PROGRAMMER'S CHOICE

Stefan Münz

Webseiten professionell erstellen

Programmierung, Design und Administration

DVD

- Videotrainings
- Tools
- Dokumentationen
- W3C-Recommendation
- HTML-Version

8 Praxisfälle für JavaScript/DOM

In Kapitel 7 wurden zwar schon diverse für die Praxis nützliche Scripts vorgestellt, doch die Beispiele hatten dort unterstützenden Charakter, um bestimmte Sprachkonzepte zu erläutern. In diesem Kapitel sollen die Beispiele im Vordergrund stehen. Das Kapitel zeigt, wie praxistypische Scripts entstehen und was dabei berücksichtigt werden sollte.

Wir konzentrieren uns auf drei wirklich nützliche Anwendungen, die JavaScript auch für professionelle Seiten salonfähig macht. Das Kapitel hat Workshop-Charakter.

Link zu JavaScript-Sammlungen:

Michael Mailer's JavaScripts:
http://www.mywebaid.de/

Scriptindex – JavaScript:
http://www.scriptindex.de/scripts/8/javascript/kat.php

JavaScript Source:
http://javascript.internet.com/toc.html

8.1 Clientseitige Formularüberprüfung

Bei Webanwendungen mit serverseitiger Datenanbindung erfolgt die Interaktion mit dem Anwender in der Regel über Formulare in HTML. Durch seine Angaben im Formular löst der Anwender nach Absenden des Formulars auf dem Webserver eine Datenverarbeitung aus. So wird beispielsweise ein Suchvorgang gestartet oder eingegebene Daten werden in eine Datenbank geschrieben oder in Dateiform gespeichert.

Auch wenn ein Formular so gestaltet ist, dass ein Anwender weiß, wie er es korrekt und sinnvoll ausfüllt, so kommt es doch immer wieder vor, dass Anwender bewusst Unsinn in ein Formular schreiben und es absen-

den oder dass sie ein wichtiges Feld vergessen auszufüllen oder – was am schlimmsten ist – dass sie versuchen, durch gefährliche Eingaben dem Server Schaden zuzufügen.

Bevor Anwendereingaben verarbeitet werden, sollten diese daher stets überprüft werden: Sind die Angaben vollständig bzw. sind alle wichtigen Felder ausgefüllt? Enthalten sie offensichtlichen Unsinn wie z. B. numerisch nicht interpretierbare Werte in einem Feld, in dem eine Zahl eingegeben werden soll? Enthalten sie Werte, die aus Server-Sicht problematisch sein können? (Wenn z. B. 1 Mio. Datensätze durchsucht werden und alle gefundenen Sätze in einem Rutsch ausgegeben werden, ist es problematisch, wenn ein Anwender nur nach dem Buchstaben »a« sucht.)

Nun gibt es zwei Orte, an denen eine Überprüfung von eingegebenen Formulardaten stattfinden kann: Der eine Ort ist das Script, welches auf dem Server die Formulardaten empfängt und die Daten verarbeitet. Der Vorteil bei dieser Lösung ist, dass sie browser-unabhängig ist und in jedem Fall funktioniert. Der Nachteil ist, dass zusätzliche Client-Server-Kommunikation zwischen Browser und Server nötig ist, falls Unzulänglichkeiten bei den Formulardaten festgestellt werden. Das Handling für den Anwender ist einfach umständlicher.

Der andere mögliche Ort einer Formularüberprüfung ist ein JavaScript, welches zusammen mit dem Formular an den Browser übertragen wird. Das Script wird dann aufgerufen, wenn der Anwender das Formular absenden will. Sind alle Eingaben in Ordnung, werden die Daten an den Server gesendet. Stellt das Script Fehler fest, kann der Anwender diese gleich korrigieren. Der Vorteil dieser Lösung ist also, dass bei Unzulänglichkeiten keine zusätzliche Client-Server-Konfiguration erforderlich ist und dass die Rechenkapazitäten des Anwenders mit der Eingabeprüfung belastet werden, statt die des Servers. Der Nachteil ist, dass bei deaktiviertem JavaScript keine Überprüfung stattfindet und die Daten ungeprüft zum Server gelangen.

In der Praxis ist daher eine doppelte Prüfung am realistischsten, wobei sich die serverseitige Prüfung auf problematische Fälle beschränken kann, während die clientseitige Prüfung auch den Charakter eines Hilfe-Assistenten annehmen kann.

8.1.1 Das Beispiel

In unserem Beispiel zur Formularüberprüfung mit JavaScript soll es um die Bestellung eines Horoskops gehen. Mit der Anmeldung geht der Anwender einen Kaufvertrag ein. Diverse Angaben sind zur Abwicklung Pflicht und einige davon müssen einem bestimmten Datenmuster entsprechen. Ein Script soll die Vollständigkeit und Korrektheit der Daten prüfen, bevor das Formular zum Server gesendet wird.

8.1.2 Quelltexte und Erläuterungen

Zunächst der Quelltext des HTML-Dokuments. Der script-Bereich zur Formularüberprüfung ist darin bereits vorgesehen, doch der Script-Code ist noch nicht eingefügt. Den JavaScript-Code werden wir noch separat behandeln. Im Quelltext des HTML-Dokuments sollen zunächst die Felder vorgestellt werden, die später mittels JavaScript auf Korrektheit überprüft werden sollen.

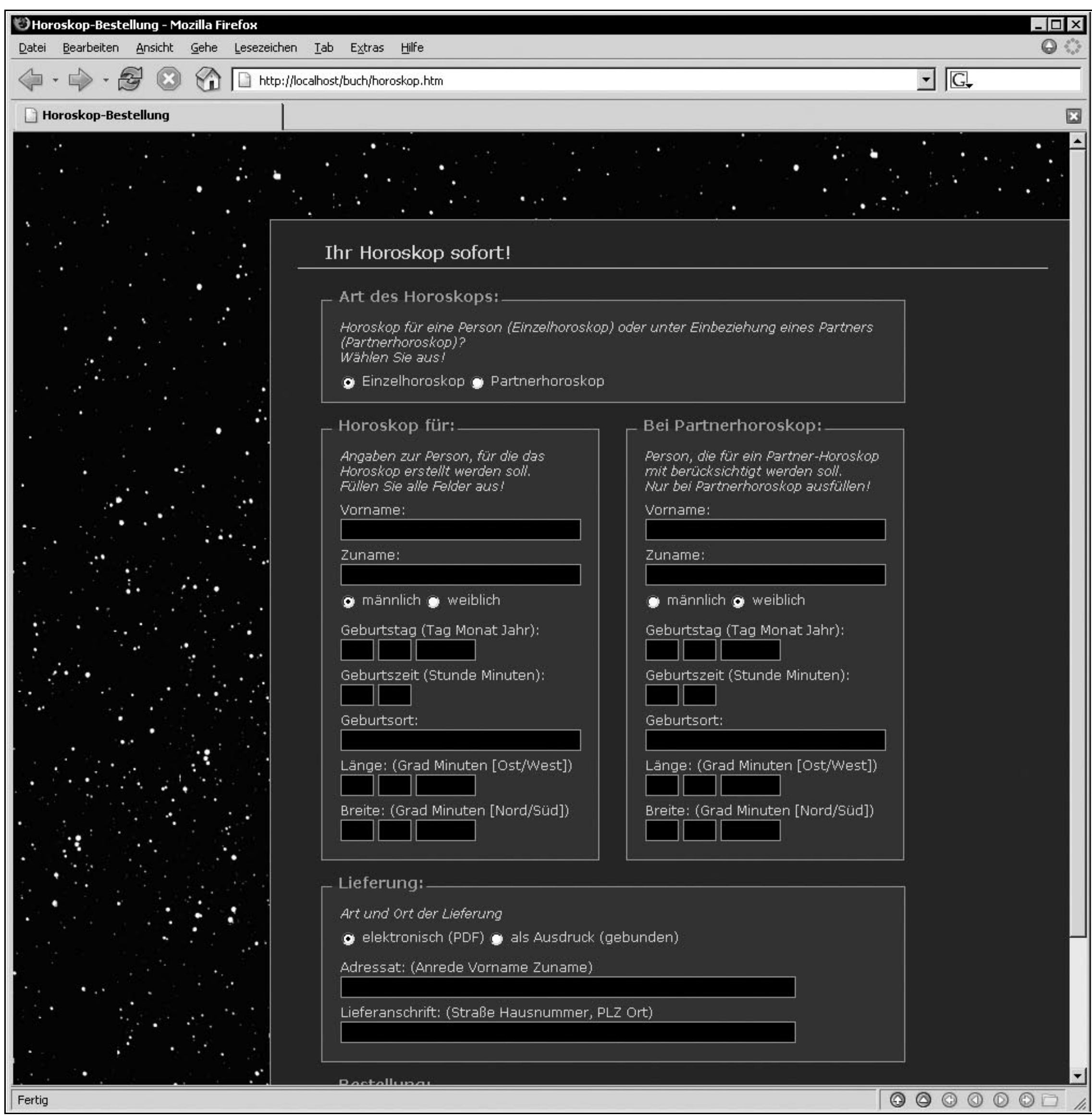


Abbildung 8.1: Formular zur Bestellung eines Horoskops

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
```

```
<title>Horoskop-Bestellung</title>
<link rel="stylesheet" type="text/css" href="horoskop.css">
<script type="text/javascript">
function check_form() {
   // hier folgt die Formularüberprüfung
}
</script>
</head>
<body>
<div id="top">
</div>
<div id="left">
</div>
<div id="content">
<h1>Ihr Horoskop sofort!</h1>
<form id="order" action="/scripts/order.php" method="post"
      onsubmit="check_form()">
<table id="form_table"><tr><td colspan="2">
<fieldset id="type"><legend>Art des Horoskops:</legend>
<div class="explain">Horoskop für eine Person (Einzelhoroskop)
oder unter Einbeziehung eines Partners (Partnerhoroskop)?<br>
Wählen Sie aus!</div>
<div style="margin-bottom:12px">
<input type="radio" class="radio" name="horoscope_type"
       value="e" checked>
Einzelhoroskop
<input type="radio" class="radio" name="horoscope_type"
       value="p">
Partnerhoroskop
</div>
</fieldset>
</td>
</tr><tr>
<td><fieldset id="person"><legend>Horoskop für:</legend>
<div class="explain">Angaben zur Person, für die das Horoskop
erstellt werden soll.<br>Füllen Sie alle Felder aus!</div>
<div>Vorname:<br>
<input type="text" name="given_name" class="text"
       style="width:218px">
</div>
<div>Zuname:<br>
<input type="text" name="family_name" class="text"
       style="width:218px">
</div>
<div style="margin-bottom:12px">
<input type="radio" class="radio" name="gender"
       value="m" checked>
männlich
```

```
<input type="radio" class="radio" name="gender"
       value="w">
weiblich
</div>
<div>Geburtstag (Tag Monat Jahr):<br>
<input type="text" name="birthday_day" class="text"
       style="width:25px">
<input type="text" name="birthday_month" class="text"
       style="width:25px">
<input type="text" name="birthday_year" class="text"
       style="width:50px">
</div>
<div>Geburtszeit (Stunde Minuten):<br>
<input type="text" name="birthtime_hours" class="text"
       style="width:25px">
<input type="text" name="birthtime_minutes" class="text"
       style="width:25px">
</div>
<div>Geburtsort:<br>
<input type="text" name="birth_place" class="text"
       style="width:218px">
</div>
<div>Länge: (Grad Minuten [Ost/West])<br>
<input type="text" name="long_deg" class="text"
       style="width:25px">
<input type="text" name="long_min" class="text"
       style="width:25px">
<input type="text" name="long_dir" class="text"
       style="width:50px">
</div>
<div>Breite: (Grad Minuten [Nord/Süd])<br>
<input type="text" name="lat_deg" class="text"
       style="width:25px">
<input type="text" name="lat_min" class="text"
       style="width:25px">
<input type="text" name="lat_dir" class="text"
       style="width:50px">
</div>
</fieldset></td>
<td>
<fieldset id="partner"><legend>Bei Partnerhoroskop:</legend>
<div class="explain">Person, die für ein Partner-Horoskop mit
berücksichtigt werden soll.<br>
Nur bei Partnerhoroskop ausfüllen!</div>
<div>Vorname:<br>
<input type="text" name="p_given_name" class="text"
       style="width:218px">
</div>
<div>Zuname:<br>
```

```
<input type="text" name="p_family_name" class="text"
       style="width:218px">
</div>
<div style="margin-bottom:12px">
<input type="radio" class="radio" name="p_gender"
       value="m">
männlich
<input type="radio" class="radio" name="p__gender"
       value="w" checked>
weiblich<br>
</div>
<div>Geburtstag (Tag Monat Jahr):<br>
<input type="text" name="p_birthday_day" class="text"
       style="width:25px">
<input type="text" name="p_birthday_month" class="text"
       style="width:25px">
<input type="text" name="p_birthday_year" class="text"
       style="width:50px">
</div>
<div>Geburtszeit (Stunde Minuten):<br>
<input type="text" name="p_birthtime_hours" class="text"
       style="width:25px">
<input type="text" name="p_birthtime_minutes" class="text"
       style="width:25px">
</div>
<div>Geburtsort:<br>
<input type="text" name="p_birth_place" class="text"
       style="width:218px">
</div>
<div>Länge: (Grad Minuten [Ost/West])<br>
<input type="text" name="p_long_deg" class="text"
       style="width:25px">
<input type="text" name="p_long_min" class="text"
       style="width:25px">
<input type="text" name="p_long_dir" class="text"
       style="width:50px">
</div>
<div>Breite: (Grad Minuten [Nord/Süd])<br>
<input type="text" name="p_lat_deg" class="text"
       style="width:25px">
<input type="text" name="p_lat_min" class="text"
       style="width:25px">
<input type="text" name="p_lat_dir" class="text"
       style="width:50px">
</div>
</fieldset></td>
</tr><tr>
<td colspan="2">
<fieldset id="delivery"><legend>Lieferung:</legend>
```

```
<div class="explain">Art und Ort der Lieferung
</div>
<div style="margin-bottom:12px">
<input type="radio" class="radio" name="delivery_type"
       value="e" checked>
elektronisch (PDF)
<input type="radio" class="radio" name="delivery_type"
       value="p">
als Ausdruck (gebunden)<br>
</div>
<div>Adressat: (Anrede Vorname Zuname)<br>
<input type="text" name="delivery_person" class="text"
       style="width:418px">
</div>
<div>Lieferanschrift: (Straße Hausnummer, PLZ Ort)<br>
<input type="text" name="delivery_address" class="text"
       style="width:418px">
</div>
</fieldset></td>
</tr><tr>
<td colspan="2">
<fieldset id="submit"><legend>Bestellung:</legend>
<p><strong>Preise:</strong><br>
Einzelhoroskop elektronisch als PDF: &euro; 28,00<br>
Einzelhoroskop gedruckt und gebunden: &euro; 38,00<br>
Partnerhoroskop elektronisch als PDF: &euro; 36,00<br>
Partnerhoroskop gedruckt und gebunden: &euro; 46,00</p>
<div>
<input type="submit" class="button" value="Absenden">
</div>
</fieldset></td>
</tr></table>
</form>
</div>
</body>
</html>
```

Listing 8.1: HTML-Quelltext mit Formular

Den Quelltext der CSS-Datei sparen wir an dieser Stelle aus, da er nicht von Belang ist.

Das HTML-Formular hat einen praxistypischen Umfang. Bei den Eingabefeldern soll Folgendes überprüft werden:

- Alle Feldgruppen mit Radiobuttons müssen jeweils eine gültige Auswahl enthalten.
- Wenn als Art des Horoskops *Einzelhoroskop* ausgewählt wurde, müssen alle Felder im Bereich *Horoskop für* einen Inhalt aufweisen. Wenn als Horoskopart *Partnerhoroskop* ausgewählt wurde, müssen sowohl im Bereich *Horoskop für* als auch im Bereich *Bei Partnerhoroskop* alle Felder ausgefüllt sein.

- Eingaben in Feldern für Vor- und Zuname sollen nicht mehr als 60 Zeichen haben dürfen, weil sie z. B. in eine Datenbank mit so begrenzten Feldlängen übernommen werden sollen.
- In den Einzelfeldern für Geburtstag und Geburtszeit müssen gültige Werte eingetragen werden: Monatstage zwischen 1 und 31, Monate zwischen 1 und 12, Jahr vierstellig, Stunden zwischen 0 und 23 und Minuten zwischen 0 und 59. Ferner soll geprüft werden, ob die Angaben zum Geburtsdatum ein gültiges Kalenderdatum ergeben.
- In den Einzelfeldern für Längengrad, Längenminuten, Breitengrad und Breitenminuten müssen ebenfalls gültige Werte eingetragen werden, also Längengrade zwischen 0 und 180, Breitengrade zwischen 0 und 90, Längen- und Breitenminuten zwischen 0 und 59 und in den Feldern für die Richtung ein Wort wie Ost oder West bzw. Nord oder Süd.
- Die Felder für den Geburtsort dürfen nicht mehr als 100 Zeichen haben. Die Felder für Adressat und Lieferanschrift dürfen nicht mehr als 160 Zeichen haben.

Bei Fehlern soll ein Meldungsfenster das aufgetretene Problem ausgeben. Im Fall von Eingabefeldern soll anschließend der Cursor direkt in das Feld mit der fehlenden oder fehlerhaften Eingabe positioniert werden, damit der Anwender den Wert gleich korrigieren kann.

Da diverse Checks öfter vorkommen, z. B. die Überprüfung eines Werts daraufhin, ob er sich in einem bestimmten numerischen Bereich von/bis bewegt, strukturieren wir den JavaScript-Code so, dass einzelne, wiederverwendbare Funktionen bestimmte Prüfungen vornehmen und lediglich zurückgeben, ob die Eingabe den Prüfungskriterien entspricht (`true`) oder nicht (`false`).

Zum Management der einzelnen Fehler spendieren wir ebenfalls eine eigene Funktion. Eine Zentralfunktion schließlich geht die einzelnen Felder durch.

Zunächst zu den wiederverwendbaren Einzelfunktionen, die bestimmte Arten von Prüfungen vornehmen oder eine Hilfsfunktionalität leisten:

```
function check_radios(name) {
   var check = false;
   for(i = 0; i < document.getElementsByName(name).length; i++) {
      if(document.getElementsByName(name)[i].checked == true)
         check = true;
   }
   return(check);
}
function check_field_length(name, length) {
   var check = false;
   if(document.getElementsByName(name)[0].value.length <= length)
      check = true;
   return(check);
}
function check_field_min_length(name, min_length) {
   var check = false;
   if(document.getElementsByName(name)[0].value.length >= min_length)
```

```
      check = true;
   return(check);
}
function check_field_str_range(name, comma_str) {
   var check = false;
   var strings = comma_str.split(",");
   for(i = 0; i < strings.length; i++) {
      if(document.getElementsByName(name)[0].value.toLowerCase() ==
strings[i].toLowerCase())
         check = true;
   }
   return(check);
}
function check_field_int_range(name, min, max) {
   var check = false;
   var x = parse0int(document.getElementsByName(name)[0].value);
   if(x >= min && x <= max)
      check = true;
   return(check);
}
function check_valid_date(day, month, year) {
   if(isNaN(day) || isNaN(month) || isNaN(year))
      return false;
   if(year < 1801 || year > 2799)
      return false;
   if(month < 1 || month > 12)
      return false;
   var feb = 28;
   if(month == 2) {
      if(year % 4 == 0) feb += 1;
      if(year % 100 == 0) feb -= 1;
      if(year % 400 == 0) feb += 1;
   }
   var days = new Array(31,feb,31,30,31,30,31,31,30,31,30,31);
   if(day > days[month-1] || day < 1)
      return false;
   return true;
}
function point_to(name) {
   document.getElementsByName(name)[0].focus();
}
function parse0int(x) {
   y = parse0int(x * 10);
   y = y / 10;
   return(y);
}
```

Listing 8.2: Wiederverwendbare Einzelfunktionen für Formularüberprüfung

Folgende Funktionen sind in diesem Listing enthalten:

- `check_radios()`: Prüft, ob von einer Gruppe von Radiobuttons einer aktiviert ist. Erwartet als Parameter den gemeinsamen Wert des name-Attributs einer Gruppe von Radiobuttons. Ist einer der Buttons aktiviert, wird `true` zurückgegeben, wenn nicht, dann `false`.
- `check_field_length()`: Prüft, ob die Anzahl Zeichen in einem Feld einen bestimmten Wert nicht überschreitet. Erwartet als ersten Parameter den Wert des name-Attributs des Eingabefelds und als zweiten Parameter die maximal erlaubte Anzahl Zeichen. Ist die Eingabe nicht länger als erlaubt, wird `true` zurückgegeben, andernfalls `false`.
- `check_field_min_length()`: Prüft, ob in einem Feld mindestens eine bestimmte Anzahl Zeichen eingegeben wurden. Erwartet als ersten Parameter den Wert des name-Attributs des Eingabefelds und als zweiten Parameter die Mindestanzahl von Zeichen, die das Feld enthalten sollte. Ist die Eingabe mindestens so lang wie angegeben, wird `true` zurückgegeben, andernfalls `false`.
- `check_field_str_range()`: Prüft, ob ein Feld einen Zeichenkettenwert aus einem bestimmten Bereich erlaubter Werte enthält. Erwartet als ersten Parameter den Wert des name-Attributs des Eingabefelds und als zweiten Parameter eine Zeichenkette, welche die erlaubten Werte enthält, wobei die einzelnen Werte durch Kommata voneinander getrennt sein müssen. Kommt der eingegebene Feldwert in den erlaubten Werten vor, wird `true` zurückgegeben, wenn nicht, dann `false`.
- `check_field_int_range()`: Prüft, ob ein Feld einen Ganzzahlwert aus einem bestimmten Bereich enthält. Erwartet als ersten Parameter den Wert des name-Attributs des Eingabefelds, als zweiten Parameter den Mindestwert des erlaubten Bereichs und als dritten Parameter den Höchstwert des erlaubten Bereichs. Kommt der eingegebene Feldwert im erlaubten Bereich vor, wird `true` zurückgegeben, wenn nicht, dann `false`.
- `check_valid_date()`: Prüft, ob ein Datum, bestehend aus Tag, Monat und Jahr, kalendarisch gültig ist. Erwartet als ersten Parameter den Wert für »Tag«, als zweiten den für »Monat« und als dritten den für »Jahr«. Alle Parameter müssen bereits numerisch übergeben werden. Ist das Datum kalendarisch korrekt, wird `true` zurückgegeben, wenn nicht, dann `false`.
- `point_to()`: Setzt den Cursor in ein bestimmtes Eingabefeld. Erwartet als Parameter den Wert des name-Attributs des gewünschten Eingabefelds.
- `parse0int()`: Hilfsfunktion, die Eingaben wie 08 oder 09 in numerischen Feldern davor bewahrt, von JavaScript als Fehler interpretiert zu werden. Wird im Script anstelle der Standardfunktion `parseInt()` aufgerufen.

Erkennbar ist, dass die Funktionen die Aufgabenstellungen widerspiegeln. Die einzelnen Check-Funktionen tun noch nichts weiter, als ihre Prüfung vorzunehmen und `true` oder `false` zurückzugeben. Erst die aufrufende Funktion entscheidet dann, was auf Grund des Rückgabewerts zu tun ist. Dies ist typisch funktionsorientiertes Programmieren nach dem Vorbild von C. JavaScript ist für eine solche Art von Programmierung bestens geeignet.

Alle Check-Funktionen greifen über die Methode `document.getElementsByName()` auf die Feldinhalte zu. Das bietet sich an, da die Formularfelder ohnehin ein name-Attribut erhalten müssen, welches für die spätere Datenverarbeitung auf dem Server von Bedeutung ist. Die Check-Funktionen greifen stets auf `document.getElementsByName()[0]` zu, da sie davon ausgehen, dass alle name-Attribute von Eingabefeldern dokumentweit eindeutig sind. Nur bei Radiobuttons, die jeweils eine Gruppe bilden, liegt der Fall anders.

Zur Funktion `check_valid_date()` ist noch anzumerken, dass sie das übergebene Jahr prüft und nur einen Wertebereich von 1801 bis 2799 zulässt. Dieser Wertebereich ist nicht streng begründbar, aber auch nicht willkürlich. Dahinter stehen Argumentationen rund um die Verbreitung und die künftige Korrektheit des gregorianischen Kalenders. Und für unser Horoskop-Beispiel sollte dieser Wertebereich allemal ausreichen. Die Funktion prüft ansonsten noch, ob der angegebene Monat überhaupt so viele Tage hat wie als Monatstag angegeben. Dabei werden, was den Februar betrifft, auch Schaltjahre inklusive 100-Jahres- und 400-Jahres-Regelung berücksichtigt. Ferner wird überprüft, ob der Monat sich im gültigen Bereich zwischen 1 und 12 befindet.

Die Funktion `point_to()` wird vom Fehlerhandling bei Auftreten eines Fehlers aufgerufen, um den Cursor nach Aufforderung des Anwenders zur Korrektur in das gewünschte Feld zu positionieren. Dabei wird die focus()-Methode verwendet, die auf HTML-Formularelemente anwendbar ist.

Ein Problem, das schon einige JavaScript-Programmierer fast verzweifeln ließ, ist, wenn die Standardmethode `parseInt()` auf eine Formulareingabe angewendet wird und deren Inhalt mit 08 oder 09 beginnt. In unserem Fall kann genau das sehr häufig vorkommen, da zahlreiche Anwender beim Ausfüllen von Tag, Monat, Stunde und Minute führende Nullen eingeben. Das Problem bei `parseInt()` besteht darin, dass es Werte mit führender 0 als Oktalwert interpretiert. Die Folgen 08 und 09 sind dann aber ungültig, da Oktalzahlen nur aus den Ziffern 0 bis 7 bestehen können. Mithilfe unserer »Tricky«-Funktion `parse0int()` lösen wir dieses Problem. Die Funktion multipliziert einfach den ihr übergebenen Wert mit 10. So wird aus 08 der Wert 80. Dieser Wert wird dann wieder durch 10 dividiert, was 8 ergibt. Dadurch ist ein ordentlicher numerischer Wert erreicht.

Damit kommen wir zur Funktion, welche das Fehlerhandling übernimmt:

```
function handle_error(number) {
   var error_text = new Array(
     "Hinweis: Keine Eingabenüberprüfung im Browser möglich!",
     "Art des Horoskops nicht ausgewählt!",
     "Keinen Vornamen eingegeben!",
     "Vorname zu lang!",
     "Keinen Zunamen eingegeben!",
     "Zuname zu lang!",
     "Kein Geschlecht ausgewählt!",
     "Geburtstag: Tag fehlt oder ist ungültig!",
     "Geburtstag: Monat fehlt oder ist ungültig!",
     "Geburtstag: Jahr fehlt oder ist ungültig!",
     "Geburtszeit: Stunden fehlen oder sind ungültig!",
     "Geburtszeit: Minuten fehlen oder sind ungültig!",
```

```
    "Geburtstag: ungültiges Kalenderdatum!",
    "Keinen Geburtsort eingegeben!",
    "Geburtsort zu lang!",
    "Länge: Gradangabe fehlt oder ist ungültig!",
    "Länge: Minutenangabe fehlt oder ist ungültig!",
    "Länge: 'West' oder 'Ost' eingeben!",
    "Breite: Gradangabe fehlt oder ist ungültig!",
    "Breite: Minutenangabe fehlt oder ist ungültig!",
    "Breite: 'Nord' oder 'Süd' eingeben!",
    "Partner: keinen Vornamen eingegeben!",
    "Partner: Vorname zu lang!",
    "Partner: keinen Zunamen eingegeben!",
    "Partner: Zuname zu lang!",
    "Partner: kein Geschlecht ausgewählt!",
    "Partner-Geburtstag: Tag fehlt oder ist ungültig!",
    "Partner-Geburtstag: Monat fehlt oder ist ungültig!",
    "Partner-Geburtstag: Jahr fehlt oder ist ungültig!",
    "Partner-Geburtstag: ungültiges Kalenderdatum!",
    "Partner-Geburtszeit: Stunden fehlen oder sind ungültig!",
    "Partner-Geburtszeit: Minuten fehlen oder sind ungültig!",
    "Partner: Keinen Geburtsort eingegeben!",
    "Partner: Geburtsort zu lang!",
    "Partner: Länge: Gradangabe fehlt oder ist ungültig!",
    "Partner: Länge: Minutenangabe fehlt oder ist ungültig!",
    "Partner: Länge: 'West' oder 'Ost' eingeben!",
    "Partner: Breite: Gradangabe fehlt oder ist ungültig!",
    "Partner: Breite: Minutenangabe fehlt oder ist ungültig!",
    "Partner: Breite: 'Nord' oder 'Süd' eingeben!",
    "Art der Lieferung nicht ausgewählt!",
    "Keinen Adressaten eingegeben!",
    "Adressat zu lang!",
    "Keine Lieferanschrift eingegeben!",
    "Lieferanschrift zu lang!"
  );
  alert(error_text[number]);
  switch(number) {
     case 0: break;
     case 1: break;
     case 2: case 3: point_to('given_name'); break;
     case 4: case 5: point_to('family_name'); break;
     case 6: break;
     case 7: point_to('birthday_day'); break;
     case 8: point_to('birthday_month'); break;
     case 9: point_to('birthday_year'); break;
     case 10: point_to('birthtime_hours'); break;
     case 11: point_to('birthtime_minutes'); break;
     case 12: point_to('birthday_day'); break;
     case 13: case 14: point_to('birth_place'); break;
     case 15: point_to('long_deg'); break;
```

```
        case 16: point_to('long_min'); break;
        case 17: point_to('long_dir'); break;
        case 18: point_to('lat_deg'); break;
        case 19: point_to('lat_min'); break;
        case 20: point_to('lat_dir'); break;
        case 21: case 22: point_to('p_given_name'); break;
        case 23: case 24: point_to('p_family_name'); break;
        case 25: break;
        case 26: point_to('p_birthday_day'); break;
        case 27: point_to('p_birthday_month'); break;
        case 28: point_to('p_birthday_year'); break;
        case 29: point_to('p_birthtime_hours'); break;
        case 30: point_to('p_birthtime_minutes'); break;
        case 31: point_to('p_birthday_day'); break;
        case 32: case 33: point_to('p_birth_place'); break;
        case 34: point_to('p_long_deg'); break;
        case 35: point_to('p_long_min'); break;
        case 36: point_to('p_long_dir'); break;
        case 37: point_to('p_lat_deg'); break;
        case 38: point_to('p_lat_min'); break;
        case 39: point_to('p_lat_dir'); break;
        case 40: break;
        case 41: case 42: point_to('delivery_person'); break;
        case 43: case 44: point_to('delivery_address'); break;
        default: break;
    }
    return false;
}
```

Die Funktion, `handle_error()`, bekommt beim Aufruf einfach eine Fehlernummer übergeben. Der längere Array aus Fehlermeldungstexten ist so definiert, dass aufgrund der übergebenen Fehlernummer die gleichlautende Array-Indexnummer die zugehörige Fehlermeldung liefert. Mit der alert()-Methode wird die so ermittelte Meldung im Anschluss an den Array auch gleich ausgegeben. An diesem Moment der Script-Ausführung bekommt der Anwender also einen bei der Eingabeprüfung aufgetretenen Fehler in einem Meldungsfenster angezeigt.

Anschließend soll in das betroffene Feld positioniert werden, damit der Anwender den genannten Fehler gleich korrigieren kann. Dazu muss die Funktion auch wieder alle denkbaren Fehlernummern einzeln unterscheiden. Das tut sie in einem switch-case-Konstrukt. Abhängig von der übergebenen Fehlernummer wird in das zugehörige Feld positioniert.

Am Ende gibt die Funktion `handle_error()` den Wert `false` zurück. Das ist nicht unwichtig, wie wir gleich noch sehen werden. Die Funktion `handle_error()` ist nämlich so programmiert, dass sie von der eigentlichen Zentralfunktion der Formularüberprüfung, nämlich der Funktion `check_form()`, optimal bedient werden kann. Zum Schluss noch der Code der zentralen Funktion:

```
function check_form() {
   if(!document.getElementsByName) {
      handle_error(0);
      return(true);
   }
   var chk = check_radios('horoscope_type');
   if(chk == false) return(handle_error(1));
   chk = check_field_min_length('given_name', 1);
   if(chk == false) return(handle_error(2));
   chk = check_field_length('given_name', 60);
   if(chk == false) return(handle_error(3));
   chk = check_field_min_length('family_name', 1);
   if(chk == false) return(handle_error(4));
   chk = check_field_length('family_name', 60);
   if(chk == false) return(handle_error(5));
   chk = check_radios('gender');
   if(chk == false) return(handle_error(6));
   chk = check_field_int_range('birthday_day', 1, 31);
   if(chk == false) return(handle_error(7));
   chk = check_field_int_range('birthday_month', 1, 12);
   if(chk == false) return(handle_error(8));
   chk = check_field_int_range('birthday_year', 1801, 2799);
   if(chk == false) return(handle_error(9));
   chk = check_field_int_range('birthtime_hours', 0, 23);
   if(chk == false) return(handle_error(10));
   chk = check_field_int_range('birthtime_minutes', 0, 59);
   if(chk == false) return(handle_error(11));
   var d = document.getElementsByName('birthday_day')[0].value;
   var m = document.getElementsByName('birthday_month')[0].value;
   var y = document.getElementsByName('birthday_year')[0].value;
   chk = check_valid_date(parse0int(d), parse0int(m), parse0int(y));
   if(chk == false) return(handle_error(12));
   chk = check_field_min_length('birth_place', 1);
   if(chk == false) return(handle_error(13));
   chk = check_field_length('birth_place', 100);
   if(chk == false) return(handle_error(14));
   chk = check_field_int_range('long_deg', 0, 180);
   if(chk == false) return(handle_error(15));
   chk = check_field_int_range('long_min', 0, 59);
   if(chk == false) return(handle_error(16));
   chk = check_field_str_range('long_dir', 'Ost,West');
   if(chk == false) return(handle_error(17));
   chk = check_field_int_range('lat_deg', 0, 90);
   if(chk == false) return(handle_error(18));
   chk = check_field_int_range('lat_min', 0, 59);
   if(chk == false) return(handle_error(19));
   chk = check_field_str_range('lat_dir', 'Nord,Süd');
   if(chk == false) return(handle_error(20));
   type = document.getElementsByName('horoscope_type');
```

```
if(type[1].checked == true) {
   chk = check_field_min_length('p_given_name', 1);
   if(chk == false) return(handle_error(21));
   chk = check_field_length('p_given_name', 60);
   if(chk == false) return(handle_error(22));
   chk = check_field_min_length('p_family_name', 1);
   if(chk == false) return(handle_error(23));
   chk = check_field_length('p_family_name', 60);
   if(chk == false) return(handle_error(24));
   chk = check_radios('p_gender');
   if(chk == false) return(handle_error(25));
   chk = check_field_int_range('p_birthday_day', 1, 31);
   if(chk == false) return(handle_error(26));
   chk = check_field_int_range('p_birthday_month', 1, 12);
   if(chk == false) return(handle_error(27));
   chk = check_field_int_range('p_birthday_year', 1801, 2799);
   if(chk == false) return(handle_error(28));
   chk = check_field_int_range('p_birthtime_hours', 0, 23);
   if(chk == false) return(handle_error(29));
   chk = check_field_int_range('p_birthtime_minutes', 0, 59);
   if(chk == false) return(handle_error(30));
   d = document.getElementsByName('p_birthday_day')[0].value;
   m = document.getElementsByName('p_birthday_month')[0].value;
   y = document.getElementsByName('p_birthday_year')[0].value;
   chk = check_valid_date(parse0int(d), parse0int(m), parse0int(y));
   if(chk == false) return(handle_error(31));
   chk = check_field_min_length('p_birth_place', 1);
   if(chk == false) return(handle_error(32));
   chk = check_field_length('p_birth_place', 100);
   if(chk == false) return(handle_error(33));
   chk = check_field_int_range('p_long_deg', 0, 180);
   if(chk == false) return(handle_error(34));
   chk = check_field_int_range('p_long_min', 0, 59);
   if(chk == false) return(handle_error(35));
   chk = check_field_str_range('p_long_dir', 'Ost,West');
   if(chk == false) return(handle_error(36));
   chk = check_field_int_range('p_lat_deg', 0, 90);
   if(chk == false) return(handle_error(37));
   chk = check_field_int_range('p_lat_min', 0, 59);
   if(chk == false) return(handle_error(38));
   chk = check_field_str_range('p_lat_dir', 'Nord,Süd');
   if(chk == false) return(handle_error(39));
}
var chk = check_radios('delivery_type');
if(chk == false) return(handle_error(40));
chk = check_field_min_length('delivery_person', 1);
if(chk == false) return(handle_error(41));
chk = check_field_length('delivery_person', 160);
if(chk == false) return(handle_error(42));
```

```
        chk = check_field_min_length('delivery_address', 1);
        if(chk == false) return(handle_error(43));
        chk = check_field_length('delivery_address', 160);
        if(chk == false) return(handle_error(44));
        if(chk == true) return true;
    }
```

Die Zentralfunktion `check_form()` geht der Reihe nach alle Felder des Formulars durch. Entsprechend der Vorgaben ruft sie eine der wiederverwendbaren Einzelfunktionen zur Überprüfung des jeweiligen Feldinhalts auf. Den Rückgabewert dieser Funktionen speichert sie in der Variablen `chk`. Nach jedem Aufruf dieser Art wird mit `if(chk == false)` geprüft, ob die Einzelprüfung des jeweiligen Feldinhalts einen Fehler ergeben hat. Ist das der Fall, so wird eine return-Anweisung ausgeführt. Als Wert wird der return-Anweisung ein Aufruf der Funktion `handle_error()` übergeben, und zwar mit der gewünschten Fehlernummer. Die Funktion `handle_error()` sorgt, wie bereits bekannt, für die Ausgabe der zugehörigen Fehlermeldung und für die Korrekturpositionierung ins zugehörige Feld. Da `handle_error()` am Ende den Boolean-Wert `false` zurückgibt, bedeutet ein Konstrukt wie `return(handle_error(44))` letztlich auch nur `return(false)`.

Der Vorteil dieser Konstrukte ist, dass die Funktion `check_form()` sofort nach Erledigung der Arbeiten von `handle_error()` beendet wird. Der Rückgabewert `false` landet beim aufrufenden Event-Handler `onsubmit` im einleitenden `<form>`-Tag. Dieser gibt ihn ebenfalls wieder mit `return` weiter:

```
<form id="order" action="horoskop2.htm" method="post"
      onsubmit="return(check_form())">
```

Damit wird der false-Wert an den Browser selbst geleitet. Dieser reagiert so, dass das Formular nicht abgesendet wird, was ja im Fehlerfall wichtig ist.

Fazit

Das Beispiel zeigt, dass bei ordentlichen Prüfungen von Formularfeldinhalten schnell eine ganze Menge Script-Code entstehen kann. Nutznießer davon sind jedoch einerseits der Anwender, der im Fehlerfall präzise Meldungen erhält und gleich das betreffende Feld korrigieren kann, und der Seitenanbieter, der ordentlich verarbeitbare Daten zum Server übertragen bekommt.

8.2 Navigation im Explorer-Stil

An kaum eine Form der Navigation haben sich Anwender in den letzten Jahren so gewöhnt wie an diejenige im Stil des Windows Explorers. Längst findet sich diese Art des Auswählens in allen erdenklichen Anwendungen wieder, egal ob Vektorgrafikprogramm, Programmieroberfläche, elektronische Steuererklärung oder Bookmarks im Browser.

In HTML gibt es von Haus aus kein Element, das eine solche Navigation realisiert (leider, denn eigentlich darf von einer Sprache, die »Hypertext« im Namen trägt, durchaus erwartet werden, dass sie mehr Möglichkeiten zur Realisierung einer Navigation bietet als einfache Hyperlinks).

Im Rahmen des CSS-Praxiskapitels (***Abschnitt 5.2***) lernten Sie bereits Möglichkeiten kennen, eine attraktive Navigation für eigene Websites zu realisieren, und zwar allein mit HTML und CSS. Für explorerartige Menüs geht es nicht ohne ein Minimum an JavaScript. Wir werden in unserem Beispiel mit überraschend wenig JavaScript auskommen. Das Beispiel erfüllt auch noch einige andere wichtige Kriterien:

- Das HTML-Markup der Navigation soll strukturell sinnvoll sein, so dass es auch bei nicht grafischen Browsern ordentlich aussieht. Gerade weil Projekte, die ein baumartiges Menü anbieten, in der Regel aus sehr vielen Unterseiten bestehen, ist ein strukturell gut nachvollziehbarer Aufbau der Navigation wichtig. Wir werden in unserem Beispiel mit verschachtelten Aufzählungslisten arbeiten.
- Die Navigation soll auch bei deaktiviertem JavaScript einwandfrei und ohne Zusatzaufwand funktionieren. Auch das ist bei unserem Beispiel der Fall.
- Das Auf- und Zuklappen von Teilbäumen soll aus Anwendersicht schnell reagieren und auch bei größeren Menüstrukturen nicht zu merklichen Verzögerungen führen.
- Der aktuelle Menüzustand (auf- und zugeklappte Teilbäume) soll an die nächste aufgerufene Seite übergeben werden, so dass er beim Seitenwechsel erhalten bleibt. Andernfalls wäre es für den Anwender mühsam, sich nach jedem Seitenwechsel wieder erneut durch die Struktur zu klicken.
- Das Script soll wiederverwendbar sein, d. h. auf beliebige andere Menüstrukturen anwendbar.

8.2.1 Das Beispiel

Navigationsstrukturen dieser Art bieten sich vor allem für größere technische Dokumentationen, für wissenschaftliche Abhandlungen und für systematische Darstellungen von Wissensgebieten an. In unserem Beispiel zeigen wir dies anhand eines kleinen Ausschnitts der Systematik der Tierwelt (Fauna).

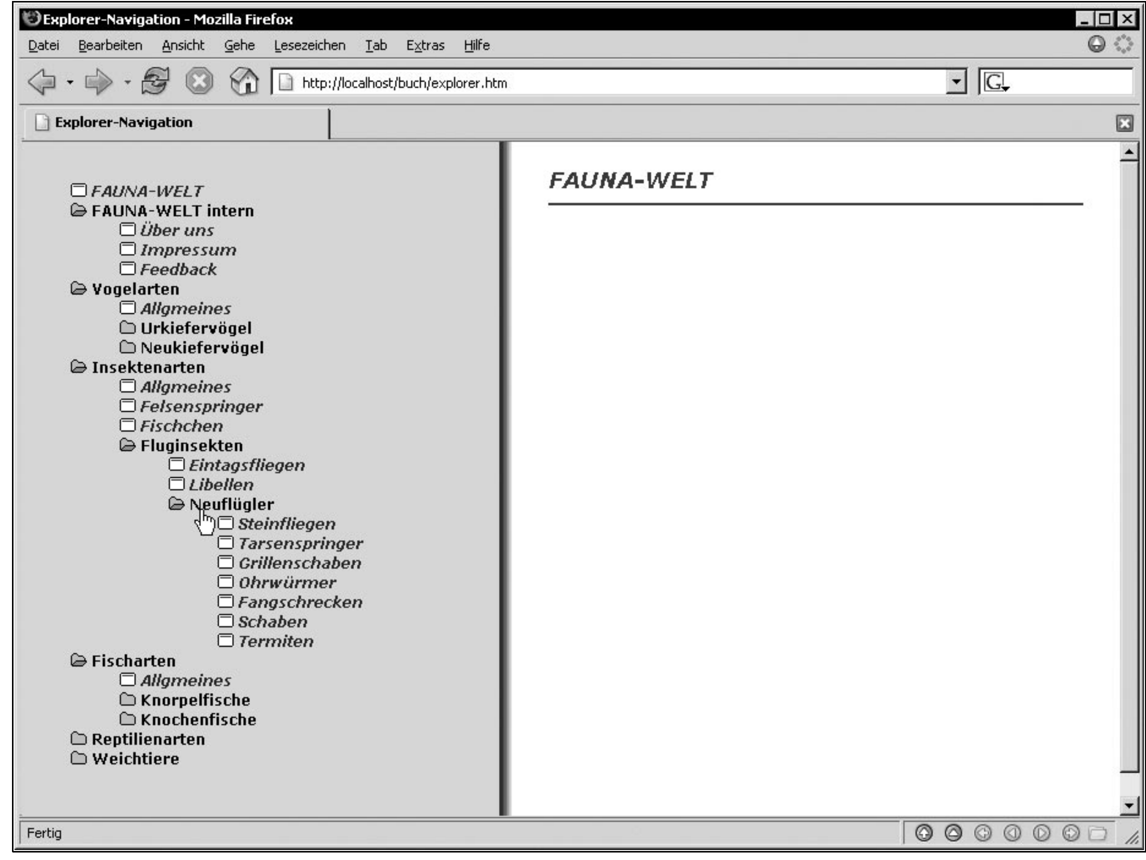


Abbildung 8.2: Angedeutete Fauna-Systematik mithilfe eines explorerartigen Menüs

8.2.2 Quelltexte und Erläuterungen

Zunächst der Quelltext des HTML-Dokuments. Der JavaScript-Code wird in eine externe Datei ausgelagert, da er ja auf allen Seiten wiederverwendet werden soll:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Explorer-Navigation</title>
<link rel="stylesheet" type="text/css" href="explorer.css">
<script type="text/javascript" src="explorer.js"></script>
</head>
```

```
<body onload="init_nav()">
<div id="navigation">
<ul class="opened" id="root">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=001">FAUNA-WELT</a></li>
<li id="faunawelt_intern">
<a class="folder" onclick="toggle('faunawelt_intern',true)">
<img src="opened.gif" alt="" id="img_faunawelt_intern">
FAUNA-WELT intern</a>
<ul id="ul_faunawelt_intern" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=002">Über uns</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=010">Impressum</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=002">Feedback</a></li>
</ul><!-- Ende: ul_faunawelt_intern -->
</li>
<li id="vogelarten">
<a class="folder" onclick="toggle('vogelarten',true);">
<img src="opened.gif" alt="" id="img_vogelarten">
Vogelarten</a>
<ul id="ul_vogelarten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=004">Allgemeines</a></li>
<li id="urkiefervoegel">
<a class="folder" onclick="toggle('urkiefervoegel',true);">
<img src="opened.gif" alt="" id="img_urkiefervoegel">
Urkiefervögel</a>
<ul id="ul_urkiefervoegel" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=024">Laufvögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=028">Steißhühner</a></li>
</ul><!-- Ende: ul_urkiefervoegel -->
</li>
<li id="neukiefervoegel">
<a class="folder" onclick="toggle('neukiefervoegel',true);">
<img src="opened.gif" alt="" id="img_neukiefervoegel">
Neukiefervögel</a>
<ul id="ul_neukiefervoegel" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=064">Hühnervögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=038">Gänsevögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=055">Kranichvögel</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=061">Schreitvögel</a></li>
```

```
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=086">Ruderfüßer</a></li>
</ul><!-- Ende: ul_neukiefervoegel -->
</li>
</ul><!-- Ende: ul_vogelarten -->
</li>
<li id="insektenarten">
<a class="folder" onclick="toggle('insektenarten',true);">
<img src="opened.gif" alt="" id="img_insektenarten">
Insektenarten</a>
<ul id="ul_insektenarten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=108">Allgemeines</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=029">Felsenspringer</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=012">Fischchen</a></li>
<li id="fluginsekten">
<a class="folder" onclick="toggle('fluginsekten',true);">
<img src="opened.gif" alt="" id="img_fluginsekten">
Fluginsekten</a>
<ul id="ul_fluginsekten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=033">Eintagsfliegen</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=012">Libellen</a></li>
<li id="neufluegler">
<a class="folder" onclick="toggle('neufluegler',true);">
<img src="opened.gif" alt="" id="img_neufluegler">
Neuflügler</a>
<ul id="ul_neufluegler" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=093">Steinfliegen</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=065">Tarsenspringer</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=090">Grillenschaben</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=035">Ohrwürmer</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=071">Fangschrecken</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=080">Schaben</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=082">Termiten</a></li>
</ul><!-- Ende: ul_neufluegler -->
</li>
</ul><!-- Ende: ul_fluginsekten -->
</li>
```

```
</ul><!-- Ende: ul_insektenarten -->
</li>
<li id="fischarten">
<a class="folder" onclick="toggle('fischarten',true);">
<img src="opened.gif" alt="" id="img_fischarten">
Fischarten</a>
<ul id="ul_fischarten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=075">Allgemeines</a></li>
<li id="knorpelfische">
<a class="folder" onclick="toggle('knorpelfische',true);">
<img src="opened.gif" alt="" id="img_knorpelfische">
Knorpelfische</a>
<ul id="ul_knorpelfische" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=074">Plattenkiemer</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=098">Seekatzen</a></li>
</ul><!-- Ende: ul_knorpelfische -->
</li>
<li id="knochenfische">
<a class="folder" onclick="toggle('knochenfische',true);">
<img src="opened.gif" alt="" id="img_knochenfische">
Knochenfische</a>
<ul id="ul_knochenfische" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=024">Flössler</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=108">Knorpelganoide</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=102">Knochenganoide</a></li>
<li id="echte_knochenfische">
<a class="folder" onclick="toggle('echte_knochenfische',true);">
<img src="opened.gif" alt="" id="img_echte_knochenfische">
Echte Knochenfische</a>
<ul id="ul_echte_knochenfische" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=049">
Knochenzünglerartige</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=067">Aalartige</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=112">Lachsartige</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=099">Hechtartige</a></li>
</ul><!-- Ende: ul_echte_knochenfische -->
</li>
</ul><!-- Ende: ul_knochenfische -->
</li>
</ul><!-- Ende: ul_fischarten -->
```

```
</li>
<li id="reptilienarten">
<a class="folder" onclick="toggle('reptilienarten',true);">
<img src="opened.gif" alt="" id="img_reptilienarten">
Reptilienarten</a>
<ul id="ul_reptilienarten" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=144">Allgemeines</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=124">Anapsida</a></li>
<li id="romeriida">
<a class="folder" onclick="toggle('romeriida',true);">
<img src="opened.gif" alt="" id="img_romeriida">
Romeriida</a>
<ul id="ul_romeriida" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=029">Diapsida</a></li>
</ul><!-- Ende: ul_romeriida -->
</li>
</ul><!-- Ende: ul_reptilienarten -->
</li>
<li id="weichtiere">
<a class="folder" onclick="toggle('weichtiere',true);">
<img src="opened.gif" alt="" id="img_weichtiere">
Weichtiere</a>
<ul id="ul_weichtiere" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=123">Allgemeines</a></li>
<li id="stachelweichtiere">
<a class="folder" onclick="toggle('stachelweichtiere',true);">
<img src="opened.gif" alt="" id="img_stachelweichtiere">
Stachelweichtiere</a>
<ul id="ul_stachelweichtiere" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=106">Wurmmollusken</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=115">Käferschnecken</a></li>
</ul><!-- Ende: ul_stachelweichtiere -->
</li>
<li id="schalenweichtiere">
<a class="folder" onclick="toggle('schalenweichtiere',true);">
<img src="opened.gif" alt="" id="img_schalenweichtiere">
Stachelweichtiere</a>
<ul id="ul_schalenweichtiere" class="opened">
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=073">Gekrümmtschaler</a></li>
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=079">Gestrecktschaler</a></li>
</ul><!-- Ende: ul_schalenweichtiere -->
</li>
```

```
</ul><!-- Ende: ul_weichtiere -->
</li>
</ul>
</div>
<div id="content">
<h1>FAUNA-WELT intern: über uns</h1>
</div>
</body>
</html>
```

Listing 8.3: HTML-Quelltext mit verschachtelter Aufzählungsliste für Explorer-Menü

Bevor wir auf die Details eingehen, ein Wort zum Thema »Vermeidung von Frames«. Da die Menüstruktur doch recht umfangreich ist und bei Verzicht auf Frames auf jeder Seite wiederholt werden muss, entsteht natürlich eine Menge vielfach wiederholter HTML-Quellcode. Solange mit statischen HTML-Dateien gearbeitet wird, ist es ohne Frames tatsächlich nicht vermeidbar, den Code auf jeder Unterseite zu wiederholen. Sobald jedoch serverseitige Techniken, egal ob Server Side Includes (SSI), PHP oder eine andere Scriptsprache zum Einsatz kommen, kann der entsprechende Code in ein Template ausgelagert werden, dass dann einfach nur dynamisch integriert wird. Auf diese Möglichkeiten gehen wir jedoch erst in Zusammenhang mit serverseitigen Techniken ein.

Der body-Bereich unseres Beispiels besteht aus zwei div-Bereichen namens *navigation* und *content*. Ein Ausschnitt aus der CSS-Datei zeigt, wie das Seitenlayout sich grundsätzlich zusammensetzt:

```
html, body {
   margin:0;
   padding:0;
   height:100%;
}
body {
   background-color:white;
   color:black;
   background-image:url(fauna_back.png);
   background-repeat:repeat-y;
   font-family:Verdana,sans-serif;
}
#navigation {
  position:absolute;
  top:20px;
  left:0px;
  width:400px;
  overflow:hidden;
}
#content {
  margin-top:20px;
  margin-right:30px;
  margin-left:430px;
}
```

Der Bereich für die Navigation ist absolut positioniert und erhält eine großzügige Breite von 400 Pixel. Das ist für stark verzweigte Baumstrukturen durchaus angemessen. Falls der Inhalt zu breit ist, wird er abgeschnitten (`overflow:hidden`). Der hellgrüne Bereich mit dem Schatteneffekt, der als Hintergrund der Navigation dient, ist durch ein Hintergrundbild realisiert. Der Bereich `content` bleibt im normalen Textfluss, damit seine Breite sich an der zur Verfügung stehenden Breite des Browserfensters orientieren kann. Sein linker Rand beträgt 430 Pixel, da 400 Pixel davon dem Bereich für die Navigation gehören.

Der Bereich *navigation* besteht aus einer umfangreichen verschachtelten Aufzählungsliste, also aus ul- und li-Elementen. Das äußerste ul-Element erhält den id-Namen `root`. Dies wird vom Script ausgewertet und ist daher Pflicht.

Zur Erinnerung noch einmal das Prinzip verschachtelter Aufzählungslisten in HTML:

```
<ul>
   <li>Listenpunkt</li>
   <li>Unterliste:
      <ul>
         <li>Listenpunkt</li>
         <li>Listenpunkt</li>
     </ul>
   </li>
</ul>
```

Eine untergeordnete Liste beginnt also damit, dass ein li-Element der übergeordneten Liste mit einem »Überschriftentext« für die Unterliste beginnt. Daran anschließend wird die innere ul-Liste komplett notiert. Abschließend wird das schließende `</li>`-Tag notiert. Untergeordnete Listen müssen also als Elementinhalt eines li-Elements der übergeordneten Liste notiert werden.

In unserem Beispiel erhält jedes ul-Element sowohl einen id-Namen (`id=`) als auch eine Klasse (`class=`), z. B.:

```
<ul id="ul_faunawelt_intern" class="opened">
```

Allen ul-Elementen der Navigation wird der Klassenname `opened` zugewiesen. In CSS ist für diese Klasse Folgendes definiert:

```
ul.opened {
  display:block;
}
```

Das mag trivial erscheinen, da es der Default-Einstellung entspricht. Im Script werden wir jedoch über den Klassennamen das Auf- und Zuklappen von Teilbäumen realisieren und deshalb wird in der CSS-Datei bereits folgende »Parallelklasse« eingerichtet:

```
ul.closed {
  display:none;
}
```

Für beide Klassen werden im Beispiel außerdem folgende Gemeinsamkeiten festgelegt:

```
ul.closed, ul.opened {
  font-size:12px;
  list-style-type:none;
  font-weight:bold;
  line-height:16px;
}
```

Der id-Name jedes ul-Elements sollte den Teilbaum, den die jeweilige ul-Liste enthält, möglichst sprechend charakterisieren. Außerdem – und das ist für unser Script wichtig – *muss* der Name mit `ul_` beginnen. Als Alternative zu sprechenden id-Namen können Sie natürlich auch ein Namensschema wie *ul_section_1_1*, *ul_section_1_2* usw. verwenden. Letzteres hat jedoch den Nachteil, dass es weniger änderungsfreundlich ist, wenn mal neue Seiten eingefügt werden sollen.

Betrachten wir nun ein li-Element, welches eine untergeordnete ul-Liste eröffnet, im Zusammenhang:

```
<li id="faunawelt_intern">
<a class="folder" onclick="toggle('faunawelt_intern'),true">
<img src="opened.gif" alt="" id="img_faunawelt_intern">
FAUNA-WELT intern</a>
<ul id="ul_faunawelt_intern" class="opened">
...
</ul>
</li>
```

Jedes li-Element, das eine untergeordnete ul-Liste enthält, erhält ebenfalls einen id-Namen. Und zwar den gleichen id-Namen wie die ul-Liste, nur ohne voranstehendes `ul_`. Im Code-Ausschnitt hat das li-Element den id-Namen `faunawelt_intern` und die innere ul-Liste folgerichtig den id-Namen `ul_faunaweld_intern`. Dieser Zusammenhang muss bei allen inneren Listen nach der gleichen Regel hergestellt werden, damit unser Script korrekt arbeitet.

Das Gleiche gilt für weitere Regeln zum Aufbau des einleitenden Teils des li-Elements vor der inneren ul-Liste. Dieser Teil besteht aus einem a-Element, welches den Klassennamen `folder` erhalten muss. Diese Klasse ist in CSS wie folgt definiert:

```
a.folder {
  cursor:pointer;
}
```

Da kein href-Attribut notiert ist, wird kein echter Link angezeigt. Der li-Eintrag soll ja auch kein echter Link sein, sondern beim Anklicken lediglich den inneren Teilbaum auf- bzw. wieder zuklappen. Damit er als anklickbar erkennbar ist, erscheint der Mauscursor beim Überfahren wie bei einem Hyperlink.

Das so definierte a-Element bei li-Texten, welche eine untergeordnete ul-Liste einleiten, muss außerdem einen Event-Handler `onclick=` erhalten. Dieser muss als Wertzuweisung einen Aufruf der JavaScript-Funktion `toggle()` erhalten. Als erster Parameter muss der Funktion der id-Name des li-Elements übergeben werden und als zweiter der Wert `true`.

Der Elementinhalt des a-Elements besteht aus einer Grafikreferenz, gefolgt von dem eigentlichen Eintragstext. Die Grafik erhält den Dateinamen *opened.gif*. Sie sollte ein geöffnetes Ordnersymbol darstellen und in der Höhe nicht mehr als 14 Pixel haben. Das img-Element der Grafikreferenz erhält ebenfalls zwingend einen id-Namen, und zwar wieder den gleichen Basisnamen wie das li-Element, jedoch angeführt von der Zeichenfolge `img_`. Auch das ist für das Script wichtig.

Bleiben noch diejenigen li-Elemente, die echte Links zu aufrufbaren Seiten enthalten. Ein Beispielauszug:

```
<li><img src="doc.gif" alt="">
<a class="nav" href="/index.php?id=002">Über uns</a></li>
```

Der Elementinhalt besteht hierbei aus einer Grafik namens *doc.gif*, welche ein Seitensymbol enthalten und ebenfalls nicht höher als 14 Pixel sein sollte. Anschließend folgt der Link zur Seite. Er erhält den Klassennamen `nav`. Für diese Klasse ist in CSS im Beispiel Folgendes definiert:

```
a.nav {
  font-weight:bold;
  font-style:italic;
}
a.nav:link, a.nav:visited, a.nav:hover, a.nav:active {
  color:rgb(8,108,8);
  text-decoration:none;
}
```

Wie aus den vorangehenden Beschreibungen deutlich wird, haben wir eine Menge »Intelligenz« bereits in den HTML-Code verlagert. Das ist erstens die Voraussetzung für ein schlankes JavaScript und zweitens wertet es den HTML-Code semantisch auf, indem durch id- und Klassennamen die inneren Bezüge verdeutlicht werden.

Damit zum Inhalt des Scripts, welches im Beispiel in eine externe Datei namens *explorer.js* ausgelagert wurde:

```
function init_nav() {
   for(i = 0; i < document.getElementsByTagName('ul').length;
       i++) {
      if(document.getElementsByTagName('ul')[i].className ==
      "opened") {
         id =
         document.getElementsByTagName('ul')[i].parentNode.id;
         toggle(id, false);
      }
   }
   if(window.name.length > 0)
      load_nav();
}
function toggle(id, save) {
   ul = "ul_" + id;
   img = "img_" + id;
   ul_element = document.getElementById(ul);
   img_element = document.getElementById(img);
```

```
    if(ul_element) {
        if(ul_element.className == 'closed') {
           ul_element.className = "opened";
           img_element.src = "opened.gif";
        }
        else {
           ul_element.className = "closed";
           img_element.src = "closed.gif";
        }
    }
    if(save == true) save_nav();
}
function save_nav() {
    var save = "";
    for(var i = 0; i < document.getElementsByTagName('ul').length;
    i++) {
       if((document.getElementsByTagName('ul')[i].className ==
       "opened" ||
       document.getElementsByTagName('ul')[i].className ==
       "closed") && document.getElementsByTagName('ul')[i].id !=
       'root')
          save = save + document.getElementsByTagName('ul')[i].id
          + "=" + document.getElementsByTagName('ul')[i].className
          + ",";
    }
    if(save.lastIndexOf(",") > 0)
       save = save.substring(0, save.lastIndexOf(","));
    window.name = save;
}
function load_nav() {
    var items = window.name.split(",");
    if(items.length > 0) {
       for(var i = 0; i < items.length; i++) {
          id_value = items[i].split("=");
          if(id_value.length == 2) {
             id = id_value[0];
             value = id_value[1];
             document.getElementById(id).className = value;
             img = "img_" + id.substring(3, id.length);
             img_element = document.getElementById(img);
             if(value == "closed")
                img_element.src = "closed.gif";
             else
                img_element.src = "opened.gif";
          }
       }
    }
}
```

Listing 8.4: JavaScript für Explorer-Menü

Im HTML-Code wird die Verbindung zum JavaScript an zwei Stellen hergestellt. Im einleitenden <body>-Tag ist onload="init_nav()" notiert. Die Funktion init_nav() ist im Script als Erste von insgesamt vier Funktionen notiert. Sie geht zunächst in einer for-Schleife alle ul-Elemente durch und erfragt in einer if-Bedingung, ob das jeweils aktuell betrachtete ul-Element den Klassennamen opened hat. Ist das der Fall, so wird das Elementobjekt des Elternelements des aktuellen ul-Elements in einer Variablen namens id gespeichert. Bei allen ul-Elementen außer der äußersten ul-Liste ist das Elternelement das li-Element, welches die innere ul-Liste einleitet. Dessen Elementobjekt wird in id gespeichert. Anschließend wird die Funktion toggle() aufgerufen. Ihr wird als erster Parameter die Variable id übergeben und als zweiter Parameter der Wert false. Auf Letzteren kommen wir später noch zurück.

Die Funktion toggle() tut Folgendes: Wenn ein Teilbaum der Navigation aufgeklappt ist, schließt sie ihn. Ist er dagegen geschlossen, klappt sie ihn auf. Dazu benutzt sie die Klassennamen, welche die ul-Listen mit dem jeweiligen Teilbaum enthalten. In HTML – wir erinnern uns – haben wir bewusst alle ul-Elemente von Teilbäumen mit dem Klassennamen opened ausgestattet. In CSS bewirkt dieser display:block also eine normale Anzeige. Der Grund ist, dass bei ausgeschaltetem JavaScript der gesamte Navigationsbaum aufgeklappt angezeigt werden soll. Im Script möchten wir jedoch anfänglich einen Zustand herstellen, der den Baum zunächst zugeklappt darstellt. Genau das leistet die for-Schleife in init_nav(), die alle betroffenen Elemente ermittelt und durch Aufruf von toggle() dafür sorgt, dass diese zugeklappt werden.

Damit aber nicht genug. Nachdem die for-Schleife abgearbeitet ist, wird abgefragt, ob die Eigenschaft window.name eine Länge von mehr als 0 Zeichen hat. In ***Abschnitt 7.3.12*** wurde diese Default-Eigenschaft vorgestellt und gezeigt, wie sich damit JavaScript-Informationen seitenübergreifend speichern lassen. Unser Script benutzt diese Eigenschaft, um den aktuellen Auf-/Zuklappzustand der Navigation zu speichern und wieder laden zu können. Allein diesem Zweck dienen die beiden letzten Funktionen im Script, save_nav() und load_nav(). Die Funktion toggle() ruft gegebenenfalls save_nav() auf, um den aktuellen Menüzustand zu speichern, und init_nav() ruft load_nav() auf, um einen gespeicherten Menüzustand wiederherzustellen. Ist in window.name also etwas gespeichert, dann kommt load_nav() zum Einsatz.

Die zweite Schnittstelle zwischen HTML und dem JavaScript-Code sind die Pseudo-Links in den li-Elementen, welche eine untergeordnete ul-Liste enthalten. Zur Erinnerung nochmals der Beispielauszug:

```
<li id="faunawelt_intern">
<a class="folder" onclick="toggle('faunawelt_intern',true)">
<img src="opened.gif" alt="" id="img_faunawelt_intern">
FAUNA-WELT intern</a>
<ul id="ul_faunawelt_intern" class="opened">
...
</ul>
</li>
```

Im Event-Handler `onclick=` wird `toggle()` aufgerufen. Dabei wird als erster Parameter der id-Name des li-Elements übergeben und als zweiter der Wert `true`. Der zweite Parameter bestimmt, ob `toggle()` den aktuellen Menüzustand in `window.name` speichern lassen soll oder nicht. In `init_nav()` wird `toggle()` mit `false` aufgerufen. In diesem Fall wird nichts gespeichert. Da beim Anklicken von Ordner-Links jedoch eine vom Anwender bewirkte Zustandsveränderung im Menü eintritt, wird `true` übergeben, damit der neue Zustand gespeichert wird.

Die Funktion `toggle()` greift auf zwei Elemente zu: auf das ul-Element des untergeordneten Teilbaums und auf die Grafikreferenz, die je nach Aufklappzustand des Teilbaums entweder ein geöffnetes oder ein geschlossenes Ordnersymbol darstellen soll. Da sie als ersten Parameter den id-Namen des den Teilbaum öffnenden li-Elements übergeben bekommt, kann sie auf Basis der vereinbarten Regel durch Voranstellen von `ul_` bzw. `img_` die id-Namen der betroffenen Elemente selbst ermitteln. Das Herzstück des Auf-/Zuklappeffekts ist das if-else-Konstrukt innerhalb von `toggle()`. Ist der aktuelle Klassenname des inneren ul-Elements `opened`, wird er in `closed` geändert, und andernfalls umgekehrt: `closed` wird zu `opened`. Da der Klasse `closed` in CSS mit `display:none` zugewiesen wurde, verschwindet die Anzeige des Teilbaums, sobald diese Klasse zugewiesen ist. Damit sich auch das Ordnersymbol entsprechend ändert, wird durch Zuweisen einer anderen Grafik an das src-Attribut der Grafikreferenz im öffnenden li-Element der Klappeffekt zusätzlich visualisiert.

Die Funktionen `save_nav()` und `load_nav()` haben wie schon erwähnt nichts unmittelbar mit der Funktionalität des Auf-/Zuklappeffekts zu tun. In `save_nav()` besorgt eine for-Schleife die Zuweisung von id-Namen zu aktuellen Klassennamen und fügt die Einzelwerte durch Kommata getrennt zu einer Zeichenkette zusammen. Diese wird am Ende in `window.name` gespeichert. In `load_nav()` wird `window.name` wieder ausgelesen und die darin gespeicherte Zeichenkette wird durch Anwendung der Zeichenkettenmethode `split()` wieder in ihre Einzelwerte zerlegt. Ähnlich wie in `toggle()` werden die ermittelten Klassennamen den zugehörigen, über id-Namen ermittelten Elementen zugewiesen, jedoch nicht nur einem einzigen Element, sondern allen Elementen des Menüs. Auf diese Weise wird nach Aufruf einer anderen Seite der Menüzustand wiederhergestellt, der nach Verlassen der letzten Seite aktuell war.

Fazit

Das Beispiel zeigt, dass man in Zeiten des DOM nicht mehr viel JavaScript benötigt und außerdem noch ein sauberes HTML-Markup einsetzen kann, um die allseits beliebten Baumnavigationen zu realisieren.

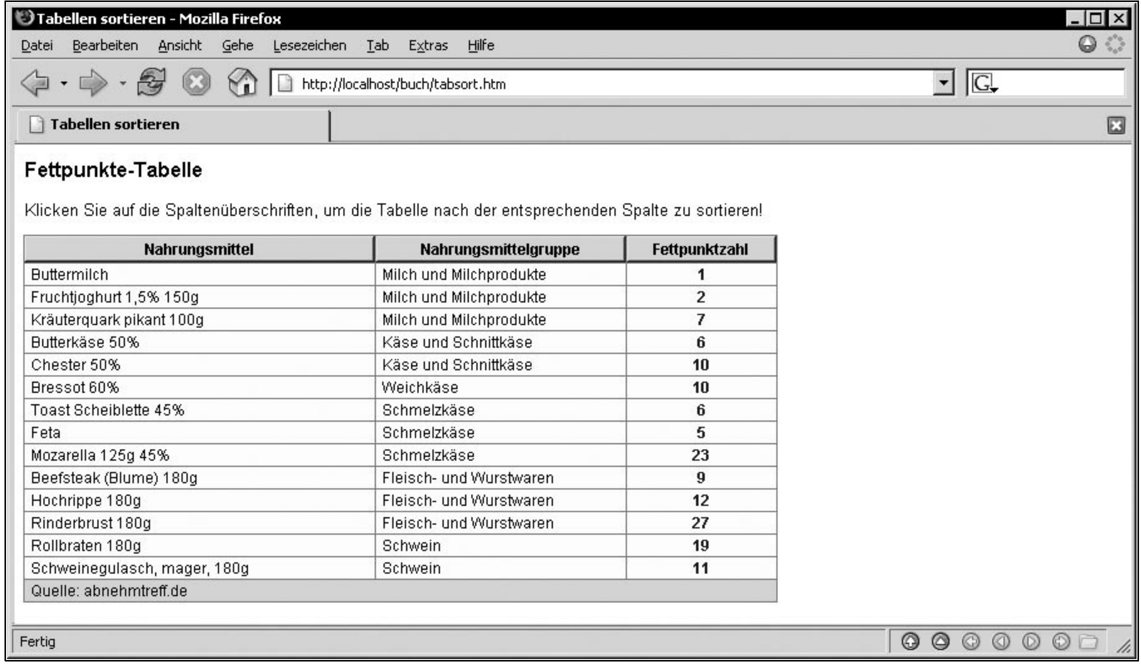
8.3 Interaktive Tabellensortierung

Es gibt zahlreiche Informationen, die sich am besten in einer Tabelle darstellen lassen. In HTML müssen Sie sich dabei entscheiden, nach welchem Kriterium Sie die Tabellendaten sortieren, oder Sie verzichten ganz auf eine Sortierung.

Gerade bei größeren Tabellen wünscht sich mancher Anwender jedoch, die Tabellenzeilen nach bestimmten Spalten sortieren zu können. Das steigert die Informationsauswertung. Gewöhnt sind die Anwender das Feature aus zahlreichen Desktop- und Internetanwendungen, etwa aus E-Mail-Programmen, wo sich Nachrichtenlisten wahlweise nach Absender, Eingangsdatum usw. sortieren lassen.

Auf Webseiten sucht man diese Möglichkeit dagegen meist vergebens. Dabei ist gerade die Möglichkeit, größere Tabellen dynamisch zu sortieren, eine der interessantesten und nützlichsten Anwendungen für JavaScript. Mit diesem Beispiel möchten wir ein Script vorstellen, das wiederverwendbar eingesetzt werden kann und mit dessen Hilfe Webseiten mit tabellarischen Informationen eine echte Aufwertung erfahren.

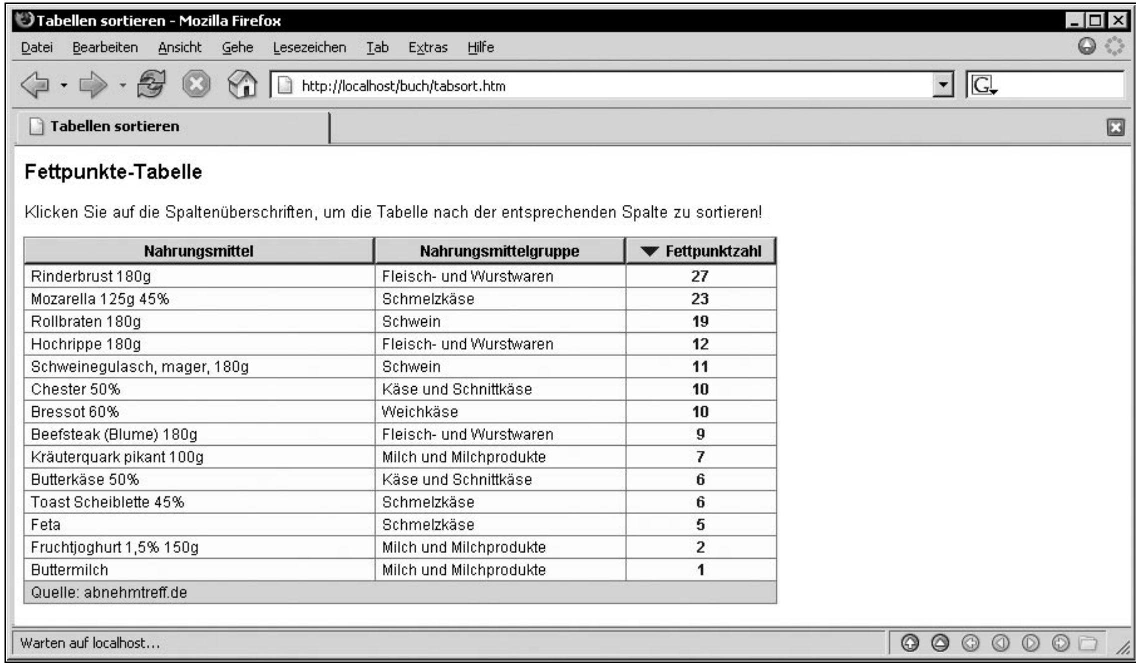
8.3.1 Das Beispiel



Nahrungsmittel	Nahrungsmittelgruppe	Fettpunktzahl
Buttermilch	Milch und Milchprodukte	1
Fruchtjoghurt 1,5% 150g	Milch und Milchprodukte	2
Kräuterquark pikant 100g	Milch und Milchprodukte	7
Butterkäse 50%	Käse und Schnittkäse	6
Chester 50%	Käse und Schnittkäse	10
Bressot 60%	Weichkäse	10
Toast Scheiblette 45%	Schmelzkäse	6
Feta	Schmelzkäse	5
Mozarella 125g 45%	Schmelzkäse	23
Beefsteak (Blume) 180g	Fleisch- und Wurstwaren	9
Hochrippe 180g	Fleisch- und Wurstwaren	12
Rinderbrust 180g	Fleisch- und Wurstwaren	27
Rollbraten 180g	Schwein	19
Schweinegulasch, mager, 180g	Schwein	11
Quelle: abnehmtreff.de		

Abbildung 8.3: Tabelle im HTML-Originalzustand

Die abgebildete Tabelle ist durch Anklicken der Spaltenüberschriften nach der angeklickten Spalte sortierbar:



Nahrungsmittel	Nahrungsmittelgruppe	▼ Fettpunktzahl
Rinderbrust 180g	Fleisch- und Wurstwaren	27
Mozarella 125g 45%	Schmelzkäse	23
Rollbraten 180g	Schwein	19
Hochrippe 180g	Fleisch- und Wurstwaren	12
Schweinegulasch, mager, 180g	Schwein	11
Chester 50%	Käse und Schnittkäse	10
Bressot 60%	Weichkäse	10
Beefsteak (Blume) 180g	Fleisch- und Wurstwaren	9
Kräuterquark pikant 100g	Milch und Milchprodukte	7
Butterkäse 50%	Käse und Schnittkäse	6
Toast Scheiblette 45%	Schmelzkäse	6
Feta	Schmelzkäse	5
Fruchtjoghurt 1,5% 150g	Milch und Milchprodukte	2
Buttermilch	Milch und Milchprodukte	1
Quelle: abnehmtreff.de		

Abbildung 8.4: Tabelle nach letzter Spalte absteigend sortiert

Eine kleine Grafik vor der Spaltenüberschrift der Sortierspalte zeigt an, nach welcher Spalte die Tabelle aktuell sortiert ist, und zwar, ob absteigend oder aufsteigend. Es kann sowohl alphanumerisch als auch numerisch sortiert werden.

Unser Beispielscript kann beliebige Tabellen mit definierten Spalten verarbeiten, auch mehrere unterschiedliche auf einer Seite.

8.3.2 Quelltexte und Erläuterungen

Zunächst der Quelltext des HTML-Dokuments mit der zu sortierenden Tabelle. Das JavaScript haben wir in eine externe Datei ausgelagert.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
        "http://www.w3.org/TR/html4/strict.dtd">
<html lang="de">
<head>
<meta http-equiv="content-type" content="text/html;
      charset=ISO-8859-1">
<title>Tabellen sortieren</title>
<style type="text/css">
table {
   border-collapse:collapse;
```

```
      border:1px solid #999999;
   }
   th {
      vertical-align:middle;
      text-align:left;
      background-color:#DDDDDD;
      border:1px solid #999999;
      padding:0px;
   }
   th button {
      width:100%;
      background-color:#DDDDDD;
      font-family:Arial,sans-serif;
      font-size:12px;
      font-weight:bold;
      text-align:left;
      margin:0px;
   }
   td {
      font-family:Arial,sans-serif;
      font-size:12px;
      text-align:left;
      vertical-align:top;
      background-color:#FFFFDD;
      border:1px solid #999999;
      padding-left:5px;
      padding-right:5px;
      padding-top:1px;
      padding-bottom:1px;
   }
   td.fp {
      font-weight:bold;
      color:blue;
      text-align:center;
   }
   td.tfoot {
      background-color:#DDDDDD;
   }
   h1 {
      font-family:Arial,sans-serif;
      font-size:16px;
   }
   p {
      font-family:Arial,sans-serif;
      font-size:13px;
   }
   </style>
   <script type="text/javascript" src="tabsort.js"></script>
   </head>
```

```
<body onload="init_tabsort('fettpunkttabelle')">
<h1>Fettpunkte-Tabelle</h1>
<p>Klicken Sie auf die Spaltenüberschriften,
um die Tabelle nach der entsprechenden Spalte
zu sortieren!</p>
<form action="javascript:void(0)">
<table id="fettpunkttabelle" border="1"
summary="Tabelle, in der für bekannte Nahrungsmittel und
Nahrungsmittelgruppen Fettpunktzahlen angegeben werden">
<colgroup>
<col style="width:280px">
<col style="width:200px">
<col style="width:120px">
</colgroup>
<thead>
<tr>
<th><button type="button"
onClick="tabsort('fettpunkttabelle','an',0)">
<span id="fettpunkttabelle_sort_img_0"></span>
Nahrungsmittel</button></th>
<th><button  type="button"
onClick="tabsort('fettpunkttabelle','an',1)">
<span id="fettpunkttabelle_sort_img_1"></span>
Nahrungsmittelgruppe</button></th>
<th><button  type="button"
onClick="tabsort('fettpunkttabelle','n',2)">
<span id="fettpunkttabelle_sort_img_2"></span>
Fettpunktzahl</button></th>
</tr>
</thead>
<tfoot>
<tr>
<td colspan="3" class="tfoot">
Quelle: abnehmtreff.de
</td>
</tr>
</tfoot>
<tbody>
<tr>
<td>Buttermilch</td>
<td>Milch und Milchprodukte</td>
<td class="fp">1</td>
</tr>
<tr>
<td>Fruchtjoghurt 1,5% 150g</td>
<td>Milch und Milchprodukte</td>
<td class="fp">2</td>
</tr>
<tr>
```

```
<td>Kräuterquark pikant 100g</td>
<td>Milch und Milchprodukte</td>
<td class="fp">7</td>
</tr>
<tr>
<td>Butterkäse 50%</td>
<td>Käse und Schnittkäse</td>
<td class="fp">6</td>
</tr>
<tr>
<td>Chester 50%</td>
<td>Käse und Schnittkäse</td>
<td class="fp">10</td>
</tr>
<tr>
<td>Bressot 60%</td>
<td>Weichkäse</td>
<td class="fp">10</td>
</tr>
<tr>
<td>Toast Scheiblette 45%</td>
<td>Schmelzkäse</td>
<td class="fp">6</td>
</tr>
<tr>
<td>Feta</td>
<td>Schmelzkäse</td>
<td class="fp">5</td>
</tr>
<tr>
<td>Mozarella 125g 45%</td>
<td>Schmelzkäse</td>
<td class="fp">23</td>
</tr>
<tr>
<td>Beefsteak (Blume) 180g</td>
<td>Fleisch- und Wurstwaren</td>
<td class="fp">9</td>
</tr>
<tr>
<td>Hochrippe 180g</td>
<td>Fleisch- und Wurstwaren</td>
<td class="fp">12</td>
</tr>
<tr>
<td>Rinderbrust 180g</td>
<td>Fleisch- und Wurstwaren</td>
<td class="fp">27</td>
</tr>
```

```
<tr>
<td>Rollbraten 180g</td>
<td>Schwein</td>
<td class="fp">19</td>
</tr>
<tr>
<td>Schweinegulasch, mager, 180g</td>
<td>Schwein</td>
<td class="fp">11</td>
</tr>
</tbody>
</table>
</form>
</body>
</html>
```

Listing 8.5: HTML-Datei mit Tabelle zum Sortieren

Eine Tabelle, welche mit unserem Beispielscript sortiert werden soll, muss im einleitenden <table>-Tag mit `id=` einen id-Namen erhalten. Weiterhin muss sie unterhalb des <table>-Tags mithilfe eines colgroup-Elements alle Spalten der Tabelle in col-Elementen vordefinieren. Die col-Elemente sind auch der beste Ort, um die gewünschten Spaltenbreiten vorzugeben. Daneben muss die zu sortierende Tabelle zumindest die Bereiche `<thead> ... </thead>` und `<tbody> ... </tbody>` explizit durch diese Tags markiert enthalten.

Der Inhalt des thead-Elements besteht aus einer Tabellenzeile mit den Spaltenüberschriften. In unserem Beispiel haben wir es so gelöst, dass der Inhalt der Kopfzellen jeweils aus einem anklickbaren Button (`<button> ... </button>`) besteht. Mit dem Selektor `th button` wird im zentralen Style-Bereich für solche Buttons unter anderem festgelegt, dass sie 100% der verfügbaren Breite einnehmen sollen. So erreichen wir, dass die Buttons die gesamte Tabellenzelle einnehmen. Jedes button-Element enthält einen Event-Handler `onclick=`, der beim Anklicken dafür sorgt, dass die Tabelle sortiert wird. Dazu später noch mehr.

Der Elementinhalt jedes Buttons besteht aus einem leeren span-Element und der Spaltenüberschrift. Das span-Element ist der Behälter für die kleinen Grafiken, welche in der Sortierspalte erscheinen und auf- oder absteigende Sortierung anzeigen. Jedes dieser span-Elemente muss einen id-Namen erhalten, der folgender Konvention genügen muss: Er beginnt mit dem id-Namen der Tabelle, gefolgt von `_sort_img_` und einer Laufnummer. Die erste Spalte bekommt eine 0, die zweite eine 1, die dritte eine 2 usw. Diese Konvention ist für das Script wichtig, um die Grafiken korrekt zu platzieren.

Ansonsten sind keine weiteren Konventionen erforderlich. Die Tabellendaten können wie üblich notiert werden.

Das Script, das die Tabelle sortiert, ist über ein script-Element in den Kopfdaten des HTML-Dokuments als externe Datei namens *tabsort.js* eingebunden. Sein Quelltext lautet:

```
var default_order = "asc"; // "asc" || "desc"
var asc_img_uri = "asc.gif";
var desc_img_uri = "desc.gif";
var asc_img_alt_text =
    "aufsteigend sortiert: ändern in absteigend sortiert";
var desc_img_alt_text =
    "absteigend sortiert: ändern in aufsteigend sortiert";
var tables = new Array();
var ti = 0;
function sort_table(id) {
   this.id = id;
   this.obj;
   this.tbody_obj;
   this.sort_cols = 0;
   this.sort_rows = 0;
   this.img_ids = new Array();
   this.tr_ids = new Array();
   this.td_ids = new Array();
   this.active_sort_col;
   this.active_sort_order = default_order;
}
function init_tabsort(table_id) {
   if(!document.getElementById) return;
   tables[ti] = new sort_table(table_id);
   var t = ti;
   ti += 1;
   tables[t].obj = document.getElementById(table_id);
   for(var i = 0; i < tables[t].obj.childNodes.length; i++) {
    switch(tables[t].obj.childNodes[i].nodeName.toLowerCase()) {
         case 'colgroup':
            for(j = 0;
            j < tables[t].obj.childNodes[i].childNodes.length;
            j++)
if(tables[t].obj.childNodes[i].childNodes[j].nodeName.
toLowerCase() == 'col')
               tables[t].sort_cols += 1;
            break;
         case 'tbody':
            tables[t].tbody_obj = tables[t].obj.childNodes[i];
            break;
         default: break;
      }
   }
   var tri = 0;
   var tdi = 0;
   for(i = 0; i < tables[t].tbody_obj.childNodes.length; i++) {
if(tables[t].tbody_obj.childNodes[i].nodeName.toLowerCase()
      == 'tr') {
         tables[t].tbody_obj.childNodes[i].id = 'tr_' + tri;
```

```
         tables[t].tr_ids.push('tr_' + tri);
         tables[t].sort_rows += 1;
         tdi = 0;
         for(j = 0;
         j < tables[t].tbody_obj.childNodes[i].childNodes.length; j++) {
if(tables[t].tbody_obj.childNodes[i].childNodes[j].nodeName.toLowerCase() == 'td') {
   tables[t].tbody_obj.childNodes[i].childNodes[j].id
   = 'tr_' + tri + '_td_' + tdi;
   tables[t].td_ids.push('tr_' + tri + '_td_' + tdi);
               tdi += 1;
            }
         }
         tri += 1;
      }
   }
}
function get_sort_table_obj_by_id(id) {
   for(var i = 0; i < tables.length; i++)
      if(tables[i].id == id)
         return(tables[i]);
}
function numsort(a,b) {
   return parseFloat(a) - parseFloat(b);
}
function tabsort(table_id, sort_type, col_n) {
   if(!document.getElementById) return;
   var delimiter = "|||||";
   var to = get_sort_table_obj_by_id(table_id);
   var sort_array = new Array();
   for(var i = 0; i < to.sort_rows; i++) {
      tr_array = new Array();
      tr_id = 'tr_' + i;
      td_id = tr_id + '_td_' + col_n;
      tr_array.push(document.getElementById(td_id).innerHTML);
      for(var j = 0; j < to.sort_cols; j++) {
          if(j == col_n) continue;
          else {
             tr_id = 'tr_' + i;
             td_id = tr_id + '_td_' + j;
             tr_array.push
             (document.getElementById(td_id).innerHTML);
          }
      }
      tr_str = tr_array.join(delimiter);
      sort_array.push(tr_str);
   }
   if(sort_type == 'an')
      sort_array.sort();
   else if(sort_type == 'n')
```

```
      sort_array.sort(numsort);
   if(to.active_sort_order == 'desc') {
      sort_array.reverse();
      to.active_sort_order = 'asc';
   }
   else
      to.active_sort_order = 'desc';
   for(i = 0; i < sort_array.length; i++) {
      tr_array = sort_array[i].split(delimiter);
      first = tr_array[0];
      col = tr_array[col_n];
      tr_array[col_n] = first;
      tr_array[0] = col;
      for(j = 0; j < to.sort_cols; j++) {
         tr_id = 'tr_' + i;
         td_id = tr_id + '_td_' + j;
         document.getElementById(td_id).innerHTML = tr_array[j];
      }
   }
   var img_html;
   if(to.active_sort_order == 'asc')
      img_html = '<img src="' + desc_img_uri + '" alt="' +
      desc_img_alt_text + '">';
   else
      img_html = '<img src="' + asc_img_uri + '" alt="' +
      asc_img_alt_text + '">';
   for(i = 0; i < to.sort_cols; i++) {
      if(i == col_n)
document.getElementById(table_id + '_sort_img_' + i).innerHTML =
img_html + " ";
      else
         document.getElementById(table_id + '_sort_img_' + i).innerHTML = "";
   }
}
```

Listing 8.6: JavaScript für HTML-Tabellensortierung

Zu Beginn des Scripts werden einige globale Variablen deklariert und initialisiert. Mit `default_order` kann festgelegt werden, ob eine Tabelle beim ersten Anklicken einer Spalte nach deren Werten zunächst aufsteigend (`asc`) oder absteigend (`desc`) sortiert werden soll. Danach wird bei jedem weiteren Klick automatisch umgeschaltet. Die beiden URIs der Grafiken, die aufsteigende und absteigende Sortierung signalisieren, lassen sich natürlich anpassen, ebenso wie die zugehörigen Alternativtexte.

Die Variable `delimiter` verdient besondere Aufmerksamkeit. Die zugewiesene Zeichenkette, im Beispiel fünf Senkrechtstriche, darf innerhalb der Tabelle nicht in den Daten vorkommen. Das Script benötigt diese Variable, um einen Tabellenzeilen-Array temporär zu einem String zusammenzusetzen und diesen anhand des Begrenzers (Delimiter) anschließend wieder in seine Einzelwerte zerlegen zu können.

Ferner wird beim Einlesen des Scripts ein Array namens `tables` angelegt sowie eine Array-Zählervariable `ti`. Alle übrigen Anweisungen des Scripts stehen innerhalb von Funktionen und werden daher erst bei Aufruf der Funktionen ausgeführt. Insgesamt kommen folgende Funktionen vor:

- **`sort_table()`:** Diese Funktion dient zur Erstellung eines Objekts. Das Script legt für jede zu sortierende Tabelle eine Instanz dieses Objekts an und speichert es im zuvor deklarierten Array `tables`. Auf diese Weise wird, falls mehrere sortierbare Tabellen auf einer Seite vorkommen, eine saubere Datenhaltung erreicht.
- **`init_tabsort()`:** Diese Funktion initialisiert die Sortierbarkeit einer Tabelle. Aufgerufen werden muss sie, bevor der Anwender einen Sortiervorgang anstößt. Der richtige Ort, um diese Funktion aufzurufen, ist der Event-Handler `onload=` im einleitenden `<body>`-Tag. Für jede sortierbare Tabelle auf einer Seite muss diese Funktion aufgerufen werden. Dabei muss ihr der id-Name der zu sortierenden Tabelle als Parameter übergeben werden.
- **`get_sort_table_obj_by_id()`:** Dies ist eine Hilfsfunktion des Script, mit deren Hilfe eine Anweisung schnell ein im Array `tables` gespeichertes Objekt des Typs `sort_table` auffinden kann.
- **`numsort()`:** Diese Hilfsfunktion wird benötigt, um bei Anwendung der sort()-Methode des Array-Objekts eine numerische Sortierung zu ermöglichen.
- **`tabsort()`:** Diese Funktion führt die eigentliche Sortierung durch. Sie wird über den Event-Handler `onclick=` beim Anklicken der Buttons in den Tabellenüberschriften aufgerufen.

Die beiden »großen« Funktionen sind also `init_tabsort()` und `tabsort()`. In `init_tabsort()` wird, nachdem alle nicht-DOM-fähigen Browser wieder heimgeschickt wurden, in der Array-Variablen `tables[ti]` ein neues Objekt des Typs `sort_table` gespeichert. Dazu wird die Objektfunktion `sort_table()` aufgerufen.

Das sort_table-Objekt hat unter anderem eine Eigenschaft `obj`. In dieser wird das DOM-Objekt der gesamten Tabelle gespeichert, indem `document.getElementById(table_id)` an `tables[t].obj` zugewiesen wird. Über `tables[t].obj` ist dadurch innerhalb der Funktion fortan der DOM-Zugriff auf die Tabelle möglich.

Im weiteren Verlauf analysiert `init_tabsort()` die Tabelle. Nun wird auch klar, warum die Tabelle in HTML ein paar Konventionen einhalten muss, die bereits beschrieben wurden. Das colgroup-Element wird gesucht und die Anzahl der Spalten wird ermittelt. Der gesamte Tabellenkörper, identifizierbar am tbody-Element, wird als Objekt in der Objekteigenschaft `tables[t].tbody_obj` gespeichert.

In der größeren, verschachtelten for-Schleife im zweiten Teil von `init_tabsort()` passiert Folgendes: jedes tr- und jedes td-Element erhält einen eindeutigen id-Namen, damit später ein möglichst einfacher Zugriff darauf möglich ist. Die id-Namensvergabe folgt dabei einem bestimmten Schema. Betrachten wir folgenden HTML-Ausschnitt:

```
<tr>
<td>Buttermilch</td>
<td>Milch und Milchprodukte</td>
```

```
<td class="fp">1</td>
</tr>
<tr>
<td>Fruchtjoghurt 1,5% 150g</td>
<td>Milch und Milchprodukte</td>
<td class="fp">2</td>
</tr>
```

Daraus macht `init_tabsort()` im Arbeitsspeicher folgenden Code:

```
<tr id="tr_0">
<td id="tr_0_td_0">Buttermilch</td>
<td id="tr_0_td_1">Milch und Milchprodukte</td>
<td id="tr_0_td_2" class="fp">1</td>
</tr>
<tr id="tr_1">
<td id="tr_1_td_0">Fruchtjoghurt 1,5% 150g</td>
<td id="tr_1_td_1">Milch und Milchprodukte</td>
<td id="tr_1_td_2" class="fp">2</td>
</tr>
```

Die tr-Elemente des sortierbaren Tabellenkörpers werden also mit `tr_0`, `tr_1`, `tr_2` usw. »nummeriert«. Innerhalb einer Tabellenzeile erhalten alle td-Elemente den tr-Namen als vorderen id-Namensteil. Angehängt wird dann für jede Spalte `_td_0` für die erste Spalte, `_td_1` für die zweite Spalte, `_td_2` für die dritte usw. Der anfängliche DOM-Zugriff auf die Tabellenzeilen und -zellen ist etwas komplex. Greifen wir zum besseren Verständnis die folgende if-Abfrage heraus:

```
if(tables[t].tbody_obj.childNodes[i].childNodes[j].nodeName.toLowerCase() == 'td')
```

Wie weiter oben bereits erwähnt, ist in `tables[t].tbody_obj` der tbody-Elementknoten der sortierbaren Tabelle gespeichert. Über `tables[t].tbody_obj.childNodes` ist der DOM-Zugriff auf alle Kindknoten dieses Elementknotens möglich. Dazu gehören neben anderen Knoten auch die Elementknoten der inneren tr-Elemente. In der herausgegriffenen if-Abfrage, die innerhalb einer for-Schleife steht, ist in `childNodes[i]` ein tr-Element ermittelt worden. Die Abfrage sucht jedoch nach td-Elementen. Also muss sie noch weiter in der Hierarchie herabsteigen und abermals nach `childNodes` suchen, diesmal nach denen des tr-Elements. Die if-Abfrage steht nicht nur innerhalb einer for-Schleife, sondern innerhalb einer zweiten. In `childNodes[j]` (`j` ist der Schleifenzähler der inneren for-Schleife, `i` derjenige der äußeren!) ist ein aktueller Kindknoten des tr-Elements (also des Elementknotens `tbody_obj.childNodes[i]`) gespeichert. Über die Objekteigenschaft `nodeName` wird der Knotenname ermittelt. Durch Anwendung von `toLowerCase()` wird dieser Wert klein geschrieben interpretiert. Wenn das Kindobjekt `td` heißt, dann ist die if-Bedingung erfüllt.

Nachdem die Funktion `init_tabsort()` beendet wurde, kann die betroffene Tabelle nach jeder ihrer Spalten auf- und absteigend, alphanumerisch oder numerisch sortiert werden. Angestoßen wird die Sortierung im Beispiel wie schon angedeutet über onclick-Event-Handler in den button-Elementen der Spaltenüberschriften. Den onclick-Event-Handlern wird dabei als Wert ein Aufruf der Funktion `tabsort()` zugewiesen. Die Funktion erwartet drei Parameter. Als erster Parameter muss der id-Name

der Tabelle übergeben werden – in unserem Beispiel lautet er `fettpunkttabelle`. Als zweiter Parameter wird entweder `an` oder `n` übergeben. Mit `an` wird eine alphanumerische Sortierung angewiesen und mit `n` eine numerische. In unserem Beispiel enthält die dritte Spalte nur Zahlenwerte. Damit diese wie gewünscht sortiert werden, wird für diese Spalte eine numerische Sortierung angewiesen. Die beiden anderen Spalten enthalten Zeichenketten als Werte, daher wird für sie `an` übergeben. Als dritter Parameter muss schließlich noch die Indexnummer der zu sortierenden Spalte übergeben werden. Bei der ersten Spalte ist dies 0, bei der zweiten 1, bei der dritten 2 usw.

In `tabsort()` wird über die Funktion `get_sort_table_obj_by_id()` das im Array `tables` gespeicherte sort_table-Objekt der betroffenen Tabelle geholt und in der Variablen `to` gespeichert. Über `to` ist damit der Zugriff auf das sort_table-Objekt der Tabelle und damit auch auf die darin gespeicherten Objekte der Tabelle selbst möglich.

Als Nächstes wird ein Array namens `sort_array` erzeugt. Was innerhalb der anschließenden for-Schleife passiert, ist Folgendes: Für jede Tabellenzeile des sortierbaren Tabellenkörpers werden die einzelnen Werte aus den Zellen dieser Zeile geholt. Aus den Werten wird eine Zeichenkette zusammengesetzt, die in der Variablen `tr_str` gespeichert wird. Zwischen den Werten der Zellinhalte wird jeweils die Zeichenfolge eingefügt, die in der globalen Variable `delimiter` gespeichert ist. Das ist auch der Grund, warum diese Zeichenfolge in den Tabellendaten selbst nicht vorkommen darf.

Angenommen, folgende HTML-Tabellenzeile wird verarbeitet:

```
<tr>
<td>Buttermilch</td>
<td>Milch und Milchprodukte</td>
<td class="fp">1</td>
</tr>
```

Daraus wird nach dem oben Beschriebenen in unserem Beispiel folgende Zeichenkette:

```
Buttermilch|||||Milch und Milchprodukte|||||1
```

Nun soll die so erzeugte Zeichenkette den Zweck erfüllen, in `sort_array` als Wert übernommen zu werden. Am Ende steht jede Tabellenzeile in dieser Form in `sort_array`. Durch Aufruf der sort()-Methode für Arrays werden die Werte sortiert. Wenn nun beispielsweise nach der dritten Spalte sortiert werden soll, müssen deren Werte aber vorne stehen, damit die Sortiermethode greift. Deshalb wird in `tr_str` der Tabellenzeileninhalt nicht einfach nur »der Reihe nach« gespeichert, sondern in der Reihe, die für die gewünschte Sortierung tauglich ist. Wird also beispielsweise eine Sortierung nach Spalte 3 gewünscht, wird Folgendes in `tr_str` erzeugt:

```
1|||||Buttermilch|||||Milch und Milchprodukte
```

Im Anschluss an die for-Schleife wird innerhalb von `tabsort()` der so erzeugte Array `sort_array()` durch Aufruf der Standard-Array-Methode `sort()` sortiert. Falls eine absteigende Sortierung gewünscht ist, muss außerdem noch `reverse()` angewendet werden, was die Elementreihenfolge im Array einfach umkehrt und ebenfalls eine Standard-Array-Methode ist. Falls eine numerische Sortierung gewünscht ist, wird der sort()-Methode der Name der Hilfsfunktion `numsort` übergeben. Diese

erzwingt eine Interpretation von zwei zu vergleichenden Werten als Fließkommazahl und bedient die sort()-Methode so, wie sie es benötigt.

Nach erfolgreicher Sortierung muss `sort_array` nun wieder verarbeitet werden. Jedes Array-Element stellt ja eine Tabellenzeile dar, mittlerweile jedoch sortiert. In einer for-Schleife wird der Array abgearbeitet Mithilfe der split()-Methode wird die aktuelle Zeile anhand der Delimiter-Zeichenfolge wieder in Array-Elemente zerlegt (`tr_array`). Die Werte können nun direkt in die Tabelle geschrieben werden. Dabei kann bequem über die von der Funktion `tabsort_init()` erzeugten id-Namen auf die einzelnen Tabellenzellen zugegriffen werden. Jede Tabellenzelle erhält mit `innerHTML` einen neuen Wert, der sich durch die Sortierung ergibt.

Fazit

Das Beispiel ist nur eines dafür, was bei HTML-Tabellen in Verbindung mit DOM und JavaScript möglich ist. Tabellensortierung bringt dem Anwender einen eindeutigen informativen Mehrwert. Mit etwas Phantasie ist mit den gleichen Bordmitteln jedoch noch viel mehr möglich. Angenommen, jede Tabellenzelle besteht aus einem input-Eingabefeld (dieses kann ja mit CSS (`border:none`) rahmenlos gemacht werden, damit es nicht so »formularmäßig« aussieht). Auf diese Weise entsteht eine vollständig editierbare Tabelle. Mit etwas JavaScript/DOM können Sie dann Ansätze einer richtigen Tabellenkalkulation auf Ihrer Webseite realisieren.