JavaScript

Inhalt

1 JavaScript – Ergänzung durch Interaktion

JavaScript ist eine Skriptsprache, mit allen üblichen programmiersprachlichen Konzepten, die als Ergänzung für die Interaktivität einer HTML-Datei dienen kann. JavaScript traditionell clientseitig, also vom Browser ausgeführt, ist mittlerweile auch serverseitig einsetzbar.
Ein Skript kann auf einem beliebigen Texteditor erstellt werden und wird dann mit der Endung *.js* gespeichert.

Mit JavaScript können HTML-Dateien verändert werden, Daten vom Anwender entgegen genommen werden, gespeichert, verarbeitet und zurückgegeben werden.

2 Das Document-Object-Model (DOM)

Das DOM ermöglicht den einheitlichen Zugriff auf ein HTML-Dokument für verschiedene Skriptsprachen in objektorientierter Form. Es dient also als Schnittstelle. Das DOM ist eine vom W3C definierte Norm.

Das DOM in Verbindung mit js, kann für folgende Funktionen verwendet werden:

- Veränderung des Seiteninhaltes eines bereits bestehenden Dokumentes
- Erstellung kompletter Dokumente
- Navigation durch ein Dokument
- Löschen und Einfügen von Elementen
- Veränderung der Attribute und Attributwerte der Elemente

2.1 Das Grundprinzip des DOM

Jedes Dokument (HTML und XML) kann als ein Baum dargestellt werden. Die Blätter (Knoten, nodes) können gezielt angesprochen werden. Dadurch bekommt jedes Dokument eine Struktur und die Elemente werden zu Vorfahren, Nachfahren und Geschwistern. Es entsteht eine strikte Hierarchie.

Es werden drei verschiedene **Knotentypen** unterschieden:

- Element-Knoten
- Attribut-Knoten (Eigenschaften der Element-Knoten und somit grafisch nicht als Knoten erkennbar.)
- Text-Knoten

Es existieren weiterhin:

- Wurzelknoten *<html>*
  er existiert nur einmal zu Beginn des Dokumentes
- Dokumentknoten *document*
  beschreibt die gesamte Struktur, also das gesamte Dokument

Alle Kindelemente eines Dokumentes liegen in einem Array mit dem Namen *childNodes[]*.

Über das DOM wird jedes HTML-Element in JavaScript als Objekt repräsentiert.
Alle Attribute der Elemente werden zu Attributen des Objekts.

Jeder Knoten hat **Eigenschaften**

- nodeName
- nodeType
- firstChild
- lastChild
- nextSibling
- previousSibling
- parentNode

2.2 Methoden zur Veränderung des DOM-Baumes

- appendChild()
  hängt einen neuen Knoten in einen existierenden Elternknoten
- hasChildNodes()
  gibt als Booleschen Wert an, ob Kindknoten existieren
- createElement()
  erzeugt einen Element-Knoten
- removeNode()
  entfernt einen Knoten und ggf. alle Kindknoten
- cloneNode()
  erzeugt eine Kopie des Knotes und ggf. aller Kindknoten
- replaceNode()
  ersetzt einen Knoten durch einen neuen Knoten
- setAttribute()
  fügt einem Knoten ein Attribut hinzu
- insertBefore()
  erzeugt einen neuen Kindknoten vor dem Kindknoten im Elternknoten

3 Einbindung von JavaScript-Dateien

Ebenso wie CSS-Dateien werden auch JavaScript-Dateien in die HTML-Datei eingebunden.
Auch hier gibt es verschiedene Möglichkeiten:

Einbindung über das <script>-Element

```
<script type="text/javascript">
     //Anweisungen
</script>
```

Einbindung als externe Datei, überlicherweise im <head>,
formal aber überall im Dokument erlaubt.

```
<script src="meineDatei.js" type="text/javascript">
</script>
```

Einbindung über ein Event und Aufruf einer oder mehrerer Funktionen,
direkt im HTML-Element (nicht unobtrusive)

```
<p onclick="alert('Hallo Welt!');"></script>
```

Alternative für Browser ohne JavaScript-Unterstützung

```
<noscript>
Huch, ... js wird nicht unterstützt oder ist nicht erwünscht
... Schade.
</noscript>
```

4 Events

Events sind Ereignisse, auf die der Browser reagieren kann.
Wird z.B. eine Aktion mit der Maus oder Tastatur ausgeführt, so ist das ein Ereignis, auf das der Browser reagieren kann. Innerhalb des JavaScript kann dieses Event dann behandelt werden.

Ausgewählte Eventhandler

```
onclick
onmouseover
onfocus
onload
...
```

Ereignisse können vom Browser selbst kommen (z.B. das Laden oder Entladen einer HTML-Datei), vom Netzwerk (z.B. die Beantwortung eines Requests an den Server), durch den Benutzer (z.B. Maus- oder Tastaturereignisse) oder zeitgesteuert (z.B. timeouts oder Intervalle).

6 Unobtrusive Design

Die Trennung von Struktur (HTML) und Funktionalität (JavaScript) sorgt für wartungsfreundliche und weniger fehleranfällige Anwendungen. Die einzelnen Bestandteile lassen sich auch für andere Anwendungen wiederverwenden.

1. js-Datei im <head> einbinden.

```
<script type="text/javascript"  src="myScript.js"></script>

Beachte: dies ist kein Standalone-Element.
```

2. in js-Datei eine start-Funktion deklarieren, die alle Funktionalitäten aufruft.

```
function init(){
/*Aufruf der Arbeitstiere, andere Funktionen*/
}
```

3. in js-Datei die start-Funktion durch Zuweisung an das Event onload referenzieren.

```
window.onload=init;
```

3. Alternativ:

```
window.onload=function(){...}
```

7 Zugriff auf HTML-Elemente über das DOM

Über das Attribut *id*

```
document.getElementById('Wert der id');
```

Über das Attribut *class*

```
document.getElementsByClassName('Wert des class-Attributes');
```

Über den Element-Namen

```
Document.getElementsByTagName('Element-Bezeichner');
```

8 Bausteine der js-Programmierung

8.1 Anweisungen (statements) und Kommentare (comments)

Alle Befehle (Anweisungen) werden sequentiell abgearbeitet.
Jede Anweisung endet mit einem Semikolon.
Anweisungsblöcke werden in {} gesetzt.
Kommentare werden nicht vom Browser „übersetzt“ und werden wie folgt angewiesen
// einzeilig
/*mehrzeilig*/

Anweisungen werden sequentiell, von oben nach unten abgearbeitet.

Mit Kontrollstrukturen kann die Abarbeitung an eine Bedingung geknüpft werden (conditional execution) oder mehrfach wiederholt werden (looping, iteration).

8.2 Variablen (values)

Variablen belegen Speicherplatz und können einen Wert enthalten. In JavaScript sind Variablen nicht typisiert. Der Typ des Wertes darf sich zur Laufzeit des Programmes verändern (z. B. von Zeichen zu Zahl).

8.3 Deklaration einer Variablen

Wird eine Variable zum ersten Mal angesprochen, dann wird sie deklariert.
Zum Zeitpunkt der Deklaration wird ihr ein Wert zugewiesen (Initialisierung).

Eine Variable wird mit dem Schlüsselwort *var* deklariert.

Eine Variable, die mit var deklariert wurde, ist gültig und sichtbar in der Funktion, in der sie deklariert wurde.

Eine Variable kann auch ohne Schlüsselwort deklariert werden, durch Verwendung. Sie ist dann global im ganzen Script gültig und sichtbar. (Achtung: nicht im ‚use strict’-Modus!)

Wenn eine Variable keinen Wert erhält, dann enthält sie den Wert *undefined*.

```
var x = 5;
```

8.4

8.5 Zuweisung (assignment)

Die Zuweisung ist eine besondere Form der Anweisung.
Der Zuweisungsoperator ist das einfache Gleichheitszeichen.

```
x = 5;
```

Auf der linken Seite steht ein Konzept, dem auf der rechten Seite ein Wert zugewiesen wird, der aus einer vordefinierten Werteliste stammt (Backus-Naur-Form)

8.6 Ausdrücke (expressions)

Alles, was einen Wert erzeugt, ist ein Ausdruck.

- 25;
- (3 + 5);
- „Hallo Welt“;
- !true;

8.7 Funktionen

Funktionen sind verpackte Anweisungsblöcke, die die eigentlichen Arbeitsanweisungen enthalten und je nach Bedarf mehrfach aufgerufen werden können. Mit Funktionen können Skripte in einzelne Blöcke aufgeteilt und strukturiert werden.

Deklaration einer Funktion

```
function doStuff(){
     //jede Menge Anweisungen, um etwas zu erledigen
}
```

Wir unterscheiden zwischen dem Aufruf einer Funktion:

```
doStuff();
```

mit dem Aufruf wird die Funktion ausgeführt.

und dem Referenzieren einer Funktion:

```
doStuff;
```

als Referenz wird die Funktion nicht sofort ausgeführt, sondern es wird lediglich darauf verwiesen.
Sie wird gespeichert und möglicherweise an anderer Stelle aufgerufen.

Funktionen werden in js wie Objekte behandelt. Dabei entspricht der Funktionsbezeichner dem Objektbezeichner und der Rumpf dem Wert. Funktionen können referenziert werden, als Argumente an andere Funktionen übergeben werden, mit *return* aus Funktionen herausgeliefert werden, in Variablen gespeichert werden und im Gegensatz zu Objekten zusätzlich aufgerufen und ausgeführt werden (*callable object*).

Eine Funktion kann Werte produzieren und, oder andere Variablen verändern.
Wenn eine Funktion einen Wert produziert, dann gibt sie ihn zurück (*return*).

8.8 Datentypen (type) und Werte (values)

JavaScript gilt als eine *nicht typisierte* Sprache. Das ist nicht ganz korrekt. Richtig ist, Variablen haben keinen Typ, als den, der sich gerade in ihnen befindet. Der Typ einer Variablen kann zur Laufzeit geändert werden. Allerdings hat **der Wert einer Variablen sehr wohl einen Datentyp**. Jeder Datentyp definiert einen Wertebereich und mögliche Operationen, die auf den enthaltenen Wert ausgeübt werden können.

In der Spezifikation (ECMAScript 5) werden die folgenden Typen aufgelistet.

- **Zahl (Number)**
  Numerische Werte, positiv und negativ, Ganzzahlen und Bruchzahlen, 64 Bit
  Arithmetische Operationen.
- **Zeichenketten (String)**
  Jedes Zeichen, eingeschlossen in einfache oder doppelte Anführungszeichen.
- **Boolean**
  Boolesche Variablen liefern nur die Werte t*rue* oder *false*.
  Boolesche Operationen (Vergleiche)
- **Nicht definierte Typen (Undefined)**
  Variablen, die keinen Wert und keinen Typ besitzen
- **„leere" Typen (Null)**
  Variablen, die keinen Wert besitzen
- **Objekte (Object)**
  Function ist ein subtype von Object, weil Funktionen in js ein *callable object* sind, oder auch *first-class-objects*.
  Ebenso fallen **Arrays** unter den Typ *object*.

- **Symbol** (neu in ES6)

8.9 Gültigkeitsbereiche (scope)

Grundsätzlich werden alle Gültigkeitsbereiche nur von Funktionen erzeugt.
Einfach Anweisungsblöcke erzeugen i.d.R. keinen scope. (Ab ES6 wird es dazu das Schlüsselwort *let* geben.)

Zu beachten ist dabei, nicht nur wo sondern auch was deklariert wird.

Variablen sind innerhalb ihres scopes ab dem Punkt der Deklaration verwendbar, bis zum Ende der Funktion, in der sie deklariert wurden.

Benannte Funktionen befinden sich innerhalb der gesamten Funktion, in der sie deklariert wurden im scope (*hoisting*).

8.10 Arrays

Die Elemente eines Arrays können jede Art von Wert oder Typ enthalten. Die Länge eines Arrays ist variable und wird durch den Inhalt bestimmt.

```
var a = []; //length: 0
var b = [1, true, [4]]; lenght: 3
```

Per default sind Arrays numerisch indiziert. Weil Arrays aber vom Typ *object* sind, können sie auch assoziativ indiziert werden. Allerdings werden keys mit einem string-Wert nicht in der Eigenschaft *length* gezählt.

```
var a = [];
a[0] = 1;
a[,foo'] = 2;
a.length //1
```

Allerdings kann dies zu Problemen bei der Abarbeitung führen, z.B. bei der Iteration über ein Array.
Grundsätzlich sollten für *key/property* – Paare Objekte verwendet werden.

8.11 Objekte

8.12 Konvertierung von Datentypen

Manchmal ist es notwendig Datentypen umzuwandeln. Sollen z.B. auf eine eingelesene Zeichenkette aus Ziffern mathematische Operationen angewendet werden, muß diese konvertiert werden.

Für die Konvertierung gibt es Funktionen, denen ein Wert oder eine Variable übergeben werden können, die dann umgewandelt werden. Die übergebenen Variablen müssen allerdings numerische Zeichen enthalten.

```
parseInt(x)    //konvertiert den Inhalt der Variablen x in eine Ganzzahl
parseFloat(x)  //konvertiert den Inhalt der Variablen x in eine Gleitkommazahl
```

8.13 Funktionen, fortgeführt

8.13.1 Deklaration von Funktionen

```
function bezeichner(Parameter)
     {//Anweisungen
     return x;}
```

Über den Bezeichner kann die Funktion, nachdem sie deklariert wurde aufgerufen werden.
Der Bezeichner ist optional und in der Eigenschaft *name* gespeichert.

In der Parameterliste können beliebig viele lokale Variablen deklariert werden, die in die Funktion hineingeliefert und dort verarbeitet werden können.

Über das optionale Schlüsselwort *return* kann aus einer Funktion auch wieder etwas hinausgeliefert werden. Was im *return* vereinbart wurde, das wird an den Aufrufer der Funktion zurückgegeben.

Funktionen können ineinander verschachtelt werden, das gilt sowohl für die Deklaration als auch für den Aufruf.

8.13.2 Anonyme Funktionen

Anonyme Funktionen haben keinen Bezeichner, sie können also nicht aufgerufen werden.
Eine anonyme Funktion wird immer benutzt, wenn sie nur einmal im jeweiligen Kontext abgearbeitet werden soll, also wenn sie nicht mehrfach aufgerufen oder referenziert werden soll.

Funktionen können sofort abgearbeitet werden (IIFE – immediately invoked function expressions).

```
(function rightAway(){...})()
```

Eine anonyme Funktion kann u.a. dafür verwendet werden, um einer Funktion, die referenziert werden soll Argumente zu übergeben.

```
function init(){
      var x=document.getElementById(‚number’);

      x.onclick=doStuff;
      /*doStuff wird hier referenziert,
      leider können keine Argumente übergeben werden.*/
      x.onclick=function(){doStuff(something);}
      /*so geht’s*/
}
```

8.13.3 Funktionsaufrufe

Die Deklaration einer Funktion ist lediglich die Bereitstellung der darin enthaltenen Arbeitsanweisungen. Ausgeführt wird die Funktion erst, wenn sie aufgerufen wird.

```
myFunktion(5);
```

Beim Aufruf der Funktion werden die bei der Deklaration vereinbarten Parameter übergeben.

Der Aufrufer der Funktion muß sich nicht an die vereinbarte Parameterliste halten.
Werden zu viele Argumente an die Parameter geliefert, dann werden sie bei der Ausführung ignoriert (eigentlich werden sie in der Variablen *arguments* gespeichert) werden zu wenige Argumente übergeben, dann bekommen die überzähligen Parameter den Wert *undefined* zugewiesen.

Eine Funktion kann auch ohne Parameterliste aufgerufen werden. In diesem Fall wird sie nichtausgeführt, sondern der Funktionsrumpf wird bereitgestellt.

```
x = myFunktion;
```

Für den obigen Fall würde myFunktion erst ausgeführt, wenn x aufgerufen wird.

8.14 AddEventListener()

Der AddEventListener() registriert ein Event für jedes beliebige Objekt des DOM-Baums, also für jedes Element bis hoch zum Dokument selbst.

Sollen für ein Element auf dasselbe Event mehrere Funktionen aufgerufen oder referenziert werden, so ist das über den AddEventListener möglich.

Der AddEventListener() erwartet drei Argumente, das Event, die abzuarbeitende Funktion und optional einen booleschen Parameter, der i.d. Regel false ist und sich auf die Abarbeitung aller Events im Dokument das sogenannte „bubbling“ bezieht.

x.addEventListener(‚click’,doStuff,false);