2.– 5. September 2013
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Script Factory Method Builder

Fortgeschrittene Techniken mit JavaScript

Frank Goraus

MATHEMA Software GmbH

Einbinung von JS

- direkt am Element
  - *<input type="text" ... onChange="alert(this.value)" />*
- besser: Script-Block in der Seite
  - *<script type="text/javascript">*
    alert(„Hallo Campus");
    </script>
- optimal: externe JS-Resource
  - bietet Chance auf Wiederverwendbarkeit statt Copy-and-Paste
  - *<script src="script.js">* **</script> (/> ist invalide)**
- Beispiele: → demo2.html

Barrierefreiheit

- JS-freie Bedienung muss gewährleistet sein
- JS nur für „Convenience“ und „Eyecandy“ verwenden

- Anti-Beispiele:
  - JS-Dialoge, welche zum Fortschritt in der Anwendung benötigt werden
  - Forms, die nur per JS abgeschickt werden (und sonst keinerlei Submit-Button besitzen)
  - Bilder, Divs, ... mit elementaren JS-Funktionen darauf

Geltungsbereich von Variablen

- innerhalb der umschließenden Funktion (function scope)
  - *var func = function() {*
    var a = b = 1;
    if (a == b) { var i = 42; }
    alert(i);
    }
    func(); //42
- global an window (global scope)
  - *var a = 42;*
    function echoA() { alert(a); }
    echoA(); //42

Closures

- Zugriff auf Kontext (= Variablen im function scope) einer nicht mehr relevanten Funktion

```
var func = function(a, b) {
  var c = a+b;
  return function(){ console.log(c); }
};
var closureFunction = func(1,2);
closureFunction(); //3
```

private Eigenschaften?

- kein explizites Sprachfeature
- per Closure „nachrüstbar“:

```
var privateTest = function() {
  var antwort = 42,
      dasLeben = function() { return antwort; };
  return { dasUniversum : function() { alert(antwort); },
           derGanzeRest : function() { return dasLeben(); } };
}
privateTest().dasLeben(); //TypeError: … has no method ...
privateTest().dasUniversum(); //42
var antwortAufDieFrage = privateTest().derGanzeRest();
```

Prototype (1/2)

- keine Klassen
- Klon eines bestehenden Objektes (= Prototyp)
  - bestehende Referenz auf den Prototypen
    → Änderungen möglich

```
function Proto() {
   this.func = function() { console.log(42); }
};
var obj = new Proto();
Proto.prototype.newFunc = function() { console.log(4711); };
obj.func(); // 42
obj.newFunc(); //4711
```

Prototype (2/2)

- Tests

  obj instanceof Proto //true

  obj.constructor == Proto //true

- Änderung des Prototypen

  → Referenz bleibt bestehen

```
function Proto2() {
   this.otherFunc = function() { console.log(0815) };
};
var obj2 = new Proto2();  //Prototype = Proto2
Proto2.prototype = Proto.prototype; //alternativ new Proto();
var obj3 = new Proto2(); //Prototype = Proto
//Prototype von obj2 = immernoch Proto2 !
```

Prototype-Vererbung (1/2)

- alle Eigenschaften und Methoden des als Prototyp fungierenden Objektes werden geerbt

```
var papa = {  vorname: 'Horst Peter',
              name: 'Testerson',
              vorstellen: function() {
                    alert('Mein Name ist '
                    +this.vorname+' '+this.name);
              }
    };
var Sohn = function () { this.vorname = 'Hans' }; //eigener Kons.
Sohn.prototype = papa; //Konstruktor des Papas (~wie super)
```

Prototype-Vererbung (2/2)

```
var hans = new Sohn();
papa.vorstellen(); // Mein Name ist Horst Peter Testerson
hans.vorstellen(); // Mein Name ist Hans Testerson
```

- überschriebene Eigenschaften können auch gelöscht werden

```
delete hans.vorname;
hans.vorstellen(); // Mein Name ist Horst Peter Testerson
```

- Überprüfen ob eine Eigenschaft vom Prototypen kommt:
  - hasOwnProperty()

```
hans.hasOwnProperty("vorname"); // false
```

Method over- und „underloading“

- *var func = function(a, b, c) {*
  return "Paramter waren "+a+", "+b+" und "+c+".";
  }
  func(1);
  //"Paramter waren 1, undefined und undefined."

- funktionsinterne Variable arguments
  var func = function(a, b, c) {
  console.log(arguments[4]);
  return arguments.length
  }
  func(1); //undefined & 1
  func(1,2,3,4,5); //5 & 5

Default Values

- Fallback bei der Erzeugung von Variablen

```
var a = b || c;
```

- super für Namespacing

```
var mathema = mathema || {};
mathema.common = mathema.common || {};
mathema.common.func = function(a, b) {
    return a+b;
}
...
var sum = mathema.common.func(1,2);
```

Immediate

- sofort ausgeführter Code

```
(function() {
  var result = 42;
  console.log(result);
}());
```

- Vorteil: benutze Variablen bleiben nicht im globalen Scope zurück

Script Templates

- Möglichkeit um HTML-Code Schnipsel auszuliefern ohne sie zu rendern

  → Alternative zu display:none

```
<script type="text/template" id="someId">
  beliebiger HTML-Code
</script>
```

  → demo3.html

Strict Mode

- deaktiviert einige „problematische Features“
- optionales Feature mit ECMAScript 5
  - am Anfang eines JS-Files, oder einer Function:
    "use strict";

- Auswirkungen:
  - with deaktiviert (und eval beschränkt, kein dynamisches Variablenbindung mehr)
  - kein implizites Anlegen von globalen Variablen
  - oktale Darstellung verboten (0100, 08, …)
  - usw...

Cross-Site-Scripting (XSS)

- Einschleußen schadhaften Codes
- „ungeprüftes“ Einbinden von Input
  - Ausgabe von Suchanfragen
  - Übernahme von Parametern

→ demo4.html

Minification - „size does matter“

- entfernen unnötiger Zeichen
  - white space, Kommentare, …
- oft in Verbindung mit weiterer Datenkompression
  - Variablenumbenennung, Entfernen von dead code, function inlining, BASE64-Kompression
- z.B. JSMin (Douglas Crockford), Closure Compiler (Google) oder YUI Compressor (Yahoo!)

statische Code Analyse

- JSLint
  - JavaScript Code Check
  - von Douglas Crockford
  - http://www.jslint.com/

  → JSLint Rhino Beispiel

AMD

- Asynchronous module definition

- modularer Aufbau von JS (Modul entspricht in etwa einer Java Klasse)
- Module können asynchron geladen werden („Inversion of Control“-Pattern)

→ demo5.html

Vielen Dank!

Frank Goraus

MATHEMA Software GmbH