JAVASCRIPT

PROF. ANDREAS HARTMANN - WEBTECHNOLOGIEN

QUELLEN, LITERATUR

- Quellenangaben

  Der Inhalt richtet sich – soweit nicht anders gekennzeichnet – aus an:

  - [1] JavaScript: das umfassende Handbuch, Christian Wenz, Galileo Press, Bonn, 2014

    (http://d-nb.info/102467035X)

  - [2] JavaScript effektiv, David Herman, dpunkt.verlag GmbH, Heidelberg, 2014

    (http://d-nb.info/1042204551)

  - [3] Professionell entwickeln mit JavaScript : Design, Patterns und Praxistipps, Philip Ackermann, Rheinwerk, Bonn, 2015

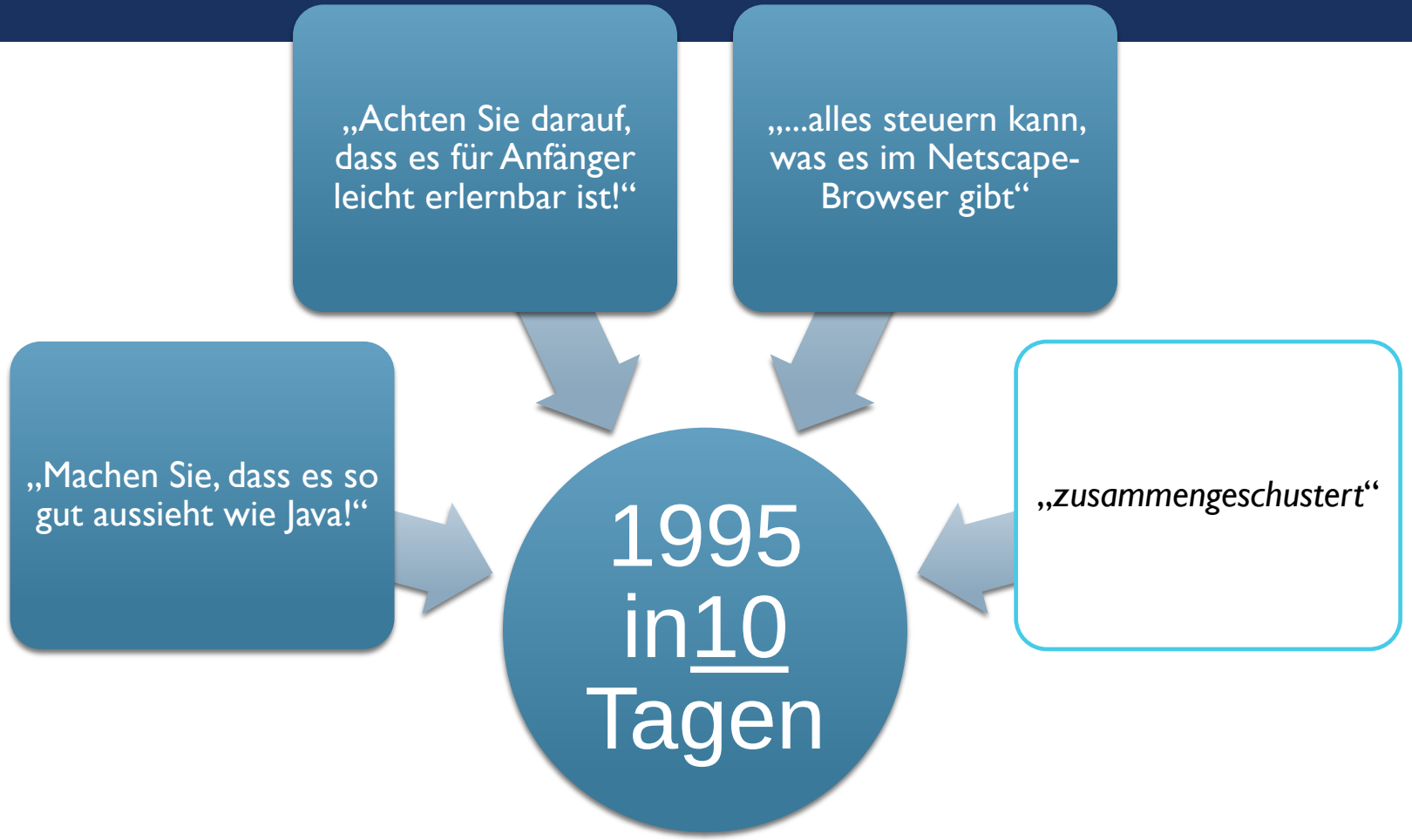
    (http://d-nb.info/1062929810)

- Es handelt sich um eigene Abbildungen, wenn keine Quelle angegeben ist.
- Einige Inhalte sind an Wikipedia angelehnt. Als Quellangaben wurden die dort angegebenen Referenzen geprüft.

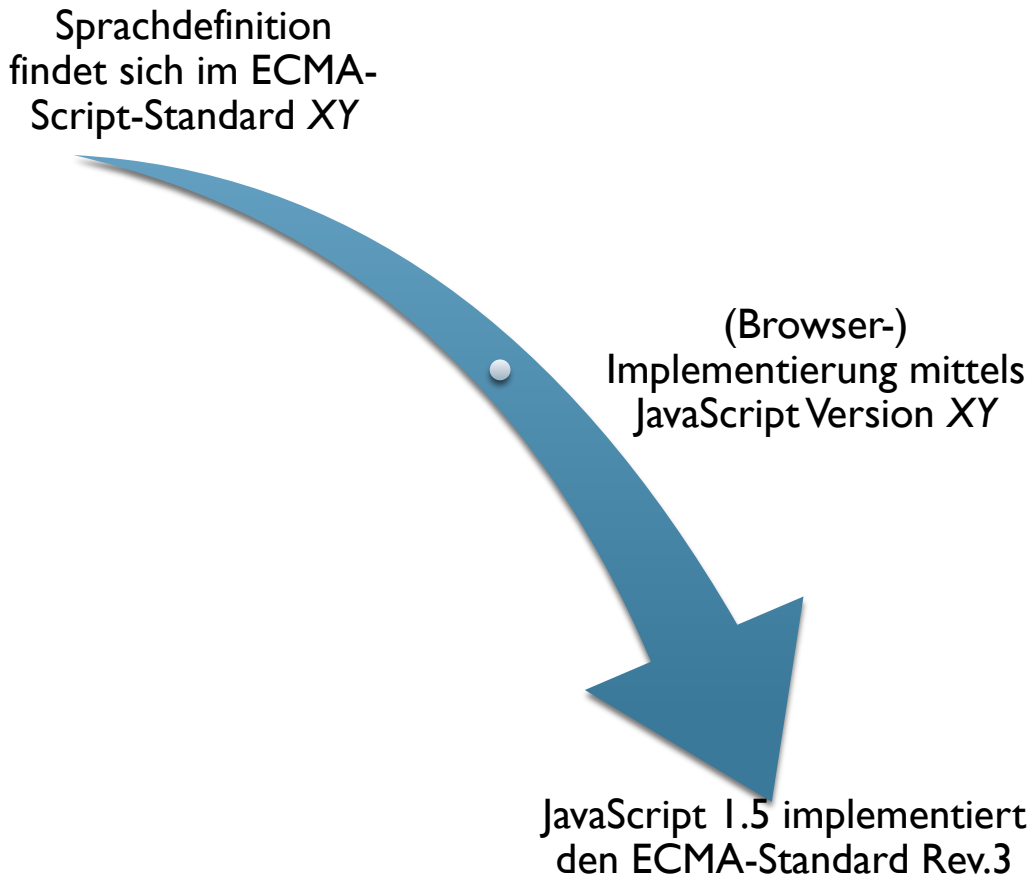
INHALTE

- Grundlagen
- Ereignisbehandlung
- DOM (-Manipulation)
- Frameworks
- TypeScript
- Versionswirrwarr

ENTSTEHUNG


„Achten Sie darauf, dass es für Anfänger leicht erlernbar ist!“
„...alles steuern kann, was es im Netscape-Browser gibt“
„Machen Sie, dass es so gut aussieht wie Java!“
1995 in10 Tagen
„zusammengeschustert“

STANDARD VS. IMPLEMENTIERUNG



Im Folgenden schauen wir uns die Grundlagen an. Sie sind ausreichend für die Projektaufgabe. Für professionelle Implementierungen ist jedoch weit mehr notwendig (siehe [2] und [3])!

GRUNDLAGEN

GRUNDLAGEN DER PROGRAMMIERSPRACHE JAVASCRIPT

INHALT

- Variablen
- Operatoren
- Kontrollstrukturen
- Datenstrukturen
- Funktionen
- Objekte

VARIABLEN

Bezeichner

Numerische

Zeichenketten

Boolesche

Deklaration

BEZEICHNER

- Buchstaben, Ziffern, Unterstrich
- Case-Sensitiv
- Einschränkungen:
  - keine Leerzeichen
  - kein Bindestrich (und einige weitere Sonderzeichen)
  - keine Ziffer am Anfang
  - keine Umlaute
  - keine Schlüsselwörter (z.B. Javascript reservierte Namen)

NUMERISCHE VARIABLEN

- keine strenge Typisierung wie in anderen Sprachen
- in Fakt werden Zahlen als *64bit Fließkommazahlen* kodiert (nach Standard IEEE 754)
- Trennzeichen ist der Dezimalpunkt
- Typ ist einfach nur: *number*
- Beispiele:
  - `Dieselpreis = 1.08;`
  - `Weltuntergang = 2012;`
  - `falscheProphezeiung = -3;`

ZEICHENKETTEN (STRINGS)

- Anführungszeichen (einfach ODER doppelt)
- Backslash als Steuerzeichen
  - z.B. `windowsC=„C:\\“;`
  - `\r \n \t \b \f` sind weitere Steuerzeichen in Zeichenketten
- Beispiele:
  - `myName = „Hartmann“;`
  - `my1stName = ‚Andreas‘;`

BOOLESCHE VARIABLEN

- Wahrheitswert: *true*/*false*
- Beispiel:
  - `koennengruenedrachenfliegen = false;`

VARIABLENDEKLARATION

- Variablendeklaration und Inititialisierung
  - `var myName = „Hartmann“;`
- neue Wertzuweisung möglich
- Javascript identifiziert selbständig den benötigten Typ
- Disziplin beim Programmierer notwendig!

OPERATOREN

Arithmetische Operatoren

Logische Operatoren

Zeichenkettenoperatoren

ARITHMETISCHE OPERATOREN

- Addition (+)
- Subtraktion (-)
- Multiplikation (*)
- Division (/)
- Modulo (%)
- Negation (-)
- Inkrement (++)
- Dekrement (- -)

LOGISCHE OPERATOREN

- UND (&&)
- ODER (||)
- Vergleichsoperatoren

  ==

  !=

  <

  >

  <=

  >=

BITWEISE ARITHMETISCHE OPERATOREN

- Besonderheit, denn hier werden Fließkommazahlen implizit zur Berechnung in Integer umgewandelt
- gerechnet wird mit 32bit Integer in Big-Endian-Format
- Beispiel:
  - `8 | 1  //ergibt 9`

Erläuterung

`8 (1000)` und `1 (0001)` werden logisch mit `OR` verknüpft
`1000 | 0001` ergibt `1001`
`1001` entspricht als Fließkommazahl der `9`

ZEICHENKETTENOPERATOREN

- Build-In
- Konkatenation/Verkettung (+)
- Zeichenketten werden als Objekte behandelt:
  - `var myName = „Hartmann“`
  - `myName.charAt(x)` – liefert das Zeichen an Stelle x
  - `myName.length` – liefert Anzahl der Zeichen
  - `myName.subString(start, end)` – liefert ein Teil der Zeichenkette

TYPISIERUNG

- Automatisch und nicht streng
- Keine implizite Prüfung (vgl. streng typisierte Sprachen)
- Typwandlung kann während einer Operation erfolgen
  - z.B. `var myAge=„31“; myAge *=1;` → ergibt 31 als Zahl
- mehr Kontrolle explizit mittels Funktionen:
  - `parseInt()`
  - `parseFloat()`
  - `toString()`

KONTROLLSTRUKTUREN

Schleifen, (break, continue)

Bedingungen

SCHLEIFEN - WHILE

```
while (Bedingung) { ... }
```

- die Bedingung wird zuerst geprüft
- danach wird der Block ausgeführt

```
do{ ... }while(Bedingung)
```

- Schleifenblock wird erst ausgeführt, danach wird die Bedingung geprüft

SCHLEIFEN - FOR

- Syntax: Initialisierung, Bedingung, Befehlsfolge
- Schleifenkörper in geschweifte Klammern
- Beispiele:
  - `for (var i=0; i<10; i++) { ... }`
  - `for (var i=0, j=„“; i<10; i++, j+=i) {...}`

BEDINGUNGEN - IF

- Schachtelung und Kurzschreibweise möglich
- bessere Schreibweise mit geschweiften Klammern

```
if (Bedingung) {
...
} else if (Bedingung2) {
...
} else {
...
}
```

BEDINGUNGEN - SWITCH

```
switch (Ausdruck) {
 case Wert1: ...; break;
 case Wert2: ...; break;
 ...
 default: ...;
}
```

FELDER/ARRAYS

- Deklaration mittels drei verschiedener Schreibweisen
  - `var woche = new Array(); woche[0] = „Montag“; woche[1]=„Dienstag“; ...`
  - `var woche = new Array(„Montag“, „Dienstag“, ...);`
  - `var woche = [„Montag“, „Dienstag“, ...];`
- Das Array kann Elemente unterschiedlichen Typs enthalten.
- Es wird als Objekt repräsentiert.

ARRAY - OPERATIONEN

- Zugriff
  - mittels Index beginnend mit 0
- Funktionen (Arrays sind Objekte!), z.B.:
  - `sort( )` – sortiert das Array
  - `join( )` – fügt ein Trennungszeichen ein und gibt Zeichenkette zurück
  - `length` – gibt die Länge zurück
  - `shift( )` – gibt das erste Element zurück und rückt den Rest nach
  - ...

FUNKTIONSDEKLARATION

- Funktionen sind ebenfalls Objekte in Javascript!
- Es gibt auch anonyme Funktionen.
- Deklaration mit Funktionskonstruktor
  - `var handleThis = new Function(„a“, „b“, „return a+b“);`
- Deklaration als Funktionsliteral
  - `var summe = function(a,b) {return a+b;}`
  - summe ist hier der Bezeichner einer Variablen und enthält als Wert eine Referenz auf die Funktion
  - `var ergebnis = summe (1,2); //liefert das Ergebnis 3`

Schauen Sie sich unbedingt noch einmal die Funktionen unter Javascript an. Insbesondere die Verwendung von anonymen Funktionen und die Behandlung als Objekte.

PARAMETERÜBERGABE

call by value

- Übergabe von Werten
- ändern sich nicht nach außen
- *Kopie*

call by reference

- Übergabe von Objekten
- ändern sich nach außen
- *Original*

OBJEKTE IN JAVASCRIPT

- JavaScript ist objektorientiert
- durch die enge Verbindung zum Browser ist der Zugriff auf einige vordefinierte Objekte möglich, z.B. `document`, `window`, `location`
- anders als z.B. in Java kommen zahlreiche Aufgaben ohne die Definition eigener Objekte aus

Wir schauen uns jetzt ein Beispiel für die Verwendung von vordefinierten Objekten im Browser an.

```
//Information holen
var ua =navigator.userAgent.toLowerCase();

//Information auswerten
var mozillaBrowser =(ua.indexOf(„gecko/“)!=-1);
var ieBrowser = (ua.indexOf(„msie“) > -1 &&
 ua.indexOf(„opera“)== -1);
var chromeBrowser = (ua.indexOf(„chrome“) > -1);

//Weiterleitung einrichten
if (chromeBrowser){
 location.href=„chrome.html“;
}
if (ieBrowser){ ... }
```

ZUGRIFF AUF DEN BROWSER

Javascript bietet Build-In Features für den Browserzugriff (über das Objekt `navigator`)

z.B. die Browsererkennung

```
<!DOCTYPE html>
<html>
  <head>
    <title>Browsererkennung</title>
  </head>
  <body>
   <h1>Script</h1>
   <script type="text/javascript">
    document.write("<h2>Browser erkennen</h2> <br>");
    //Information holen
    var ua =navigator.userAgent.toLowerCase();
    //Information auswerten
    var mozillaBrowser = (ua.indexOf("gecko/") != -1);
    var ieBrowser = (ua.indexOf("msie") > -1 && ua.indexOf("opera") == -1);
    var chromeBrowser = (ua.indexOf("chrome") > -1);

    document.write("Browser ist Chrome: "+chromeBrowser);
   </script>
  </body>
</html>
```

VOLLSTÄNDIGES BEISPIEL (HTML)

Script wurde in HTML-Datei eingebunden

Ausgabe über `document`

EREIGNISBEHANDLUNG

JAVASCRIPT

JAVASCRIPT / BROWSER

- Erinnern Sie sich - JavaScript wurde primär für den Browser entwickelt und soll den Zugriff unterstützen!
- Javascript-Unterstützung **ist** eine Implementierung des Browserherstellers (vgl. JIT)
- Achtung: Ereignisse werden von Browserherstellern ggf. unterschiedlich implementiert

EREIGNISBEHANDLUNG

- Im Browser „passiert“ etwas – wir nennen es ein ***Ereignis***
- Ereignisse sind insbesondere:
  - Nutzeraktionen (Mausklick, Tastatureingabe)
  - Systemereignisse
- Ereignisse sind an Objekte respektive Elemente gebunden
- JavaScript liefert vordefinierte Objekte und Funktionen, um Ereignisse abzufragen, d.h. zu behandeln.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Ereignisbehandlung</title>
  </head>
  <body>
      <h1>Script - Ereignisse</h1>
      <form>
        <input type="button" value="hier klicken"
                onclick="window.alert('click');"
        />
      </form>
  </body>
</html>
```

MAUSEREIGNIS ABFRAGEN

vordefinierte Funktion `onclick`

Ereignis an Button „gebunden“

öffnet ein Warnfenster

DIESE BEGRIFFE SOLLTEN SIE KENNEN.

EIGENE EVENT-LISTENER

- ggf. werden eigene/zusätzliche Event-Listener benötigt
- `addEventListener()` und `removeEventListener()`
- sie beziehen sich i.d.R. auf ein bestimmtes Ereignis
  - daher ist der Name des Ereignisses = erster Parameter im Funktionsaufruf
- Beispiel (mit anonymer Funktion)

```
...
var einElement = document.//Zugriff auf ein
Element;
einElement.addEventListener(
 „click“,
 function() {
  alert(„Geklickt“);
 },
 false);
...
```

JAVASCRIPT UND DOM

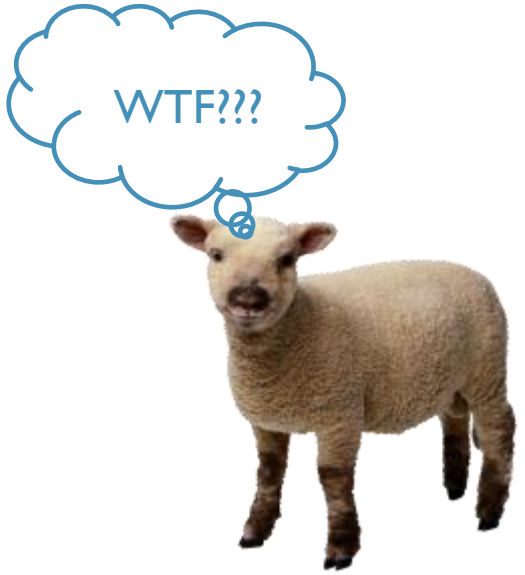
JAVASCRIPT

ZUR ERINNERUNG

- ECMA-Standard definiert die Sprache
- Browserhersteller implementieren JavaScript
- Browserhersteller implementieren HTML-Unterstützung
- Der Browser...
  - ... modelliert DOM
  - ... compiliert Script (JIT)

DIE IDEE

- Die Idee ist, dass alle Browserhersteller ein gleiches Modell implementieren (DOM) und natürlich auch den Sprachstandard ECMA für Javascript.
- Somit könnten Entwickler auf eine saubere und vom Browser unabhängige Umgebung aufsetzen.
- Mit Javascript könnte kontrolliert in die HTML-Umgebung eingegriffen werden.



Betrachten wir noch einmal unser HTML-Beispiel.

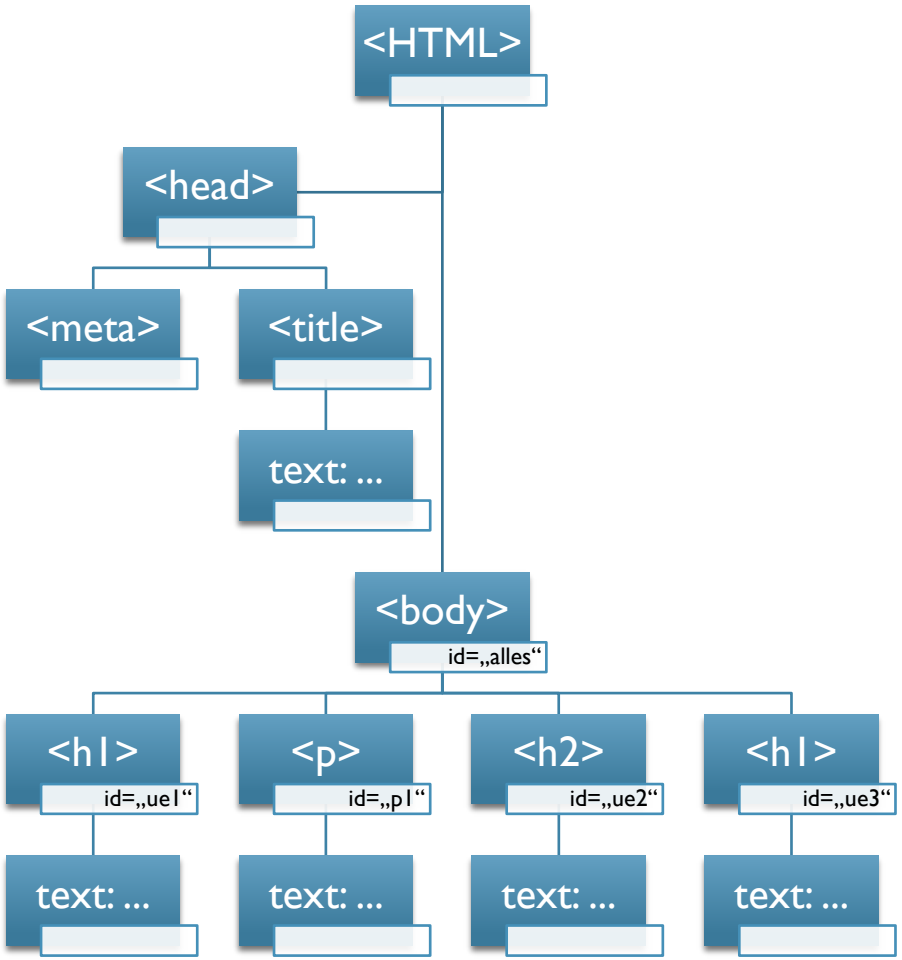
```
<!doctype html>
<html>
 <head>
<meta charset="UTF-8">
  <title>Titel der Webseite</title>
 </head>
 <body id=„alles“>
  <h1 id=„ue1“>Überschrift der 1. Ebene</h1>
  <p id=„p1“>Text, Absatzmarke</p>
  <h2 id=„ue2“>Überschrift der 2. Ebene</h2>
  <h1 id=„ue3“>Noch eine Überschrift</h1>
 </body>
</html>
```

WIEDERHOLUNG DOM

Unser Beispiel aus HTML

erweitert mit ID‘s

Und so würde der DOM-Tree aussehen (Auszug).


<HTML>
<head>
<meta>
<title>
text: ...
<body>
id=„alles“
<h1>
id=„ue1“
<p>
id=„p1“
<h2>
id=„ue2“
<h1>
id=„ue3“
text: ...
text: ...
text: ...
text: ...

Zum Verständnis ist ein kleiner Ausflug zur Graphentheorie notwendig.

EXKURS: GRAPHENTHEORIE

Der DOM-Baum kann als ***gewurzelter Baum*** i.S. der Graphentheorie betrachtet werden, wobei die Wurzel dem Element <html> entspricht.

Es handelt sich um einen gerichteten Graphen (Out-Tree)

die Knoten werden auch als ***Blätter*** bezeichnet.

Jedes Blatt kann von der Wurzel kreisfrei über genau einen ***Pfad*** erreicht werden.

D.h. wir können den DOM-Baum mit bekannten Algorithmen bearbeiten bzw. parsen!

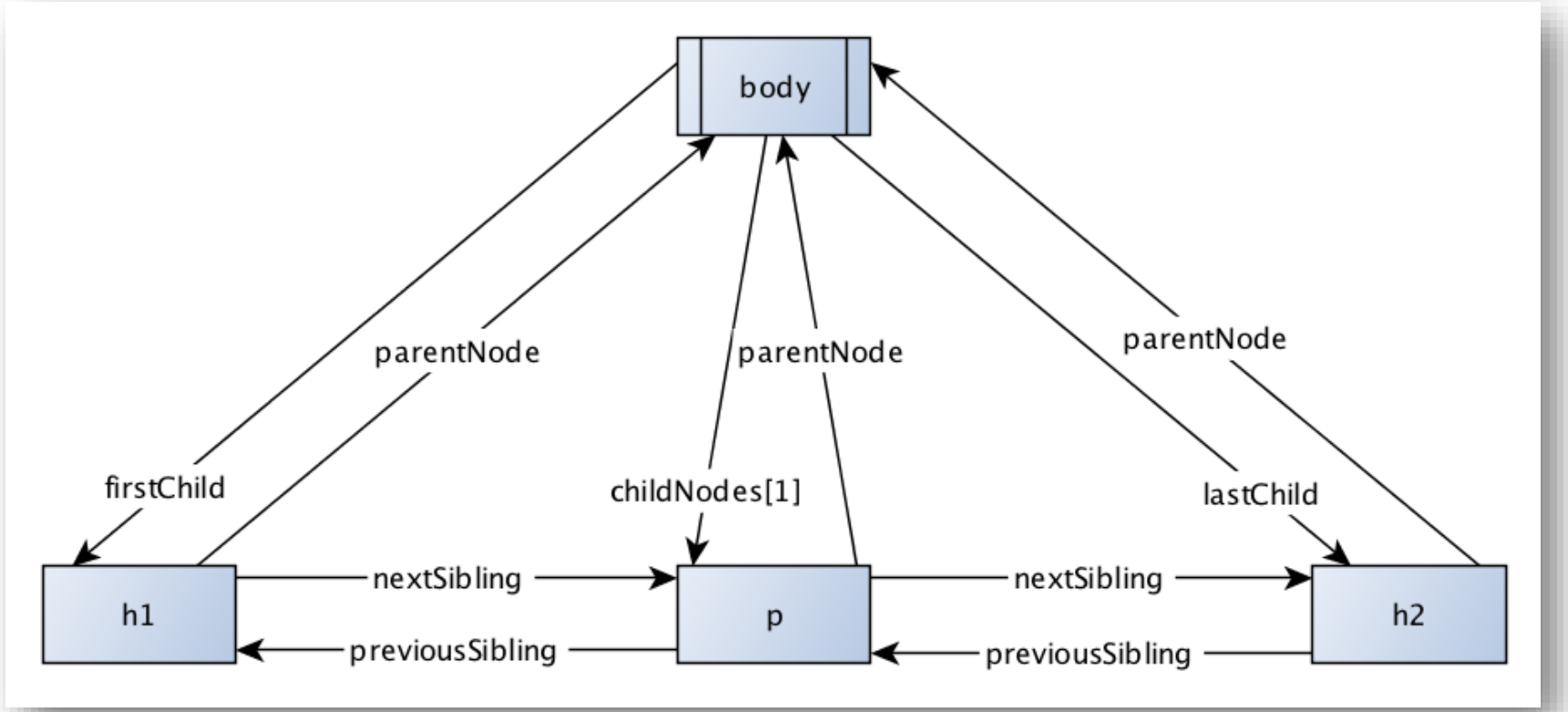
ZUGRIFF AUF DAS DOM

- das DOM wird vom Browser aus HTML-Dokument erstellt
- ist über das `document`-Objekt referenzierbar
- Kind-Knoten sind über das Array `childNodes` im Eltern-Knoten referenzierbar
- Beispiel:
  - `document.getElementById(„alles“).childNodes;`
  - ...liefert alle Kind-Knoten von `<body id=„alles“>` aus unserem Beispiel

WEITERE EIGENSCHAFTEN VON KNOTEN

`firstChild`	• (Kante zum ersten Kind)
`lastChild`	• (Kante zum letzten Kind)
`nextSibling`	• (Kante zum nächsten Kind bzw. Nachbarn)
`parentNode`	• (Kante zum Eltern-Knoten)
`previousSibling`	• (Kante zum vorherigen Kind bzw. Nachbarn)
`nodeName`	• (HTML-Tag des Knotens als String)
`nodeType`	• (Tag, Attribut oder Text)

NAVIGATION IM DOM



MANIPULATION DES DOM

Methode	Syntax	Beschreibung
appendChild	parent.appendChild(Child)	Hängt Child als Kind-Knoten an Parent an.
cloneNode	original.cloneNode(alles)	Erzeugt eine identische Kopie von Original; wenn alles==true dann werden die Kinder ebenfalls kopiert.
createElement	anode.createElement(tag)	Erzeugt einen neuen Knoten (HTML-Tag)
hasChildNodes	anode.hasChildNodes()	Boolescher Wert
insertBefore	anode.insertBefore(child,sibling)	Fügt einen Knoten child als Kind von anode vor sibling ein.
removeNode	anode.removeNode(alles)	Entfernt den Knoten anode aus dem DOM (einschließlich Kinder, wenn alles==true)
replaceNode	anode.replaceNode(newnode)	Der Knoten anode wird durch newnode ersetzt.
setAttribute	anode.setAttribute(name,value)	Das Attribute name wird neu gesetzt.

OBJEKTORIENTIERUNG

→ SELBSTSTUDIUM (SIEHE LITERATURVERWEISE)

FEHLERBEHANDLUNG

JAVASCRIPT

FEHLERBEHANDLUNG

- „The good news is...“ – es gibt einen Event-Handler
- leider nicht so ausgereift, wie z.B. bei Java
- Ein Beispiel...

```
<!doctype html>
<html>
 <head>
 <meta charset="UTF-8">
  <title>Titel der Webseite</title>
   <script type="text/javascript">
    function viewError(txt){
     alert(txt);
     return true;
    }
    window.onerror = viewError;
    var test = hoppla;
   </script>
 </head>
 <body>
  <h1>Überschrift der 1. Ebene</h1>
  <p>Text, Absatzmarke</p>
  <h2>Überschrift der 2. Ebene</h2>
 </body>
</html>
```

FEHLER ABFANGEN

Der Fehler wird über einen vordefinierten Event-Handler abgefangen

Die auszuführende Funktion muss als Referenz übergeben werden!

Alternativ können Sie wie immer eine anonyme Funktion deklarieren.

```
<!doctype html>
<html>
 <head>
 <meta charset="UTF-8">
  <title>Titel der Webseite</title>
   <script type="text/javascript">
       try{
           var test = hoppla;
       }catch(e){
           alert(e);
       }finally{
           //do something else
       }
   </script>
 </head>
 <body>
  <h1>Überschrift der 1. Ebene</h1>
  <p>Text, Absatzmarke</p>
  <h2>Überschrift der 2. Ebene</h2>
 </body>
</html>
```

FEHLER BEHANDELN

Fehleranfälliger Code kann in einem `try-catch`-Block ausgeführt werden.

FENSTER

ZUGRIFF, FENSTER ERZEUGEN, FENSTER BEARBEITEN

DAS HAUPTFENSTER

- ... haben wir bereits kennengelernt. Es ist über das vordefinierte Objekt `window` zugreifbar und stellt die Basisumgebung für das auszuführende Script dar.

FENSTER ERZEUGEN

- Fenster mit festem Kontext
  - `window.alert()`
  - `window.confirm()`
  - `window.prompt()`
- „eigene“ Fenster
  - `window.open(`*URL, Name, Option*`)`

OPTIONEN FÜR FENSTER

- `var myWindow = window.open(url, name, options);`
- Übergabe als Liste „`location=yes, status=1`“
- Optionen können z.B. sein:
  - `location = yes`
  - `status = 1`
  - `toolbar = no`
  - `resizable = yes`
  - `width = 768`
  - ... siehe [1] Seite 195 ff.

```
var myWindow = window.open(url,name,options);

myWindow.document.open();
myWindow.document.write(„<html>“);
myWindow.document.write(„<body>“);
...
myWindow.document.write(„</html>);
myWindow.document.close();
...
```

FENSTER MIT INHALT BEFÜLLEN

Natürlich können Sie in das neue Fenster HTML-Code einfügen – denkbar einfach!

FRAMEWORKS

WIE GESAGT, HEUTE FINDET EIN HOHER ANTEIL AN JAVASCRIPT-ENTWICKLUNG MIT UNTERSTÜTZUNG VON FRAMEWORKS STATT.

EINIGE FRAMEWORKS... (UND TECHNOLOGIEN)

- Ajax, JSON
- Javascript und CSS
- Cookies
- jQuery
- Canvas und zeichnen
- Node.js

AJAX

- Ajax ist keine neue Technologie
- ... wurde von Jesse J. Garrett geprägt zunächst als
  - Asynchronous JavaScript + XML  schnell wieder vergessen!
- ... beschreibt folgende Funktionalität
  - im Hintergrund wird ein XMLHttpRequest abgesetzt und anschließend ausgewertet
  - das Ergebnis wird auf der Webseite dargestellt (ohne neu zu laden)
- prominente Vertreter sind Google und Amazon
  - kontextabhängige Suchvorschläge während der Nutzereingabe
- Browser implementieren dafür ein vordefiniertes Objekt
  - `var xhr = windows.XMLHttpRequest();`

JSON

- ***J**ava**S**cript **O**bject **N**otation*
- standardisiert die Serialisierung von Arrays und Objekten
- d.h. die Ausgabe als Zeichenkette
- das funktioniert auch umgekehrt – `JSON.parse()`

- Anwendung? Serialisierung ist immer dann notwendig, wenn Daten übertragen werden sollen. So z.B. bei Ajax.

JAVASCRIPT UND CSS

- Die im CSS definierten Eigenschaften und Werte können über die Knoten des DOM referenziert werden.
- Schreibweise muss angepasst werden:
  - aus `background-color` wird `backgroundColor`

SCRIPT

```
var myParagraph = document.getElementById(„myP“);
myParagraph.style.color = „white“;
myParagraph.style.backgroundColor = „black“;
```

HTML

```
<p id=„myP“>...
```

```
...
<style type=„text/css“>
 .normal {color: black; background-color: white;}
 .invers {color: white; background-color: black;}
</style>
<script type=„text/javascript“>
 function setzeKlasse(klasse){
  var Absatz = document.getElementById(„Absatz“);
  Absatz.className = klasse;
 }
</script>
...
<p id=„Absatz“> ...
...
<form>
 <input type=„button“ value=„Normal“
  onclick=„setzeKlasse(‚normal‘);“/>
 <input type=„button“ value=„Invers“
  onclick=„setzeKlasse(‚invers‘);“/>
</form>
...
```

STYLE MIT JAVASCRIPT SETZEN

Nützlich ist der Zugriff auf den Klassenselektor eines Elements

besonders in Verbindung mit dem Einsatz von Frameworks!

COOKIES

- das HTTP-Protokoll ist zustandslos, d.h. Informationen gehen bei Verlassen der Seite verloren
- ungünstig wenn es sich z.B. um Kundendaten oder Credentials einer Anmeldung handelt
- Cookies erlauben das permanente Speichern von Informationen auf dem Client
- Textdateien, die Spezifikation stammt von Netscape

JAVASCRIPT UND COOKIES

- Cookies sind über die Referenz `document.cookies` erreichbar
- als Zeichenkette gespeichert
- Werte können direkt als String übergeben werden
- Zum Lesen des Cookies muss die gesamte Zeichenkette durchsucht werden! (→ Serialisierung einsetzen)

JQUERY

- Hierbei handelt es sich um eine umfangreiche Bibliothek (Framework) für Javascript, bestehend aus:
  - jQuery Core
  - jQuery UI (UI-Effekte)
  - QUnit (ein Test-Framework)
  - Sizzle (gehört zum Core, Selektion zur Auswahl von Elementen)
- gepackte/ungepackte Version
- zentrale Funktion: `$()`

```
$(„#Liste“).append($(„<li>“)
        .append($(„<a>“)
        .attr(„href“, „http://www.jquery.com“)
        .text(„jQuery“)));
```

MALEN MIT JAVASCRIPT

- Basis ist die Leinwand – `canvas` von HTML5
- der Kontext definiert, ob es um 2D oder 3D gehen soll
- da die Leinwand als Objekt angesprochen werden kann, können Funktionen zum Malen aufgerufen werden

```
...
var c = document.getElementById(„c“);
var ctx = c.getContext(„2d“);
ctx.fillStyle = „red“;
ctx.fillRect(0,0,50,100);

...
<canvas id=„c“ width=„1024“ height=„768“></canvas>
```

NODE.JS

- Javascript für Serverseitigen Einsatz
- browserunabhängige Ausführung (JIT)
  - JavaScript-Engine V8 von Google (http://code.google.com/p/v8)
- Node.js bringt umfangreiche Bibliotheken mit, für die Entwicklung und Unterstützung von z.B.:
  - Serverapplikationen
  - Datenbankzugriff (mySQL, MongoDB, ...)
  - Dateizugriff
  - IT-Sicherheit
  - Loadbalancing

TYPESCRIPT

JAVASCRIPT „ERWEITERN“

TYPESCRIPT

- JavaScript bietet einige Freiheiten, die jedoch auch die Anfälligkeit für Fehler erhöhen (z.B. fehlende Typsicherheit).
- TypeScript liefert wichtige zusätzliche Features, wie z.B. Typisierung oder Klassenkonzepte.
- Dabei bleibt JavaScript eine Teilmenge von TypeScript.
- Im einfachsten Fall „übersetzt“ der TypeScript-Compiler den Quellcode 1:1 nach JavaScript.
- In der fortgeschrittenen Programmierung erzeugt er gültiges JavaScript – kombiniert mit den genannten Vorteilen.

ENDE