HOCHSCHULE DER MEDIEN

JavaScript

von
Patrick Schmidt

Vortrag am
30. April 2009

Inhaltsverzeichnis

JavaScript

Einführung und Einordnung

JavaScript wurde 1995 von Brendan Eich, in Kooperation mit Sun Microsystems, für den Netscape Navigator entwickelt. Die Intention war in erster Linie eine Möglichkeit zu schaffen beispielsweise Formulardaten vor dem Abschicken an den Server zu validieren. Damit ist auch klar, dass JavaScript auf der Client Schicht einzuordnen ist.
Im September des selben Jahres erschien der erste Webbrowser unter dem Namen Navigator 2.0 mit einer eingebetteten Skriptsprache, die der oben genannten Anforderung gerecht wurde. Diese Skriptsprache wurde auf den Namen LiveSript getauft – allerdings nach kurzer Zeit in JavaScript umbenannt. Diesen Namen hat JavaScript dem zur selben Zeit vorherrschenden Java Hype zu verdanken. Außer der Namensähnlichkeit und einer an Java angelehnten Syntax ist JavaScript grundlegend von Java verschieden.
Mit kurzer Verzögerung brachte nun auch Mircosoft mit dem Internet Explorer 3 seine eigene Skriptsprache unter dem Namen JScript auf den Markt und entfacht dadurch einen Browserkrieg.
Es war zu befürchten, dass sich die oben genannten Technologien in zwei Richtungen entwickeln und es für Web Entwickler immer schwieriger werden wird browserunabhängige Websites zu programmieren. Um dem entgegen zu wirken schaltete sich die European Computer Manufacturers Association (ECMA) – eine Organisation zur Normung von Informations- und Kommunikationssystemen – ein. Diese standardisierte die Grundfunktionalität von JavaScript unter dem Namen ECMAScript (oder auch ECMA-262). Somit war eine Grundlage geschaffen, an der sich Entwickler künftiger Browser orientieren konnten, um eine einigermaßen einheitliche Implementierung von JavaScript zu gewährleisten.

JavaScript liegt mittlerweile in Version 1.8 vor. In naher Zukunft soll JavaScript 2.0 veröffentlicht werden. 97% der Internetnutzer haben bei ihrem Browser JavaScript aktivert. 99% unterstützen Version 1.5 oder besser. Ein Grund dafür ist sicherlich auch der, dass viele Websites im Zeitalter des Web 2.0, mit seinem hohen Maß an Benutzerinteraktion, ohne aktiviertes JavaScript überhaupt nicht mehr ordnungsgemäß funktionieren.

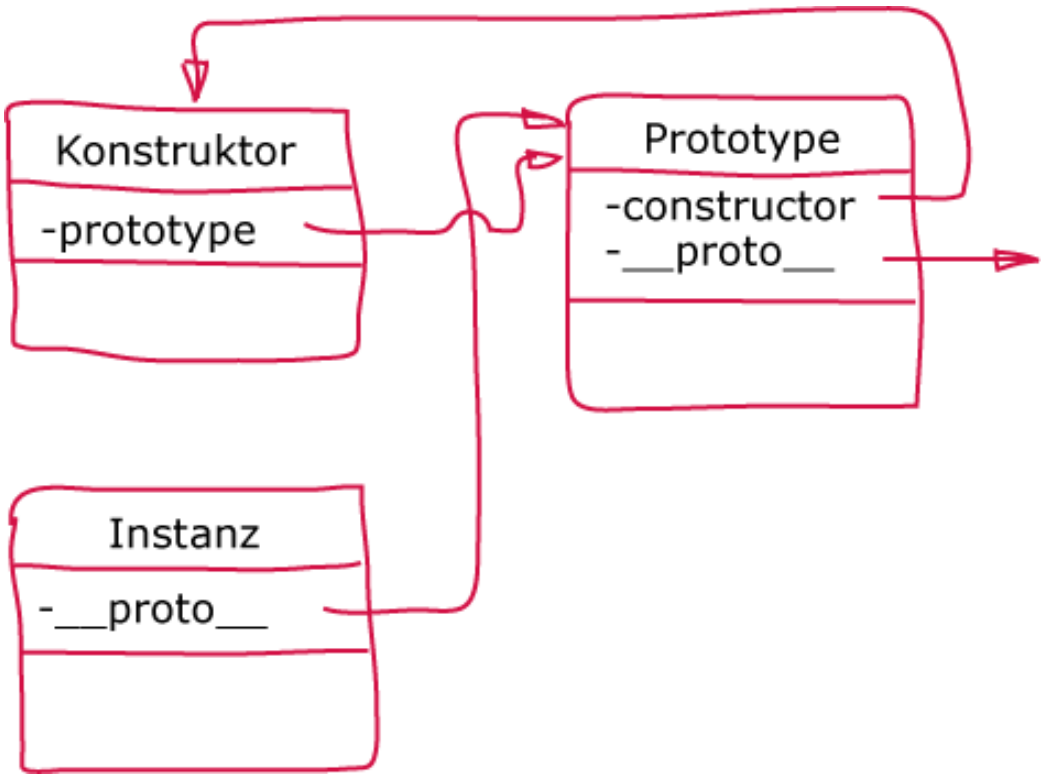
Objekte

JavaScript ist eine dynamisch typisierte, klassenlose und objektbasierte Skriptsprache.
Dynamisch typisiert meint, dass Variablen bei der Deklaration noch kein Typ zugewiesen ist. Diese Typzuweisung geschieht erst im Laufe des Programmflusses. Dabei kann sich der Variablentyp auch ändern.
Klassenlos und objektbasiert ist in JavaScript mit sogenannten Prototypen realisiert. Grundsätzlich sind in JavaScript alles Objekte, so sind selbst Funktionen oder Strings Objekte.
Intern sind Objekte als assoziatives Array realisiert, wobei die Keys den Eigenschaften und Methoden und die Values den entsprechenden Werten oder Referenzen auf weitere Objekte – etwa Funktionsobjekte – entsprechen. Zugriff auf die Eigenschaften und Funktionen eines Objekts hat man durch den Dot-Operator in der Form `variablenName.eigenschaft` bzw `variablenName.mehtode()`.

Alternativ kann genauso wie auf ein assoziatives Array zugegriffen werden `variablenName['eigenschaft']`.
Eine äußerst wichtige Rolle in JavaScript spielen Funktionen. Diese können in Verbindung mit dem new-Operator zum Erzeugen neuer Objekte aufgerufen werden. Dabei erhält die Funktion immer implizit eine Referenz auf das aufrufende Objekt, auf das dann per `this` zugegriffen werden kann. Diese Referenz steht nicht nur beim Aufruf mit dem new-Operator sondern grundsätzlich bei jedem Funktionsaufruf zur Verfügung. Wird eine Funktion in Verbindung mit dem new-Operator aufgerufen fungiert die Funktion als Konstruktorfunktion und erzeugt und initialisiert eine neue Instanz.

Prototypen

Grundsätzlich kann man sich Prototypen so vorstellen, dass alle Objekte ein Prototyp-Objekt referenzieren, dessen Eigenschaften und Methodern quasi „vererbt" werden.
Mit jedem Funktions-Objekt wird gleichzeitig jeweils ein eigner sogenannter Prototyp instanziiert, den das Funktions-Objekt über die Eigenschaft prototype referenziert. Per default handelt es sich bei dem Prototyp um eine Instanz des Object-Objektes. Mittels `funktionsName.prototype=new AndererPrototype` kann der Funktion auch ein beliebiger anderer Prototyp zugewiesen werden. Der Prototyp kann auch zur Laufzeit geändert werden.
Jede Instanz eines Objektes erhält implizit eine interne Referenz auf das Prototyp-Objekt der erzeugenden Konstruktor-Funktion. Auf diese Eigenschaft ist allerdings nicht mit allen Browsern aus zugreifbar. Im Mozilla Firefox etwa kann auf die Referenz mittels der Eigenschaft `__proto__` zugegriffen werden. Wird nun eine Eigenschaft oder Methode auf einer Instanz aufgerufen und dort nicht direkt gefunden wird die Prototype-Kette durchlaufen, bis die gewünschte Eigenschaft oder Methode gefunden wurde oder die interne Referenz auf den nächsten Prototype `null` ist.



Wichtig ist, dass zwischen Lesen und Schreiben von Eigenschaften eines Prototyps über eine Instanz eines Objektes unterschieden werden muss. Während beim lesenden Zugriff einfach der Wert geliefert wird, wird beim schreibenden Zugriff bei der Instanz eine neue Eigenschaft mit dem neuen Wert hinzugefügt. Ansonsten würde sich ein schreibender Zugriff auf alle Instanzen mit einer Referenz auf diesen Prototyp auswirken. Das selbe gilt entsprechend für das Hinzufügen beziehungsweise Entfernen von Funktionen bei Objektinstanzen. Dem Prototyp kann auch zur Laufzeit Eigenschaften und Methodern hinzugefügt werden, die dann implizit an alle entsprechenden Instanzen „weitervererbet" werden.

```
function meinParentObjekt() {
   this.eigenschaftEins = "wert1";
}
function meinObjekt(parameter) {
   this.eigenschaftZwei = parameter;
   this.methode = function () {
   alert(this.eigenschaftZwei);
   }
}
//ersetzt den default Prototyp durch ein meinParentObjekt
meinObjekt.prototype = new meinParentObjekt();

var meineInstanz = new meinObjekt("wert2");

meineInstanz.methode();                    //output:   wert2
alert(meineInstanz.eigenschaftEins);       //output:   wert1
```

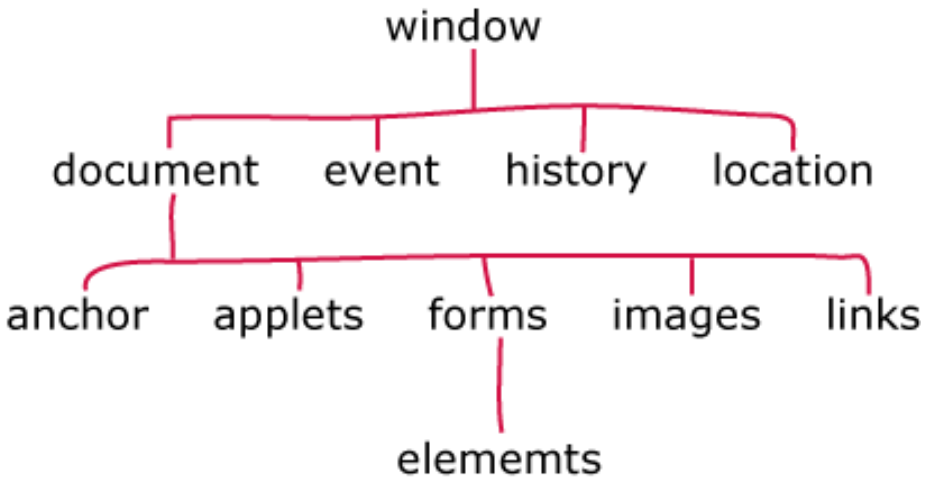
Wie oben beschrieben bieten Prototypen eine extreme Flexibilität und Freiheit. Bei leichtsinnigem Umgang kann es jedoch schnell zu unvorhersehbaren Effekten kommen.

Host Objekte

Im klassischen JavaScript lassen sich die vordefinierten Objekte in zwei Gruppen einteilen. Die erste Gruppe bilden dabei die sogenannten Host Objekte – oder auch Core Objects, wie sie in der JavaScript Referenz von Mozilla ganannt werden. Diese Gruppe von Objekten hat nichts direkt mit dem Anzeigefenster zu tun, stellt aber dafür wichtige Daten und Funktionalitäten bereit.
Beispiele für Objekte dieser Gruppe sind das Array-Objekt, das Date-Objekt oder das Math-Objekt. Ein wichtiges Core Objekt ist das navigator-Objekt, das Daten zum verwendeten Browser liefert. Damit können dann etwa Browserweichen realisiert werden, da sich die Interpretation von JavaScript auf den einzelnen Browsern in einigen Punkten immer noch ziemlich unterscheidet. Das Math- und navigator-Objekt haben die Besonderheit, dass sie nicht instanziiert werden müssen, sondern direkt mit ihnen gearbeitet werden kann.

Fest implementierte Objekte

Die zweite Gruppe der vordefinierten Objekte in JavaScript sind die fest implementierten Objekte. Diese beziehen sich direkt auf das Browserfenster und sind hierarchisch gegliedert.

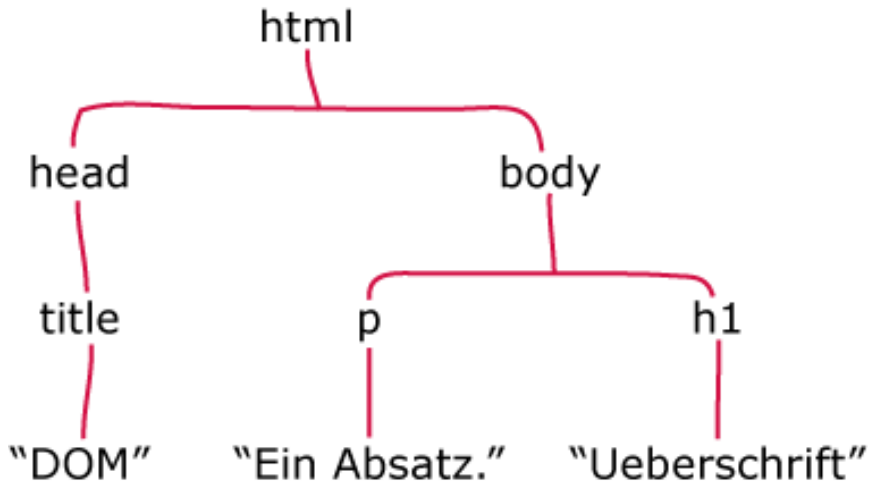


Auf der ersten, obersten Hierarchiestufe befindet sich das window-Objekt. Mit ihm ist es möglich das Browserfenster zu manipulieren also etwa die aktuelle Fenstergröße auszulesen und zu setzten oder neue Fenster zu öffnen beziehungsweise zu schließen. Es handelt sich hierbei um ein sogenanntes Root-Objekt. Sollen Eigenschaften oder Methoden des window-Objektes angesprochen werden muss nicht explizit `window.eigenschaft` aufgerufen werden – es genügt `eigenschaft` beiziehungsweise `methode()` zu schreiben. Derartige Aufrufe werden implizit auf dem window-Objekt ausgeführt.
Die zweite Hierarchiestufe wird von vier weiteren Objekten gebildet. Dem Location-Objekt, mit dem die aktuelle URL manipuliert werden kann, dem History-Objekt, mit dessen Hilfe im Browserverlauf navigiert werden kann, und dem Event-Objekt, das später noch im Eventhandling von Bedeutung sein wird. Das wichtigste Objekt auf dieser Stufe ist das document-Objekt, das das eigentliche HMTL Dokument repräsentiert und gleichzeitig Ausgangspunkt für die Objekte der dritten Hierarchiestufe ist.
Bei den Objekten in der dritten Hierarchiestufe handelt es sich um Arrays von HTML Elementen, auf die mittels dieser Arrays zugegriffen werden kann. Allerdings beschränkt sich der Zugriff auf Anchors, Applets, Forms, Images und Links – was wohl zu Anfangszeiten von JavaScript völlig ausgereicht haben mag. Allerdings bleibt etwa der gezielte Zugriff auf Tabellen oder Überschriften verwährt. An dieser Stelle schafft das Document Object Model abhilfe.

Document Object Model

Im Document Object Model kurz DOM wird das komplette XML- beziehungsweise HMTL-Dokument in einem Baum repräsentiert. Dabei werden die einzelnen Elemente als Knoten im Baum sichtbar. Textueller Inhalt wird ebenfalls in Form eines Knotens oder präzieser als Blatt in den Baum eingefügt.

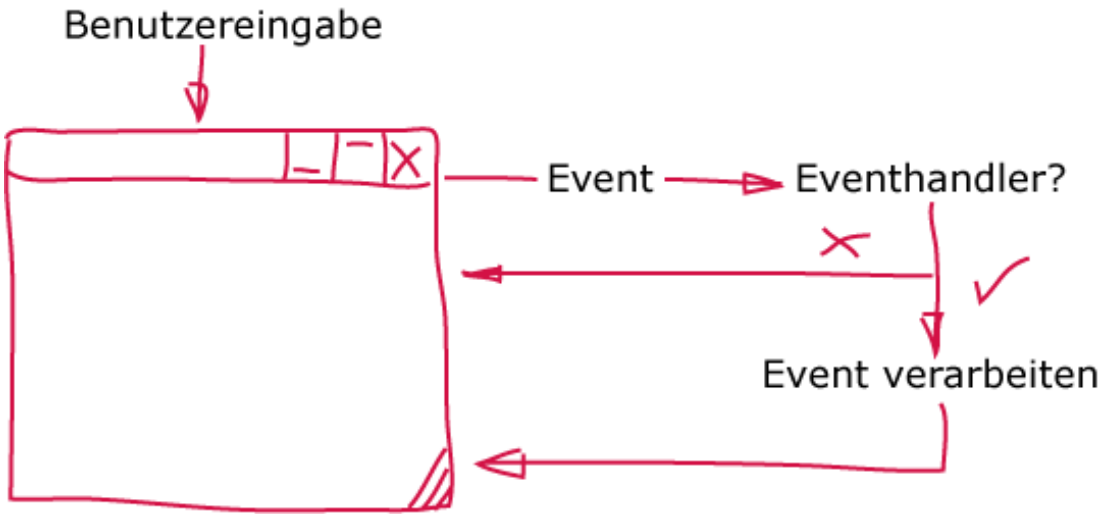
```
<html>
   <head>
         <title>DOM</title>
   </head>
   <body>
         <h1>Ueberschrift</h1>
         <p>Ein Absatz.</p>
   </body>
</html>
```



Das Document Object Model existiert in der Form schon seit Mitte der 90er Jahre. Es war allerdings nicht von Anfang an Bestandteil von JavaScript, sondern wurde erst mit der JavaScript Version 1.5 voll unterstützt.
Dazu wurde das document-Objekt um einige Funktionen erweitert. Mittels `document.getElementById()`, `document.getElementsByName()` und `document.getElementsByTagName()` kann direkt und gezielt auf Knoten im DOM Baum zugegriffen werden. Diese Methoden liefern Objekte vom Typ Node zurück, welche wiederum Funktionalitäten zum Traversieren und Manipulieren des DOM Baumes bereitstellen.
Mittels DOM ist es jetzt nicht nur möglich auf alle Elemente eines Dokumentes zuzugreifen, sondern theoretisch und praktisch auch möglich ein komplettes Dokument „on-the-fly" zu erzeugen, sprich dem Dokument neue Elemente hinzuzufügen – wie es etwa beim Google Web Toolkit praktiziert wird.

Event handling

Eine zentrale Rolle im Bezug auf DHTML und Web 2.0 spielen Events, die bei jeder Benutzeraktion gefeuert werden. Events sind ebenfalls wieder Objekte. Beispiele für diese Events sind Verändern der Browsergröße, Überfahren von HTML-Elementen mit der Maus oder Tastaturanschläge.
Damit diese Events verarbeitet werden können müssen passende Eventhandler für die entsprechenden Events registriert werden. Ist kein passender Eventhandler registriert verpufft das Event.



Eventhandler sind Mehtoden die automatisch zum Verarbeiten des Events aufgerufen werden. Erkennbar sind sie, da sie immer mit „on" beginnen – etwa `onhover`, `onclick`, `onkeydown`,... Die Verarbeitende Funktion bekommt implizit eine Referenz auf das Event-Objekt als Parameter mit übergeben und kann somit direkt auf das Event-Objekt zugreifen und beispielsweise den Eventtyp auslesen.

```
var heading = document.getElementsByTagName(„h1“)[0];
heading.onmouseover = function (e) {
   alert(e.type);
}

//output:      mouseover
```

Eventhandler ermöglichen somit die Realisierung äußerst dynamischer Websites, die an gewohnte Desktop Applikationen sehr nahe herankommen.

Sicherheit

JavaScript befindet sich in einer Sandbox. Das heißt es ist mit reinem JavaScript weder möglich auf das Dateisystem noch auf sonstige Ressourcen des Betriebssystemes zuzugreifen. Die einzige Möglichkeit etwas clientseitig mit Hilfe von JavaScript zu speichern sind Cookies, welches eine Eigenschaft des document-Objekts ist.

```
document.cookie = “var1=wert1“;
var cookieInhalt = document.cookie;
```

Somit bringt es aus dem Aspekt der Sicherheit keinen Vorteil JavaScript zu deaktivieren.