werden kann. Das Format sieht beispielsweise so aus:

1 *HTTP erlaubt die Erweiterung um zusätzliche Header-Felder und damit können sitzungsspezifische Daten vom Server zum Client versendet werden.*

2 *Mozilla-Browser speichern etwa alle Einträge in einer einzigen Klartextdatei namens cookies.txt, während Opera Cookies in kodierter Form in der Datei cookie.dat ablegt. Dagegen legt der Internet Explorer gewöhnlich für jedes Cookie eine eigene Klartextdatei im Cookies-Verzeichnis des Betriebssystems an.*

Listing 10.1: Das Format eines Ablaufdatums in einem Cookie

```
Wdy, DD-Mon-YY HH:MM:SS GMT
```

`Wdy` ist eine Stringrepräsentation des Tags der Woche (englisch), `DD` zweistellig der Monatstag, `Mon` eine aus drei Buchstaben bestehende String-Repräsentation des Monats (englisch) und `YY` das zweistellige Jahr. `HH`, `MM`, und `SS` sind zweistellige Angaben für die Stunden, Minuten und Sekunden (optional). Aber auch andere Formate sind denkbar.

Weitere optionale Eigenschaften eines Cookies sind `domain`, worüber der URL der Domain als String spezifiziert werden kann, für welche ein Cookie gültig ist. Wenn die Angabe gesetzt ist, kann ein geschriebenes Cookie nur von einem Server aus dieser Domain wieder ausgelesen werden. Die Eigenschaft `path` dient zur Angabe von Verzeichnissen innerhalb dieser Domain (fehlt die Angabe, wird der Pfad des aktuellen Dokuments angenommen) und `secure` als Boolean-Wert zur Festlegung, dass ein Cookie nur über einen sicheren (d.h. verschlüsselten) Pfad übertragen werden darf.

Die an die `cookie`-Eigenschaft von `document` zu übergebende Zeichenkette muss die gewünschten Angaben als – durch Parameter getrennte – Werte enthalten. So weit ist es also ganz einfach, ein Cookie zu setzen. Leider ist es nicht ganz so leicht, wenn Sie komplexere Informationen aus einem Cookie wieder auslesen möchten. JavaScript bietet da erstmal keine große automatische Unterstützung für die verschiedenen Aufgaben, die dabei erledigt werden müssen. Cookie-Werte dürfen zum Beispiel keine Leerzeichen und keine Sonderzeichen enthalten. Deshalb müssen Sie beim manuellen Erstellen und Auslesen von Cookies solche Zeichen mit JavaScript-Funktionen wie `escape()`[3] und `unescape()` kodieren und wieder dekodieren. Des Weiteren müssen verschiedene Aktionen wie das Bestimmen der Länge eines Werts oder das Trennen an bestimmten Zeichen im String durchgeführt werden. Glücklicherweise müssen Sie das aber nicht wirklich von Hand machen, denn es gibt für das Cookie-Management mit JavaScript bereits vorgefertigte Funktionsbibliotheken. Einen populären Satz solcher freien Funktionen hat zum Beispiel ein Programmierer namens Bill Dortch entwickelt und im Internet frei verfügbar bereitgestellt. Mit dessen bekannten Funktionen `SetCookie(name, value, expires, path, domain, secure)` und `GetCookie(name)` zum Setzen und Lesen von beliebigen Angaben wird das Cookie-Management unter JavaScript wirklich zum Kinderspiel. Das nachfolgende Beispiel zeigt die beiden Funktionen. Die zusätzliche Funktion `DeleteCookie(name, path, domain)` erlaubt das Löschen eines Cookie, indem das Verfalldatum auf den 1. Januar 1970 gesetzt wird (es gibt keine Möglichkeit zum direkten Löschen).

Das Beispiel zeigt ein kleines Formular, das bei einem Klick auf die eine Schaltfläche ein Cookie schreibt, in das der Wert des einen Eingabefelds geschrieben wird. Beim erneuten Laden des Formulars wird dieses wieder ausgelesen und das Formularfeld bereits vorbelegt:

3 *Die Funktion haben wir in Zusammenhang mit der Datenübermittlung der* `open()`*-Methode schon gesehen.*

Listing 10.2: Das Setzen und Auslesen von Cookies mit JavaScript

```
<html>
<script language="JavaScript">
function GetCookie (name) {
  var arg = name + "=";
  var alen = arg.length;
  var clen = document.cookie.length;
  var i = 0;
  while (i < clen) {
   var j = i + alen;
   if (document.cookie.substring(i, j) == arg)
    return getCookieVal (j);
   i = document.cookie.indexOf(" ", i) + 1;
   if (i == 0) break;
  }
  return null;
}
function SetCookie (name,value,expires,path,domain,secure) {
  document.cookie = name + "=" + escape (value) +
   ((expires) ? "; expires=" + expires.toGMTString() : "") +
   ((path) ? "; path=" + path : "") +
   ((domain) ? "; domain=" + domain : "") +
   ((secure) ? "; secure" : "");
}
function getCookieVal (offset) {
  var endstr = document.cookie.indexOf (";", offset);
  if (endstr == -1)
   endstr = document.cookie.length;
  return unescape(document.cookie.substring(offset, endstr));
}
function schreibeCookie() {
  wert = document.mF.id.value;
  haltbarDatum = new Date(2006,11,31,1,0,0);
  document.cookie = SetCookie("typ",wert,haltbarDatum);
}
function leseCookie() {
  wert = GetCookie("typ");
  if(wert != null) document.mF.id.value=wert;
}
</script>
<body onLoad="leseCookie()">
<form name="mF" method="GET"  onSubmit="schreibeCookie()">
Eingabe: <input name="id"><br>
<input type="Submit" value="Ok">
</form>
</body>
</html>
```

Beachten Sie, dass die Zeilen 3 bis 29 die besagten Funktionen enthalten, die frei im Internet verfügbar sind. Erst ab Zeile 30 finden Sie Code, der selbst erstellt wurde. In der Funktion `schreibeCookie()` (Zeilen 30 bis 34) wird der Wert des Eingabefelds und ein Haltbarkeitsdatum in der Zukunft in ein Cookie namens `typ` geschrieben. In den Zeilen 35 bis 38 finden Sie die Funktion `leseCookie()`, die über den spezifizierten Namen den gespeicherten Wert ausliest und dem ersten Eingabefeld des Formulars wieder zuweist. Beachten Sie die Zeile 37 mit `if(wert != null) document.mF.id.value=wert;`. Mit dem `if`-Konstrukt wird sichergestellt, dass das Formularfeld nicht mit `null` gefüllt wird, wenn die Seite das erste Mal geladen wird. In dieser Situation ist das Cookie natürlich noch nicht vorhanden und ein Lesevorgang wird ins Leere laufen. Die Wertzuweisung wäre sinnlos und würde den Anwender mit so einer unsinnigen Angabe nur verwirren.

Tipp

Viele Anwender haben Cookies in ihrem Webbrowser deaktiviert. Dies ist sicher auch eine der größten Schwächen der Zustandsverwaltung und Speicherung von Sitzungsdaten beim Client. Sie können nicht gewährleisten, dass das Verfahren überhaupt funktioniert. Wenn Sie Cookies verwenden wollen, sollten Sie im Vorfeld testen, ob das bei einem Anwender überhaupt möglich ist. Dazu schreiben Sie am besten ein Dummy-Cookie und lesen den geschriebenen Wert unmittelbar wieder ein. Klappt dies, können Sie Cookies für Ihre eigentlichen Aufgaben verwenden.

Achtung

Auch wenn Sie bei einem Besucher Cookies schreiben können, dürfen Sie sich keinesfalls darauf verlassen, dass die Cookies bei der nächsten Stippvisite des Besuchers in Ihrem Webangebot noch verfügbar sind, selbst wenn Sie das Haltbarkeitsdatum lange genug setzen. Viele Anwender löschen Cookies regelmäßig manuell, automatisch beim Schließen des Browsers oder mittels eines Cookie-Managers. Das Cookie-Verfahren ist schlicht und einfach unzuverlässig, was dessen größte Schwäche darstellt. Darüber hinaus betrachten viele Besucher argwöhnisch den Einsatz von Cookies. Die Angst vor der Estellung von Anwenderprofilen ist auch keinesfalls unbegründet. Sie reduzieren durch den Einsatz von Cookies möglicherweise die Akzeptanz Ihres Webangebots.

10.3 Nachladen von Daten mit Frames

Ein sehr interessanter Ansatz, Daten asynchron vom Anwenderverhalten in Webseiten nachzuladen, führt über **Frames** und vor allem **IFrames** (Inline-Frames – das sind in eine Webseite eingebettete einzelne Frames). Frames waren ein Stilmittel in den 90er Jahren zur Aufteilung des Anzeigebereichs eines Browsers in einzelne Segmente, die unabhängig mit Inhalt gefüllt werden können. Zwar gelten Frames in Webseiten heutzutage als absolute NoNos[4], aber für das asynchrone Nachladen von Daten kann man sie immer noch gebrauchen. Die Frame-Technologie hat zwei Facetten. Es gibt einmal die Frames selbst, welche den Anzeigebereich des Browsers gliedern und dennoch eigenständige Fenster mit allen HTML-Darstellungsmöglichkeiten bilden. Diese einzelnen Frames sind zusammen in einer umgebenden Struktur – dem so genannten **Frameset** – enthalten. Die Frameset-Datei ist diejenige, welche in den Browser geladen wird. Sie enthält nur Informationen über Name, Größe und Position der einzelnen im Anzeigefenster enthaltenen Frames. Insbesondere ist der eigentliche Inhalt der Frames nicht in der Frameset-Datei notiert.

Eine HTML-Datei, welche als Frameset-Datei für eine Frame-Struktur verwendet werden soll, hat einen anderen Aufbau als eine gewöhnliche HTML-Datei. Es wird vor allem kein `<body>`-Tag mehr benötigt (obgleich er nicht verboten ist). Dessen Funktion wird von dem `<frameset>`-Container übernommen. Bei der Definition der Frameset-Datei müssen genaue Angaben darüber gemacht werden, wie das Anzeigefenster aufgeteilt werden soll und welche Dateien konkret in ein einzelnes Frame zu laden sind.

Es muss bei der Angabe der Aufteilung je nach Konzept entweder Reihen oder Spalten (Anzahl und Größe) oder auch beides definiert werden. Die Definition der Reihen und Spalten erfolgt als Erweiterung des `<frameset>`-Tag. Über den Parameter `rows="[Angabe in Prozent vom Browserfenster oder eine Pixelangabe]"` legen Sie die Anzahl von Reihen und deren anfänglichen, prozentualen Anteil an der Fenstergröße fest. So viele durch Kommata getrennte Prozentwerte dort notiert werden, so viele Zeilen hat der Frameset. Analog werden die Spalten im einleitenden `<frameset>`-Tag festgelegt, falls man eine Aufteilung nach Spalten wünscht. Dazu dient der Parameter `cols="[Angabe in Prozent vom Browserfenster oder eine Pixelangabe]"`.

Wenn also über den `<frameset>`-Tag und seine Parameter ein äußeres Gerüst für eine Frame-Struktur aufgebaut ist, wird im Inneren für jede dort definierte Zeile oder Spalte ein einzelnes Frame angegeben. Der Inhalt der Frames wird mit der Syntax `<frame src= "[URL]" name="[Name]">` spezifiziert. Über die `scr`-Angabe wird der URL des Inhalts des jeweiligen Frames gesetzt, `name` definiert einen internen Namen des Frames, über den er in Verweisen per HTML, aber auch aus JavaScript heraus angesprochen werden kann. Der Name eines Frames ist relativ frei wählbar. Es gelten nur

4 *Sie widersprechen dem barrierefreien Web, da viele Lesegeräte für Behinderte damit nicht klar kommen, das Layout ist altbacken, es können wichtige Teile der Webseite bei einer falschen Auflösung unsichtbar sein, mitten in der Webseite können Bildlaufleisten auftauchen etc.*

die üblichen Regeln für HTML-Anweisungen (keine Leerzeichen, so gut wie keine Sonderzeichen – mit Ausnahme des Unterstrichs) und die Namen sollten recht sprechend sein. Bestimmte Namen für Frames sind verboten, da diese reservierten Fensternamen bei Verweisen eine spezielle Bedeutung haben.

Tipp

Wenn Sie tatsächlich auf Frames zum Nachladen von Daten setzen, besteht ein Trick darin, einen Frame-Bereich auf ein Pixel Größe zu beschränken. Dort kann der Inhalt einer Datei unsichtbar geladen werden und von dort per DHTML in ein anderes Frame befördert werden[5].

Eine zum Nachladen von Daten oft verwendete Alternative zu »normalen« Frames sind IFrames. Diese sind einfach Fenstersegmente in einer gewöhnlichen Webseite, die unabhängig von der restlichen Webseite mit Inhalt gefüllt werden. Sie werden als `<iframe>`-Element an einer beliebigen Stelle in der Webseite notiert. Über den `src`-Parameter wird der URL der angezeigten Datei angegeben. Damit bieten sie sich wie ein `<div>`- oder `<span>`-Element an, um Inhalt unabhängig vom Inhalt der restlichen Seite anzuzeigen. In einen IFrame kann auch nach dem Anzeigen der Webseite – wie in einen normalen Frame – eine andere Datei geladen werden, ohne dass die restlichen Segmente der Webseite verändert werden. Eine Applikation mit dem Nachladen von Daten in einem IFrame ist damit sehr nahe an dem Verhalten einer AJAX-Applikation.

10.4 Vorratshaltung mit DHTML

Ein weiterer interessanter Ansatz, Daten asynchron vom Anwenderverhalten in Webseiten bereitzustellen, führt über die Vorratshaltung der Daten mit JavaScript und DHTML. Wohlbemerkt – es geht hier in den meisten Situationen[6] nicht um das asynchrone Nachladen von Daten, sondern um das asynchrone Bereitstellen. Mit anderen Worten – alle Daten werden zusammen mit der Webseite angefordert, aber teilweise noch nicht angezeigt. Erst wenn eine Anwenderaktion die Anzeige neuer Daten erfordert, werden diese aus einem »JavaScript-Cache« oder einem unsichtbaren Bereich der Webseite geholt und ohne Nachladen in der Webseite mit DHTML-Techniken angezeigt. Das Verfahren hat die Vorteile, dass die Webseite nicht neu nachgeladen werden muss und auch die neuen Daten blitzschnell verfügbar sind.

5 *Das verfolgen wir hier aber nicht – es gibt bessere Lösungen, auch in Verbindung mit Frames.*

6 *Eine Ausnahme sind zum Beispiel Bilder, die asynchron per JavaScript nachgeladen und angezeigt werden können.*

Zur Realisierung können Sie bei Bildern zum Beispiel mit `new Image()` in JavaScript Bildobjekte erzeugen und deren `src`-Eigenschaft konkrete Bilddateien zuweisen. Wir haben im Laufe des Buchs gesehen, wie diese dann in der Webseite angezeigt werden können. Oder aber Sie laden bereits beliebige Inhalte in `<div>`- oder `<span>`-Container und positionieren diese im Offscreen-Bereich. Bei Bedarf holen Sie diese hervor, indem eine Position zugewiesen ist, die im sichtbaren Bereich der Webseite liegt. Genauso effektiv ist das temporäre Sichtbar- und Unsichtbarmachen von Teilbereichen der Webseite. Oder aber Sie halten Daten in JavaScript-Variablen vor und schreiben mit `document.write()` eine Webseite bei Bedarf neu. Oder Sie nutzen die Inhalte, um mit Methoden von `nodes` neue Elemente zu erzeugen, auszutauschen etc. Die Techniken sind allesamt mit etwas JavaScript- und DHTML-Erfahrung leicht umzusetzen und wurden im Laufe des Buchs bereits in Verbindung mit nachgeforderten Daten gezeigt. Aber sämtliche Techniken der Vorratshaltung kranken daran, dass Sie möglicherweise sehr viele Daten auf Vorrat laden und vorhalten müssen, die vielleicht niemals gebraucht werden. Und ab einer bestimmten Datenmenge ist das Verfahren auch nicht mehr praktikabel. Denken Sie im Extremfall an Google suggest. Soll dem Besucher erst einmal der gesamte Datenbestand von Google übertragen werden, um ihm dann Vorschläge machen zu können ;-)?

Tipp

Falls Sie Daten auf Vorrat in einer Webseite nutzen wollen, notieren Sie einen Skript-Container mit den, beim Laden erst einmal versteckten, Daten ganz am Ende der Webseite, am besten nach dem `</body>`-Tag. Dann sieht der Anwender vor dem Laden der erst einmal unerheblichen Daten schon einmal alle Informationen, die ihn am Anfang interessieren. Während sich der Besucher auf Ihrer Seite umsieht, laden Sie im Hintergrund und vom Besucher unbemerkt diejenigen Daten weiter, die auf Vorrat in JavaScript oder einem unsichtbaren HTML-Container bereitgehalten werden sollen. Bei nicht zu großen Datenmengen ist das recht effektiv. In der Regel führt der Besucher durch die Zeitspanne zur ersten Orientierung erst dann eine Aktivität aus, die diese Daten notwendig macht, wenn sie geladen wurden.

10.5 Dateninseln

Wenn man sich anschaut, was AJAX eigentlich genau leistet, sollten Kenner ein Déjà-vu erleben. Denn bereits zu Zeiten des Internet Explorer 4 hatte Microsoft mit dem XMLHTTP-Objekt und so genannten **Dateninseln** (**Data Islands**) ein Konzept bereitgestellt, über das man XML-Daten vom Server nachladen konnte. Nur ging es im Fall von Dateninseln ausnahmsweise Microsoft so, wie es sonst dessen Konkurrenten geht – eine im Grunde geniale Technik wurde vorgestellt und keiner wollte sie. Kein Konkurrenzbrowser implementierte Dateninseln. Und auch die Verbindung mit der hoch

kritischen ActiveX-Technologie sorgte dafür, dass Dateninseln sich nicht durchsetzten. Dennoch – im Ansatz liefern Dateninseln bereits alles, was sich mit AJAX nun auf breiter Front etabliert.

Bei Dateninseln wird ein (X)HTML-Dokument direkt mit XML-Tags gemischt. Das bedeutet, in einer Webseite wird ein Bereich mit XML-Daten gefüllt – eben eine Dateninsel innerhalb eines anderen Datenformats. Es ist selbstverständlich auch möglich, mehrere Dateninseln innerhalb einer Webseite zu verwenden. Jede Dateninsel wird in der Webseite in ein `<xml>`-Element eingeschlossen. Dies ist kein offizieller (X)HTML-Tag des W3C, sondern ein spezifischer Microsoft-Tag, den entsprechend rein W3C-konforme Browser nicht unterstützen. Um sowohl den Zugriff auf Dateninseln mit JavaScript als auch die Anzeige der Daten zu ermöglichen, ordnen Sie einem `<xml>`-Element eine ID zu. Sofern eine **interne Dateninsel** vorliegt, wird im Inneren des `<xml>`-Elements direkt XML-Code notiert. Beispiel:

Listing 10.3: Eine Webseite mit einer internen Dateninsel

```
<html>
 <body>
  <h1>Die lieben Leute</h1>
  <hr>
  <xml id="adressliste">
   <adressen>
    <eintrag>
     <anrede>Frau</anrede>
     <name>Maria Meier</name>
    </eintrag>
    <eintrag>
     <anrede>Frau</anrede>
     <name>Paula Punkt</name>
    </eintrag>
    <eintrag>
     <anrede>Herr</anrede>
     <name>Hans Schmitz</name>
    </eintrag>
    <eintrag>
     <anrede>Herr</anrede>
     <name>Willi Schmidt</name>
    </eintrag>
    <eintrag>
     <anrede>Herr</anrede>
     <name>Karl Kaiser</name>
    </eintrag>
    <eintrag>
     <anrede>Herr</anrede>
     <name>Otto Fisch</name>
    </eintrag>
   </adressen>
```

```
32   </xml>
33   <hr>
34  </body>
35 </html>
```

Das Beispiel zeigt eine weitgehend konventionelle Webseite. Allerdings erstreckt sich von Zeile 5 bis Zeile 32 die interne Dateninsel.

Hinweis

Bei der Anzeige der Webseite aus dem letzten Beispiel werden die XML-Daten im Browser in der Regel nicht vernünftig dargestellt. Der Internet Explorer weiß bei den XML-Elementen nicht, wie er sie darstellen soll, und ignoriert alle XML-Daten (sowohl die Elementknoten als auch die Textknoten).

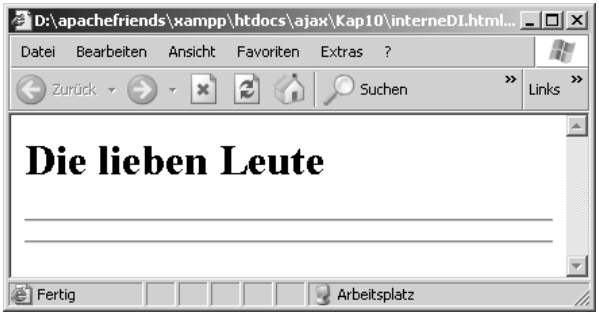


Abbildung 10.1: Nix zu sehen – aber das ist kein Fehler.

Der Internet Explorer kann die XML-Daten jedoch in Tabellen mit einem speziellen Parameter darstellen, wie wir sie gleich sehen. Dabei wird die XML-Datei als Datenquelle angegeben und die Daten werden Zeile für Zeile ausgegeben. Anders ist es mit allen anderen Browsern. Die kennen den proprietären `<xml>`-Tag nicht und ignorieren ihn ebenso wie alle folgenden XML-Tags[7] nach dem Prinzip der Fehlertoleranz. Sie stellen aber den reinen Text in den Elementen dar. Das ist aber in der Regel keine sinnvolle Verwertung der Dateninsel.

7 *Es sei denn, Sie verwenden dort Tags, die mit gültigen (X)HTML-Tags übereinstimmen – das wäre dann vom Browser aber eine Interpretation als (X)HTML- und nicht als XML-Tag.*

Abbildung 10.2: Sämtliche Browser mit Ausnahme des Internet Explorers werden alle Tags in der Dateninsel ignorieren und die Textknoten als reinen Text darstellen.

Es gibt neben internen Dateninseln auch die Möglichkeit **externer Dateninseln**. Und dies ist der Schlüssel zum asynchronen Nachladen von Daten. Wie bei der internen Dateninsel wird auch die externe Dateninsel mit `<xml>` eingebunden. Der Tag wird jedoch mit dem Parameter `src` erweitert, um die externe XML-Datei anzugeben, und als leeres Element definiert. Damit wird aus der obigen Webseite Folgendes:

Listing 10.4: Eine Webseite mit einer externen Dateninsel

```
01 <html>
02  <body>
03   <h1>Die lieben Leute</h1>
04   <hr>
05   <xml id="adressliste" src="externedi.xml"></xml>
06   <hr>
07  </body>
08 </html>
```

In Zeile 5 befindet sich die Referenz auf die externe Dateninsel. Die XML-Datei sieht so aus:

Listing 10.5: Die XML-Datei

```
01 <?xml version="1.0" encoding="utf-8" ?>
02  <adressen>
03     <eintrag>
04      <anrede>Frau</anrede>
05      <name>Maria Meier</name>
06     </eintrag>
07     <eintrag>
08      <anrede>Frau</anrede>
09      <name>Paula Punkt</name>
10     </eintrag>
```

```
11     <eintrag>
12      <anrede>Herr</anrede>
13      <name>Hans Schmitz</name>
14     </eintrag>
15     <eintrag>
16      <anrede>Herr</anrede>
17      <name>Willi Schmidt</name>
18     </eintrag>
19     <eintrag>
20      <anrede>Herr</anrede>
21      <name>Karl Kaiser</name>
22     </eintrag>
23     <eintrag>
24      <anrede>Herr</anrede>
25      <name>Otto Fisch</name>
26     </eintrag>
27    </adressen>
```

Das ist einfach eine wohlgeformte XML-Datei.

Eine externe Dateninsel löst das Problem, dass W3C-konforme Browser den <xml>-Tag und die enthaltenen XML-Tags ignorieren und die Textinhalte in unerwünschter Weise darstellen. Wir haben hier ja nur noch ein leeres Element vorliegen. Aber auch bei Verwendung der externen Dateninsel ist das Problem noch nicht gelöst, dass die Daten auch im Internet Explorer nicht dargestellt werden. Aber dafür kann der Internet Explorer im <table>-Tag mit einem speziellen Parameter umgehen, der die XML-Daten als Datenquelle angibt. Dabei wird die XML-Datei als Datenquelle angegeben und die Daten werden Zeile für Zeile ausgegeben.

Hinweis

Dateninseln sind weitgehend auf die Darstellung von tabellenartig aufgebauten XML-Dokumenten in HTML-Tabellen beschränkt. Das ist aber keine wirkliche Einschränkung, denn mit Tabellen mit einer Zelle oder Style Sheets und DHTML können Sie mit Dateninseln nahezu alle Möglichkeiten von AJAX realisieren.

Dies ist die erweiterte Webseite:

Listing 10.6: Anzeige der XML-Daten in einer Tabelle

```
01 <html>
02  <body>
03   <h1>Die lieben Leute</h1>
04   <hr>
05   <xml id="adressliste" src="externedi.xml"></xml>
06   <table border="1" datasrc="#adressliste">
```

```
07     <thead>
08       <tr><th>Anrede</th><th>Name</th></tr>
09     </thead>
10     <tr><td><span datafld="anrede"></span></td>
11     <td><span datafld="name"></span></td></tr>
12   </table>
13   <hr>
14  </body>
15 </html>
```

Die Webseite enthält nun eine Tabelle, die aber scheinbar nur aus zwei Zeilen besteht. Der Parameter `datasrc` des `<table>`-Tag in Zeile 6 verweist jedoch auf die ID des `<xml>`-Elements (`<table border="1" datasrc="#adressliste">`) und gibt dieses damit als Datenquelle an. Von Zeile 7 bis 9 wird ein Kopf für die Tabelle definiert. Das verhindert, dass diese Zeile wiederholt wird, wenn die Datenquelle Datensatz für Datensatz ausgegeben wird. In Zeile 10 und 11 finden Sie den Parameter `datafld` bei den `<td>`-Elementen. Diese referenzieren auf die Spalten in der XML-Datenquelle. Beim Laden der Webseite in den Internet Explorer wird die gesamte Datenquelle zeilenweise durchlaufen und daraus werden dynamisch so viele Zeilen in der Tabelle generiert, wie es Datensätze gibt.

Abbildung 10.3: Die Daten aus der XML-Datenquelle werden in einer Tabelle angezeigt.

Nun können Sie natürlich jederzeit mit der ID des `<xml>`-Elements die Wertzuweisung des `src`-Parameters per JavaScript ändern und damit dynamisch Daten nachfordern. Ebenso lässt sich mit dem Parameter `datapagesize` beim `<table>`-Tag die Anzahl der maximal angezeigten Zeilen einstellen. Und die ID stellt in dem Microsoft-Konzept

ein Objekt da, das Methoden wie `previousPage()` oder `nextPage()` bereitstellt, um damit aus JavaScript heraus neue Seiten in dem Tabellenbereich anzuzeigen. Dies werden wir hier aber nicht weiter ausführen.

10.6 Zusammenfassung

Sie haben in diesem Kapitel einige Alternativen für AJAX oder zumindest Teilbereiche der AJAX-Funktionalität gesehen. Die Palette ist dabei sehr groß, denn über die Zeit des Webs erfanden Entwickler ständig neue Ideen, um den Einschränkungen von HTTP begegnen zu können – seien es Cookies zur Zustandsverwaltung und Speicherung von kleinen Datenmengen auf dem Client, sei es das Laden von Daten auf Vorrat oder auch das Bereitstellen von Daten in IFrames oder Frames. Aber erst AJAX vereint die Vorteile der verschiedenen Techniken und vermeidet die meisten Nachteile – obwohl man festhalten muss, dass Dateninseln und das XMLHTTP-Objekt von Microsoft die wesentlichen Ideen von AJAX bereits beinhalten.

11 AJAX-Frameworks

Sie haben in diesem Buch nun alle erforderlichen Techniken kennen gelernt, um AJAX-Applikationen zu erstellen. Nur sollte Ihnen klar geworden sein, dass AJAX-Applikationen für die Zukunft des Webs sehr viele Vorteile bieten, jedoch auch ein hohes Maß an Tests und Anpassung an verschiedene Webbrowser und Plattformen mit sich bringen können. Auch wenn JavaScript-Code in den meisten AJAX-Anwendungen nicht sonderlich kompliziert und auch serverseitige Programmierung mit Java oder anderen Sprachen selten Hexenwerk ist, warten zahlreiche Fallstricke. Besonders tückisch ist, dass bei einer AJAX-Applikation mehrere Stellen beteiligt sind. Eine AJAX-Anfrage durchläuft ein Browser-Skript, das `XMLHttpRequest`-Objekt, den Webserver, eventuell eine nachgeschaltete Applikation wie ein Datenbanksystem und dann wieder den Webserver, erneut das `XMLHttpRequest`-Objekt und zum Schluss wieder das Browser-Skript. Es kann im Fehlerfall sehr mühsam sein, herauszufinden, an welcher Stelle ein Fehler auftrat. Ebenso werden bestimmte Aufgaben bei einer AJAX-Applikation immer wieder zu lösen sein. Sehr bequem ist es dabei vor allem, wenn etwas verzwicktere Probleme der AJAX-Technologie bereits mit vernünftigen Bibliotheken gelöst werden. So funktioniert das Setzen von Lesezeichen bei einer AJAX-Applikation erst einmal nicht vernünftig und es wird ja bei AJAX die Grundfunktionalität der ZURÜCK-Schaltfläche im Browser ausgehebelt. Genau genommen kann die gesamte Benutzerführung recht diffizil werden, wenn Anwender ältere Inhalte wie gewohnt reproduzieren wollen. Natürlich kann man das manuell mit JavaScript, HTML-Ankern etc. kompensieren, aber dazu muss man recht trickreich programmieren und kann für vorgefertigte Lösungen dankbar sein.

So genannte **Frameworks** für AJAX stellen Ihnen solche Funktionsbibliotheken mit getesteten und hoch funktionellen Lösungen bereit, damit Sie nicht jedes Mal das Rad neu erfinden und vor allem dessen einwandfreie Funktionalität umfangreich testen müssen. Insbesondere werden Frameworks oft von mehreren Leuten oder gar ganzen Firmen vorangetrieben und damit wird in die Entwicklung manchmal mehr Manpower investiert als ein einzelner Programmierer im ganzen Leben vor seiner Kiste sitzt. Frameworks sind zum Teil als reine JavaScript-Erweiterungen für den Browser konzipiert, also Bibliotheken mit vielen nützlichen Funktionen. Andere Frameworks laufen auf dem Webserver mit Programmiersprachen wie PHP, ASP.NET, Java oder Python. Bei vielen Frameworks wird vor allem das reibungslose Zusammenspiel zwischen der serverseitigen und der clientseitigen Welt gewährleistet. Andere Frameworks gestatten selbst visuelle Erstellung der Oberflächenparts einer AJAX-Applika-

tion (so genannte **Web-Controls** oder auch **Komponenten**[1]). Hinzukommt das Zuordnen von spezifischen Events zu bestimmten Komponenten des Webinterface, wenn ein Framework eine grafische Entwicklungsumgebung mitbringt oder sich darin integrieren lässt. In diesem abschließenden Kapitel sollen einige wichtige AJAX-Frameworks kurz vorgestellt werden. Dabei soll und kann dies aber nur eine Auswahl aller verfügbaren Lösungen darstellen, denn die Entwicklung von AJAX-Frameworks ist gerade erst angelaufen und gewaltig in Bewegung.

11.1 Für wen lohnen sich Frameworks?

AJAX-Frameworks lohnen sich nicht für jeden. Sie können sich nicht auf den Standpunkt stellen, dass Sie jetzt alles vergessen können, was Sie in den bisherigen Kapiteln über AJAX erfahren haben, da Sie jetzt ein Framework einsetzen wollen. Um ein AJAX-Framework zu nutzen, müssen Sie auf jeden Fall die Idee und Arbeitsweise von AJAX verstanden haben und alle Grundtechnologien zumindest im Ansatz begreifen. Das gilt insbesondere für die Frameworks, die als reine Funktionsbibliotheken konzipiert sind. In jedem Fall erfordert der Einsatz von AJAX-Frameworks von Ihnen die Einarbeitung in die jeweiligen Funktionsbibliotheken und Arbeitsweisen eines Frameworks (auch bei visuell unterstützenden Frameworks). Das kann recht kompliziert sein. Diese Mühe lohnt sich bei kleineren Webpräsenzen mit wenigen asynchronen Datenanforderungen kaum. Des Weiteren sollten Sie bei der Verwendung von Frameworks einkalkulieren, dass gerade visuell arbeitende Frameworks (mit WYSIWYG-Modus und dem visuellen Generieren von Events) oft sehr großen und ineffektiv erstellten Quellcode bewirken, der zudem oft nur in einigen Browsern ohne Probleme funktioniert. Denken Sie als Beispiel nur an die reine Webseitenerstellung. Wenn Sie von einem Tool wie Frontpage erstellten Code einer Seite betrachten und im Vergleich dazu eine von Hand erstellte Seite, die identisch aussieht und sich identisch verhält, werden Sie bei der WYSIWYG-Seite einen vielfach umfangreicheren Code finden[2].

Auf der anderen Seite werden Sie nach dem Mühsal der Einarbeitung Ihre AJAX-Applikationen mit einem Framework viel effektiver, effizienter und robuster entwickeln können. Das wirkt sich besonders bei großen Webangeboten mit umfangreichen asynchronen Datenanforderungen aus. Als Faustregel kann gelten: Je größer die Webpräsenz und je mehr asynchrone Datenaustausche Sie benötigen, desto sinnvoller ist der Einsatz eines Frameworks. Für welches Framework Sie sich aber entscheiden, hängt an vielen Faktoren. Das beginnt bei den Gegebenheiten bei Ihrem Internet Provider respektive Webserver und geht über die gewünschten Basistechnologien, das Wissen der beteiligten Programmierer bis hin zu Ihrem persönlichen Geschmack. Und bevor Sie sich für ein Framework entscheiden, sollten Sie genau wissen, welche Aufgaben Sie mit AJAX erledigen wollen. Und nicht zuletzt sollten Sie auf ein Projekt

1 *Schaltflächen, Eingabeboxen etc.*
2 *Die reine Größe des Bytecodes kann im unglücklichsten Fall bis zu tausend (!) Mal größer sein.*

setzen, von dem zu erwarten ist, dass es weiter gepflegt wird. Sonst kann es bei Updates von Browsern zu Fehlern kommen und Sie müssen mit hohem Aufwand das Framework wechseln. In jedem Fall begeben Sie sich bei der Verwendung eines Frameworks in gewisse Abhängigkeit von diesem Projekt respektive der dahinter stehenden Firma.

11.2 Verschiedene Frameworks

Nachfolgend finden Sie eine Auswahl wichtiger AJAX-Frameworks. Diese Auswahl ist aber in keiner Weise vollständig. Der Markt der AJAX-Frameworks ist zurzeit gewaltig in Bewegung und diese Auswahl kann deshalb auch nur eine Momentaufnahme darstellen.

11.2.1 Atlas

Eines der wichtigsten Frameworks ist sicher Atlas von Microsoft (*http://atlas.asp.net*). Insbesondere aus historischer Sicht, denn mit dem Atlas-Framework hat Microsoft seine Idee der Dateninseln wieder neu aufgegriffen, konsequent weiterentwickelt und in seine gesamte Entwicklungsstrategie aufgenommen. Allerdings befindet sich das Atlas-Framework zum Zeitpunkt der Bucherstellung noch in einer sehr frühen Phase. Zudem setzt das Framework explizit auf die .NET-Plattform auf, was die Wahl des genutzten Webserver-Umfelds ziemlich begrenzt. Gerade im Apache-Umfeld werden meist andere Plattformen eingesetzt.

Mit der Atlas zugrunde liegenden AJAX.NET-Bibliothek hat Microsoft nun ein auf allen relevanten Browsern unterstützten Weg geschaffen, um zwischen Server und Client ohne Neuladen der Seite Daten auszutauschen. Die AJAX.NET-Bibliothek stellt Methoden zur Verfügung, die direkt in JavaScript auf Client-Seite verwendet werden können. Dabei werden für den Client dynamisch so genannte JavaScript-Wrapper (Hüllfunktionen) erstellt, die in eine Webseite eingebunden werden können und mit dem Server-System kommunizieren. Dazu bietet Microsoft mit der **Client Script Library** eine umfangreiche JavaScript-Erweiterung mit Funktionalitäten an, die bereits aus der OOP-Welt[3] bekannt sind und auf regulären Sprachelementen und Objekten von JavaScript aufsetzen. Aber auch die Datentypen von JavaScript wurden erweitert und an die .NET-Datentypen angepasst. Einige Objekte aus .NET wurden ebenso mit JavaScript nachprogrammiert. Da aber die gesamten Erweiterungen auf purem JavaScript aufsetzen, sollen nach Aussage von Microsoft alle diese Erweiterungen eben nicht nur vom Internet Explorer, sondern von allen gängigen Browsern unterstützt werden. Erste Erfahrungen zeigen jedoch, dass es in einigen Browsern mit Mozilla-Basis im aktuellen Stadium Probleme gibt.

Die Besonderheit von Atlas selbst ist jetzt, dass Sie damit nicht nur Daten austauschen können, sondern auch im Rahmen der .NET-Entwicklungstools von Microsoft gra-

3 *Schnittstellen, Vererbung, Namensräume etc.*

fisch Weboberflächen erstellen können. Wenn Sie im Rahmen einer solchen IDE Ihre AJAX-Applikation erstellen, können Sie Controls aus der Toolbox mit Drag&Drop auf die Entwurfsansicht der Webseite ziehen und in der grafischen Ansicht bereits die Reaktionsmuster für die jeweiligen Komponenten festlegen.

11.2.2 Sajax

Sajax (Simple AJAX Toolkit – *http://www.modernmethod.com/sajax/*) ist – wie der Name schon andeutet – ein sehr einfaches Framework, dessen hauptsächliche Ausrichtung darauf liegt, mittels JavaScript aus einer Webseite heraus serverseitige Funktionen in verschiedenen Programmiersprachen aufzurufen (unter anderem PHP, Perl, Python und Ruby). Es ist also eine reine Sammlung von JavaScript-Funktionen. Für jede unterstützte Programmiersprache beinhaltet das Framework eine Bibliothek, die das Einbinden von Funktionen der jeweiligen Sprache in eine AJAX-Anwendung ermöglicht. Die Details unterscheiden sich selbstverständlich je nach Sprache. Aber das grundsätzliche Vorgehen bleibt immer gleich. Wie im Atlas-Framework gibt es Wrapper-Funktionen, die den eigentlichen Aufruf kapseln.

11.2.3 Prototype JavaScript Framework

Hinter dem **Prototype JavaScript Framework** (*http://prototype.conio.net/*) verbirgt sich eine relativ komplexe JavaScript-Bibliothek, um objektorientierte Techniken in JavaScript zu ergänzen. Die Namensgebung deutet das bereits an, denn Prototyping ist ja in JavaScript eine Art Ersatz für Vererbung. Insbesondere beinhaltet die Bibliothek einige Funktionen für den Umgang mit AJAX. Um die Bibliothek zu verwenden, muss sie bloß als externe JavaScript-Datei in eine Webseite eingebunden werden.

11.2.4 Rico AJAX Framework minimal

Das **Rico Framework** (*http://openrico.org/*) ist ein sehr einfaches Interface in Form einer JavaScript-Bibliothek, damit Ihre Webapplikation Drag&Drop von Teilen der Webseite unterstützt. In Rico können Sie einfach irgendein HTML-Element oder JavaScript-Objekt als verschiebbar und irgendein anderes HTML-Element oder JavaScript-Objekt als Zielzone registrieren. Rico kümmert sich um den Rest. Dazu bietet Rico noch einige visuelle Effekte.

11.2.5 AJAX Toolkit Framework

Ein sehr zukunftsträchtiges Projekt ist das **AJAX Toolkit Framework**, das im Rahmen des Apache-Projekts angesiedelt ist. Genau genommen wurde es für den Incubator (Brutkasten) PMC von Apache (*http://incubator.apache.org/*) als strategisches Framework für Entwicklungsumgebungen vorgeschlagen. Es soll als Plug-in in die Entwicklungsumgebung Eclipse integriert werden und ein Framework für verschiedene interaktive AJAX-Entwicklungsumgebungen bieten. Damit kann die Erstellung von

AJAX-Applikationen erheblich vereinfacht werden, denn ein JavaScript-Editor mit Syntaxüberprüfung während der Eingabe samt DOM-Browser und JavaScript-Debugger verspricht Unterstützung, die es derzeit im JavaScript-Umfeld selten gibt. Das Framework basiert auf dem Mozilla XULRunner, JavaConnect und Eclipse WTP. Außerdem sind die freie JavaScript-Implementierung Rhino, der JavaScript-Überprüfer JSLint, die JavaScript-Bibliothek Rico für AJAX und der Web-Mailer und Webkalender Zimbra enthalten.

11.2.6 Sarissa

Sarissa (*http://sarissa.sourceforge.net/doc/*) war eine makedonische Stoßlanze aus dem 4. Jahrhundert v. Chr. Damit haben die Macher für dieses Projekt wie bei AJAX einen Namen aus der griechischen Geschichte gewählt, was sicher kein Zufall ist. Bei Sarissa handelt es sich um eine reine JavaScript-Bibliothek zur Erweiterung von Browser-Funktionen, die in erster Linie eine einheitliche Schnittstelle zu den XML-APIs verschiedener Browser bietet. Hauptfokus des Frameworks liegt also auf dem Umgang mit XML-Dokumenten. Dabei wird ein objektorientierter Ansatz gewählt. XML-Dokumente werden über Objekte gekapselt.

11.2.7 Weitere Frameworks

Es gibt derzeit noch zahlreiche weitere Frameworks rund um AJAX. Und es werden ob des AJAX-Hypes ständig mehr. Das **Flexible Ajax Framework** (*http://tripdown.de/flexible_ajax_intro.php*) ist zum Beispiel ein sehr junges Projekt, das einiges verspricht. Auch **JSPAN** (*http://jpspan.sourceforge.net*) lohnt es sich anzusehen. **PAJAX** ist ein Framework, um PHP-Objekte AJAX-kompatibel zu machen. PAJAX (*http://www.auberger.com/pajax/3/*) stellt einen Konnektor zur Verfügung, der ein JavaScript-Interface für XMLHttpRequests emuliert. Und unter *http://net.sf.tacos.ajax.components* finden Sie diverse Java-Klassen zur Unterstützung von AJAX. Die Links zu zahlreichen weiteren Frameworks finden Sie unter *http://ajaxpatterns.org/wiki/index.php?title=AJAXFrameworks*.

11.3 Zusammenfassung

Der Einsatz von AJAX-Frameworks lohnt sich beileibe nicht für jedermann. Gerade für kleinere Webpräsenzen mit wenigen asynchronen Datenanforderungen kann der Einsatz eines Frameworks bedeuten, dass Sie mit Kanonen auf Spatzen schießen. Ebenso gibt es viele Programmierer, die gerne die vollständige Kontrolle über ihren Code behalten wollen und lieber eigenen Lösungen vertrauen[4]. Aber bei großen

4 *Ich zähle mich definitiv dazu. Für meine Homepage habe ich – statt mich in ein CMS einzuarbeiten und es anzupassen – lieber ein solches selbst programmiert. Zwar viel einfacher und beileibe nicht so professionell wie die auf dem Markt vorhandenen, aber dafür kenne ich mein Projekt vorwärts wie rückwärts. Dennoch – das kann man sich nur leisten, wenn man viel Zeit hat und Spaß daran.*

Webangeboten mit umfangreichen asynchronen Datenanforderungen wird schnell der Break-Even erreicht. Sie erhalten zum Beispiel ausgereifte und getestete Funktionen, die Sie in Ihren eigenen Code integrieren können, oder gar vollständige Entwicklungsumgebungen, die Ihnen selbst eine visuelle Erstellung Ihrer AJAX-Applikationen gestatten.

A Anhang

In diesem Anhang finden Sie Quellen im Internet sowie Tabellen zu den AJAX-Kerntechnologien und ein Glossar.

A.1 Quellen

Beschreibung	URL
AJAXIAN.COM	*ajaxian.com/by/topic/ajax/*
AJAX-NET.de – ein deutschsprachiges Portal zu AJAX vom Autor dieses Buchs. Hier gibt es auch die Buchbeispiele zum Download.	*www.ajax-net.de*
Ajaxpatterns.org	*ajaxpatterns.org*
AjaxWhois	*www.ajaxwhois.com*
Das AJAX Toolkit Framework bzw. der Incubator PMC von Apache	*incubator.apache.org*
Das Atlas-Framework	*atlas.asp.net*
Das Flexible Ajax Framework	*tripdown.de/flexible_ajax_intro.php*
Das JSPAN-Framework	*jpspan.sourceforge.net*
Das offizielle Essay von Jesse James Garrett zu AJAX: A New Approach to Web Applications	*www.adaptivepath.com/publications/essays/archives/000385.php*
Das PAJAX-Framework	*www.auberger.com/pajax/3/*
Das Prototype JavaScript Framework	*prototype.conio.net/*
Das Rico Framework	*openrico.org/*
Das Sajax-Framework	*www.modernmethod.com/sajax/*
Das Sarissa-Framework	*sarissa.sourceforge.net/doc/*
Das W3C	*www.w3.org*
Der Conglomerate XML-Editor	*www.conglomerate.org*
Der Gnome XML-Parser	*www.xmlsoft.org/*

Tabelle A.1: Quellen

Beschreibung	URL
Der HTML-Validierungsservice des W3C	*validator.w3.org*
Der HTML-Editor Nvu	*www.mozilla.nightrat.net/nvu/*
Der HTML-Editor Phase 5	*www.ftp-uploader.de/*
Der KXML-Editor	*kxmleditor.sourceforge.net*
Die Homepage des Apache-Projekts	*www.apache.org*
Die Homepage des Autors	*www.rjs.de*
Die Homepage des Eclipse-Projekts	*www.eclipse.org*
Die Homepage des XAMPP-Projekts	*www.apachefriends.org/de/*
Die Homepage des XMLBuddy-Projekts	*xmlbuddy.com/*
Die offiziellen DOM-Spezifikationen auf den Seiten des W3C	*www.w3.org/DOM*
Google Maps	*maps.google.com*
Google Pages	*www.google.com/accounts/ServiceLogin*
Google suggest	*www.google.com/webhp?complete=1&hl=en*
Informationen zu Flex	*www.macromedia.com/software/flex/*
Informationen zu OpenLaszlo	*www.openlaszlo.org/*
Links zu diversen AJAX-Frameworks	*ajaxpatterns.org/wiki/index.php?title=AJAXFramework*
Microsoft Live.com	*www.live.com*
Offizielle Beschreibung von XML beim W3C	*www.w3.org/XML*
Offizielle Informationen zu CSS 1 auf den Seiten des W3C	*www.w3.org/TR/REC-css1/*
Offizielle Informationen zu CSS 2 auf den Seiten des W3C	*www.w3.org/TR/PR-CSS2/*
Offizielle Informationen zu DTDs auf den W3C-Seiten	*www.w3.org/TR/REC-xml*
Offizielle Informationen zu Namensräumen auf den W3C-Seiten	*www.w3.org/TR/REC-xml-names*
Offizielle Informationen zu XML-Schemata auf den W3C-Seiten	*www.w3.org/XML/Schema*
Offizielle Informationen zu XSL und XSLT auf den W3C-Seiten	*www.w3.org/Style/XSL* *www.w3.org/TR/xsl* *www.w3.org/TR/xslt*
Offizielle Informationen zu ECMAScript	*www.ecma-international.org*
Peter's XML-Editor	*www.iol.ie/~pxe/*

Tabelle A.1: Quellen (Forts.)

Beschreibung	URL
Writely	*www.writely.com*
Yahoo -- Flickr	*www.flickr.com*
Yahoo -- Maps	*maps.yahoo.com*

Tabelle A.1: Quellen (Forts.)

A.2 Die JavaScript-Schlüsselwörter

JavaScript-Schlüsselwörter sind Bezeichner, die in JavaScript eine feste Bedeutung haben (so wie `var`). Wenn diese Wörter als Variablen- oder Funktionsname verwendet werden, kann der JavaScript-Interpreter nicht zwischen dem Variablen- bzw. Funktionsname und dem JavaScript-Schlüsselwort unterscheiden. Deshalb dürfen Sie sie dafür nicht verwenden.

Hinweis

Nicht alle reservierten Wörter von JavaScript sind in Gebrauch. Einige Wörter sind für zukünftige Sprachvarianten reserviert. Aber auch diese nicht benutzten reservierten Wörter dürfen Sie nicht als Variablen- oder Funktionsnamen verwenden. Diese Reservierung ist von Java beeinflusst. Ein wesentlicher Grund für die Sperrung dieser Wörter dürfte darin zu sehen sein, dass Missverständnisse mit Java vermieden werden sollen. Die reservierten Wörter sind dort teilweise in Verwendung und es steht zu erwarten, dass der Einfluss von Java diese Wörter auch irgendwann in JavaScript hinüber befördert.

Die nachfolgende Tabelle enthält die reservierten Wörter von JavaScript.

Schlüsselwort	Beschreibung
`abstract`	reserviert
`boolean`	reserviert
`break`	Abbruch in Schleifen
`byte`	reserviert
`case`	Auswahlfall bei einer `switch`-Unterscheidung
`catch`	Einsatz im Rahmen des Exceptionhandlings zum Auffangen von Ausnahmen (erst in JavaScript 1.4 bzw. 1.5)
`char`	reserviert
`class`	reserviert

Tabelle A.2: Reservierte Wörter von JavaScript

Schlüsselwort	Beschreibung
`const`	reserviert
`continue`	Fortsetzung in Schleifen
`default`	Default-Auswahlfall bei einer `switch`-Unterscheidung
`delete`	Löschen eines Array-Elements oder einer selbst definierten Objekteigenschaft
`do`	Beginn einer fußgesteuerten Schleife (`do-while`)
`double`	reserviert
`else`	Einleitung des alternativen Blocks in einer `if`-Entscheidung
`export`	Objekte oder Funktionen für fremde Skripts ausführbar machen
`extends`	reserviert
`false`	der Wert falsch
`final`	reserviert
`finally`	reserviert
`float`	reserviert
`for`	Einleitung von `for`-Schleifen
`function`	Einleitung von Funktionen
`goto`	reserviert
`if`	Einleitung von `if`-Entscheidungen
`implements`	reserviert
`import`	Objekte oder Funktionen eines fremden Skripts importieren
`in`	bedingte Anweisungen in `if`-Entscheidungen
`instanceof`	reserviert
`int`	reserviert
`long`	reserviert
`native`	reserviert
`new`	Definition von Objekten
`null`	reserviert
`package`	reserviert
`private`	reserviert
`protected`	reserviert
`public`	reserviert
`return`	Übergabe eines Rückgabewerts in Funktionen

Tabelle A.2: Reservierte Wörter von JavaScript (Forts.)

Schlüsselwort	Beschreibung
`short`	reserviert
`static`	reserviert
`super`	reserviert
`switch`	Fallunterscheidung
`synchronized`	reserviert
`this`	Bezug auf die aktuelle Instanz eines Objekts. Über dieses Schlüsselwort haben Sie Zugang auf das aktuelle Objekt, in dem Sie gerade arbeiten.
`throw`	Einsatz im Rahmen des Exceptionhandlings zum Auswerfen von Ausnahmen (erst in JavaScript 1.4 bzw. 1.5)
`throws`	Einsatz im Rahmen des Exceptionhandlings zum Dokumentieren von Ausnahmen (erst in JavaScript 1.4 bzw. 1.5)
`transient`	reserviert
`true`	der Wert wahr
`try`	Einsatz im Rahmen des Exceptionhandlings zum Umschließen von kritischen Anweisungen, die Ausnahmen auswerfen können (erst in JavaScript 1.4 bzw. 1.5)
`typeof`	Typ eines Elements
`var`	Definition einer Variablen
`void`	leerer Funktionstyp
`while`	Einleitung einer `while`-Schleife
`with`	erlaubt es, mehrere Anweisungen mit einem Objekt durchzuführen

Tabelle A.2: Reservierte Wörter von JavaScript (Forts.)

A.3 Glossar

In diesem Glossar finden Sie die wichtigsten Begriffe im Umfeld von AJAX. Dabei wird ausdrücklich kein Anspruch auf Vollständigkeit erhoben:

Begriff	Beschreibung
ANSI	ANSI ist die Abkürzung für American National Standards Institute und bezeichnet im Grunde die US-amerikanische Stelle zur Normung industrieller Verfahrensweisen. Allerdings wird der Begriff oft als Synonym für eine Gruppe von Zeichensätzen verwendet, die auf dem ASCII-Zeichensatz basieren.
ASCII	Der American Standard Code for Information Interchange ist ein standardisierter Zeichensatz zur Textdarstellung. Er basiert auf dem lateinischen Alphabet, wie es im modernen Englisch benutzt wird.
CSS	Cascading Style Sheets sind die populärste Form der Formatvorlagen im WWW. Sie sind eine eigene Sprache und werden wie XML vom W3C standardisiert.
Dateninseln	XML-Daten können nicht nur als eigenständige Dateien gespeichert werden, sondern auch in andere Dateien mit fremden Formaten eingebettet werden, beispielsweise in HTML-Dateien. In diesem Fall spricht man von Dateninseln in dieser fremden Datei. Für die Darstellung und den Zugriff auf diese Dateninseln bieten sich andere Möglichkeiten als in reinen XML-Dokumenten. So kann man bei einer Dateninsel in einem HTML-Dokument etwa mit JavaScript darauf zugreifen und HTML- oder CSS-Formatierungen anwenden. Allerdings sind Dateninseln eine Technologie, für die es kaum Anwendungsprogramme gibt. Im Grunde unterstützt nur der Internet Explorer in den neuesten Versionen Dateninseln, die in HTML-Dateien eingebettet werden.
DOM	Das Document Object Model ist ein Konzept des W3C zum Zugriff auf Bestandteile eines strukturierten Klartextdokuments. Insbesondere kann man aus JavaScript DOM verwenden. Aber auch der Zugriff aus anderen Programmiertechniken ist möglich.
DTD	Eine Document Type Definition beschreibt allgemein ein Regelwerk, mit dem die Struktur, der Inhalt und die Bedeutung von XML-Dokumenten festgelegt werden können. DTD ist eine Konkurrenz zu XML-Schemata, aber explizit nicht aus XML aufgebaut. Die DTD ist eine eigene Sprache und wird wie XML vom W3C standardisiert. DTDs sind bedeutend älter als Schemata und liegen als Grammatik sowohl HTML als auch XML selbst zugrunde.
HTML	Die Hyper Text Markup Language ist eine auf die Darstellung von Informationen in einem Hypertextsystem optimierte Auszeichnungssprache und die Basis des WWW.
HTTP	Das Hyper Text Transfer Protocol ist das Standardübertragungsprotokoll des WWW.

Tabelle A.3: Verschiedene Abkürzungen und Fachausdrücke aus dem AJAX-Umfeld

Begriff	Beschreibung
ISO 8859-1	ISO 8859-1 bzw. ISO/IEC 8859-1 ist ein Standard zur Zeichenkodierung, der in den ersten sieben Bit dem ASCII-Zeichensatz mit führendem Nullbit entspricht. Zusätzlich zu den damit dargestellten ASCII-Zeichen kodiert ISO 8859-1 weitere Zeichen, um möglichst viele Sonderzeichen westeuropäischer Sprachen abzudecken.
MIME	Multipurpose Internet Mail Extensions bedeutet einen Internetstandard zur Spezifizierung von Dateitypen bei der Kommunikation zwischen einem Server und einem Client (in der Regel ein Browser) im WWW. Sowohl der Server als auch der Client kennen bestimmte Dateitypen. Beim Übertragen vom Server zum Client wird über das HTTP-Protokoll der MIME-Typ mit übertragen. Aufgrund seiner Liste mit MIME-Typen kann der Client eine Datei eines bekannten Typs korrekt behandeln. Werden unbekannte Dateitypen geladen, kann das Programm nicht direkt damit umgehen, sondern muss die Datei speichern oder über zusätzliche Informationsquellen (etwa einen Benutzerdialog) nach der Behandlungsweise forschen.
MSXML	Die Microsoft eXtensible Markup Language zählt zu den XML-Basisdiensten von Microsoft-Betriebssystemen und ist in modernen Varianten meist automatisch integriert. Sie stellt den XML-Parser bereit, den Windows-Applikationen wie der Internet Explorer zur Verarbeitung von Daten im XML-Format verwenden. MSXML-Parser gibt es in verschiedenen Versionen, die teilweise inkompatibel (auch nicht abwärtskompatibel) sind.
OpenLazlo	Ein Framework zur Erstellung von interaktiven Webanwendungen auf Basis von Flash und einer objektorientierten, ereignisgesteuerten XML-Programmiersprache mit Namen LZX (Kurzform für Lazlo).
Parser	Ein Parser ist ein Tool, mit dem ein Dokument analysiert, interpretiert und entsprechend aufbereitet werden kann. Im Fall von XML-Dokumenten stehen verschiedene XML-Parser wie Xerces, Crimson oder MSXML bereit.
SGML	Die Standard Generalized Markup Language ist eine Auszeichnungssprache, die die Basis von den meisten modernen Auszeichnungssprachen wie HTML oder XML darstellt. SGML ist extrem leistungsfähig und umfangreich, aber sehr komplex und schwer zu handhaben.
SOAP	Das Simple Object Access Protocol ist aus dem XML Remote Precedure Call (XML-RPC) hervorgegangen. Über diesen Standard ist der Aufruf von entfernten Methoden und der Datenaustausch zwischen Applikationen auf Basis von XML möglich. SOAP bildet die Grundlage der so genannten Web Services.
Token	Ein Zeichen oder eine Zeichenkombination, die in einer Sprache als Sinnzusammenhang zu verstehen ist. In XML ist das beispielsweise die Zeichenkette »xml« oder in Java ein Operator oder ein Schlüsselwort.
UDDI	Universal Description, Discovery and Integration bezeichnet einen auf SOAP basierenden Verzeichnisdienst im Umfeld von Web Services.

Tabelle A.3: Verschiedene Abkürzungen und Fachausdrücke aus dem AJAX-Umfeld (Forts.)

Begriff	Beschreibung
Unicode	Unicode bezeichnet einen internationalen Standard zur Kodierung von Zeichen. Mit Unicode wird versucht, das Problem der verschiedenen inkompatiblen Kodierungen in unterschiedlichen Ländern zu beseitigen. Dazu verwendet der Coderaum von Unicode zwei oder mehr Byte, die zudem seit der Version 2.0 um Bereiche (so genannte Planes) erweitert werden können.
URL	Ein Uniform Resource Locator ist ein vereinheitlichtes Adressierungsschema, das vor allem im Internet zur Angabe einer Ressource verwendet wird.
UTF	Das UCS Transformation Format ist eine Kodierung, um Unicode kompatibel zu ASCII-Code speichern zu können. UTF-16 verwendet dabei zwei Byte zur Kodierung von Zeichen und UTF-8 ein Byte zur Kodierung. Bei UTF-8 kann es vorkommen, das mehr als ein Byte zur Kodierung eines Zeichens verwendet wird (wenn ein Zeichen mit hohem Unicode-Wert gespeichert wird).
XHTML	Die eXtensible Hyper Text Markup Language bezeichnet XML-konformes HTML, das auf Basis von XML reimplementiert wurde.
XML Schemata	Mit einem XML Schema beschreibt man allgemein ein Regelwerk, mit dem die Struktur, der Inhalt und die Bedeutung von XML-Dokumenten festgelegt werden können. Es gibt mehrere Sprachen, um ein XML Schema aufzubauen (etwa XSD, RELAX NG, Schematron oder Examplotron). Ein XML Schema ist eine XML-konforme Alternative zu einer DTD, die zudem bedeutend mehr Möglichkeiten als eine DTD bietet.
XPath	XPath ist eine eigenständige Spezifikation für die Navigation in XML-Dokumenten. Sie ist die Grundlage für viele weitere Technologien im XML-Umfeld. XPath ist eine Art Abfragesprache für XML und wird oft als SQL (Standard[1] Query Language) für XML bezeichnet.
XSD	Die XML Schema Definition Language dient als Strukturbeschreibungssprache (Schemata) für XML-Dokumente.
XSL	Die eXtensible Stylesheet Language ist eine Sprache zur Erstellung von Formatvorlagen. Die XSL-Spezifikation besteht aus zwei Teilen. Die XSL-Transformation (XSLT) ermöglicht die Überführung von XML-Dokumenten in ein anderes Format. Das kann eine andere Ausprägung von XML oder auch ein anderes Format wie HTML sein. Mit den (bekannteren) XSL Formating Objects (XSL-FO) hat man die Möglichkeit zur Definition der Darstellung von Informationen. XSL-FO erlaubt sehr detaillierte Angaben zum Layout eines Dokuments (ähnlich wie CSS, die im Web hauptsächlich eingesetzt werden) und eine genau angepasste Darstellung für die unterschiedlichsten Medien.
XUL	Die XML UI Language stammt aus dem Umfeld von Mozilla und stellt eine Beschreibung der Benutzerschnittstelle des Browsers selbst dar.

Tabelle A.3: Verschiedene Abkürzungen und Fachausdrücke aus dem AJAX-Umfeld (Forts.)

1 *Bis vor wenigen Jahre stand SQL für Structured Query Language, weshalb man diese Erklärung noch häufig in der Literatur findet.*

Stichwortverzeichnis