Dynamic HTML

Inhaltsverzeichnis:

Quellen:

http://www.w3.org/TR/REC-DOM-Level-1/
http://www.w3.org/TR/DOM-Level-2-Core/
http://www.w3.org/TR/1999/CR-DOM-Level-2-19991210/
http://www.mozilla.org/docs/dom/samples/

Literatur:

Goodman, Danny: Dynamic HTML: The definitive reference. – Cambridge [u.a.]: O'Reilly, 1998.

1. DHTML und DOM

1.1. Dynamic HTML

Mit dem Aufkommen von Netscape 4.0 gab es kaum neue Tags, aber ein neues Schlagwort: DHTML – verbunden mit Cascading Style Sheets sollten Möglichkeiten zu pixelgenauer Positionierung und frei wählbaren Schriften eröffnet werden. Dynamisches HTML ab es nun auch beim Internet Explorer 4.0, nur eben mit anderen Techniken und anderer Programmierung.

DHTML ist nun aber keine Bezeichnung eines Standards oder einer Technologie, vielmehr bezeichnet DHTM das Zusammenwirken verschiedener Technologien:

- HTML,
- JavaScript,
- Cascading Style Sheets (CSS),
- Document Object Model (DOM),

wobei natürlich die Technologien aufeinander abgestimmt sind.

Mittlereweile wird der Begriff aber sehr weit gefasst und umfasst auch Anwendungen, die bereits mit dem Aufkommen von JavaScript ermöglicht wurden, wie:

- Browser-Check,
- Dynamisches Auswechseln von Bildern,
- Überprüfung von Formularfeldern,
- Wechsel von Frames.

Bei genauer Betrachtung ist dies nicht ganz falsch, unbestritten ist aber, dass DHTML mit dem Aufkommen des DOM einen qualitativen Schritt nach vorn getan hat. Somit soll das nachfolgende Skript seinen Schwerpunkt auf dem DOM haben.

Vielleicht ist hier ein kleiner Schritt in die Geschichte der Browser-Entwicklung sehr nützlich:

- Dynamik gab es schon mit den ersten Browsern: Die Farbe von Links (<a href="...">) wurde danach gewählt, ob der Link bereits besucht wurde.
- Der nächste wichtige Schritt war die Einführung von JavaSript als Programmiersprache für die Verwendung in einem HTML-Dokument. Dabei ist das Bedeutende nicht so sehr die an C angelehnt objektorientierte Sprache selbst, sondern der Einbau eines Anwendungsprogramminterfaces (API: application programming interface), das auch eine Interaktion mit ausgewählten HTML-Objekten über vordefinierte Objektklassen erlaubt:
  - document.anchors (seit Netscape 2.0),

  - document.applets (seit Netscape 4.0),
  - document.forms (seit Netscape 2.0),
  - document.images (seit Netscape 3.0),
  - document.links (seit Netscape 3.0).
- Mit dem mittlerweile verbannten Tag `<font>` kam zwar keine Dynamik, aber Farbe: Der Web-Autor hat erstmals Einfluss auf das Aussehen, Gestaltung und die physische Auszeichnung der einzelnen Seiten-Bereiche.
- In gewisser Weise verfolgt die Einführung von Cascading Style Sheets (CSS) dasselbe Ziel, aber auf einem qualitativ höherem Niveau: die physische Auszeichnung wird nicht mehr an das Dokument in Form von Ein- und Ausschaltern gebunden, sondern an die Tags selbst. Neben der eindeutigen Trennung von Inhalt und Form (Tags sind logische Elemente, ihre physische Auszeichnung wird in einer auswechselbaren Stilvorlage definiert) werden Tag-Elemente positionierbar – eine der wichtigsten Voraussetzungen für dynamisches HTML.
- Es ist nicht verwunderlich, dass mit der Einführung von CSS auch eine Erweiterung der JavaScript-Objekthierarchie erfolgte mit dem Ziel, auch eine Schnittstelle zum CSS zu besitzen. Allerdings gehen Microsoft und Netscape getrennte Wege:
  - Netscape führt im Netscape 4.0 ein neues Tag ein: `<layer>`. Über das Objekt `document.layers` kann jedes `<layer>`-Tag in seinen Eigenschaften modifiziert werden. Eine weitere Zugriffsmöglichkeit besteht über den Zugriff über einen Namen (Identifier), z.B. ist `<div id="idname">` über `document.idname` ansprechbar, womit es auch möglich wird, `<div>`-Tags zu modifizieren. Alle anderen Tags sind auf diesem Wege nicht modifizierbar.
  - Microsoft führt im Internet Explorer 4.0 das Objekt document.all ein, das eine Auflistung aller Tags im Dokument enthält, über die sie modifiziert werden können. Es ist ebenfalls möglich, einzelne Tags über ihren Namen anzusprechen: `document.all.idname`.

  Dabei kommen aber zwei Probleme zum Vorschein:
  1. Die Inkompatibilität beider Ansätze erfordert eine Standardisierung, um den Aufwand der Programmierung auf einen Ansatz reduzieren zu können.
  2. Sowohl das Objekt `document.layers` als auch `document.all` sind statisch, d.h., das Einfügen und Löschen von Objekten ist nicht möglich.
- Mit dem Document Object Model (DOM) legt das W3-Konsortium (http://www.w3c.org) einen Standard vor, der oben genannte Probleme beseitigt: ein HTML- oder XML-Dokument erhält ein hierarchisches System von Dokument-Tags, die mit Eigenschaften versehen und modifiziert werden können. Selbst das hierarchische System ist modifizierbar: das ist nun Dynamik pur. Damit ist das DOM deutlich allgemeiner und umfangreicher die früheren Ansätze von Netscape und Microsoft. Das

DOM ist im Internet Explorer 5.0 und in der Gecko-Engine bzw. Netscape 6.0 implementiert.

1.2. Document Object Model (DOM)

Das Document Object Modell ist eine Programmierschnittstelle (API: application programming interface) für die Anwendung in HTML- und XML-Dokumenten. Es basiert auf einer hierarchischen (baumartigen) Objektstruktur, die ein eindeutiges Abbild der Dokumentstruktur darstellt, auf die zugegriffen werden und die modifiziert werden kann.

Dabei soll das Wort Dokument bewusst in einem breiten Sinne benutzt werden, lassen sich doch z.B. mit XML auch mathematische Ausdrücke (MathML)[1] oder Vektorgrafiken (SVG)[2], für die das DOM ebenfalls angewendet werden kann.

Mit dem DOM ist man in der Lage, Dokumente zu erstellen, in ihrer Struktur zu navigieren, Elemente bzw. deren Inhalt zu ergänzen, zu modifizieren oder zu löschen. Dies betrifft im Prinzip alles, was in einem HTML- oder einem XML-Dokument enthalten ist.

Möglichkeiten zur Behandlung von Ereignissen stehen ebenfalls zur Verfügung (DOM level 2).

Das Programminterface ist so gestaltet, das es auf möglichst vielen Plattformen unter Nutzung verschiedenster Programmiersprachen (z.B. Java und JavaScript (ECMASrcipt)) benutzt werden kann.

[1] Mathematical Markup Language. Siehe auch: W3C's Math Home Page (http://www.w3.org/Math/).

[2] Scalable Vector Graphis. Siehe auch: Scalable Vector Graphics (SVG) 1.0 Specification (http://www.w3.org/tr/svg/); Scalable Vector Graphics (http://www.adobe.com/svg/); Heyer, Jürgen: SVG – Grafik für das Web. PC Professionell 2/2000 S. 186–189.

2. Das Document-Object-Modell (DOM)

2.1. Begriffe

- Dokument:
  Das Dokument ist eine im Speicher befindliche Hierarchie von Knotenobjekten (node objects), die eine HTML-Seite repräsentieren. Das Dokument ist ein Knoten mit einem Element: das Dokument selbst.

- Knoten (node):
  - Ist Oberbegriff für alle Objekte, die in einem Dokument enthalten sein können: Dokument, Elemente, Textknoten.
  - Elemente sind alle Knoten-Objekte eines Dokuments außer den Texten. Sie haben Attribute. Elemente können Eltern (parents) von anderen Elementen sein, die dann in dieser Beziehung Kinder (childs) heißen.
  - Textknoten behandeln den Text in einem Dokument.

- Attribute:
  Sind Eigenschaften von Elementen, somit also keine child-Knoten eines Elementes und somit auch nicht Bestandteil des Dokumentbaumes.

- N-ärer Baum (n-ary tree):
  Darstellung der Dokument-Hierarchie in Baum-Form. Da pro Knoten mehr als zwei Kind-Elemente auftreten können, wird die Bezeichnung n-är benutzt.

- document
  - head
    - title
    - meta
    - meta
  - body
    - h1
    - p
      - b
    - table
      - tr
        - td
        - td
    - p

2.2. Ausgewählte Methoden und Eigenschaften

Zugriff auf Elemente

- Einzelelement: `var myElement = getElementById("id");`

- Liste: `var myElements = getElemetsByTagName("tagname");`
  Über die Eigenschaft `length` erfährt man die Anzahl *n* der Elemente. Die Elemente können dann über die Nummern 0...*n*-1 angesprochen werden: entweder mit `[i]` oder mit `item(i)`.

  Beispiel:
  ```
  var myElements = getElemetsByTagName("p");
  for (var i = 0; i < myElements.length; i++)
  { myElements[i].style.color = "red" }
  ```

- Liste von untergeordneten Kind-Knoten:
  `var myElements = element.childNodes();`
  Liefert die Liste aller Kind-Knoten einschließlich der Textknoten zurück.

Vordefinierte Eigenschaften von Knoten-Objekten

- `style` Objekt zur Verwaltung ausgewählter Stileigenschaften.
- `attributes` Liste der zugehörigen Attribute.

- `nodetype` Liefert Typ des Knotens zurück, Eigenschaft kann nur ausgelesen werden. Wichtige Werte sind: 1 für Element-Objekt, 3 für Textknoten.
- `nodename` Liefert das zugehörige Tag eines Element-Objektes, z.B. `p`, `h1`, `li`.

- `childnodes` Liste alle Kind-Knoten-Objekte. `childnodes.length` liefert die Anzahl *n* der Kindknoten, auf die über ihren Index 0...*n*-1 zugegriffen werden kann, z.B. `childnodes[2]`.
- `firstchild` Erstes Kindknotenobjekt der `childnotes`-Liste.
- `lastchild` Letztes Kindknotenobjekt der `childnotes`-Liste.

- `parentnode` Bezug eines Knotens auf seinen Elternknoten.
- `nextSibling` Nachfolger des Elternknotens in der Liste, in der Elternknoten Kind-Objekt ist.
- `previousSibling` Vorgänger des Elternknotens in der Liste, in der Elternknoten Kind-

Objekt ist.

- `nodeValue` Liefert den Inhalt eines Textknotens. Für alle anderen Knoten wird der Wert `null` zurückgeliefert.

Definition von Element-Eigenschaften

1. Über `setAttribute`-Methode: `setAttribute("Attributename", "Wert")`.
   Mit dieser Methode lassen sich alle (sinnvollen) Eigenschaften eines Elements setzen, die auch im Tag angegeben werden könnten.
   Beispiel: `myElement.setAttribute("class", "divbox");`

2. Über `style`-Objekt (siehe nächstes Kapitel).
   Die wichtigsten Stileigenschaften lassen sich auch über den Zugriff über das `style`-Objekt setzen.
   Beispiel: `myElement.style.color="black";`

Behandlung von Kindelementen

- Erstellung von Knoten:
  - `document.createElement(tagname);`
    Erstellt und liefert ein Element für den angegebenen Tagnamen.
    Beispiel: `var myElement=document.createElement("div");`
  - `document.createTextNode(textdaten);`
    Erstellt und liefert ein Textknotenobjekt mit dem angegebenen Inhalt.
    Beispiel: `var myElement=document.createTextNode("Dies ist der Text");`

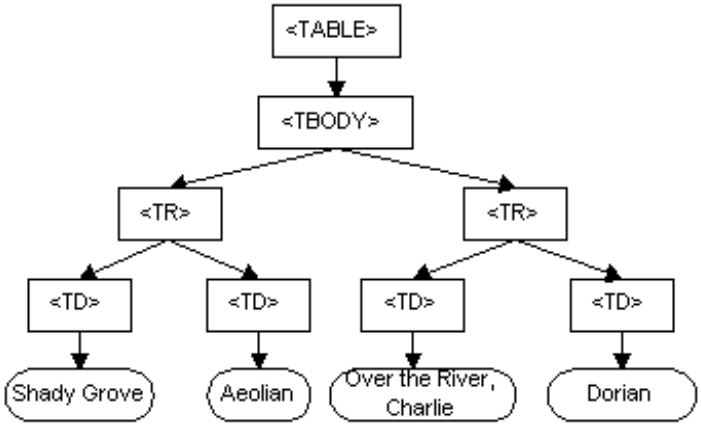
- Einfügen und Entfernen von Kind-Knoten aus Hierarchiebaum:
  - `node.insertBefore(childnode, nodePosition);`
    Fügt das angegebene Kind-Knoten-Objekt `childnode` als Kind in das Knotenobjekt `node` ein. `nodePosition` ist dasjenige Kind-Knoten-Objekt, vor dem das neue Objekt eingefügt werden soll.
  - `node.appendChild(childnode);`
    Fügt das angegebene Kind-Knoten-Objekt `childnote` als letztes Kind-Objekt an.
  - `node.removeChild(childnode);`
    Entfernt das angegebene Kindobjekt `childnote`.

  - `node.replaceChild(new_childnote, old_childnote);`
    Ersetzt das alte Kind-Knoten-Objekt `old_childnote` durch `new_childnote`. Das ursprüngliche Knotenobjekt wird zurückgeliefert.

- Methoden für Textknoten:
  - `textnode.appendData(arg);`
    Fügt Zeichenkette `arg` an das Ende der bereits bestehenden Zeichenkette an.
  - `textnode.insertData(offset, arg);`
    Fügt Zeichenkette `arg` an der Offsetposition `offset` ein.
  - `textnode.deleteData(offset, count);`
    Löscht `count` Zeichen ab der Zeichenposition `offset`.
  - `textnode.replaceData(offset, count, arg);`
    Löscht ab der Position `offset` `count` Zeichen und setzt Zeichenkette `arg` an die Position `offset`.

2.3. Beispiel: Erstellung einer Tabelle

Das nachfolgende Beispiel ist mit Netscape 6.0 bzw. Mozilla und im Internet Explorer 5.x darstellbar. Wenn das `tbody`-Tag vergessen wird, läuft das Beispiel im Internet-Explorer 5.x nicht.




```
<script language="JavaScript" type="text/javascript">
  var myParent, myTable, myTBody, myRow, myCell, myText, newText;

function CreateTable() {

  myParent=document.getElementsByTagName("body").item(0);
  myTable=document.createElement("table");
  myTBody=document.createElement("tbody");
  myRow=document.createElement("tr");
    myCell=document.createElement("td");
    myText=document.createTextNode("Die erste Zelle");
    myCell.appendChild(myText);
    myRow.appendChild(myCell);
    myCell=document.createElement("td");
    myText=document.createTextNode("Die zweite Zelle");
    myCell.appendChild(myText);
    myRow.appendChild(myCell);
  myTBody.appendChild(myRow);
  myTable.appendChild(myTBody);
  myParent.appendChild(myTable);

  }

</script>
```

2.4. Eigenschaften von style

Einsetzbar seit Gecko, Meilenstein 8, IE 5.

Beispiele unter Verwendung der Definition:

```
var myElement=document.getElementById(id);
```

style

`backgroundColor`	Setz Hintergrundfarbe des Elements auf Wert `backgroundColor`. Beispiel: `myElement.style.backgroundColor="magenta";`
`backgroundImage`	Setz Hintergrundbild des Elements auf Wert `backgroundImage`. Beispiel: `myElement.style.backgroundImage="url('fig.gif2')";`
`backgroundRepeat`	Bestimmt das Wiederholverhalten des Hintergrundbildes mit dem Wert `backgroundRepeat`: keine Wiederholung (`no-repeat`), Wiederholung nur in X-Richtung (`repeat-x`), Wiederholung nur in Y-Richtung (`repeat-y`), Wiederholung in beiden Richtungen (`repeat`). Beispiel: `myElement.style.backgroundRepeat="repeat";`
`color`	Setzt Textfarbe eines Elements auf `color`. Beispiel: `myElement.style.color="black";`
`fontFamily`	Setzt Schriftfamilie eines Elements auf `fontFamily`. Beispiel: `myElement.style.fontFamily="Times";`
`fontSize`	Setzt Schriftgröße eines Elements auf `fontSize`. Beispiel: `myElement.style.fontSize="50px";`
`fontWeight`	Setzt Schriftgewicht eines Elements auf `fontWeight` (Mögliche Zahlenangaben 100, 200, 300, 400, 500, 600, 700, 800 oder 900; Wörter `normal`, `bold`, `bolder`, `lighter` etc.). Beispiel: `myElement.style.fontWeight=500;` Beispiel: `myElement.style.fontWeight="bold";`
`height`	Setzt Höhe eines Elements auf `height` (in Pixeln). Beispiel: `myElement.style.height=100;`
`left`	Setzt linke Kante des Elements auf `left` (in Pixeln). Beispiel: `myElement.style.left=50;`
`letterSpacing`	Setzt den Buchstabenabstand eines Elements auf `letterSpacing`. Beispiel: `myElement.style.letterSpacing ="30px";`
`opacity`	Eigenschaft der Gecko-Engine für Testzwecke: Ein- und Ausblendeigenschaft, so wie sie für CSS 3 (`opacity` [Undurchsichtig-

	keit]) vorgeschlagen wird. Könnte umbenannt oder verworfen werden. Wertebereich 0...1. Beispiel: `myElement.style.opacity="0.5";`
`textAlign`	Setzt Zeilenausrichtung (`left`, `center`, `right`) eines Elements auf `textAlign`. Beispiel: `myElement.style.textAlign="left";`
`top`	Setzt obere Kante des Elements auf `top` (in Pixeln). Beispiel: `myElement.style.top=50;`
`visibility`	Anzeige (`visible`)[3] oder Ausblenden (`hidden`) eines Elements durch Setzen des Wertes `visibility`. (Gecko, Navigator Ver 4.x, Internet Explorer 4.x). Als dritte Möglichkeit besteht die Möglichkeit der Übernahme vom parent (`inherit`). Beispiel: `myElement.style.visibility="hide";`
`width`	Setzt Breite eines Elements auf `width` (in Pixeln). Beispiel: `myElement.style.width=300;`

[3] Im Layers-Modell von Netscape heißen die Optionen `show` und `hide`.

3. Cross-Browser-Funktionalität

3.1. Einführung

Mit der vierten Browsergeneration haben sowohl Netscape als auch Microsoft Verfahren eingeführt, die DHTML zulassen sollten. Beide Wege waren mitnichten kompatibel, so dass für beide Browser unterschiedliche Code-Sequenzen geschrieben werden müssen.

Das W3-Konsortium hat einen neuen Standard vorgelegt, der sich in weiten Teilen mit der Implementation von Microsoft deckt. Netscape 6 bzw. die Gecko-Engine und der Internet-Explorer beherrschen das vom Konsortium vorgeschlagene DOM, Netscape 6 bzw. die Gecko-Engine aber konsequenterweise nur dieses Modell. Damit besteht die Notwendigkeit, letztendlich drei Programmversionen bereitzustellen.

Das wichtigste zu Beginn ist also, die Browserfähigkeiten zu bestimmen (Test über Browser-Versionsnummer ist ungeeignet):

```
var DhtmlType = 0;
// NS4 = 1, IE4 = 2, DOM = 3

if (document.getElementById) { DhtmlType = 3 }
   else { if (document.all) { DhtmlType = 2 }
      else { if (document.layers) { DhtmlType = 1 }
      }
   };
```

Die Methode `getElementById` ist erst im W3C DOM definiert, somit ist mit ihrem Vorhandensein eben dieses Modell realisiert.

Das Objekt `document.all` ist nur im Internet Explorer definiert, die Nutzung ist nur im Fall des IE 4.x sinnvoll (die Version 5 orientiert sich am W3C DOM), wenn man obige Abfragereihenfolge einhält, wird im IE 5.x mit dem W3C DOM gearbeitet.

Das Objekt `document.layers` ist nur im Netscape 4.x definiert.

3.2. Netscape 4.x

Es besteht eine Zugriffsmöglichkeit für `<div id="idname">`- und `<layer id="idname">`-Tags, diese über ihren Identifikator anzusprechen und Werte auszutauschen:

```
with (document.idname) {
   left=70;
   top=10;
   bgColor="black"; }
```

Wichtige Eigenschaften:

- `background` Hintergrundbild.
- `bgColor` Hintergrundfarbe.
- `clip` Anzeigebereich.
- `left` Links-Wert der linken Kante, relativ.
- `name` Name des Layers.
- `pageX` Links-Wert der linken Kante, absolut.
- `pageY` Oben-Wert der oberen Kante, absolut.
- `top` Oben-Wert der oberen Kante, relativ.

Um den Inhalt eines `<div>`- bzw. `<layer>`-Tags neu zu beschreiben, muss man die entsprechenden `document`-Methoden benutzen:

```
with (document.imgcontainer1) {
   document.open();
   document.write('<table><tr><td><span class="divbox">Text</span></td></tr>
   </table>');
   document.close(); }
```

3.3. Internet Explorer 4.x

Es besteht eine Zugriffsmöglichkeit für alle Tags, diese über ihren Identifikator anzusprechen und Werte auszutauschen:

```
with (document.all.idname.style) {
   left=70;
   top=10;
   backgroundColor="black"; }
```

Wichtige Eigenschaften von `style`:

- `background` Hintergrundbild.
- `backgroundColor` Hintergrundfarbe.
- `backgroundImage` Hintergrundbild.
- `backgroundRepeat` Wiederholung des Hintergrundbildes in X- bzw. Y-Richtung.
- `clip` Eingrenzung des Anzeigenbereiches.
- `color` Textfarbe.
- `font` Schrift.
- `fontFamily` Schriftfamilie.
- `fontSize` Schriftgröße.
- `fontWeight` Schriftgewicht.
- height Höhe des Elements.
- `left` Position der linken Kante.
- `top` Position der oberen Kante.
- `width` Breite des Elements.

Um den Inhalt eines Elements neu zu beschreiben, kann man das `innerHTML`-Eigenschaftsfeld benutzen:

```
with (document.all.idname) {
   innerHTML='<table><tr><td><span class="divbox">Text</span></td></tr>
   </table>' }
```

Auch wenn die Internet-Explorer 5.x noch die Funktionalität von seinem Vorgänger unterstützt (insbesondere das `document.all`-Objekt, die `innerHTML`-Eigenschaft), so sollten doch hier konsequenterweise nur die Methoden und Eigenschaften des W3C DOM benutzt werden.

4. Umstellung von Code für IE4 und Netscape 4 auf DOM

Quelle:

http://www.mozilla.org/docs/web-developer/upgrade_2.html

Proprietäres Feature	W3C DOM-Ersatz
Nav4 LAYER	HTML 4.0 DIV
Nav4 ILAYER	HTML 4.0 IFRAME
Nav4 LAYER SRC=, ILAYER SRC=, DIV SRC=	HTML 4.0 IFRAME SRC=
IE3/4/5 MARQUEE	HTML 4.0 DIV
Nav2/3/4/5 BLINK	CSS1 text-decoration:blink
IE 3/4/5 BGSOUND	HTML 3.2 EMBED
Nav4 document.layers[] IE4/5 document.all	DOM level 1 document.getElementById()
Nav4 element.visibility = value; IE4/5 element.style.visibility = value;	DOM level 2 element.style.visibility =value;
Nav4 element.left IE4/5 element.style.pixelLeft	DOM level 2 parseInt (element.style.left)
Nav4 element.top IE4/5 element.style.pixelTop	DOM level 2 parseInt (element.style.top)
Nav4 element.moveTo(x, y); IE4/5 element.style.pixelLeft = x; element.style.pixelTop = y;	DOM level 2: element.style.left = value +"px"; element.style.top = value +"px";
Nav4 document.tags, document.ids, document.classes, Nav4/IE4/5 document.elementName	DOM access methods
IE4/5 document.styleSheets[].addRule (selector, declaration); Nav4 document.contextual()	DOM Level 2 CSS Interface
Nav4 handleEvent()	W3C DOM Level 2 dispatchEvent()
Navigator 4 .jar files and IE4+ .cab files	XPI files for XPInstall
Plug-ins using LiveConnect to access the Java API from JS	Mozilla Plug-in API

5. Ereignisbehandlung

###