Workshop JavaScript ISBN 3-8273-1718-6

3 Das Objektmodell

Von Christine Kühnel, kuehnel@screenexa.net

3.1 Grundlagen

JavaScript-Programme laufen nicht im luftleeren Raum, sie brauchen eine Umgebung. Da sie nicht wie viele andere Programme übersetzt werden, muss diese Umgebung die Syntax der Sprache beherrschen, um die Skripte[1] interpretieren zu können.

3.1.1 Die Sprache in ihrer Umgebung

Wir beschäftigen uns hier mit JavaScript im Bereich Web-Publishing, deswegen ist unsere Umgebung der *Web-Browser*[2]. Der Browser übernimmt also die Rolle des Interpreters für unsere Skripte.

Das Wichtige und Interessante daran ist, der Browser interpretiert nicht nur, er bietet auch Objekte an, auf die man zugreifen kann. Sie werden beim Schreiben Ihrer eigenen Skripte ständig mit diesen Objekten zu tun haben, ohne sie geht (fast) nichts. Sie sind einigen wenigen Objekten sogar schon hier in diesem Buch begegnet, auch wenn Ihnen das vielleicht gar nicht bewusst geworden ist. Sie erinnern sich gewiss an diese Anweisung:

```
alert("Hallo Du!");
```

Objektmodell

Schon dieses einfache Meldungsfenster setzt voraus, dass der Browser überhaupt in der Lage ist, es sichtbar zu machen. Er bietet im `window`-Objekt die Methode `alert()` an, die genau das realisiert, ein guter Grund also, dem *Objektmodell* (OM) – so nennt man die Struktur dieser Objekte – erhöhte Aufmerksamkeit zu widmen.

Aber unternehmen wir zuvor einen kurzen Abstecher für diejenigen unter Ihnen, denen der Denkansatz der objektorientierten Programmierung noch nicht so vertraut ist, alle anderen dürfen schon mal weiter blättern.

1. Ich werde von jetzt an den Begriff »Skript« verwenden. Sollten andere Autoren »Programm«, »JavaScript-Programm« oder »Script« sagen, lassen Sie sich nicht verwirren, es ist dasselbe.
2. Der Vollständigkeit halber sei hier erwähnt, dass es im Bereich Web-Publishing auch serverseitiges JavaScript gibt. Das ist jedoch nicht Thema dieses Workshops.

3.1.2 Denken in Objekten

Gebrauchen Sie das Wort *Objekt* ruhig im Wortsinn, denn genau darin besteht die Idee. Man benutzt eine aus dem täglichen Leben geläufige Art, Dinge zu betrachten. Wir veranschaulichen uns das Prinzip am besten an einem ganz einfachen Beispiel.

Sie halten gerade ein Buch in den Händen. Dieses Buch ist ein Objekt. Einverstanden? Wenn Sie zunächst ein Buch überhaupt und dann speziell dieses Buch beschreiben sollten, wie würden Sie anfangen? Vielleicht so?

1. Ein Buch besteht aus einem Einband und einzelnen Seiten. Dieses Buch hier ist ein Taschenbuch, es hat einen vorwiegend gelben Einband, ...

Sie analysieren das Objekt, beschreiben seine Eigenschaften und die seiner Bestandteile. Nichts anderes tut die objektorientierte Programmierung. Zur Darstellung dieser Hierarchie bedient man sich einer speziellen Syntax. Man beginnt mit dem Objekt auf der obersten Ebene, fügt dessen *Eigenschaften* durch einen Punkt getrennt an. Ist diese Eigenschaft (engl. *Property*) selbst wieder ein Objekt, hat also eigene Eigenschaften, dann fährt man ganz einfach nach diesem Prinzip fort.

In einem Objekt `Buch` könnte man also z. B. Folgendes finden:

```
Buch.Einband
Buch.Seiten
Buch.Titel
Buch.Verlag
```

Natürlich sind das längst nicht alle Eigenschaften, aber im Alltag erwähnen Sie sicher auch nicht immer jedes einzelne Detail, sondern greifen die Eigenschaften heraus, die gerade interessant und wichtig sind. Das tun wir hier auch, und das werden Sie auch in Ihren Skripten so machen.

Wir erkennen außerdem, einige Eigenschaften sind wiederum so komplex, dass es mit einer einfachen Wert-Zuweisung wohl nicht getan ist, der `Einband` zum Beispiel ruft geradezu nach einem eigenen Objekt. Vielleicht verwirrt es Sie, dass Objekte Eigenschaften anderer Objekte sein können? Dann versuchen Sie es zur Verdeutlichung einmal mit folgender gar grausigen Formulierung:

> Ein `Buch` hat die *Eigenschaft*, einen `Einband` zu haben. :-)

Die folgende Syntax dafür ergibt sich aus den bisherigen Erklärungen. Ihnen ist sicher sofort klar, warum man zum Beispiel das hier schreibt:

```
Buch.Einband.Farbe
Buch.Einband.Art
Buch.Seiten.Anzahl
```

Bisher haben wir das Objekt `Buch` ganz allgemein angesehen. Die erwähnten Eigenschaften existieren für jedes Buch; unterschiedlich sind dagegen die *Werte* dieser Eigenschaften.

Betrachten wir dazu neben diesem Buch, in dem Sie gerade jetzt lesen, noch ein zweites. Der noch nicht zu Ende gelesene Roman auf Ihrem Nachtschrank ist ganz sicher auch ein (Objekt) `Buch`, hat ebenfalls (die Eigenschaften) `Einband`, `Seiten` und `Titel`. Wie unterscheiden sich aber die beiden Bücher, und wie unterscheidet man sie syntaktisch? Tabelle 3.1 verdeutlicht Zusammenhänge und Unterschiede. Dazu nennen wir die beiden Bücher, oder besser, die beiden Objekte `diesesBuch` und `meinRoman`.

Eigenschaft	Wert	Eigenschaft	Wert*	Typ
`diesesBuch`		`meinRoman`		
`.Einband`		`.Einband`		Object
`.Einband.Art`	'Taschenbuch'	`.Einband.Art`		String
`.Einband.Farbe`	'gelb'	`.Einband.Farbe`		String
`.Seiten`	`300`	`.Seiten`		Integer
`.Titel`	'JavaScript Work-Shop'	`.Titel`		String
`.Verlag`	'Addison-Wesley'	`.Verlag`		String

*Die Werte für meinRoman setzen Sie bitte selbst ein. Ich weiß nicht, was Sie gerade lesen.

Tab. 3.1: Beispiel-Objekte, Eigenschaften, Werte

Die (Werte der) Eigenschaften eines Buches, die wir gerade eben angesehen haben, können Sie nicht verändern, die Farbe des Einbandes wird immer so sein, wie sie jetzt ist (von eventuellen Gebrauchsspuren durch intensive Benutzung wollen wir hier einmal absehen). Das wird Ihnen auch künftig begegnen, nur lesbare Eigenschaften und solche, deren Werte überschrieben werden können.

Auch bei einem Buch ist das so. Das Buch, das Sie in den Händen halten, wird sehr wahrscheinlich aufgeschlagen sein, sonst könnten Sie das hier ja nicht lesen. Es hat also auch eine Eigenschaft

```
diesesBuch.offen
```

Im Moment ist der Wert wahr (`true`). Aber irgendwann werden Sie das Gelesene erst einmal verdauen müssen und das Buch zuschlagen., der Wert wird sich dabei ändern, anschließend falsch (`false`) sein. Eine Aktionen, das Zuschlagen, sorgt dafür. Diese Aktion, sie ahnen es, ist eine *Methode* des Objektes `diesesBuch`. Methoden ruft man so auf:

```
diesesBuch.zuschlagen()
```

Dabei lassen sich in Klammern Parameter angeben, die der Aktion erforderliche Informationen liefern. So etwas kennen Sie schon von Funktionen her. Beim Zuschlagen eines Buches ist es sicher nicht notwendig, näher zu spezifizieren, aber bereits beim Gegenstück, dem Aufschlagen, kann das durchaus nützlich sein. Mit

```
diesesBuch.aufschlagen(140)
```

könnte man der Methode sagen, dass das Buch auf Seite 140 aufgeschlagen werden soll.

An der Stelle beenden wir unseren kleinen Ausflug. Wie man in JavaScript Objekte definiert, ihnen Eigenschaften und Methoden zuordnet, damit befassen wir uns im folgenden Kapitel genauer. Behalten Sie also das Buch in den Händen und das Objekt `diesesBuch` im Gedächtnis.

Kehren wir zunächst zurück zum Objektmodell des Browsers und den darin enthaltenen »vorgefertigten« Objekten.

Objekte in der Umgebung, oder einfacher – das OM des Browsers

Schauen wir uns so ein Objektmodell aus der Nähe an. Abbildung 3.1 zeigt stark vereinfacht die Struktur, die wir bei allen bekannten Browsern wiederfinden.[3]

Wir können hier nicht auf alle Objekte eingehen. Das ist aber auch gar nicht erforderlich. Haben Sie einmal die Grundstruktur erkannt, herausgefunden, wie man damit umgeht, ist alles andere nicht mehr wirklich schwer. Werfen wir also nur einen Blick auf einige ausgewählte Objekte.

window

Es muss nicht nur ein window geben

Sie erkennen an der schematischen Darstellung, das Objekt `window` spielt eine zentrale Rolle. Um besser zu verstehen, nehmen Sie es ruhig wörtlich. `window` repräsentiert tatsächlich das geöffnete Fenster Ihres Browsers. Und damit ist sofort klar, es muss nicht nur ein `window`-Objekt geben. Sie könnten ja mehrere Browserfenster gleichzeitig verwenden. Alles, was in (und später auch mit) dem Fenster geschieht, spielt sich innerhalb dieses Objektes ab.

Für die Praxis merken wir uns noch, dass von einem Skript aus das Fenster, in dem es läuft, immer über `window` zu erreichen ist. Damit haben wir den Startpunkt, um alle nachgeordneten Objekte anzusprechen, um zum Beispiel auf das Dokument in diesem Fenster zuzugreifen: `window.document`

In vielen Fällen kann man auf die Angabe `window` verzichten, weil es vom Browser standardmäßig eingesetzt wird. Das heißt, `window.document` und `document` sagen dasselbe.

3. Korrekt müsste es heißen: fast bei allen bekannten Browsern. images gibt es erst ab JavaScript 1.1, also NN3, wurde aber hier ganz bewusst dennoch aufgenommen.

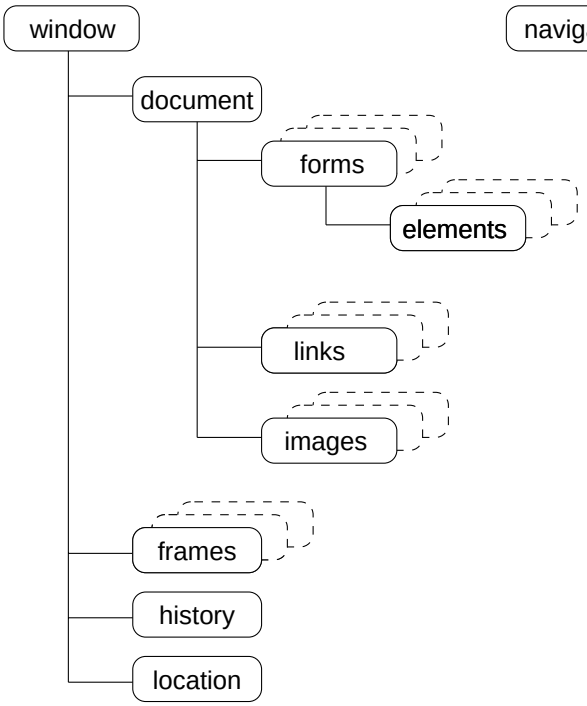

Abbildung 3.1: Objektmodell stark vereinfacht

Ich werde jedoch jetzt und hier davon zunächst wenig Gebrauch machen, und empfehle das für den Anfang auch Ihnen. Sie schreiben bewusster, wenn Sie die vollständige Angabe wählen. Auch später ist es durchaus oft sinnvoll, für sich selbst die lange Form zu wählen. Spätestens beim Durcharbeiten von Kapitel 7, wenn es um Referenzierungen über Frame- und Fenstergrenzen hinweg geht, werden Sie mir vermutlich zustimmen.

Ein Frame ist auch ein window

Gerade haben Sie verstanden, was das `window`-Objekt ist, und schon muss ich das eben Gesagte etwas einschränken. Dieses Objekt begegnet uns nicht nur in Form des Browserfensters, sondern auch noch in einer anderen, nämlich als Frame. Sie haben in Abbildung 3.1 zum Beispiel `frames[]` gesehen. Das ist nichts anderes als ein Array, bestehend aus `window`-Objekten, für jeden Frame eins, das nicht unabhängig existiert, sondern fest zum Hauptfenster gehört.

Um uns an den Umgang mit den browsereigenen Objekten heranzutasten, wählen wir zunächst die Eigenschaft `location`, das ist eine vergleichsweise übersichtliche Eigenschaft.

location

`location` ist selbst ein Objekt, nämlich der Repräsentant des aktuellen URLs im Fenster. Von dessen Eigenschaften soll uns hier zunächst nur `href` interessieren; ihr Wert gibt uns den aktuellen URL an.

Probieren Sie es aus! Schreiben Sie dazu dieses kleine Skript, das die Information in einem Meldungs-Fenster zeigt:

```
<script language="JavaScript" type="text/javascript">
alert(window.location.href);
</script>
```

Auf noch eine Eigenschaft von `window` wollen wir an dieser Stelle kurz einen Blick werfen, auf das wohl komplexeste Objekt, `document`.

document

`document` beinhaltet all das, was das gerade aktuelle Dokument im Fenster betrifft. Es ist damit durchaus folgerichtig, dass alles, was Sie in einer HTML-Datei spezifizieren, unterhalb dieses Objektes wiederzufinden ist. Das sind. z. B. alle Links der Seite, enthaltene Formulare und Grafiken, Farbangaben für Text und Hintergrund etwa, und noch sehr vieles mehr.

An zwei ausgewählten Objekten werden wir nun den Umgang etwas genauer betrachten. Machen Sie sich keine Sorgen, wenn Sie am Ende dieses Kapitels nicht absolut fit sind. Es soll Ihnen die Grundlagen vermitteln. Gelegenheit zum intensiveren Üben finden Sie später in diesem Buch.

Beginnen wir mit einem oft gebrauchten Objekt, dem Form-Objekt, denn Formulare in HTML-Dateien sind Ihnen vertaut.

Form-Objekt

Eine Datei wie die folgende ist Ihnen schon oft begegnet, ein einfaches Formular mit zwei Textfeldern zum Ausfüllen, drei Radio-Buttons zur Auswahl und einem Button, der am Ende für Action sorgt:

```
<html>
<head>
<title>Angaben zur Person</title>
</head>
<body>
<h1>Angaben zur Person</h1>
<form name="Person">
Vorname:<br>
<input type="text" name="Vorname" value="" size=20><br>
Nachname:<br>
<input type="text" name="Nachname" value="" size=20><br>
<input type="radio" name="Geschlecht"> weiblich
```

```
<input type="radio" name="Geschlecht"> männlich
<input type="radio" name="Geschlecht" checked="checked"> keine
Angabe<br>
<input type="button" value="Anzeigen" name="Anzeige">
</form>
</body>
</html>
```

Der Button »Anzeigen« ist im Moment noch vollkommen funktionslos. Seine Aufgabe soll es später sein, uns die eingegebenen Werte per `alert()` zu zeigen.

Um die Werte zu ermitteln, müssen die einzelnen Elemente angesprochen werden. Dafür gibt es verschiedene Möglichkeiten. Wir entscheiden uns hier zunächst für die, wie ich finde, anschaulichste. Vorsorglich wurde allen Elementen über `name`-Attribute Namen[4] gegeben, über die sie zu erreichen sind:

```
window.document.Person.Vorname
window.document.Person.Nachname window.document.Person.Anzeigen
```

Es fällt sofort auf, der Name »`Geschlecht`« ist im Beispiel nicht eindeutig gewählt, er steht für alle drei Radio-Buttons. Wie also den einzelnen Radio-Button erreichen? Dafür bietet der Browser eine andere Möglichkeit. Für die drei Buttons existiert ein Array `Geschlecht[]` mit drei Elementen[5]:

```
window.document.Person.Geschlecht[0]
window.document.Person.Geschlecht[1]
window.document.Person.Geschlecht[2]
```

Damit fehlen zur Bewältigung der Aufgabe, die Werte zu ermitteln, nur noch Eigenschaften, in denen sich diese Werte finden.

- Für die Textfelder ist das `value` mit dem enthaltenen Text als String
- Für die Radiobuttons gibt es `checked` als Boolean

Mit den nun vorhandenen Kenntnissen ist es nicht mehr schwer, eine Funktion vorzubreiten, die die gewünschte Ausgabe generiert. In einem Script-Block im `head` sieht das so aus:

```
<script language="JavaScript" type="text/javascript">
<!--
function zeigen()
{
   var Anrede = '';
   if (window.document.Person.Geschlecht[0].checked)
      Anrede = 'Frau ';
```

4. Der HTML-Standard empfiehlt, statt name besser id zu verwenden, aber in der Praxis können viele der vorhandenen Browser leider nicht damit umgehen.
5. Lassen Sie bitte NN2 nicht an diese Arrays, der schafft es oft nicht, die Indizes der Reihe nach zu vergeben.

```
   else
      if (window.document.Person.Geschlecht[1].checked)
         Anrede = 'Herr ';
   var Name = window.document.Person.Vorname.value +
              ' ' +
              window.document.Person.Nachname.value;
   var Nachricht = 'Sie sind\n' + Anrede + ' ' + Name;
   alert(Nachricht);
}
// -->
</script>
```

Was noch fehlt, ist der Aufruf dieser Funktion. Die Anzeige soll bei Klick auf den Button »Anzeigen« erfolgen. Dafür benutzen wir den Event-Handler `onclick`. Darauf, was ein Event-Handler ist, kommen wir gleich zurück. Zunächst benutzen wir diesen hier, ohne weiter darauf einzugehen. Dazu muss im HTML-Code von oben lediglich dem `input`-Element ein entsprechendes Attribut hinzugefügt werden:

```
<input type="button" value="Anzeigen" name="Anzeige"
       onclick="zeigen()">
```

Eine andere Möglichkeit, dieselben Objekte anzusprechen, soll hier noch erwähnt werden. Gerade ist uns ein Array begegnet. Der Browser bietet für eine Reihe von Objekten Arrays an. Zum Beispiel könnte man die Werte der Textfelder so ermitteln:

```
var x = window.document.forms[0].elements[0].value;
var y = window.document.forms[0].elements[1].value;
```

Das Prinzip: In `document` gibt es ein Array `forms[]` mit allen Formularen, im Form-Objekt wiederum eins mit allen Elementen des Formulars, nämlich `elements[]`, jeweils in der Reihenfolge ihres Auftretens.

Verlassen wir hier das Form-Objekt, Sie werden an anderer Stelle Gelegenheit haben, sich eingehender damit zu beschäftigen. Hier sollte Ihnen zunächst nur deutlich werden, wie sich Ihnen vertraute HTML-Elemente in der Objekthierarchie des Browsers wiederfinden. Deswegen werfen wir nur noch kurz einen Blick auf ein anderes immer wieder anzutreffendes Objekt, das Image-Objekt.

Image-Objekt

Wir benutzen wieder eine ganz einfache HTML-Datei:

```
<html>
<head><title>Image-Objekt</title>
</head>
<body>
```

```
<h1>Bilder eben</h1>
<p>ein Bild:
<img src="Apfel.gif" name="erstesBild"
     width="100" height="100" alt="ein Bild">
</p>
<p>und noch ein Bild:
<img src="Birne.gif" name="zweitesBild"
     width="100" height="100" alt="noch ein Bild">
</p>
</body>
</html>
```

Mit Hilfe von `alert()` wollen wir diesmal sehen, welche Grafik-Datei im zweiten `img`-Element benutzt wird.

Unser Beispiel zeigt es, auch `img`-Elemente treten mehrfach auf. Sie kennen das inzwischen aus dem Formular-Beispiel, es existieren auch hier mehrere Image-Objekte in einem Array

```
window.document.images
```

Ansprechen kann man sie deswegen ganz analog:

```
window.document.images[0]
window.document.images[1]
```

Da das `name`-Attribut verwendet wurde, geht auch:

```
window.document.erstesBild
window.document.zweitesBild
```

An dieser Stelle sei eine weitere Möglichkeit erwähnt. Die `name`-Eigenschaft dient dem Browser dazu, Arrays als assoziative Arrays[6] anzubieten:

```
window.document.images['erstesBild']
window.document.images['erstesBild']
```

Ein anderer Begriff wird Ihnen ganz gewiss noch begegnen, wenn es um in Arrays »gesammelte« Objekte geht: *Collection*[7]

Kehren wir zu unserer Aufgabe zurück. Dazu müssen wir jetzt nur noch wissen, dass die Eigenschaft, in der der URL der Grafikdatei zu finden ist, `src` heißt, dann haben wir alle Voraussetzungen. Wir probieren bei der Gelegenheit gleich alle drei Möglichkeiten, unser Image-Objekt anzusprechen, die Ergebnisse sollten identisch sein:

6. Sie werden später sehen, es muss nicht name sein, vor allem im DHTML-Teil wird Ihnen id in dieser Rolle begegnen.
7. Auch wenn das Wort »collection« sich vergleichsweise problemlos in unsere Sprache übersetzen lässt, tun wir sicher gut daran, es auch weiterhin beim englischen Original zu belassen.

```
<html>
<head><title>Image-Objekt</title>
</head>
<body>
<h1>Bilder eben</h1>
<p>ein Bild:
<img src="Apfel.gif" name="erstesBild"
     width="100" height="100" alt="ein Bild">
</p>
<p>und noch ein Bild:
<img src="Birne.gif" name="zweitesBild"
     width="100" height="100" alt="noch ein Bild">
</p>
<script language="JavaScript" type="text/javascript">
<!--
alert(window.document.images[1].src)
alert(window.document.zweitesBild.src)
alert(window.document.images['zweitesBild'].src)
// -->
</script>
</body>
</html>
```

Welche Form Sie in der Praxis benutzen, das hängt zum einen sicher vom konkreten Problem ab und zum anderen auch von Ihrer persönlichen Vorliebe. Ich verwende häufig die dritte Form, die hat einen oft sehr nützlichen Vorteil, Funktionen und Methoden kann man mitteilen, welches konkrete Objekt angesprochen werden soll, indem man Ihnen als Parameter einen String übergibt, der den Namen enthält. Sie werden sicher in den Beispielen und Übungen dieses Buches noch deutlicher sehen, was ich meine. Denken Sie bei der Wahl auch daran, dass die Verwendung numerischer Indizes wenig änderungsfreundlich ist. Bei jeder Änderung im HTML-Code müssen Sie nachschauen, ob die Indizes noch stimmen.

Eigenschaften definieren

Sie vermuten richtig; es ist kein Zufall, dass es eine Eigenschaft `src` gibt, die den URL der Grafikdatei enthält, und dass der Wert dieser Eigenschaft durch das `src`-Attribut im `img`-Tag festgelegt wurde.

Attribute in HTML-Tags legen Eigenschaften von HTML-Elementen fest. Findet sich für so ein Element ein Objekt in Objektbaum, dann hat es im Allgemeinen Eigenschaften, die den Attributen entsprechen, deren Werte man wie gerade gesehen über Attribute festlegen kann.

width und height zum Beispiel sollten im Image-Objekt also auch zu finden sein. Probieren Sie es aus!

```
alert(window.document.images['zweitesBild'].width);
alert(window.document.images['zweitesBild'].height);
```

Event-Handler

Sie kennen das; wenn Sie Links mit der Maus berühren, verändern sich Grafiken. Weil es das Paradebeispiel für den Einsatz von *Event-Handlern* ist, werde auch ich keine Ausnahme machen und genau diesen Effekt benutzen, um Ihnen *Ereignisse (engl. events)* etwas näher zu bringen.

Das Eintreten bestimmter Ereignisse ist erkennbar

Die Veränderung ist natürlich nicht wirklich eine Veränderung, sondern lediglich die Zuweisung eines anderen Wertes (URL der angezeigten Grafik) zur src-Eigenschaft des Image-Objektes. Der Trick, der eigentlich gar keiner ist, besteht lediglich darin, dass der neue Wert in einem ganz bestimmten Moment zugewiesen wird. Man macht sich zunutze, dass es der Browser ermöglicht, das Eintreten bestimmter Ereignisse zu erkennen. Die bekanntesten Ereignisse sind das Berühren eines Links mit der Maus bzw. das Verlassen des Links. Ereignisse sind an ein Objekt gebunden, hier ist es das Link-Objekt. Bei dem wollen wir vorerst auch bleiben. Das große Thema »Event-Handling« und seine Vielfalt werden Ihnen später noch mehrfach in diesem Workshop begegnen.

Aber zurück zu unserem Beispiel. Die Ereignisse »Maus berührt Link« und analog »Maus verlässt Link« zeigen so genannte Event-Handler an, die das Link-Objekt besitzt. Für den dritten Link innerhalb des Dokuments tun das zum Beispiel

```
window.document.links[2].onmouseover
window.document.links[2].onmouseout
```

Sie sehen bei der Gelegenheit, die Link-Objekte finden sich ebenfalls als Elemente eines Arrays, nämlich document.links

Häufiger verwendet wird eine andere Schreibweise. Im Formular-Beispiel haben wir sie für onclick auch schon eingesetzt. So, wie sich Eigenschaften von Objekten sowohl im Script als auch als Attribut im HTML-Element festlegen lassen, ist auch bei Event-Handlern beides möglich.

Damit haben wir schon alles, was für die einfachste Form, so einen Wechsel zu initiieren, erforderlich ist. Und so könnte er aussehen.

```
<html><head>
<title>Events</title>
</head>
<body>
<a href="nachirgendwo.html"
onmouseover="window.document.tauschmich.src='aus.gif'"
onmouseout="window.document.tauschmich.src='ein.gif'"><img
```

```
src="grafik1.gif" name="tauschmich" width="74" height="23"
alt="mal so, mal so"></a><
</body>
</html>
```

Dass das so nicht die eleganteste Lösung für Bildwechsel ist, man in der Praxis noch auf ein paar Dinge am Rande achten sollte, lassen wir an dieser Stelle einmal außen vor. Das Prinzip ändert sich dadurch nicht, es wird Ihnen bei allen Bildwechseln immer wieder begegnen, so gut, vielleicht sogar raffiniert sie auch gemacht sein mögen.

Eigene Variablen ordnen sich in die Objekthierarchie des Browsers ein

Bisher haben wir uns Objekte angesehen, die der Browser mitbringt. In Kapitel 2 haben Sie Variablen definiert. Die existieren nicht etwa neben den Browserobjekten, sondern finden sich in der Objekthierarchie wieder.

Erinnern Sie sich bitte an das vorangegangene Kapitel. Da ging es um Konten.

```
<script language="JavaScript" type="text/javascript">
<!-
var kontonummer  = 0;
var kontoinhaber = "";
var kontostand   = 0.0;
// -->
</script>
```

Probieren Sie aus, bauen Sie das hier kurz ein:

Eigene Variablen finden sich im Objektbaum

```
alert(window.kontonummer);
alert(window.kontoinhaber);
alert(window.kontostand);
```

Das Ergebnis ist deutlicher als es jede Erklärung sein könnte. Ihre Variablen sind Eigenschaften von `window`.

Das gilt so natürlich nur für Variablen, die auch im ganzen Fenster bekannt sind. Erinnern Sie sich bitte an Kapitel 2 und daran, welche Variablen wo sichtbar sind.

Nur was der Browser kennt, kann man verwenden

Bis hierher ist Ihnen ganz bestimmt klar geworden, dass man auf die Objekte angewiesen ist, die der Browser anbietet. Es muss ein `window` geben, ein `document`, `document` muss ein Array `forms[]` enthalten usw.

Skripte, die Sie für das Web schreiben, laufen nicht in einer bekannten oder bestimmbaren Umgebung, sondern auf dem Client. Im Voraus wissen Sie nicht, welchen Browser der Besucher der Seiten verwendet, kennen also die Umgebung nicht, in der die Skripts angestoßen werden. Nun ist die Existenz der Objekte aber Voraussetzung, es stellt sich also eine wichtige Frage:

3.1.3 Sind die Objektmodelle der Browser identisch?

Nein!

JavaScript ist vergleichsweise jung. Was Navigator 2 von Netscape, der erste Browser mit JavaScript-Fähigkeit, konnte und kannte, waren allererste Anfänge. Die Entwicklungen, die man insbesondere bei den Browsern von Netscape und Microsoft seither beobachten kann, sind gravierend. Die Sprache selbst hat sich dabei gar nicht so sehr gewandelt, die Objektmodelle sind es, die sich so stürmisch verändern. Betrachtet man das OM des Navigator 2 heute, so nimmt es sich vergleichsweise bescheiden aus.

Und die Entwicklung steht ganz sicher noch lange nicht still. Für uns, die wir JavaScript für das Web einsetzen, heißt das, wir müssen mit Unterschieden leben, sie kennen und berücksichtigen. Seiten, mit denen der Besucher nichts anfangen kann, die ihn vielleicht sogar mit Fehlermeldungen verjagen, gehören zum Unangenehmsten, was uns passieren kann.

Welcher Browser bietet was?

Ich würde Ihnen diese Frage gern beantworten, wenn ich es könnte. Ich fürchte allerdings, eine nur annähernd vollständige Liste füllt ein eigenes wesentlich dickeres Buch als dieses hier.

Hilfreich ist es, etwas zur Geschichte der Browser zu wissen:

- Das, was der erwähnte Navigator 2 konnte und kannte, wird nachträglich als JavaScript 1.0 bezeichnet.[8]
- Etwas später zieht Microsoft mit dem Internet Explorer 3 nach, was die JavaScript-Fähigkeit der Browser anging. Die Unterschiede zum Navigator 2 sind nicht wirklich gravierend, zumindest aus heutiger Sicht nicht mehr.
- Danach kommt der nächste von Netscape, die Version 3. Er bringt die ersten wirklichen Neuerungen mit. Mit ihm erscheint auch die JavaScript-Versionsangabe und die Möglichkeit, sie in <`script`> anzugeben. Und mit ihm kommt das Image-Objekt.

Ab hier gibt es das Image-Objekt

- Den Umfang, den NN3 in seinem OM anbietet, nimmt dann auch MS in Version 4 seines Internet Explores auf. [9]
- Aber das ist nicht alles, was MS seiner Version 4 mitgibt. Obwohl etwas jünger als NN4, folgt MSIE4 von jetzt an Netscape nicht mehr.

8. Die Versionsangaben von Netscape beziehen sich nicht ausschließlich auf die Sprache, sondern eben auch auf die Objektmodelle der Browser. Das ist vom Begriff her sicher etwas irreführend.
9. Man macht an dieser Stelle in der Praxis sehr oft einen Schnitt, unterscheidet Browser, die JavaScript 1.1 nicht kennen, von denen, die es kennen. Ganz praktisch landen dabei NN2 und MSIE3 in der Gruppe der »Alten« (Browser anderer Anbieter einmal nicht beachtet).

- Sind es bis dahin Details, die Browser unterscheiden, und lediglich Erweiterungen in JavaScript 1.1 seitens Netscape, so beginnt zu dieser Zeit das, was den JavaScript-Programmierern fortan gelegentliche Kopfschmerzen bescheren soll: Netscape und Microsoft gehen getrennte Wege. Alles, was von nun an Neues kommt, ist nicht mehr wirklich vergleichbar.

Dennoch kann man vieles, was über die Fähigkeiten von NN3 hinausgeht, sowohl mit Netscape- als auch mit MS-Browsern erreichen. Man muss dabei allerdings mit schon vom Ansatz her unterschiedlichen Objektmodellen arbeiten. Dafür, diese Unterschiede am Ende doch unter einen Hut zu bringen, hat sich der Begriff Crossbrowser-Programmierung eingebürgert.

Schlagen Sie jetzt bitte nicht dieses Buch verzweifelt zu. All das spielt bei einer Vielzahl der Anwendungen noch gar keine Rolle. Erst wenn Sie tiefer einsteigen, sich mit DHTML beschäftigen, werden Sie nicht umhin können, sich damit eingehend zu befassen. Die Kapitel 9 bis 12 in diesem Buch werden Ihnen auch dabei helfen.

DOM – Document Objekt Model

Und noch eine gute Nachricht: Es gibt bereits Licht am Ende des Tunnels. W3C hat ein *Document Objekt Model* (DOM) spezifiziert ([W3CDOM]), das Netscape 6 zugrunde gelegt hat, dem aber auch der Microsoft Internet Explorer 5 bereits weitgehend folgt.

Die Kenntnis der Unterschiede ist gut und schön, nur nützt sie erst dann etwas, wenn man erkennen kann, welche Fähigkeiten der Browser des Besuchers hat. Damit sind wir bei der nächsten wichtigen Frage:

Wie findet man heraus, was der Browser kennt?

Eine oft anzutreffende Methode ist es, zu ermitteln, welchen Browser der Besucher benutzt, um sich in den Skripten danach zu richten. Dafür verwendet man das `navigator`-Objekt.

navigator

Blättern Sie bitte noch einmal an den Anfang dieses Kapitels zurück, in Abbildung 3.1 ist dieses Objekt erwähnt. Die wichtigsten Eigenschaften schauen wir uns am besten an, indem wir nachsehen, was bei einem bestimmten Browser in ihnen steckt. Probieren Sie das folgende Skript mit Ihrem Browser aus:

```
<html>
<head>
<title>navigator</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
<!--
document.writeln('<PRE>');
document.writeln('navigator.userAgent     ' +
```

```
                navigator.userAgent+'<BR>');
document.writeln('navigator.appCodeName   ' +
                navigator.appCodeName+'<BR>');
document.writeln('navigator.appVersion    ' +
                navigator.appVersion+'<BR>');
document.writeln('navigator.appName       ' +
                navigator.appName+'<BR>');
document.writeln('navigator.language      ' +
                navigator.language+'<BR>');
document.writeln('navigator.platform      ' +
                navigator.platform+'<BR>');
document.writeln('</PRE>');
// -->
</script>
<noscript>
Ihr Browser kann leider ohne JavaScript keine Informationen über ein
JavaScript-Objekt liefern.
</noscript>
</body>
</html>
```

Ich habe gerade Netscape 4.72 unter Windows 98 hier. Damit erhalte ich:

```
navigator.userAgent     Mozilla/4.72 [en] (Win98; I)
navigator.appCodeName   Mozilla
navigator.appVersion    4.72 [en] (Win98; I)
navigator.appName       Netscape
navigator.language      en
navigator.platform      Win32
```

Und mein Internet Explorer 5 zeigt:

```
navigator.userAgent
     Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
navigator.appCodeName  Mozilla
navigator.appVersion
     4.0 (compatible; MSIE 5.0; Windows 98; DigExt)
navigator.appName      Microsoft Internet Explorer
navigator.language     undefined
navigator.platform     Win32
```

Sie sehen, es sind Strings, die man natürlich auf ihren Inhalt hin untersuchen kann, und Sie sehen auch, dass `navigator.userAgent` die wichtigsten Informationen enthält. Je nachdem, wonach Sie unterscheiden wollen, untersuchen Sie dazu vielleicht noch andere Eigenschaften. Dafür lässt sich nur schwer ein Schema angeben, die Gründe, warum man unterscheiden will, sind zu vielfältig.

indexOf() leistet gute Dienste

Weil es um Auswertung von Zeichenfolgen geht, sei hier noch ein nützliches Hilfsmittel kurz vorgestellt: `indexOf()` bietet sich geradezu an. Das ist eine Methode, die Objekte vom Typ String haben, also alle Zeichenketten. Sie gibt Ihnen die Anfangsposition einer enthaltenen Zeichenfolge zurück. Und das ist die allgemeine Syntax:

```
stringName.indexOf(searchValue, [fromIndex])
```

Wichtig und nützlich ist die Tatsache, dass `-1` zurückgegeben wird, wenn die gesuchte Zeichenfolge nicht enthalten ist. Die Benutzung ist denkbar einfach:

```
if(indexOf(navigator.userAgent,' MSIE')>-1)
  alert('Aha, ein Browser von Microsoft!');
```

Einen Hinweis muss ich zur Browsererkennung über `navigator`-Objekt unbedingt anbringen: Es wird oft davon ausgegangen, dass `navigator.appName` die Erkennung von Netscape-Browsern ermöglicht. Das stimmt so nicht. Zum Beispiel findet man auch bei Opera dort »Netscape«. Verwenden Sie bitte deswegen nie ausschließlich `navigator.appName`!

Eine Übersicht darüber, was bei einer Reihe von Browsern in den Werten des `navigator`-Objektes alles so steckt, finden Sie im Web unter *http://www.netz-notizen.de/javascript/tipps/navigator.html*.

Unterscheidung per navigator-Objekt ist nicht unbedingt der beste Weg

Ich verwende derartige Browsererkennungen nur selten. Es gibt eine sicherere Methode: Man fragt den Browser, ob er die benötigte Eigenschaft des Objektes kennt, bevor man sie verwendet. Damit ist man etwas weniger auf Kenntnisse über jeden einzelnen Browser angewiesen.

```
if (document.images)
```

liefert zum Beispiel abhängig davon, ob es `document.images` gibt oder nicht, `true` oder `false`. Will man also vermeiden, dass Browser, die gar kein Image-Objekt kennen, die Besucher mit lästigen Fehlermeldungen nerven, statt Bildwechseleffekte zu zeigen, führt man die Anweisungen nur im `true`-Fall aus.

Deswegen sind die Beispiele zum `image`-Objekt weiter oben gar nicht besucherfreundlich, denn gerade das `image`-Objekt kennen die ersten Browser tatsächlich nicht. Wir hätten besser vorher nach seiner Existenz gefragt, zum Beipiel so:

```
<html>
<head><title>Image-Objekt</title>
<script language="JavaScript" type="text/javascript">
<!--
function wechseln()
{
   if (!document.images) return;
      window.document.Wechselbild.src='ein.gif';
   return;
}
```

```
// -->
</script>
</head>
<body>
<a href="nachirgendwo.html" onmouseover="wechseln()"><img
          src="aus.gif" name="Wechselbild"
          width="100" height="100"
          alt="Bild"></a>
</body>
</html>
```

Nun ist es sicher kaum praktikabel, nach jeder Eigenschaft zuvor zu fragen. Das ist auch nicht erforderlich. Auch hier ist Wissen um die Fähigkeiten der Browser sehr hilfreich. Man kann von der Existenz bestimmter Objekte ausgehend darauf schließen, dass auch anderes bekannt ist. Und genau das tun die meisten JavaScript-Programmierer auch. Das gerade erwähnte `image`-Objekt ist **das Beispiel** dafür. Es kam mit dem Navigator 3, also JavaScript 1.1, und man geht davon aus, dass ein Browser, der dieses Objekt kennt, auch alles andere beherrscht, was diese Version enthält.

3.2 Übungen

1. Übung

Wer interpretiert Ihre Skripte?

1. Das Betriebssystem des Client-Rechners
2. Der Web-Server
3. Der Web-Browser

Was ist neben der Sprache selbst die wichtigste Grundlage, auf die JavaScript-Programme angewiesen sind?

2. Übung

Wie könnte man den Browser veranlassen, eine andere Datei zu laden?

3. Übung

Sie haben alternative Seiten vorbereitet: eine Version für Browser, die nichts mit JavaScript anfangen können oder nur etwa den Umfang von JavaScript 1.0 beherrschen, eine andere für alle anderen. Wie würden Sie dafür sorgen, dass jeder Browser die ihm zugedachte Seite zeigt?

4. Übung

Sie haben eine HTML-Datei, deren `body` so aussieht:

```
<body>
<form name="erstesFormular">
<input type="hidden" name="versteckt" value="geheim">
</form>
<form name="zweitesFormular">
<input type="text" name="meinTextfeld" value="">
<input type="button" value="los geht's - Nr.1"
onclick="schreibeText(1)">
<input type="button" value="los geht's - Nr.2"
onclick="schreibeText(2)">
<input type="button" value="los geht's - Nr.3"
onclick="schreibeText(3)">
</form>
<body>
```

Sie wollen in das Textfeld des zweiten Formulars jeweils bei Klick auf einen der Buttons einen Text schreiben. Da das mit Hilfe der Funktion `schreibeText()` geschehen soll, haben Sie die Event-Handler, die den Aufruf der Funktion veranlassen sollen, schon mal vorsorglich eingefügt. Es fehlt noch die Funktion selbst. Schreiben Sie diese Funktion, wählen Sie aber jeweils eine andere Schreibweise, um das Textfeld anzusprechen.

3.3 Tipps

Tipp zu 2

Denken Sie bitte an das `location`-Objekt und dessen Eigenschaft `href`.

Tipp zu 3

Das Image-Objekt wurde mit JavaScript 1.1 eingeführt.

Tipps zu 4

Welcher Button angeklickt wurde, erkennen Sie am übergebenen Paramter.

Für die Wahl der verschiedenen Schreibweisen denken Sie an Namen, Arrays und assoziative Arrays.

3.4 Lösungen

Lösung zu 1

Der Web-Browser interpretiert Ihre JavaScripts, und auf sein Objektmodell (OM) sind Sie angewiesen.

Lösung zu 2

Diese Anweisung sorgt dafür, dass nach ihrer Ausführung der Wert von `window.location.href` den URL der zu ladenden Datei enthält und damit der Browser veranlasst wurde, diese Datei in das entsprechende Fenster (`window`) zu laden:

```
window.location.href = 'andereSeite.html';
```

Lösung zu 3

Wir verwenden die Idee der vorherigen Übung, um weiterzuleiten, wenn der Browser über die geforderten Fähigkeiten verfügt. Wir prüfen in der Datei, die für Browser ohne oder mit zu geringen JavaScript-Fähigkeiten gedacht ist, ob `document.images` existiert, schließen wenn ja daraus, dass der Browser unsere Anforderungen erfüllt, und laden sofort die passende Datei.

```
<html>
<head><title>Weiterleitung</title>
<script>
if (document.images)
   window.location.href = 'schoenereSeite.html';
</script>
</head>
<body>
<p>
Das ist die Seite für Browser ohne oder JavaScript oder mit etwas
magerer JavaScript-Fähikeit.
</p>
<body>
</html>
```

Eine wichtige Anmerkung zu dieser Weiterleitung möchte ich Ihnen nicht vorenthalten: Diese Lösung zeigt das Prinzip, und sie funktioniert auch. Aber sie hat einen Nachteil: man kommt von der so geladenen Seite nicht per Back-Button weg. Probieren Sie es aus oder schauen Sie auf der CD nach, dort finden sie diese Lösung.

Deswegen sollte man besser die Methode `window.location.replace()` benutzen, die dafür sorgt, dass die gerade aktuelle Seite nicht in die History aufgenommen wird. Leider existiert `replace()` aber erst seit JavaScript 1.1 Für den

Fall also, dass auch noch ältere Browser weitergeleitet werden sollen, kommt man nicht umhin, entweder die gerade gezeigte unschöne Lösung zu verwenden oder zusätzlich eine Unterscheidung einzubauen.

Für unser Beispiel geht es ohne Unterscheidung; wir wollten ja ohnehin nur Browser weiterleiten, die etwas von JavaScript 1.1 verstehen. `location.replace()` erwartet als Parameter den URL der neuen Seite. Unser Script könnte also so aussehen:

```
<script>
if (document.images)
window.location.replace('schoenereSeite.html');
</script>
```

Lösung zu 4

Diese drei Möglichkeiten, das Textfeld anzusprechen und dessen Wert zu verändern, sind mir gerade eingefallen:

```
<script>
function schreibeText(1)
{
 if (x == 1)
  window.document.forms[1].elements[0].value =
   'Bei Nacht';
 if (x == 2)
      window.document.zweitesFormular.meinTextfeld.value =
         'sind alle Katzen';
   if (x == 3)

window.document.forms['zweitesFormular'].
       elements['meinTextfeld'].value =
         'grau.';
}
</script>
```