PEARSON Education

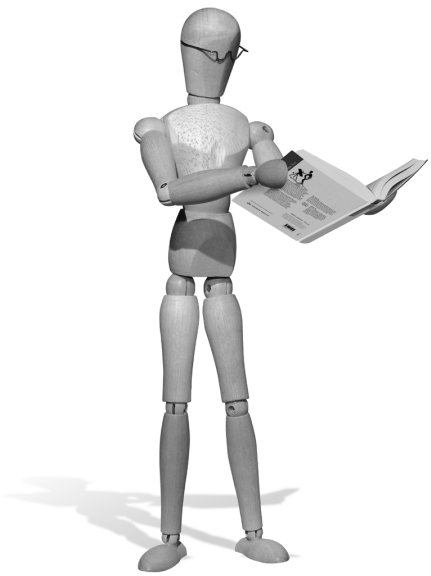
ADDISON-WESLEY

open source library

Ralph Steyer

AJAX mit PHP

Beschleunigte Webapplikationen für das Web 2



ADDISON-WESLEY

An imprint of Pearson Education

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

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. Es gibt überdies verschiedene historische Schreibweisen, die aber alle im Wesentlichen das Gleiche meinen. Microsoft schreibt seine Variante **Dynamic HTML** (mit großem D), Netscape dagegen ursprünglich **dynamic HTML** (mit kleinem d).

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** dennoch 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.

Mittlerweile hat sich flächendeckend die Schreibweise DHTML durchgesetzt. Und 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, obgleich 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 Browsermodelle, 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 Browserhersteller durch die reine Marktdominanz von Microsoft mittlerweile eine gewisse Unterstützung für `all` bereitstellen, ist diese meist unvollständig 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. 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 realisieren zu können, 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 teilweise nur in bestimmten Browsern, aber teilweise 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 dem Namen des Objektfelds samt Index möglich – eventuell auch über mehrere Ebenen und 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 im Zusammenhang mit Schleifen. Aber sie haben 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 oder zumindest indiziert sein. In diesem Fall ist der Zugang über `window.document` und dann dem Namen des Elements möglich – eventuell auch wieder über mehrere Ebenen und mit verkürztem Zugang unter Weglassung von `window`. Wenn eine Grafik zum Beispiel wie folgt in der Webseite referenziert wird:

Listing 7.2: Ein Grafik-Element 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 zustand

```
document.zustand
```

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>
```

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

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
```

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 zu der Schreibweise der Methode `getElementById()` achten. Es heisst `getElement`**`s`**`ByName()` (Mehrzahl) und nicht `getElementByName()` (Einzahl).

7.2.3 Zugriff über eine ID

Wenn Sie einem Element in einer Webseite eine Style Sheet-Klasse zuweisen, erfolgt das über das Attribut `class`. Über JavaScript kann man diese Klasse ansprechen und auch austauschen. Zumindest im Internet Explorer, aber auch einigen anderen neueren Browsern. Dazu ist das zusätzlich notwendige 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 document.all

```
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"
```

2 *Dies haben wir bisher schon oft gemacht.*

7.2.4 Zugriff über den Typ des HTML-Elements

Das DOM-Konzept sieht auch vor, dass Sie auf Elemente einer Webseite über den 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. Dieser ergibt sich sequenziell. Beispiele:

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 DOMs, sondern des allgemeinen Kern-DOMs für beliebige XML- bzw. SGML-basierte Dokumente. Sie gehört also eigentlich zu dem Objekttyp `nodes` und liefert eine Knotenmenge im Sinn von XML, mit der Sie operieren.

Die verschiedenen Formen des Zugriffs 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. Das können Sie nutzen, um bei Datennachforderungen per AJAX dem Anwender dies deutlich zu machen (sofern es von der Benutzerführung 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, mit der Anweisung `new Image();` in JavaScript von Hand 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 zu dem 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`).
`height`	Die korrespondierende Eigenschaft zu dem 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 Bildes.
`hspace`	Der horizontale Abstand zwischen einer Grafik und ihren benachbarten Elementen. Korrespondierende Eigenschaft zu dem 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 zu dem gleichnamigen HTML-Parameter.
`name`	Der Name der Grafik. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter.
`src`	Der URL der Grafikdatei. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter.
`vspace`	Der vertikale Abstand zwischen einer Grafik und ihren benachbarten Elementen. Korrespondierende Eigenschaft zu dem gleichnamigen HTML-Parameter. Wenn die Angabe im HTML-Tag nicht gesetzt ist, hat `vspace` den Wert 0
`width`	Die Breite der Grafik. Korrespondierende Eigenschaft zu dem 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 Bildes.

Tabelle 7.1: Die Eigenschaften eines Bildobjekts

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 ein `<img>`-Tag dafür an. Auch wenn dieses 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 `nodes`.

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 Bildes 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 mit 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:

Listing 7.13: Die Webseite mit dem Nachladen der Informationen

```
01 <html>
02 <script language="JavaScript" src="image1.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 <img src="leer.gif" height="50"
10   style="position:absolute;top:100px;left:500px;">
11 </body>
12 </html>
```

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 Sheets 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.

Zugriff über Objektfelder

Auf JavaScript-Seite 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 so lange einen Selbstaufruf,

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.php',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    nachladen();
31  }
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 gegen eine 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.

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 auf Grund 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 Ähnliches 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 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 gute Lösung.

Zugriff über Namen

Zum Umformulieren des Beispiels zur Fortschrittsanzeige des Ladevorgangs von eben 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 mit 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 einer Webseite zu platzieren.

3 *Dazu siehe Seite 187.*

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:

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 im `<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 Standardwert 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 Standardklasse `stil1` positioniert das Element, dem die Klasse zugewiesen wird, und hisst dabei die ostfriesische Kriegsflagge[4]. Damit ist der Text in dem -Container solange unsichtbar, wie die Klasse `stil1` zugewiesen ist. Die Klasse `stil2` wird den Text sichtbar machen.

In der externen JavaScript-Datei ist wieder nur folgende Stelle 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    nachladen();
31  }
32  P1.className="stil1";
33  handleResponse();
34 }
```

In der Funktion `sndReq()` wird in Abhängigkeit von dem 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 wieder der Aufruf der Funktion `handleResponse()`.

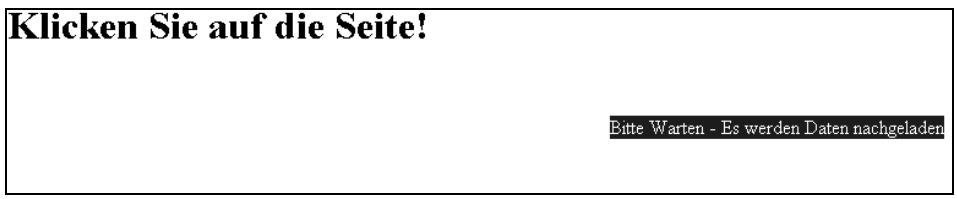


Abbildung 7.2: Die Stilklasse wird zur Laufzeit dynamisch ausgetauscht.

Achtung

Diese Art des Zugriffs funktioniert im Grunde nur im Internet Explorer. Sie macht als allgemeine Lösung keinen Sinn. Dazu sollten Sie im allgemeinen Fall `getElementById()` oder `getElementsByName()` verwenden.

4 *Weißer Adler auf weißem Grund ;-). Hier ist es weiße Schrift auf weißem Hintergrund.*

Tipp

Die Effekte, die Sie mit dynamischem Verändern der Stilklasse erreichen können, sind vielfältig. Sie können zum Beispiel Daten bereits im Offscreenbereich[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 der 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 und auch die Vorhaltung im Offscreenbereich kann man hervorragend dazu einsetzen, um das Nachladen von Informationen deutlich zu machen oder aber unabhängig vom physikalischen Laden von Informationen sehr leistungsfähig Daten in einer Webseite anzuzeigen. Das schließt nicht aus, dass zusätzlich mit asynchroner Datennachforderung gearbeitet wird. Es ist nur ein zusätzliches Mittel, um die jeweils beste Möglichkeit zu nutzen.

Die nachfolgende CSS-Datei spezifiziert eine Klasse mit positioniertem, aber unsichtbarem Inhalt:

Listing 7.20: Eine CSS-Datei mit einer Klasse

```
01 .still {
02   color : white;
03   background-color : blue;
04   visibility : hidden;
05   position : absolute;
```

5 *Der Offscreenbereich 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 Offscreenbereich verwenden wie den normalen sichtbaren Bereich der Webseite.*

```
06   top : 100px;
07   left : 500px;
08   }
```

Die HTML-Datei ist mittlerweile gewohnte Hausmannskost:

Listing 7.21: Der -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    nachladen();
31  }
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 im Schwerpunkt 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 zwi-

schen `innerHTML` und `innerText` ist recht gering. Über `innerHTML` wie 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 bei `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-Objektmethoden 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 und 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 weiteren Element-Kindknoten eines Knotens ansprechen. Dazu nutzen Sie die spezifischen Eigenschaften und Methoden eines `node`-Objekts, die insbesondere über XPath-Syntax genauer spezifiziert werden. Die Details führen ohne XML zu weit, aber wir werden im folgenden Kapitel zu XML die 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 in 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 wir XML-Daten verwerten (im nächsten Kapitel). Und dann kommt auch das `node`-Objekt zum Einsatz.

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 wurde, und die Webseite ohne Neuladen verändern. Damit lassen sich Effekte zur Rückmeldung an den Besucher bei AJAX-Datentransfers realisieren und auch die neuen Informationen in die Webseite einfügen.