Javascript - The Good Parts

Eine Einführung in Programmierung von Webapplikationen mit Javascript.

Die Entstehungsgeschichte von JavaScript

Brendan Eich entwickelt 1995 die Sprache Mocha/LiveScript für den Netscape Navigator 2.0

NETSCAPE®

18. September

1995

NETSCAPE®

Sun Microsystems und Netscape kooperieren - Ziel: Javaapplets werden mit LiveScript gesteuert.

LiveScript wird in Javascript 1.0 umbenannt, Netscape stellt LiveConnect zur Verfügung.

4. Dezember

1995

ECMA - European Computer Manufacturer Association

Die ECMA übernimmt die Standardisierung des allgemeinen Teils von Javascript:

Die einfachen Variablentypen 'number', 'string' und 'boolean' werden genau beschrieben, deren Deklaration und Verhaltensweisen.

Ebenso die komplexen Typen 'array', 'object' und 'function'.

Dazu kommen einige Funktionsobjekte/-konstruktoren: Date, Math, RegExp

Die Methode 'var' wird Pflicht, um Variablen zu deklarieren, Die 'use strict' Klausel trennt ECMA-konformes Scripting von 'freiem' Schreiben.

Heute gilt der ECMA Standard 5.1. In bereits anstehenden ECMA Standard 6.0 werden weitere prägnante Entscheidungen getroffen, so werden zum Beispiel 'class', 'private', public', 'protected' und weitere OO-Muster dazukommen.

Die Mozilla Foundation:
DOM - Javascript

Die Mozilla Foundation als Nachfolger der ehemaligen Netscape Entwicklergruppe entwickelt den DOM-Part von Javascript, der für das Verhalten im Browser verantwortlich ist und implementiert dies in seinen Browser 'Firefox'

Darin befinden sich alle DOM-Objekte und Methoden, wie 'window', 'document', 'navigator' etc. Diese werden von allen Browserherstellern weitestgehend übernommen. Microsoft entwickelt hier teilweise eigene Lösungen, die besser zu Architektur der Microsoft-Software passen: zum Beispiel das ActiveXObjekt, das bei Mozilla XMLHttpRequest heisst.

Mozilla entwickelt das NICHT-DOM-Objekt console zur Ausgabe von Werten und Typen in der Browserconsole. (Firebug war auch die erste Konsole für das Debugging im Browser).

Die Javascript Engines in den Browsern

Jeder Browser verwendet eine eigene Engine zur Interpretation und Kompilierung von Javascript.

Chrome, Safari und Opera verwenden eine Version der Opensource Javascript Engine V8. Microsoft hat xxx entwickelt und verwendet diese ab der Versionen 10 des Internet Explorer.

Alle folgen mittlerweile den Vorgaben der ECMA und von Mozilla,

Sie implementieren zusätzlich spezifische Objekte und Methoden, so gibt es zum Beispiel in Googles Chrome ein 'chrome' Objekt, das die Grundlagen für Chrome-Apps liefert.

Bekannte Anwendungen von Javascript

Browser

Adobe Flash

Adobe PDF

Node.js

Jurassic

JavaScript in HTML einbinden

Deprecated: Direct JavaScripts

```
<head>
   <script language="javascript" type="text/javascript">
   <!--
      hier kommt das Script ...
   // -->
   </script>
</head>
```

Auskommentieren des Skriptcodes

Der HTML-Kommentar versteckt das Javascript vor sehr alten Browser, die so daran gehindert werden, den Javascript Code als irgendetwas anderes zu interpretieren.

```
<script language="javascript" type="text/javascript">
<!--
   hier kommt das Script ...
// -->
</script>
```

Die Javascript-Zeile

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

① Es kommt ein Script

② Es ist JavaScript (nur bis HTML 4)

③ Der MIME-type ist "text/javascript"

Die meisten Validierer monieren ein fehlendes TYPE, aber die Browser führen das Skript dennoch aus.

In HTML 4.01 ist es Pflicht, in HTML5 nur optional.

async und defer ④ ⑤

```
<script ... [async="async"] [defer="defer"]>
```

④

async

Das Script wird asynchron zum Rest der Seite ausgeführt. Es startet, wenn es geladen ist.

NEU seit HTML5!

~~④~~ ⑤

defer

Das Script wird in der Ladereihenfolge ausgeführt. Deferred Scripts laufen vor allen anderen geladenen Skripten.

~~④~~ ~~⑤~~

Das Skript wird unmittelbar nach dem Lesen ausgeführt, bevor der Browser mit dem Parsen der übrigen Seite fortfährt.

JavaScript Includes

Das Einbinden externer Javascript-Dateien hat einige Vorteile:

Es muss nur eine (einzige) Datei aktualisiert werden, alle Seiten nutzen automatisch das (geänderte) Skript.

Die HTML Seiten sind übersichtlicher und bleiben sauber;

```
<html>
<head> ... </head>
<body>
   ...
   <script src="_scripts/javascript.js" ... ></script>
</body>
</html>
```

Wo wird Javascript aufgerufen?

Im Allgemeinen wird Javascript heute am Ende des Body Elementes der HTML Datei. Das stellt einen reibungslosen Ladevorgang des DOM und CSS sicher, bevor mit dem Parsen und Kompilieren von Javascript begonnen wird.

charset

```
<script ... charset="UTF-8"></script>>
```

⑥

charset

Das charset Attribut spezifiziert das Character Encoding der inkludierten Datei.

Es wird benötigt, wenn die inkludierte Datei einen anderen Zeichensatz verwendet, als die HTML Datei.

JavaScript Includes

Wenn das src-Attribut verwendet wird, muss das Script-Element leer bleiben!

```
<script src="_scripts/javascript.js" ... >
<!--
   hier steht das Skript
-->
</script>
```

Die Komponenten von JavaScript

Javascript

Javascript ist eine in Objekten "denkende" Sprache.

Objektbasiert, nicht objektorientiert.

Objekt bedeutet hier entweder ein Objekt aus dem Document Object Model, sprich ein HTML-Knoten (Node), oder ein selbst erzeugtes.

Ähnliche Sprachen finden wir mit Actionscript 3 oder Python.

Objekte in JavaScript

Objekte in JavaScript bestehen aus Eigenschaften, die als Name/Wert-Paar realisiert werden. Dabei wird nicht zwischen Attributen und Methoden des Objektes unterschieden.

Jedes Objekt – auch durch Literale erzeugte Objekte – erbt vom Prototyp des globalen Object-Konstruktors.

Vordefinierte Objekte

Eingebaute Objekte, die von ECMAScript definiert werden.

Das namenlose globale Objekt, das alle Variablen und Objekte enthält.

Object als allgemeiner Prototyp, von dem alle Objekte abgeleitet sind

Function als Prototyp für Funktionen

Array als Prototyp für Arrays

String als Prototyp für Zeichenketten

Boolean als Prototyp für boolesche Variablen

Number als Prototyp für Zahlen (64-Bit-Gleitkommazahlen gemäß IEEE 754)

Math stellt Konstanten und Methoden für mathematische Operationen bereit. Math kann nicht als Konstruktor dienen.

Date für Operationen mit Daten bzw. Zeitpunkten und Datumsformaten

RegExp für reguläre Ausdrücke

Browserobjekte

Die restlichen Objekte, die beim clientseitigen JavaScript verwendet werden, entstanden historisch vor allem durch die Netscape-Spezifikationen (window, document usw.).

Das window-Objekt selbst ist dabei de facto das globale Objekt, indem einfach einer Variablen window das globale Objekt zugewiesen wurde.

Objekte des Document Object Model

Zahlreiche Unterobjekte von document wurden mittlerweile durch DOM HTML standardisiert (title, images, links, forms usw.). Aktuelle Browser unterstützen zudem DOM Core und andere W3C-DOM-Standards sowie Erweiterungen von Microsoft JScript.

window für Browser-Fenster,

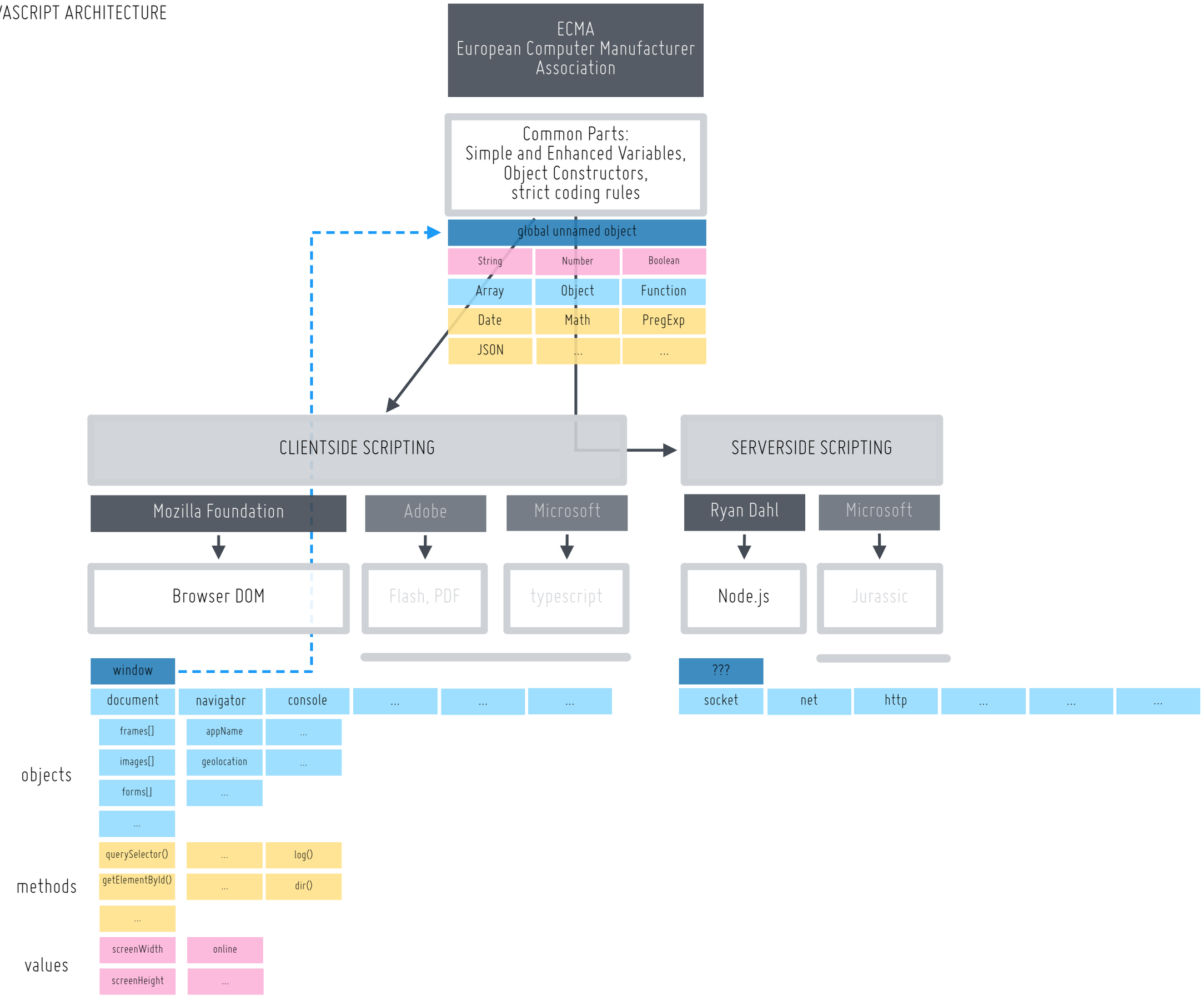
navigator für allgemeine Browsereigenschaften wie Plugins, Browserversion usw.,

screen für Daten zum Bildschirm/Display des Benutzers,

history für die Liste zuvor besuchter URLs

location für die Adresszeile des Browsers

THE JAVASCRIPT ARCHITECTURE



Schreiben von Javascript

Editor (z.B. Notepad++, Sublime) und Browser

WebStorm (IDE mit Debugger)

Netbeans, Eclipse

Visual Studio 11

Es empfiehlt sich ein Editor mit Intellisense und Syntaxprüfung, da JavaScript durchaus Fehler zulässt und erst im Browser kompiliert wird.

Eigenschaften von Javascript

(C/C++/Java) – ähnliche Syntax

Objekte, aber keine Klassen

Wird im Browser kompiliert

Funktionsscopes, keine lokalen Gültigkeitsbereiche

Typen, aber keine Typenüberprüfung

ECMAScript 5

ES5 löst einige der kritischen Bestandteile (bad parts) von Javascript durch Standardisierung.

Die meisten aktuellen Browser unterstützen ES5.

"use strict";
unterstützt das Schreiben von besserem Javascript Code.

Verwendung nur innerhalb von Funktionsscopes, nicht global!

Wortreservierungen

ECMAScript
break case catch continue debugger default delete do else finally for function if in instanceof new return switch this throw try typeof var void while with

Zukunft (1):
class enum export extends import super

Zukunft (2) (strict):
implements interface let package private protected public static yield

Source : https://developer.mozilla.org/en/JavaScript/Reference/Reserved_Words

Die Sprachelemente von JavaScript

Kommentare

Kommentarzeichen

```
// slashslash für einzeiligen Kommentar

/*
    slashstar
    für einen Block
    Kommentar
*/
```

Verwendung von Javadoc-kompatiblen Kommentaren ist gut fürs Teamwork

```
/* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: */
/**
 * Short description for file
 *
 * Long description for file (if any)...
 *
 * written in Javascript 1.7, jQuery 1.7.2
 *
* @category    mySoftwareCategory
* @package     myApplication
* @author      Michael Reichart <reichart@michaelreichart.de>
* @author      Christian Marx
* @copyright   1999-2012 Michael Reichart
* @license     http://www.michaelreichart.de/license/1.txt
* @version     SVN: $Id$
* @link        http://host.net/package/myApplication
* @since       File available since Release 1.0.0
* @deprecated  File deprecated in Release 2.0.0
*/
```

Standardkommentare für Funktionen

```
/**
 * Short description for Method
 *
 * Long description for method (if any)...
 *
 * @author     Michael Reichart
 * @author     Your Name
 * @version    1.0.0
 * @since      Method available since Release 1.0.0
 * @deprecated File deprecated in Release 2.0.0
 *
 * @param  type $varName
 * @return type
 */
```

Variablennamen und Bezeichner

Variablenname beginnen mit

- einem Buchstaben
- Oder _
- Oder $.

```
var a, _b, $c;
```

Gefolgt von keinem oder mehr Buchstaben,
Ziffern, _ oder $.

```
var _, $;
var a1, b_3, c$, myCamelCaseVariablesName
```

Nach der allgemeinen Konvention beginnen alle Variablen, Parameter, Member, oder Funktionsnamen
mit einem kleinen Buchstaben.

```
var a = 2;
var myVariableName = null;
var obj = {};
var array = [];
```

Konstruktorfunktionen beginnen mit einem Grossbuchstaben.

```
function Auto () { ... }
var myAuto = new Auto();
```

Ein initialer _ sollte für Implementierungen reserviert bleiben.

$ sollte für die Verarbeitung durch Maschinen reserviert bleiben.

Funktionen schreiben

Eine einfache Log - Funktion:

```
function log(msg) {
  console.log('log: ' + msg);
}
```

Ein Funktionsaufruf

```
log('hallo Welt');
```

Objektnotationen

JavaScript Object Notation - JSON

Ein Objekt ist eine dynamische Sammlung von Eigenschaften. Jede Eigenschaft hat eine String als Namen. Der String ist unique! Die Attribute sind public !

```
var tier = {
   art     : "Hund",
   beine   : 2,
   fluegel : 0,
   bellen  : function(){
      log('wau!');
   }
};
```

Get, set und delete

get

object.name
object[expression]

set

object.name = value;
object[expression] = value;

delete

delete object.name
delete object[expression]

Scopes -
Gültigkeitsbereiche von Variablen

Erzeugen globaler Variablen

```
<script>
a = 2;          // global
var c = 3;        // global
function f () {
   b = 3;       // auch global
   var d = 4;
}
</script>
```

Dabei spielt es keine Rolle, ob die Variable innerhalb oder ausserhalb einer Funktion gesetzt wird.

Implizite globale Variablen!

Jede Variable, die nicht sauber deklariert wurde, wird als globale Variable behandelt.

Für Programmierer, die sich um die Konzepte von Kapselung von Funktionen keine Gedanken machen, ist es so sehr einfach, zu Ergebnissen zu kommen.

Aber globale Variablen machen Software unzuverlässig und anfällig.

(global) Object

Das Objekt, das seinen Namen nicht nennt.

Es ist der Container für alle globalen Variablen und in Javascript enthaltenen Objekte.

In Browsern ist window ist das globale Objekt.

Es ist nicht empfehlenswert, globale Objekte oder Variablen zu verwenden, da sie von überall her überschrieben werden können und leicht in Konflikt mit anderen globalen Variablen oder Objekten geraten.

Benutzen Sie niemals implizite globale Variablen!

Und überhaupt:
Vermeiden sie auch jede globale Variablen so gut es geht.

Global variables should be as rare as hen’s teeth,
and should stick out like a sore thumb.

ALLCAPS

Schreiben Sie nach ECMAScript 5.1

Ein "use strict"; am Anfang einer Funktion wird bei einem fehlendes var (in manchen Browsern) einen Fehler werfen.

Es gilt innerhalb des Gültigkeitsbereiches der Funktion.

Ein "use strict" am Anfang eines Javascriptes würde innerhalb des global Gültigkeitbereiches für alle nachfolgenden Skripte gelten.

Das scheint schön - man braucht es nur einmal zu setzen

Ist aber nicht schön - Jedes Skript muss sich nun an der Standard halten, auch externe Libraries, die gar nicht von ihnen sind.

Selbst jQuery ist nicht durchgängig nach ECMA 5.1 geschrieben und melden bei einer entsprechenden Prüfung Fehler.

Lokale Variablen -"var" innerhalb von Funktionen

```
function f () {
    "use strict";
    var a;     // Lokale Variable mit var
        .
        .
        .;
}
```

Typen

Typen in Javascript

Variables haben keinen festen Typ!

a = 5; a = 'Hallo';

Objekte haben einen Typ!

Typen in Javascript

number (kein int, float, double)

string (no char); a = 'Hallo'; a="H";

boolean a = 3>4; a=false;

object a = {};

array a = [];

function a = function () {};

Numbers

Numbers

Only one number type; no integer types

64-bit floating point

IEEE-754 (aka “Double”)

Numbers

```
var n1 = 0xa3; // hex

var n2 = 078; // 078 was Octal

var n3 = 9.81; // dec
```

Good that we don’t have int

a > 0

b > 0

c = a + b;

Possible results:

c < a

c < b

Numerische Literale

.01024e4

1.024e+3

10.24E2

102.4E+1

1024.e0

1024.00

1024

10240e-1

Dezimalzahlen sind Näherungswerte

```
a = 0.1;

b = 0.2;

c = 0.3;

(a + b) + c === a + (b + c)  // false
```

Methoden für numerische Werte

var a = 4

a.toExponential()

a.toFixed()

a.toLocaleString()

a.toPrecision()

a.toString()

a.valueOf()

Zahlen sind wie Objekte - eine Zahl kann in einer Variaben gespeichert werden.

Als Parameter übergeben werden.

Von einer Funktion zurückgegeben werden.

In einem Objekt gespeichert werden.

Eine Zahl darf Methoden haben.

Math object

abs

acos

asin

atan

atan2

ceil

cos

exp

floor

log

max

min

pow

random

round

sin

sqrt

tan

Number methods

```
if (!Number.prototype.trunc) {
    Number.prototype.trunc =
        function trunc(number) {
            return Math[
                number >= 0 ? 'floor' : 'ceil‘
            ](number);
        };
}
```

NaN

Ein ganz besonderer Zahlwert: Not a Number

Ist das Ergebnis von nicht definierten
oder fehlerbehafteten Rechenoperationen.

NaN ist giftig: Jede arithmetische Operation,
die mit einem NaN rechnet, wird NaN als Ergebnis liefern.

NaN ist mit nichts anderem vergleichbar,
auch mit sich selbst nicht.

```
NaN === NaN    // false
NaN !== NaN          // true
```

Boolean

Boolean

```
var un; var obj = {};

Boolean(null);      // false

Boolean(un);        // false

Boolean(0);         // false

Boolean(1);         // true

Boolean("");        // false

Boolean("A");       // true

Boolean(parseInt('AA'));
                                // NaN:
        // false

Boolean(obj);               // true

Boolean(undefined == null);
        // true

Boolean(undefined !== null);
        // true
```

String

Strings

Eine Folge von 0 oder mehr 16 bit Unicode Buchstaben (UCS-2, nicht etwa UTF-16)

No awareness of surrogate pairs

Kein eigener Typ "char"
Einzelne Buchstaben sind ein String mit der Länge 1.

String Literale können einfache oder dopptelte Anführungszeichen verwenden. Bei Vermengung kann mit \ escaped werden.

```
// http://unicode.org/charts/
var uni = "\u0993\u0994\u0995\u0996\u0997\u0998";
log(uni);
```

String length

string.length

Die "length" Eigenschaft gibt die Anzahl der 16-bit Zeichen in einem String zurück.

Erweiterte Zeichen werden als 2 Zeichen gezählt.

+

+ kann Strings verbinden oder addieren.

'$' + '1' + '2' === '$12'

'$'.concat('1').concat('2')

'$' + 1 + 2 = ‚$12‘

1 + 2 + „$“ = „3$“

Wie man eine Zahl in einen String konvertiert

Die number Methode:

str = num.toString();

Oder die String Methode

str = String(num);

Wie man einen String in eine Zahl konvertiert

Mit der Number Funktion: num = Number(str);

Mit dem + Präfixoperator: num = +str;

Mit der parseInt/parseFloat Funktion: num = parseFloat(str);

Die parseInt Funktion

parseInt(str, 10)

Konvertiert einen Wert in eine Zahl.
Die Konvertierung endet beim ersten nicht Ziffern-Zeichen.

parseInt("12em") === 12

Das Anhängsel (10) sollte immer verwendet werden.

parseInt("08") === 0 (ES3)

parseInt("08", 10) === 8

String Methoden

charAt

charCodeAt

compareLocale

concat

indexOf

lastIndexOf

localeCompare

match

replace

search

slice

split

substring

toLocaleLowerCase

toLocaleUpperCase

toLowerCase

toString

toUpperCase

trim

valueOf

Array

Arrays

array = ['value', 'value', 'value'];

Arrays erben von Objekten.

Arrayindizes werden zu Strings konvertiert und können zur Werteermittlung verwendet werden.

array[0]; array[1];

Sehr effizient für kleine Arrays, aber in den meisten anderen Fällen nicht.

Ein Vorteil: Array benötigen bei ihrer Erzeugung
keine Angabe der Arraylänge.

length

Arrays haben, nicht wie Objekte, eine besondere length Eigenschaft.
Sie ist immer um eines größer als der höchste im Index eingetragene Integerwert.

Man kann also das gewohnte for Statement verwenden:

```
for (i = 0; i < a.length; i++) {
      ...
}
```

Dagegen kann das "for ... in" in Array nicht benutzt werden.

Arrayliterale

Die Arrayschreibweise verwendet ein []
Sie kann jeglichen Ausdruck beinhalten, getrennt von Kommas:

```
myList = ['oats', 'peas', 'beans'];
```

Neue Einträge können einfach hinzugefügt werden:

```
myList[myList.length] = 'barley';
myList.push('barley')
```

Allerdings sollte die Punktnotation nicht für Arrays verwendet werden.

Array methods

concat

every

filter

forEach

indexOf

join

lastIndexOf

map

pop

push

reduce

reduceRight

reverse

shift

slice

some

splice

toLocaleString

toString

unshift

sort

```
var n = [4, 8, 15, 16, 23, 42];

n.sort();

// n is [15, 16, 23, 4, 42, 8]
```

Elemente eines Array löschen

delete array[number]

Löscht das Element,
hinterlässt aber eine Lücke in der Arrayelementefolge.

array.splice(number, 1)

Löscht ein Element und
nummeriert die restlichen Elemente neu.

Elemente eines Array löschen

```
myArray = ['a', 'b', 'c', 'd'];

delete myArray[1];

// ['a', undefined, 'c', 'd']

myArray.splice(1, 1);

// ['a', 'c', 'd']
```

Arrays vs. Objects

Benutzen sie Objekte, wenn die Indizes aus willkürlichen Literalen bestehen;

Benutzen sie Arrays, wenn der Index aus einer sequentiellen Folge von Integerzahlen besteht.

Lassen sie sich vom Begriff "assioziatives Array" nicht durchenander bringen.

Date

Die Date Funktion basiert auf der Date Klasse von Java.

Date

```
var now = new Date();
var t1 = now;
log('year ' + now.getFullYear());
log('month ' + now.getMonth() + 1); // Jan: 0!!!
log('day ' + now.getDate()); // not getDay
log('day of week ' + now.getDay());
for (var i = 0; i < 1000000; ++i) {
var a = i * i * i / Math.PI;
}
var t2 = new Date();
log('took ' + (t2 - t1) + ' msec.'); // Performance
```

RegExp

/\/(\\[^\x00-\x1f]|\[(\\[^\x00-\x1f]|[^\x00-\x1f\\\/])*\]|[^\x00-\x1f\\\/\[])+\/[gim]*/i

RegExp

Ein regulärer Ausdruck spezifiziert eine Syntax eines Suchausdrucks für Texte.

RegExp erlaubt keine Leerzeichen oder Kommentare
→ Sie sind schwer zu lesen.

Ein einfacher regulärer Ausdruck in Javascript:

```
var re = /abc/;
```

Besondere Zeichen und Ausdrücke

//	Start – Ende eines RegExp
\	escaped Slash (*, …)
^	zu Beginn
$	am Ende
.	Wildcard, beliebiges Zeichen
|	Alternative
[]	Set/Bereich von Zeichen [0-9], [A-Za-z0-9] findet jedes dieser Zeichen

Zeichenbereiche

[xyz] Zeichenbereich, der alle notieren Zeichen erfasst

[^xyz] Erfasst keines der notierten Zeichen

Besondere Zeichen und Ausdrücke

+	einen oder mehrere der vorangehenden Zeichegruppe
?	Keinen oder einen ...
*	keinen oder mehr ...
{x}	x mal ...
{x,}	x mal oder mehr ...
{x,y}	x bis y mal ...

Besondere Zeichen und Ausdrücke

\b	Wortgrenze (Anfang/Ende)
\s	Jedes Leerzeichen, Tabulator oder Umbruch
\d	Jede Ziffer
\w	Jedes Vorschubzeichen
\n	Zeilenvorschub
\w	Jeder Buchstabe, inkl. _

Gruppierungen

(…) Gruppe; höchstens 9 Gruppen

?:(…) Ausschlußgruppe

Parameter, auch Kombinationen sind möglich

/…/g globale Suche

/…/i unabhängig von Groß- und Kleinschreibung

/…/m Multiline

JSON

Das JSON - Objekt

`JSON.parse(text [, reviver])`

Schreibt ein als JSON-String gegebenes Objekt in ein Objekt in den Speicher. So kann ein per Ajax gelieferter JSON-String weiterverarbeitet werden.

`JSON.stringify(value [, replacer] [, space])`

Schreibt ein im Speicher liegendes Objekt in einen String um. Der String kann nun zum Server geschickt oder gespeichert werden.

Notierenswertes

Alle Werte sind Objekte

Außer null und undefined.

null

Ein Wert, der überhaupt nichts ist.

undefined

Ein Wert, nicht das ist, was er vorgibt.
Undefined ist der Standardwert für Variablen und Parameter.
Es ist der Wert für fehlende Member in Objekten.

typeof

Der typeof Präfixoperator gibt den Typ eines Wertes als String zurück.

type	*typeof*	
object	'object'	
function	'function'	
array	'object'	!!!
number	'number'	
string	'string'	
boolean	'boolean'	
null	'object'	!!!
undefined	'undefined'	

```
Array.isArray

if (typeof Array.isArray !== 'function') {
    Array.isArray = function (value) {
        return Object.prototype.toString.apply(value)
                === '[object Array]';
    };
}
```

Falsy values

false
null
undefined
"" (empty string)
0
NaN

Alle anderen Werte sind truthy (einschließlich aller Objekte).
"0" "false"

Schwach typisiert

Jeder der Typen kann in einer Variable gespeichert werden, oder als Parameter in einer Funktion übergeben.

Javascript ist hat keinesfalls keine Typisierung.

Referenzen

Objects können als Argumente in Funktionen übergeben werden. und können von Funktionen als Returnwert zurückgegeben werden.

Objekte werden dabei als Referenz behandelt.

Der === Operator vergleicht Objektreferenzen, und nicht deren Werte.
Nur wenn beide Operanden ein und dasselbe Objekt sind.

Herstellen einer trim Funktion, falls nicht vorhanden

```
if (typeof String.prototype.trim !== 'function') {
    String.prototype.trim = function () {
        return this.replace(/^\s*(\S*(\s+\S+)*)\s*$/,
"$1");
    };
}
```

supplant

```
var template =   '<table border="{border}">'
             +   '<tr><th>Last</th><td>{last}</td></tr>'
             +   '<tr><th>First</th><td>{first}</td></tr>'
             +   '</table>';

var data = {
    first: "Carl",
    last: "Hollywood",
    border: 2
};

mydiv.innerHTML = template.supplant(data);
```

supplant

```
if (typeof String.prototype.supplant !== 'function') {
    String.prototype.supplant = function (o) {
        return this.replace(/{([^{}]*)}/g,
        function (a, b) {
            var r = o[b];
            return typeof r === 'string' ? r : a;
        });
    };
}
```

Operatoren

Operatoren

Arithmetrisch	+ - * / %
Vergleichend	== != < > <= >= === !==
Logisch	&& || !
Bitweise	&	^ >> >>> <<
Ternary	? :

+

Addition und Stringverkettung

Wenn beide Operanden numerisch sind,
dann addiere
sonst
wandle sie in Strings und verkette sie

'$' + 3 + 4 = '$34'

+

Unäre Operatoren können Strings in Zahlen konvertieren.

+"42" = 42

Auch: Number("42") = 42

Und: parseInt("42", 10) = 42

+"3" + (+"4") = 7

/

Die Teilung von zwei Integerzahlen
kann einen Nicht-Integerwert hervorbringen.

10 / 3 = 3.3333333333333335

== !=

Ist gleich und ist nicht gleich

Beide Operatoren können eine Typenänderung erzwingen.

Es ist daher immer besser === und !== zu verwenden, hier bleiben die Typen erhalten. Daher ist der Vergleich typensicher.

Typenänderungen erzwingen ist nicht gut.

```
'' == '0'           // false
0 == ''             // true
0 == '0'            // true
false == 'false'    // false
false == '0'        // true
false == undefined  // false
false == null       // false
null == undefined   // true
' \t\r\n ' == 0     // true
```

||

Der "default" Operator, das logische Oder

Wenn der erste Operand truthy ist
dann ist der erste Operand das Ergebnis
sonst ist der zweite Operand das Ergebnis.

Damit können Defaultwerte gesetzt werden:

var last = input || nr_items;

Wenn input truthy ist, dann last = input;
sonst ist last = nr_items.

Wenn der erste Operand eine 0 ist,
funktioniert das nicht ganz wie erwartet. 0 ist falsy.

&&

Der Wächteroperator, das logische Und

Wenn der erste Operand truthy ist, dann ist der zweite Operand das Ergebnis sonst der erste.

Das logische Und kann verwendet werden, um null Referenzen zu verhindern:

```
if (a) {
	return a.member;
} else {
	return a;
}
```

Kann auch geschrieben werden als:

```
return a && a.member;
```

!

Präfixoperator für eine logisches Nicht.

Wenn der Operand truthy ist, ist das Ergebnis false.

Im anderen Fall true.

!! produziert ein boolsches Ergebnis.

Bitweise

& | ^ >> >>> <<

Die bitweisen Operatoren konvertieren die Operanden zu einer 32-bit vorzeichenfähigen Integerzahl und liefern das Ergebnis wieder als 64-bit Fließkommazahl ab.

Statements

Statements

expression

if

switch

while

do

for

break

continue

return

try/throw

For statement

Iteriert durch alle Elemente eines Arrays:

```
for (var i = 0; i < array.length; i++) {
    // within the loop,
    // i is the index of the current member
    // array[i] is the current element
}
```

For in statement

Iteriert durch alle Elemente eines Objektes:

```
for (name in object) {
    if (object.hasOwnProperty(name)) {
        // within the loop,
        // name is the key of current member
        // object[name] is the current value
    }
}
```

Switch statement

Mehrwegverzweigung

Der Switch-Wert muss keine Zahl, sondern kann auch ein String sein.

Die einzelnen Cases können Ausdrücke und Programmzeilen beinhalten.

Gefährlich: Cases fallen in den nächsten Case durch, solange der Case nicht durch ein bread unterbrochen und beendet wird.

Switch statement

```
switch (expression) {
    case ';':
    case ',':
    case '.':
        punctuation();
        break;
    default:
    noneOfTheAbove();
}
```

```
while () {}
```

```
while (condition) {
    doSomethingAwesome();
}
```

```
var a = 0;
while(a<4) {
   doSomethingAwesome();
   a++;
}
```

do {} while ()

```
do {
    doSomethingAwesome();
} while (condition)
```

```
var a = 0;
do {
   doSomethingAwesome();
   a++;
} while(a<4)
```

Throw statement

```
throw new Error(reason);

throw {
    name: exceptionName,
    message: reason
};
```

Try statement

```
try {
      ...
} catch (e) {
      switch (e.name) {
          case 'Error':
          ...
          break;
      default:
          throw e;
      }
  }
```

Try Statement

Die JavaScript Implementierung kann folgende Exception names ausgeben:

'Error'
'EvalError'
'RangeError'
'SyntaxError'
'TypeError'
'URIError'

Die Browserobjekte in Javascript

flash.js

```
function hasFlash() {
   var hasFlash = false;
   for (i = 0; i < navigator.plugins.length; i++) {
      if(navigator.plugins[i].name.indexOf('Flash') > 0)
      {
         hasFlash = true;
      }
   }
   if(hasFlash) alert('Flash!');
   else window.location = 'deadend.html';
}
```

Pluginliste

Mit dem folgenden Code kann die Liste aller installierten Plugis ausgegeben werden. Es verwendet das document Objekt und Methoden aus dem DOM.

Pluginliste

```
function listPlugins() {
   var b = document.body;
   var p = document.createElement('div');
   p.id = 'plugins';
   document.body.appendChild(p);

   p.innerHTML = '<h3>Liste der Plugins</h3>';
```

Pluginliste

```
for (i = 0; i < navigator.plugins.length; i++) {
   var tmp = document.createElement('p');
   tmp.innerHTML = navigator.plugins[i].name;
   tmp.className = 'plugin';

   p.appendChild(tmp);
}
```

Das Window Objekt

Das windows Objekt liefert Informationen über das aktuell angesprochene Fenster (das das Dokument enthält).

Es kennt u.a. folgende Eigenschaften:

```
window.frames
window.location
window.name
window.opener
window.parent
window.status
```

Das Window Objekt hat Methoden

Methoden sind Fähigkeiten (was kann ...) eines Objektes.

```
windows.alert()
windows.blur()
windows.clearTimeout()
windows.close()
windows.confirm()
windows.focus()
windows.prompt()
windows.scroll()
windows.setTimeout()
```

Außerdem reagiert es auf Events

Das Window Objekt kennt u.a. folgende Eventhandler:

```
onBlur = ...          onFocus = ...
onLoad = ...          onUnload = ...
```

Zum Beispiel stellt der folgende Event sicher, dass das, was innerhalb der anonymen Funktion steht,erst ausgeführt wird wird, wenn das Dokument vollständig geladen ist.

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

Das Navigator Objekt

Das Navigator Objekt liefert Informationen über den verwendeten Browser.

`navigator.plugins` liefert zum Beispiel ein Array mit allen installierten Plugins.

Das Navigator Objekt

Es liefert außerdem folgende Werte

```
navigator.appName
navigator.appVersion
navigator.appCodeName
navigator.mimeTypes
navigator.plugins
navigator.userAgent
```

Eine vollständige Objektreferenz

http://de.selfhtml.org/javascript/objekte/index.htm

DOM Manipulation

Objekte des DOM im Javascript

document – Zugriff auf HTML-Elemente

window – Zugriff auf den Browser

console – Ausgaben zum Debugging

Objekte in Javascript

Ein Objekt ist eine dynamische Sammlung von Eigenschaften. Jede Eigenschaft hat eine String als Namen. Der String ist unique! Die Attribute sind public !

Es gibt bereits Browserobjekte wie window oder document.

Weitere Objekte können selbst geschrieben werden.

"Irgendwie ist in Javascript alles ein Objekt!"
Michael Reichart, beim letzten Seminar

Methoden und Attribute von window

window ist ein globales Objekt

document ist ein Attribut von window

```
window.document.title = 'Hello World';

===

document.title = 'Hello World';
```

Methoden und Attribute von window

innerwidth/innerheight

-> Größe des HTML Bereiches

setTimeOut()

-> Timer, wird für Animationen verwendet

Dokumentenknoten (DOM Nodes)
sind Objekte

```
<html>
  <head>...</head>
  <body>
    <div id="box"> ...</div>
  </body>
</html>

 ->

document.getElementById('#box');
```

Methoden, um den DOM zu manipulieren:

```
var node = document.createElement(‚h1‘);

node.setAttribute(‚class‘, ‚h1-main‘);

var child = document.createTextNode(‚text‘);

node.appendChild(child);

document.body.appendChild(node);
```

Identifizieren von Knoten

```
var node = document.getElementById('content');
```

```
getElementsByClassName("green")
getElementsByTagName("p")
querySelectorAll("p.green")
querySelector()
```

Interaktion verwendet Events, Events kann man hören

```
document.getElementById('catch01')
   .addEventListener("keydown",
         function (event) {
            log('catch01 '+event.keyCode);
         }
   );
```

Erst erzeugen, dann einfügen ...
- auch Textknoten sind Knoten

```
var head     = document.createElement('h1');
var headText = document.createTextNode('Schule ist
doof');

head.appendChild(headText);

document.body.appendChild(head);
```

Eigene Objekte,
deren Notationen und Einsatz

Objektnotationen

Ein Objekt ist eine dynamische Sammlung von Eigenschaften. Jede Eigenschaft hat eine String als Namen. Der String ist unique! Die Attribute sind public !

JavaScript Object Notation - JSON

```
var tier = {
   art     : "Hund",
   beine   : 2,
   fluegel : 0,
   bellen  : function(){
      log('wau!');
   }
};
```

Ein leeres Objekt erzeugen ...

```
var obj = {};

// aka
var obj = new Object();
```

Werte setzen und abfragen?
Alle Keys sind public.

```
var tier = {
    // Attribute
    type  : 'Hund',
    legs  : 4,
    wings : 0,

    // Methoden
    say : function () { console.log('wau'); },
    getType : function () { ... },
    setType : function (value) { ... },
 };

// Getter ? -> alle Attribute und Methoden sind public!
console.log(tier.type);
console.log(tier['legs']);

// Setter ?
tier.type = 'Katze';
tier.say = function () { console.log('miau')};
```

Get, set und delete

get

```
object.name
object[expression]
```

set

```
object.name =
value;
object[expression]
= value;
```

delete

```
delete object.name
delete
object[expression]
```

Funktionsobjekte

Eine Funktion als Objekt.

```
var animal = function (type, legs, wings, sound) {

    var type  = type  || 'Hund',
        legs  = legs  || '4',
        wings = wings || '0',
        sound = sound || 'wau'
    ;

    function say () { growl(sound) }
};

// Getter ?
growl('Direkter Zugriff: ' + animal.type);
```

Immediate Functions

Funktionen, die sich selbst ausführen.
Sie werden vor allen anderen Funktionen initialisiert
und können nebenbei dazu verwendet werden,
private Keys und Methoden zu verwenden.

Eine Immediate Function

```
()();

/*
Was, das soll eine Funktion sein?

Ja. Das erste Klammernpaar ist eine
Javascriptausdruck für eine anonyme,
sich nach dem Kompilieren sofort selbst
ausführende Funktion.

Das zweite Klammernpaar ist eine
Argumente-Klammer, mit der Argumente in
die Funktion übergeben werden können.

*/
```

Beispiel einer Immediate Function

```
var extArgs;

( function (intArgs) { ... } )(extArgs);

/*
innerhalb der Immediate Function wird in der Regel eine anonyme
Funktion platziert, die durch die Immediate Function ausgeführt
wird. Ihr werden die Argumente übergeben.
*/
```

Immediate Function mit Rückgabewert

```
var extArgs
    ExtObj;

extObj =  (function (intArgs) {
   var intObj = {};
   ...
   return intObj;
} )(extArgs);

/*
Da die Funktion sofort ausgeführt ist, gibt sie anstelle eines
Funktionszeigers ein Ergebnis zurück.
*/
```

Ein Objekt mit public und private Keys.

```
var animal = (function (type, legs, wings, sound) {

    var type  = type  || 'Hund',
        legs  = legs  || '4',
        wings = wings || '0',
        sound = sound || 'wau'
    ;
    function getType () { return type; }
    function setType (value) { type = value; }
    return ({
        getType : getType,
        setType : setType
    });
})();

// Getter!
console.log('Getterzugriff: ' + animal.getType());
```

Prototypische Objekte

Prototypische Verebung

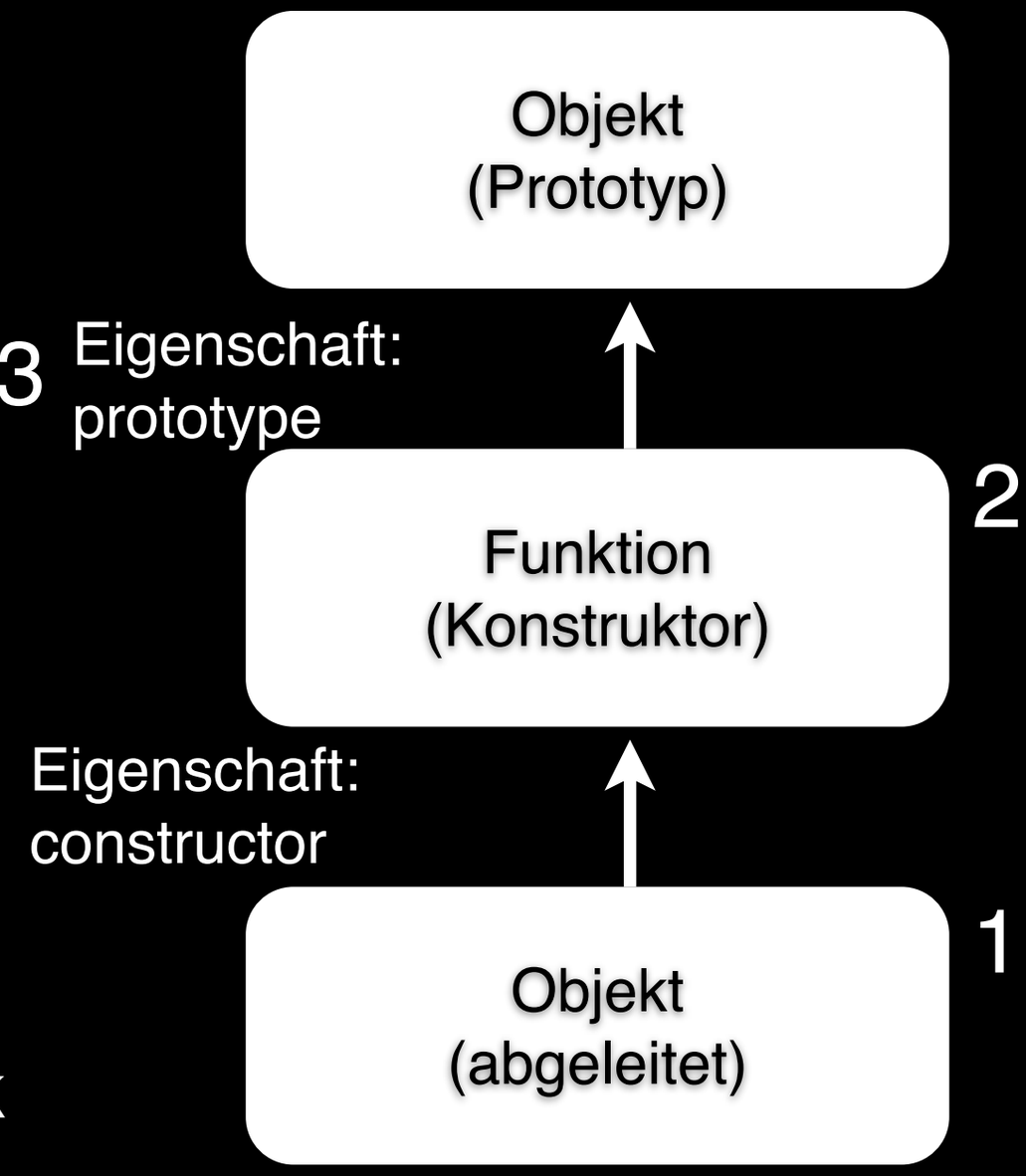
Die Prototypen Kette

1 Hole das Attribut
vom Konstruktorobjekte

2 Wenn keines da ist,
nimm das vom Prototypenobjekt

3 Wenn der Prototyp existiert
gehe zu 1.

4 Ansonsten gibt undefined zurück



Konstruktoren

Eine Funktion, mit dern neue Objete erzeugt werden muss mit den new prefix aufgerufen werden. Dabei wird ein leeres Objekt erzeugt, das auf das neue Objekt verweist.

```
function Tier(a) {
    this.art = a; // attribut
};

var tier = new Tier('tier');
```

Prototypen

Funktionen haben eine prototype-Eigenschaft

```
function Insekt(b) {
    this.beine = b;
    this.art = 'Insekt';
}

Insekt.prototype = tier;
var insekt = new Insekt(6);
```

Vereinfachung zur Erzeugung eines Objektes

```
function create(o) {
    var F = function() {};
    F.prototype = o;
    return new F();
};

// Anwendung:
    var insekt = create(tier);

// oder nach ES5: Object.create()
```

Auslesen eines Objektes

Loopt über die Eigenschaften eines Objektes

for att in obj { ... }

Das for in Problem

Vererbte Funktionen enthalten beim Durchzählen auch den Prototypen.

```
for (name in object) {
    if (object.hasOwnProperty(name) {
        …
    }
}
```

Das kann kann mit der hasOwnProperty Eigenschaft abgefangen werden.

Objekte mit privaten Eigenschaften und Methoden

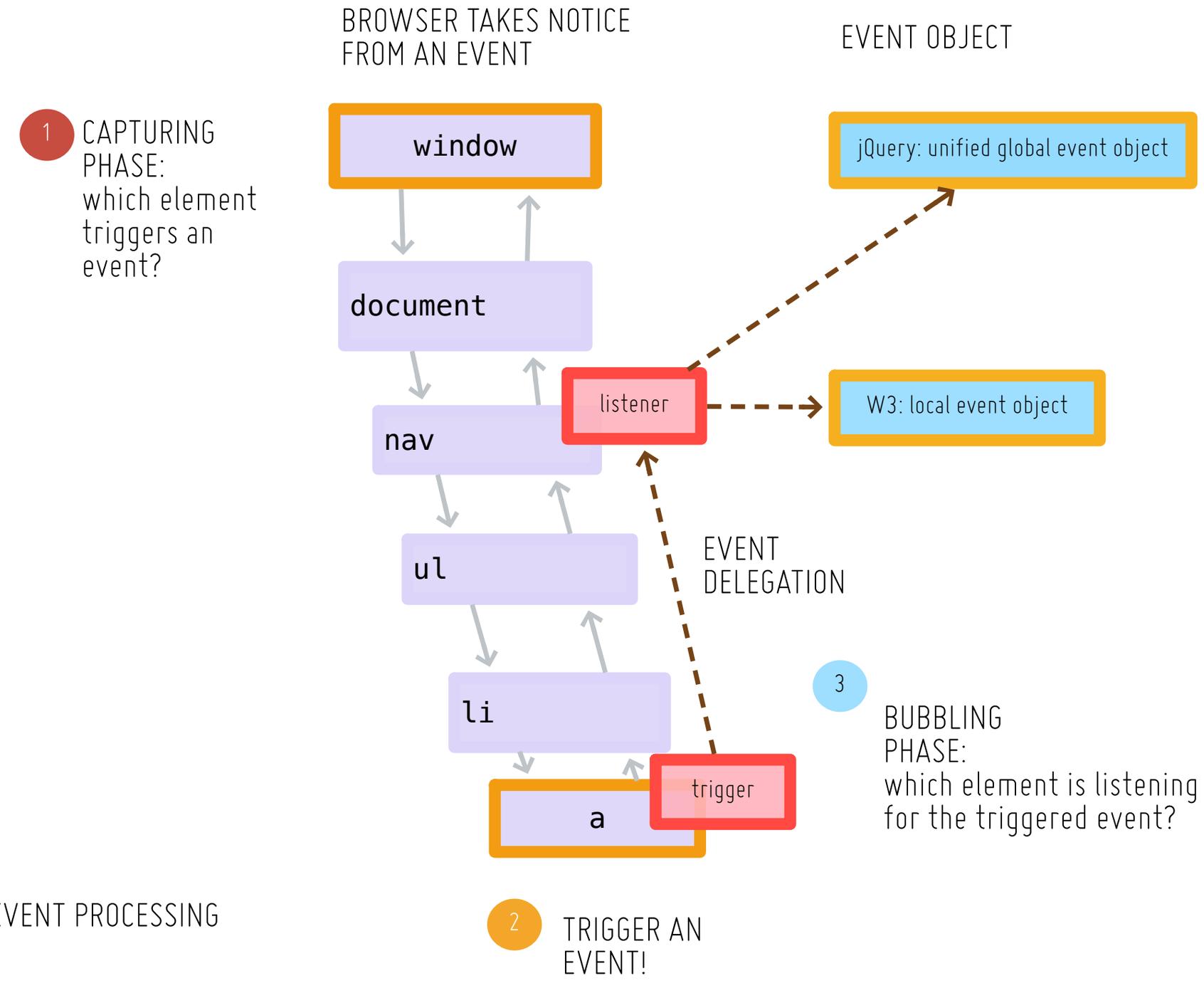
```
var obj = (function(){
	// Properties
	var publicKey  = 'value 1';
	var privateKey = 'value 2';

	// Public
	function setup () {
		log('Objekt setup!');
	}

	// Private
	function destroy () { log('Objekt destroy"');}

	return { setup : setup, publicKey : publicKey } // MAKE PUBLIC !
})()
```

Javascript Events



DOM EVENT PROCESSING

Events

Alle Elementobjekte und weitere Objekte besitzen die Methode addEventListener.

Ein Eventlistener enthält einen Ereignistyp, ein Elementobjekt und eine Handler-Funktion.

http://molily.de/js/event-handling-fortgeschritten.html

```
element.addEventListener(   "event",
                            handlerfunktion,
                            capturing);
```

"event" ist ein String und enthält den Ereignistyp:
"click", "mouseover", "load", "submit" ...

Das Handler-Funktionsobjekt, das ausgeführt werden soll.

Der dritte Parameter bestimmt, für welche Event-Phase der Handler registriert werden soll. Die Werte sind true oder false.

false	steht für die Bubbling-Phase (sollte als Standard verwendet werden).
true	für die Caputuring-Phase.

http://molily.de/js/event-handling-fortgeschritten.html

Ein Beispiel

```
window.addEventListener("load", start, false);


function start () {
   var pElement = document.getElementById("interaktiv");
   pElement.addEventListener("click", klickverarbeitung, false);
}

function klickverarbeitung () {
   document.getElementById("interaktiv").innerHTML +=
      " Das ist dynamisch generierter Text.";
}
```

http://molily.de/js/event-handling-fortgeschritten.html

Event-Handler entfernen: removeEventListener

Um die mit addEventListener registrierten Handler wieder zu entfernen, gibt es die Methode removeEventListener.

Die Methode erwartet dieselben Parameter, die addEventListener beim Registrieren bekommen hat.

```
function beenden () {
   pElement.removeEventListener("click", klickverarbeitung,
false);
}
```

http://molily.de/js/event-handling-fortgeschritten.html

Eventlistener in IE < 9

```
window.attachEvent("onload", start);


function start () {
   var pElement = document.getElementById("interaktiv");
   pElement.attachEvent("onclick", klickverarbeitung);
}

function klickverarbeitung () {
   document.getElementById("interaktiv").innerHTML +=
      " Das ist dynamisch generierter Text.";
}
```

http://molily.de/js/event-handling-fortgeschritten.html

Events in IE < 9 entfernen

```
function beenden () {
   pElement.detachEvent("onclick",
klickverarbeitung);
}
```

http://molily.de/js/event-handling-fortgeschritten.html

Eine browserübergreifende Eventfunktion

```
function addEvent (obj, type, fn) {
   if (obj.addEventListener) {
      obj.addEventListener(type, fn, false);
   } else if (obj.attachEvent) {
      obj.attachEvent('on' + type, function () {
         return fn.call(obj, window.event);
      });
   }
}
```

Eine bessere unter:
http://therealcrisp.xs4all.nl/upload/addEvent_dean.html

http://molily.de/js/event-handling-fortgeschritten.html

Das Eventobjekt mit preventDefault

```
function zeigeVollbild (eventObjekt) {
   // Browserübergreifender Zugriff auf das Event-Objekt
   if (!eventObjekt) eventObjekt = window.event;

   // Existiert die Methode preventDefault? Dann rufe sie auf.
   if (eventObjekt.preventDefault) {
      // W3C-DOM-Standard
      eventObjekt.preventDefault();
   } else {
      // Andernfalls setze returnValue
      // Microsoft-Alternative für Internet Explorer < 9
      eventObjekt.returnValue = false;
   }
};
```

http://molily.de/js/event-handling-objekt.html

THE EVENT OBJECT

```
event
 target
 delegateTarget

 which
 type
 meta-/shift-/alt-/ctrlKey

 pageX
 pageY

 preventDefault()
 stopPropagation()


 . . .
```

AN ONCLICK EVENT HANDLER

```
fn = {
  onclick : function (event) {
    event.preventDefault();
    event.stopPropagation();
    url = $(event.target).attr('href');
    fn.loadContentOf(url)
  },
  loadContentOf : function (url) {...}
}
```

```
USE event.target, NOT this!
```