JavaScript

Dr. Günter Kolousek

2012-08-30

Übersicht

Merkmale

- Objekt-basierte, prototypen-basierte Programmiersprache
- von C, Java, Python, Perl, Scheme, Self beeinflusst
  - kein block-level scoping (außer mit `let`), sondern function-level
- plattformunabhängig (OS, Rechnerarchitektur)
- Fokus auf Browser (Server: $\rightsquigarrow$ node.js, rhino)
- dynamisch und schwach getypt
  - (großzügige) Typumwandlungen!!!
- reguläre Ausdrücke in Sprache eingebaut
- Garbage collector (GC)
- Security $\rightsquigarrow$ sandbox

Geschichte

Versionsnummern gemäß Mozilla Browser:

- 1996: JS 1.0, Netscape, jetzt geschützte Marke von Oracle
- 1998: JS 1.3
  JS von Netscape bei ECMA (European Computer Manufacturer's Asscoiation) $\rightsquigarrow$ ECMAScript 1 und ECMAScript 2
- 2000: JS 1.5 (ECMAScript 3)
- 2005: JS 1.6 (= JS 1.5 + Array Extras...)
- 2006: JS 1.7 (+Pythonic Generators, +Iterators, +Array Comprehensions)
- 2008: JS 1.8 (+Generator Expressions, +Expression Closures)
- 2010: JS 1.8.5 (+ECMAScript 5.1 Kompatibiltät)

Alle außer FF: ECMAScript 3!

Allgemeines

- Unicode-Zeichen: UTF-16
- Identifier
  - aus Buchstaben, Ziffern, `$`, _
  - keine Ziffer an erster Stelle
- kein Typ bei Deklaration von Variablen
  - $\leadsto$ dynamisch getypt

Variable

- Zugriff: zuerst lokal, dann umschließender Scope,...
  - $\rightsquigarrow$ ReferenceError
- Zuweisung: zuerst lokal, dann umschließender Scope,...
  - global ... Eigenschaft des globalen Objektes
- `var`
  - kann *nicht* mit `delete` gelöscht werden
  - ohne `var` $\rightsquigarrow$ global
- function-level scope (siehe myscope2)

```
1  var scope = "global";  // globale Variable
2  myscope2 = "global";  // globale Var, ohne var
3  function test() {
4    console.log(myscope2);  // -> undefined
5    scope = "lokal";  // globale Var!
6    myscope = "global";  // neue globale Var (kein var!)
7    var myscope2 = "lokal";  // lokale Var
8    console.log(myscope2);   // -> lokal
9  }
```

$\rightsquigarrow$ **immer** deklarieren!

Semikolon optional

- wenn separate Zeilen und
- folgende Zeile nicht als Fortsetzung interpretiert werden kann
  - außer
    - `break`, `return`, `continue`
    - `++` und `--`
  - Beispiel

```
1  x = 1
2  y = 2
3
4  // Aber:
5  var y = x + f
6  (a + b).toString()
7
8  // wird interpretiert wie
9  var y = x + f(a+b).toString();
```

- besser: Semikolon verwenden!!!

Reservierte Wörter:

- Schlüsselwörter
  - `if`, `else`, `switch`, `case`, `default`
  - `while`, `for`, `do`, `break`, `continue`
  - `try`, `catch`, `finally`, `throw`, `default`
  - `this`, `instanceof`, `typeof`, `delete`
  - `function`, `return`, `var`, `null`
  - `void`, `debugger`, `with`
- reserviert, aber derzeitig nicht genutzt
  - `class`, `extends`, `super`, `const`, `enum`, `export`, `import`
- im ‘strict mode’:
  - reserviert: `implements`, `let`, `private`, `public`, `yield`, `interface`, `package`, `protected`, `static`
  - nicht als Var, Fkt, Par: `arguments`, `eval`
- ECMAScript 3 reserviert alle Schlüsselwörter von Java

Vordefiniert...

- `Array`, `Date`, `Error`, `RegExp`
- `Math`, `Number`, `Boolean`, `String`, `Object`, `Function`, `JSON`
- `Infinity`, `isFinite`, `NaN`, `isNaN`, `parseInt`, `parseFloat`
- `eval`, `encodeURI`, `encodeURIComponent`, `decodeURI`, `decodeURIComponent`
- `arguments`, `undefined`
- `EvalError`, `RangeError`, `ReferenceError`, `SyntaxError`, `TypeError`, `URIError`

Einteilung

- elementare Typen (immutable!)
  - Zahlen
  - Strings
  - boolesche Wahrheitswerte
    - Typumwandlung in `false`: `undefined`, `null`, 0, `NaN`, `""`
  - Werte: `null` und `undefined`
    - `null`: `typeof null` ⇝ `"object"`
    - `undefined`: Variable nicht initialisiert oder Objekteigenschaft nicht definiert.
    - `null == undefined`, aber `null !== undefined`
- Objekttypen
  - Sammlung von Eigenschaften
  - ⇝ globales Objekt
    - Eigenschaften sind global definierte Symbole! (⇝ `window`)
    - z.B. `undefined`, `isNaN()`, `Date()`, `Math()`
  - Arrays sind Objekte
  - Funktionen sind Objekte (first class objects)
  - Funktion mit `new` aufgerufen ⇝ Konstruktor

Zahlen und Mathematisches

- keine Unterscheidung in ganze Zalen und Gleitkommazahlen (64-Bit)
- `Math` Funktionen und Konstanten
  - z.B.: `Math.pow(2,53)` oder `Math.PI`
- `Number`
  - `Infinity`, `Number.POSITIVE_INFINITY`, `Number.MAX_VALUE + 1`, `1/0`
  - `0/0`, `NaN`, `Number.NaN`
  - `.1+.1+.1+.1+.1+.1+.1+.1+.1+.1 == 1` ⇝ `false`

Strings

- einfache oder doppelte Anführungszeichen
- Escape-Zeichen ganz “normal”, z.B. `\n`

```
1  var s = "abcdef";
2  s.charAt(0);
3  s[0];  // ECMAScript 5
4  s.slice(0, -1);  // -> abcde
5  s.slice(-3);  // -> def
6  // aber nicht: s.charAt(-1)!
7  s.indexOf("d");  // -> 3
8  "abc".charAt(0);  // "abc" ist kein Objekt!!
```

Wrapper-Objekte

- `"abc".charAt(0)` wie `new String("abc").charAt(0)`
  - Implementierung **muss kein** richtiges Objekt anlegen
- `15.toString(2) == "1111"`
  - Wrapper-Klasse `Number`
- `true.toString()`
  - Wrapper-Klasse `Boolean`
- keine Wrapper-Klassen für `null` und `undefined`

Typumwandlungen

- Implizit
  - beliebiger Typ in booleschen Wert (siehe boolesche Werte)
  - beliebiger Typ in String
    - `10 + "Objekte"` ⇝ `"10 Objekte"`
    - `NaN + " Objekte"` ⇝ `"NaN Objekte"`
  - beliebiger Typ in Zahl
    - oder `NaN`
    - `"7" * "4"` ⇝ `28`
    - `"2" * "3 Orangen"` ⇝ `NaN`
    - `1 - "x";` ⇝ `NaN`
    - `"0" == 0` bzw. `"0" == false` ⇝ `true`
- Explizit
  - Achtung: ohne `new`
  - `Boolean([])` ⇝ `false`
  - `Number("3")` ⇝ `3`
    - `Number("3 Objekte")` ⇝ `NaN`
    - `parseInt("3 Objekte")` ⇝ `3`
  - `String(false)` ⇝ `"false"`

Objekt-basierte Sprache

- keine Klassenanweisung, keine Vererbung
- dafür: Prototypen
- Konstruktor “definiert” Klasse
- Eingebaute Klassen
  - `Array`
  - `Date`
  - `RegExp`
    - Literaltyp, z.B. `/[1-9][0-9]*/`
  - `Error`

Objekte

```
1  var p1 = {
2     "x" : 1,
3     y : 1,
4  }
5
6  p.x = 0;
7  p["y"] = 0;
8
9  var points = [p1, {x:1, y:1.0}];
10
11 points.dist = function() {
12      var p1 = this[0];
13      var p2 = this[1];
14      var a = p2.x - p1.x;
15      var b = p2.y - p1.y;
16      return Math.sqrt(a * a + b * b);
17 }
18
19 points.dist();  // -> 1.414...
```

Objekte und Methoden - 1

```
1  var person = {
2    firstName : "Max",
3    lastName : "Mustermann",
4    greet : function() {
5        console.log("Hi, " + this.firstName);
6    }
7  };
8
9  var sayHi = person.greet; // Methode in Variable ablegen
10
11 sayHi(); // -> Hi, undefined
12 // this referenziert jetzt globales Objekt!!!
```

Objekte und Methoden - 2

```
1  var person = {
2    firstName : "Max",
3    lastName : "Mustermann",
4    greet : function(greeting, punctation) {
5        console.log(greeting + " " + this.firstName + punctation
6    }
7  };
8
9  var sayHi = person.greet; // Methode in Variable ablegen
10
11 sayHi.call(person, "Hi", "!")
12 sayHi.apply(person, ["Hallo", "!!"])
```

Getter und Setter

```
1  var p = {
2      x: 1,
3      y: 1,
4      get r() { return Math.sqrt(this.x * this.x +
5                                  this.y * this.y)},
6      set r(val) {
7        var oldval = Math.sqrt(this.x * this.x +
8                               this.y * this.y);
9        var ratio = val / oldval;
10       this.x *= ratio;
11       this.y *= ratio;
12     }
13 };
14
15 alert(p.r);  // -> 1.414...
16 p.r = 2;
17 alert(p.x + " " + p.y);  // -> 1.414... 1.414...
```

Arrays - 1

```
1  var primes = [2, 3, 5, 7, 11];
2  var a = new Array(10);  // a.length == 10
3  a[0];  // -> undefined
4  var b = new Array(2, 3, 5, 7, 11, "Test", false);
5
6  var c = ["a"];
7  c[0];  // "a"
8  c[1] = 3.14;
9  c.length;  // -> 2
10 c["a"] = true;
11 c.a;  // -> true
12 c.length;  // -> 2
```

Arrays - 2

```
1  var d = [1, 2, 3];
2  delete d[1];
3  1 in d;  // -> false: prueft ob Index in Array
4  console.log(d.toString());  // -> 1,,3
5  console.log(d);  // -> [1, 2: 3]    (Chrome)
6  d.length;  // -> 3!
7  for (var i in d) console.log(d[i]);  // -> 1 3 (2 Zeilen)
8  // -> 1 undefined 3
9  for (var i=0; i < d.length; i++) console.log(d[i]);
10 // 1 undefined 3
11
12 o = {};
13 o[1] = "x";  // 1 wird in einen String konvertiert
14 o[1];  // -> "x"
15 o["1"];  // -> "x"
16 o.length;  // -> undefined
```

Ausdrücke

- ziemlich wie Java
- Operator `typeof` ... Typ ermitteln
- Operator `instanceof` ... Instanz dieser Klasse?
  - `[1, 2, 3] instanceof Array` ⇝ `true`
- Operator `delete` ... Eigenschaft löschen
- Operator `in` ... Eigenschaft in Objekt?
- `"x" in {x:0, y:1}` ⇝ `true`
- `"toString" in {x:0, y:1}` ⇝ `true`
- `"5" in [4,5,6]` ⇝ `true`
- Operator `==` ... Gleichheit
- Operator `===` ... strenge Gleichheit
  - keine Typwandlungen

`typeof` - Operator

- `typeof undefined` ⇝ `"undefined"`
- `typeof null` ⇝ `"object"`
- `typeof true` ⇝ `"boolean"`
- `typeof 3.14` ⇝ `"number"`
- `typeof "abc"` ⇝ `"string"`
- `typeof Math.sqrt` ⇝ `"function"`
- `typeof {x:1,y:2}` ⇝ `"object"`
- `typeof [1, 2, 3]` ⇝ `"object"`

`delete` - Operator

```
1  var o = {x:1, y:2};
2  delete o.x;  // -> true
3  typeof o.x;  // -> undefined
4  delete o.x;  // -> true
5  delete o["y"]  // -> true
6  // deklarierte Var koennen nicht geloescht werden
7  delete o;  // -> false
8
9  var a = [1,2,3];
10 delete a[2];
11 a.length;  // -> 3
```

Funktionen - 1

```
1  function fact(x) {
2      if (x <= 1) return 1;
3      return x * fact(x - 1);
4  }
5
6  var square = function(x) { return x * x; }
7
8  var f = function fact2(x) {
9            if (x <= 1) return 1;
10             return x * fact2(x - 1);
11         };
```

Funktionen - 2

```
1  function sum(x, y, /* optional */ z) {
2      if (z === undefined) z = 0;
3      return x + y + z;
4  }
5
6  sum(1, 2);   // -> 3
7  sum(1, 2, 3);   // -> 6
8
9  function max(/* ... */) {
10     var max = Number.NEGATIVE_INFINITY;
11     for (var i=0; i < arguments.length; i++) {
12         if (arguments[i] > max) max = arguments[i];
13     }
14     return max;
15 }
16
17 max(1, -3, 42, 9);   // -> 42
```

Funktionen - 3

```
1  function sum2(x, y) {
2      if (arguments.length < 2)
3          throw new Error("zu wenig Parameter");
4
5      var res = x + y;
6
7      if (arguments.length > 2)
8          for (var i = 2; i < arguments.length; i++)
9              res += arguments[i];
10
11     return res;
12 }
13
14 alert(sum2(1));  // -> Error: zu wenig Parameter
15 alert(sum2(1,2));  // -> 3
16 alert(sum2(1,2,3,4,5));  // -> 15
```

“Klassen” - 1

```
1  function Point(x, y) {
2      this.x = x;
3      this.y = y;
4  }
5  // damit wird Point zu einem "Konstruktor"
6  var p1 = new Point(1, 1);
7  Point.prototype.dist0 = function() {
8      return Math.sqrt(this.x * this.x + this.y * this.y);
9  };  // Point.prototype ist Objekt "Point"
10
11 p1.dist0();  // -> 1.414
12 p1.constructor === Point;  // -> true
13 p1.constructor.prototype === Point.prototype; // -> true
14 Point.prototype.isPrototypeOf(p1); // -> true
15 Object.getPrototypeOf(p1) === Point.prototype;  // -> true
16 p1.__proto__ === Point.prototype; // -> true (nicht Standard!)
```

Jede Funktion hat eigenes “Prototypen”-Objekt!

“Klassen” - 2

Prinzip der “Vererbung”: `o.x`

1. Hat `o` eine Eigenschaft `x`? ja,...
2. nein: Hat Prototyp eine Eigenschaft `x`? ja,...
3. nein: Hat Prototyp des Prototyps eine Eigenschaft `x`?...

```
1  var o1 = {x: 1, y: 2};
2  var o2 = Object.create(o1);
3  var o3 = Object.create(o2);
4  o3.x;  // -> 1
5  o3.hasOwnProperty("x");  // -> false
6  o2.hasOwnProperty("x");  // -> false
7  o1.hasOwnProperty("x");  // -> true
```

“Klassen” - 3

- Instanz erbt von Prototyp
  - Prototyp nicht “direkt” als Eigenschaft zugreifbar.
- Prototyp kennt Konstruktor
  - Instanz erbt von Prototyp die Konstruktor-Eigenschaft (`o.constructor`)
- Konstruktor kennt Prototyp-Eigenschaft (`o.constructor.prototype`)

```
1  var o1 = {x:1, y:2};
2  var o2 = Object.create(o1);  // o1 ist Prototyp von o2
3  o1.isPrototypeOf(o2);  // -> true
4  o2.constructor === Object;  // -> true
5  o2.constructor.prototype === o1;  // -> false
6  Object.prototype.isPrototypeOf(o2);  // -> true (transitiv!)
7  // Object.prototype "ist" das Objekt "Object"
8  // nicht die Funktion "Object"
9  o2.constructor.prototype === p;  // -> false!
10 o2.constructor.prototype === Object.prototype;  // -> true
```

“Klassen” - 4

```
1  var p1 = new Point(1,1);
2  var p2 = Object.create(p1);
3  p1.isPrototypeOf(p2);  // -> true
4  p2.constructor === Point;  // -> true
```

Unter Chrome:

```
1  Object.prototype  // -> Objekt "Object"
2  Object.prototype.__proto__  // -> null
3  Object.prototype.constructor  // -> Funktion "Object"
4  Object.__proto__  // -> Funktion "Empty"
5  Object.__proto__.__proto__  // -> Objekt "Object"
6  Object.__proto__.prototype  // -> undefined
```

Anweisungen

- `for`, `while`, `do`, `if`, `switch` wie in Java 7
- Alternative Form von `for`

```
1  var o = {x:0, y:1};
2  for (var key in o)
3      console.log(o[key]);  // o.key geht natuerlich nicht!
4  // aber:
5  for (i in ["a", "b", "c"])
6      console.log(i);  // 0, 1, 2
```

JS im Browser

- JS im Browser: single-threaded
  - $\rightsquigarrow$ HTML5: WebWorker
- Einbettung
  - externe Datei mittels `<script src="...">...</script>` (präferiert)
  - innerhalb von `script`
    - XHTML $\rightsquigarrow$ CDATA! (`<![CDATA[ ... ]]>`)
  - Event-Handler in HTML

    ```
    <input type="checkbox" name="opt"\
    value="pepper" onchange="...">
    ```
  - JavaScript-URLs: `<a href="javascript:alert(\` `new Date().toLocaleTimeString();">`

JS im Browser - Ausführung

1. Request los und `document` Objekt beim Response anlegen (`document.readyState == "loading"`).
2. Parsen $\rightsquigarrow$ HTML Elemente erzeugen und hinzufügen (siehe DOM).
3. `<script>` Element (kein `async` oder `defer`) $\rightsquigarrow$ Skript laden und synchron ausführen.
   - `async` ... parallel laden, asynchron ausführen.
   - `defer` ... laden, warten bis Dokument geparst.
4. Dokument vollständig geparst, dann `document.readyState == "interactive"`
5. `defer` Skripte starten (Reihenfolge wird beachtet).
6. Dananach (alle `defer` fertig): Verarbeitung der Events (`async` Skripte müssen noch nicht fertig sein).
7. Alle Ressourcen (z.B. Bilder) geladen, dann `document.readyState == "complete"` $\rightsquigarrow$ load-Event!

`defer` und `async` im Vergleich

- nicht in Opera!
- weder `defer` noch `async`
  - werden sofort geladen und gestartet
- `defer`
  - werden gestartet, wenn DOM fertig
  - Reihenfolge der Ausführung wird eingehalten.
- `async`
  - Start prinzipiell sofort
  - Reihenfolge der Ausführung wird eingehalten.
- beide, dann wie `async`

Window-Objekt

- Window-Objekt ist im Browser **globales** JavaScript-Objekt!
- jedes Fenster hat eigenes `window` Objekt
- auch `iframe` hat eigenes `window` Objekt
- jeder Tab hat eigenes `window` Objekt
- Ein Window kann
  - ein anderes Window öffnen und schließen
  - auf Inhalt des anderen im Rahmen der CORS zugreifen

Window-Objekt - Timeout

- `setTimeout(f, t)` ... Funktion `f` nach `t` ms
  - Return-Wert kann an `clearTimeout()` übergeben werden
- `setInterval(f, t)` ... `f` alle `t` ms aufrufen
  - Return-Wert kann an `clearInterval()` übergeben werden
- Beispiel

```
1  var cnt = 0;
2  function incr() {
3      cnt++;
4      if (cnt < 10) {
5          console.log(cnt);
6          setTimeout(incr, 1000);
7      }
8  }
9  setTimeout(incr, 1000);
```

Window - open

- Öffnen kann prinzipiell vom Browser blockiert werden
- Name ist wichtig beim `target` Attribut von `<a>` bzw. `<form>`
  - "_blank" (default) . . . neues, unbenanntes Fenster

```
1  var w = open("http://www.orf.at",
2            "news",  // name, optioal
3            // durch , getrennt; nicht standardisiert;
4            // kann ignoriert werden
5            "width=400,height=350",
6            false);  // neuer Eintrag in History (T: replace)
7  setTimeout(function() { w.close(); }, 10000);
8  w.opener != null; // enthaelt window-Objekt des Erzeugers
9  w = open("http://derstandard.at",
10           "news");  // kein neues Fenster!
11 // Fenster w wird nach 10s geschlossen
```

Window-Objekt - `location`

- `location` - Objekt
  - `window.location == document.location`
- Eigenschaften
  - `protocol`, `host`, `port`, `pathname`, `search`, `hash`
- Methoden
  - `reload()`
- Zuweisung
  - `location = "http://www.orf.at";`
  - auch relative URL, wie z.B.
    `location = "page2.html";`
  - auch, z.B. nur Fragment, z.B. `location = "#top";`
    - wenn keine id mit “top”, dann zum Anfang

Window - Methoden

- `alert()` ... Meldung anzeigen
- `confirm()` ... Meldung anzeigen, OK und Cancel, booleschen Wert zurückliefern
- `prompt()` ... Meldung anzeigen, Textfeld, String zurückliefern
- `confirm` und `prompt` blockieren!
  - `alert` i.d.Regel auch, aber nicht zwingendermaßen
- maßvoll einsetzen!
- hauptsächlich `alert` zum Debuggen

Document Object Model (DOM)

- API zum Zugriff auf alle Elemente
- Zugriff über `window.document`
- Baumstruktur des HTML Dokumentes mit allen HTML Elementen
- root-Element ist das `html` Element
- Klassenhierarchie

```
Node
  Document
    HTMLDocument
  CharacterData
    Text
    Comment
  Attr
  Element
    HTMLElement
      HTMLHeadElement
      ...
```

Eigenschaften von `Node`

- `parentNode`
- `childNodes` ... Array-ähnliches Objekt
- `firstChild`, `lastChild`
- `nextSibling`, `previousSibling`
- `nodeType` (Element=1, Text=3, Document=9,...)
- `nodeValue` ... Inhalt eines Text- oder Comment-Knotens
- `nodeName` ... Tagname

Elemente auswählen

- `id`
  - `sect1 = document.getElementById("sect1");`
- `name`
  - `rad = document.getElementsByName("color");`
  - Liefert eine `NodeList` zurück
  - IE auch Elemente mit gleicher `id`!
- Tagname
  - `s = document.getElementsByTagName("span");`
- CSS Klasse
  - `w = document.getElementsByClassName("wrn");`
- CSS Selectors API
  - `querySelector` und `querySelectorAll`

$\rightsquigarrow$ jQuery!

HTMLElement-Objekt

- Ein HTML (XML) Element besteht aus:
  - Start-Tag mit Attributen
  - Inhalt
  - Ende-Tag
- Inhalt
  - `<p>Ein <em>wichtiger</em> Absatz</p>`
  - `elem.innerHTML` ⇝ `Ein <em>wichtiger</em> Absatz`
  - `elem.textContent` ⇝ `Ein wichtiger Absatz`
  - iterieren über Kind-Knoten

HTMLElement-Objekt - Attribute

- Attribute können gelesen und geschrieben werden:
  - reservierte Wörter, dann “html” davorsetzen
    - `for` Attribut des `label` $\rightsquigarrow$ `htmlFor`
  - z.B.

```
var img = document.getElementById("myimage")
img.src = "img2.png";
```

  - über `attributes`
    - `img.attributes[0]`
    - `img.attributes.src`
    - `img.attributes["src"]`

HTMLElement-Objekt - CSS

- über Attribut `style`: Objekt vom Typ `CSSStyleDeclaration`
  - Alle Werte als Strings!
    - z.B.: `e.style.left = "300px";`
  - CSS Eigenschaften mit Bindestrichen, dann camelCase!
    - z.B.: `e.style.fontSize = "24pt";`
  - CSS Eigenschaft als Schlüsselwort in JavaScript, dann `css`!
    - z.B.: `e.style.cssFloat = "left";`
- Klassen
  - über `className` ... setzt `class` Attribut (besser wäre `classNames`...)
    - z.B.: `e.className = "warning";`
  - über `classList` ... array-ähnlich (dzt. nicht in IE)

Eigenschaften des Document - Elementes

- `cookie`
  - setzen mittels Zuweisung
  - auslesen mittels Zugriff und splitten an “;”
  - löschen mittels Wert auf Leerstring und Zeit abgelaufen
- `domain` ... siehe same-origin-policy
- `location`
- `referrer` ... Achtung “rr” (wie HTTP-Header “Referer”)
  letzte Seite, die zu diesem Dokument geführt hat

Cookies

```
1  function setCookie(name, value, days) {
2    if (days) {  var date = new Date();
3      date.setTime(date.getTime() + (days*24*60*60*1000));
4      var expires = "; expires=" + date.toGMTString();
5    } else var expires = "";
6    document.cookie = name+"="+value+expires+"; path=/";
7  }
8
9  function getCookie(name) {
10   var nameEQ = name + "=";
11   var ca = document.cookie.split(';');
12   for(var i=0;i < ca.length;i++) {
13       var c = ca[i];
14       while (c.charAt(0)==' ') c = c.substring(1,c.length);
15       if (c.indexOf(nameEQ) == 0)
16         return c.substring(nameEQ.length,c.length);
17   } return null; }
18
19 function deleteCookie(name) {
20     setCookie(name,"",-1); }
```

Event-Typen

- “alte” Events: load, click, mouseover,. . .
- “neue” Events:
  - DOM Level 3 Events: “alte” + neue, wie focusin, focusout, mousenter, mouseleave, textinput
  - HTML5 Events, z.B. für Video und Audio,. . .
  - Touch-Events: touchstart, touchmove, touchend, touchcancel

Event-Handler

- Definition
  - via HTML, z.B.
    `window.onload = function() { ... };`
    - lediglich ein Event-Handler!
    - Rückgabewert = `false`: keine Standardaktion
  - `addEventListener`
    - `attachEvent` bis IE8!
    - `e.preventDefault()` oder `e.returnValue = false` (bis IE8): keine Standardaktion
    - `e.stopPropagation()` oder `e.cancelBubble = true` (bis IE8): Abbruch der Event-Propagation.
- Aufruf
  - meist mit einem Event-Objekt (außer IE: globales `event`)
    - außerdem: Unterschiede je Implementierung!

Überblick Event-Arten - 1

- Tastatur - Events
  - steigen auf
  - keydown, keyup: low-level
  - keypress: ausgebbares Zeichen
    - Firefox: auch nicht ausgebbare Zeichen
    - Eigenschaft `keyCode` (Firefox: `charCode`)
  - input: Eingabe schon passiert, keine Information was geändert wurde
- Mouse - Events
  - steigen auf (außer mouseenter und mouseleave)
  - click, dblclick
  - mousedown, mouseup
  - mousemove, mouseover, mouseout
  - mouseenter, mouseleave

Überblick Event-Arten - 2

- Formular - Events
  - steigen auf (außer blur und focus)
  - click . . . Buttons
  - change . . . Textfelder, Checkbox, Radiobox
    - Checkbox, Radiobox, Select . . . sofortiges Triggern des Events
    - “Text” . . . triggert erst beim Fokuswechsel
  - focus, blur . . . Fokus erhalten bzw. verlieren
  - focusin, focusout
  - submit, reset
- Window
  - load, resize, scroll

Ajax - Einführung

- HTTP mittels JavaScript
  - `location` eines Window-Objektes setzen
  - `submit()` eines Formular-Objektes aufrufen
  - $\rightsquigarrow$ Seite wird neu geladen (synchron!)
- Asynchronous JavaScript and XML
  - lange Zeit ein “Buzzword”
  - geskriptete HTTP-Requests
- Alternativen
  - `src` von `img` auf Skript setzen und Server liefert leeres Bild (1x1 transparent) $\rightsquigarrow$ eine Richtung!
  - `src` von `iframe` ... $\rightsquigarrow$ lokales JavaScript kann Inhalt abfragen.
  - `src` von `script` ... $\rightsquigarrow$ oft JSON kodiert

JSON

- Javascript Object Notation
- Datenformat, text-basiert, gültiges JavaScript (per `eval()`)
- Datentypen
  - `null`, `true`, `false`
  - Zahl, String (in doppelten Anführungszeichen)
  - Array in eckigen Klammern
  - Objekt mit Key (String) und Wert (JSON Wert)
- Beispiel

```
1  { "id": 4712,
2    "firstname": "Maxi",
3    "phone": {
4      "home": "02622/2781",
5      "office": ["02622/27871", "02622/27178"]
6    },
7    "salary": 1500.42,
8    "locked": false,
9    "departement": null
10 }
```

AJAX - Charakterisierung

- Begriff
  - asynchronous
  - JavaScript
  - XML
    - prinzipiell beliebiges Datenformat
    - heute meist (da einfacher): JSON
- Implementierung
  - Klasse `XMLHttpRequest` (XHR)

XHR - 1

```
1  var request = new XMLHttpRequest();
2
3  request.open("POST", "/send");
4
5  // fuer POST notwendig
6  request.setRequestHeader("Content-type",
7                           "text/plain;charset=UTF-8");
8
9  // bei GET: null verwenden (kein Datenblock)
10 request.send(msg);
11 // kein warten: asynchron!
12 // synchron, wenn 3. Parameter von open: false, aber...
```

XHR - 2

- die folgenden Header werden selbstständig hinzugefügt:
  - Content-Length, Date, Referer, User-Agent
- XHR - Objekt hat Eigenschaft `readyState`
  - 0 . . . UNSET
  - 1 . . . OPENED
  - 2 . . . HEADERS_RECEIVED
  - 3 . . . LOADING
  - 4 . . . DONE
- `readystatechange` Event

XHR - 3

```
1  var request = new XMLHttpRequest();
2  request.open("GET", url);
3  request.onreadystatechange = function() {
4    if (request.readystatechange === 4 && request.state === 200)
5      var type = request.getResponseHeader("Content-Type");
6      if (type.match(/^text/))
7        callback(request.responseText);
8      else if (type === "application/json")
9        callback(JSON.parse(request.responseText));
10   }
11 };
12 request.send(null);
```

Sicherheit

- JavaScript unterstützt nicht, z.B.:
  - beliebige Daten schreiben oder löschen bzw. Verzeichnisse zu lesen (aber HTML5: File API)
  - beliebige Netzwerkverbindungen aufbauen und beliebige Daten versenden
- JavaScript schränkt ein, z.B.:
  - kann Browser-Fenster öffnen
  - kann Browser-Fenster schließen, wenn selbst geöffnet
  - kann kein `value` eines HTML FileUpload - Elementes setzen.
  - kann den Inhalt von anderen Dokumenten nicht lesen, wenn von anderem Server $\leadsto$ Same-Origin-Policy

Same-Origin-Policy ("Gleiche-Herkunfts-Richtlinie")

- z.B. bei Öffnen eines Window oder bei `<iframe>`
- gleiche Herkunft des Dokumentes $\rightsquigarrow$ Zugriff auf anderes Dokument erlaubt
- gleiche Herkunft, wenn
  - gleiches Protokoll
  - gleicher Host
  - gleicher Port
- Herkunft des Skripts **unerheblich**!
- gilt auch für XMLHttpRequests (aka Ajax)
  - Requests nur an Website mit gleicher Herkunft!

Möglichkeiten der Lockerung

- Motivation: “developers.example.com” möchte auf “orders.example.com” zugreifen.
- Lösung 1: `domain` Eigenschaft des `document` Objektes auf “example.com” setzen (auf beiden `document` Objekten!)
  - default: Host von dem es gelesen wurde
  - nur gültiges Domänensuffix
    - z.B. “example.com”
    - nicht: “com” (keine TLD, d.h. mind. ein Punkt)
    - nicht: “office.example.com”
  - Achtung: “evils.example.com” könnte ebenfalls auf “example.com” setzen!
- Lösung 2: Cross-Origin Resource Sharing
- Lösung 3: Cross-Document Messaging
  - mittels `postMessage()` Objekt versenden
  - mittels `onmessage` Event empfangen
  - sonst kein Zugriff!

Cross-Origin Resource Sharing (CORS)

- Working Draft: http://www.w3.org/TR/cors/
- “developers.example.com” an “orders.example.com”:
  - C→ S: `Origin: http://developers.example.com`
  - S→ C: `Access-Control-Allow-Origin: http://developers.example.com`
  - C: erlaubt daraufhin den Zugriff
- beliebiger Zugriff: `Access-Control-Allow-Origin: *`
- Aktivieren am Server: http://enable-cors.org/
- Weiters möglich (bei Verwendung zusätzlicher Header!)
  - beliebige Methoden: POST, PUT, DELETE,...
  - Cookies
  - Alle Header beginnen mit “Access-Control-” außer “Origin:”
  - Weitere Infos:
    - http://www.html5rocks.com/en/tutorials/cors/
    - http://dev.opera.com/articles/view/dom-access-control-using-cross-origin-resource-sharing/

JSONP

- “JSON with Padding”
- Hilfslösung für Zugriff auf Daten anderer Domänen
- Prinzip
  - Einbetten der Antwort in `<script>` Element (da von Same-Origin Policy ausgenommen!!)
  - Idee: Antwort als JSON Dokument
    - aber: Block `{ ... }` kein gültiges JavaScript
  - JSONP: auffüllen (padding) mit z.B.
    - Funktionsaufruf: `parseData({ ... })`
    - Zuweisung: `data = {...}` (u.U. `JSON.parse()` verwenden)
- Vorteil: Browser benötigt keine spezielle Implementierung
- Nachteil: Security (CSRF möglich), nur GET, keine Cookies