3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

JavaScript goes Enterprise

Mit JavaScript Business-Anwendungen erstellen

Stefan Hildebrandt

consulting.hildebrandt.tk

@hildebrandttk

Agenda

- Motivation
  - Beispiel
- Testen mit qunit
- Die Sprache JavaScript
  - JSLint
  - Coffeescript
- Die Schnittstelle DOM
- Unobtrusive-Javascript
- MVC

Motivation

1. Erkenntnis:

   Umsetzung von zusätzlichen Funktionen wie direkte Validierung und bessere Benutzerführung ist mit Javascript für den Anwender schneller und besser umzusetzen als z.B. mit JSF!

2. Erkenntnis:

   Unbedarfte Herangehensweise von Java-Entwicklern bei der JavaScript Programmierung erzeugt schrecklichen Code

3. Erkenntnis:

   Anerkannte Hilfsmittel bei der Java-Entwicklung (Strukturen, Code-Analysen, verlässlicher Build, CI, Unit-Tests) werden weggelassen

Motivation

4. Erkenntnis:

   Java und Javascript haben nicht viel mehr als einen Teil des Namens gemeinsam

Beispiel Anwendung

- Einfaches Inventar
  - Speicherung von Items (Name, Beschreibung, GLN(EAN)-Code, Lagerort, Anzahl) in Kategorien strukturiert
    - Kategorien sind erst einmal vorgegeben
      - Bücher, CDs, DVDs, BDs, Handies, Schrauben
    - Bis auf das Feld GLN-Code sind alles Pflichtfelder
- Was macht ein Entwickler wenn er eine Lösung für eine Aufgabe sucht?
  - Googlen!?!
    - Good-Luck-Treffer

Was dabei rauskommen kann!

- Siehe Code-Beispiel

Probleme

- Aufruf der Events aus der HTML-Seite heraus
  - Vermischung von View und Control
- Javascript-Code in der HTML-Seite
  - Wiederverwendbarkeit nur mittels Copy-Paste
- Redundante Informationen (z.B. Name des Feldes)
  - Fehlerträchtig
- Tests?

qunit

- Testframework aus dem jQuery-Umfeld
- Funktionsweise und Vorgehensweise sind analog zu JUnit
- Strukturierung der Tests
  - Erstellung einer html-Seite je Testbereich
  - Ermöglicht die Einbindung von Mock-html und Tests darauf
- Alternative
  - Erstellen einer Testsuite in einer Datei die auf Test.js endet und mit dem Bibliotheksnamen startet
  - Ermöglicht die automatische Ausführung in Maven, …

qunit

```
<html>
<head>
  <meta charset="utf-8">
  <title>QUnit Example</title>
  <link rel="stylesheet" href="qunit-1.9.0.css">
</head>
<body>
  <div id="qunit"></div>
  <script src="qunit-1.9.0.js"></script>
  <script src="../scripts.js"></script>
  <script src="../scriptsTest.js"></script>
  </body>
</html>
```

qunit

```
test(<Beschreibung>,
  function() {
    <Ausführung der Tests>
});
```

- Checks
  - ok(state, message)
  - equal(actual, expected, message)
  - notEqual(actual, expected, message)
  - deepEqual(actual, expected, message)
  - notDeepEqual(actual, expected, message)
  - strictEqual(actual, expected, message)
  - notStrictEqual(actual, expected, message)
  - raises(block, expected, message)

qunit

```
test("test ean validation length",
  function() {
    equal(validateEan("121212121212"),
      "Length must be 13");
    equal(
      validateEan("141414141414"),
      "Length must be 13");
    equal(validateEan("4008110310541"),
      true);
});
```

Beispiel 2

- Validierung der GLN (Prüfsumme) mit qunit-Tests entwickelt

GLN-Kriterien

- Basisnummer (sieben bis neun Stellen) der Global Location Number. Sie besteht aus:
- Länderpräfix der GS1-Mitgliedsgesellschaft (drei Stellen), zum Beispiel 400 bis 440 für Deutschland, 760 bis 769 für die Schweiz und Liechtenstein, 900 bis 919 für Österreich
- Unternehmensnummer (ähnlich alte BBN Bundeseinheitliche Betriebsnummer)
- Artikelnummer des Herstellers (5, 4 oder 3 Stellen in Abhängigkeit zur Basisnummer, **sodass die Gesamtlänge immer 13 bleibt**)
- **Prüfziffer (1 Stelle)**
  (Quelle: Wikipedia)

Berechnung der Prüfziffer

- Die Prüfziffer der GTIN (ehem. EAN), die letzte Ziffer, errechnet sich, indem die einzelnen Ziffern von rechts nach links, beginnend mit der vorletzten (), **abwechselnd mit 3 und 1 multipliziert und anschließend diese Produkte addiert werden**:
  - (). Die Prüfziffer () ergänzt diese Summe dann zum nächsten Vielfachen von 10.
- Beispiel (siehe oben), EAN: `5 449000 09624-15*1 + 4*3 + 4*1 + 9*3 + 0*1 + 0*3 + 0*1 + 0*3 + 9*1 + 6*3 + 2*1 + 4*3  = 5 + 12 + 4 + 27 + 0 + 0 + 0 + 0 + 9 + 18 + 2 + 12=  89`
- Daraus folgt: `Prüfziffer = 1`

Beispiel 2 - Code

- Siehe Beispiel 2

Testen mit Mockups

- Verändern von Objekten für den Tests
- Definition von Mockups in anderen Dateien

Ausblick: Funcunit

- Arbeitet auf HTML-Seiten
  - Test komplexer Logik möglich
  - Integrationstests
- Automatische Ausführung der Tests über Selenium-Integration → CI
- Syntax basiert auf jQuery
  - In der Regel „natürlicher“ lesbar
- Bestandteil von javascript-mvc

Ausblick: Funcunit

```
steal("funcunit", function(){
   module("inventory test", {
      setup: function(){
         S.open("//inventory/inventory.html");
      }
});

test("Copy Test", function(){
   equals(S("h1").text(),
         "Welcome to JavaScriptMVC 3.2!","welcome text");
   });
})
```

JavaScript Grundlagen – Datentypen und Variablen

- Datentypen
  - String
  - Number
  - Boolean
  - Object
- var string = "abc";
- var number = 1.2;
- var int = 1;
- var bol = true;

JavaScript Grundlagen – Variablen

- Nicht typisiert!
- Typ änderbar

```
var name = 'Mustermann';
name = 1;
```

- Mit `var` wird eine lokale Variable definiert
  - Scope ist die Funktion in der sie deklariert wird
    - **Auch vor der Deklaration!**

      → Variablen immer am Anfang einer Methode deklarieren
- Wird `var` weggelassen (oder vergessen), wird implizit eine globale Variable angelegt!

JavaScript Grundlagen – Arrays

- Arrays
  - `var array = new Array();`
  - `var anOtherArray = [];`
  - `var fibs = [1,1,2,3,5,8,13,21,34,...]`
  - `var firstFib = fibs[0];`
- Manipulation am Ende
  - `fibs.push(55);`
  - `fibs.pop(); //liefert 55;`
- Manipulation am Anfang
  - `fibs.shift(0); //liefert fibs[0]`
  - `fibs.unshift(0); // fügt 0 vor dem ersten Element ein`
- Diverse weitere Methoden zum sortieren, spalten, ...

JavaScript Grundlagen – Funktionen

- Deklaration einer Methode

```
function add(a, b) {
    return a+b;
}
```

- Eine Funktion ist ein Objekt

```
var sum = function(a, b) {
    return a+b;
}
```

- Aufruf

```
var summe = add(1,3);
var summe = sum(1,3);
```

JavaScript Grundlagen – Objekte

```
var bar = new Object();
bar.foo = 1;
bar.add = function(a){
    foo = foo + a;
}
```

- Alterativ als Literal:

```
var bar2 = {
   foo : 1;
   add : function(a){
     foo = foo + a;
   }
}
```

→ Problem: Es gibt nur eine Instanz!

JavaScript Grundlagen – Objekte

- Erstellen eines neuen Objekts

```
function MyObject(){
    this.foo=1;
    this.add = function(a){
       this.foo = this.foo+a;
    }
}
var myNewObject1 = new MyObject();
myNewObject1.add(2);
```

JavaScript Grundlagen – Bedingungen

- Langform

```
if (d > 1) {
    d=0;
}else{
    d=-d;
}
```

- Ternärer Operator

```
d>1 ? d =0 : d=-d;
```

JavaScript Grundlagen – Schleifen

```
for (var i = 0; i< 10; i++){
    a = a+i;
}
```

JavaScript Grundlagen – Exceptions

- Exception werfen:

```
throw {
 name: 'ValidationError',
 message: 'Name must be filled'
 };
```

- … und fangen:

```
try {
 doSomething();
} catch (e) {
 alert(e.name + ': '+ e.message);
}
```

JavaScript Grundlagen – Closures

```
function outer() {
    var foo = 1;
    var inner = function() {
       alert("foo is "+foo);
    }
}
outer();
```

JavaScript Grundlagen – Wrapper

```
for(var i = 0; i < 10; i++){
    function outer() {
       var inner = function() {
          alert("This is element "+i);
       }
       elements[i].onclick = inner;
    }
    outer();
}
```

JavaScript Grundlagen – anonymer Wrapper

```
for(var i = 0; i < 10; i++){
  (function() {
   someElements[i].onclick = function() {
    alert("This is element "+i);
   }
  })();
}
```

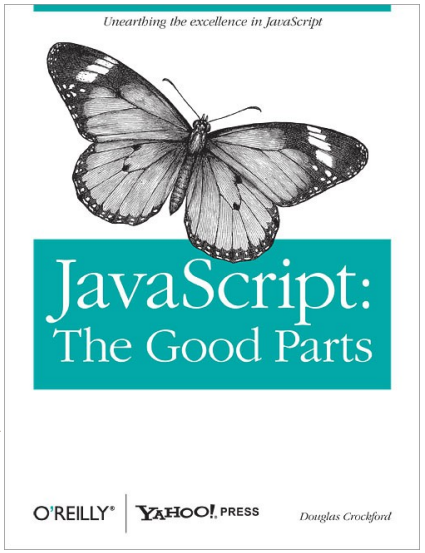
JavaScript Grundlagen – Prototypes

- Im Gegensatz zu Java, deren Basis statische Klassen sind, nutzt Javascript einen prototypischen Ansatz, bei dem sich Objekte erweitern lassen.
- Erstellen eines neuen Objekts

```
function MyObject(){
 this.foo=1;
}
MyObject.prototype.add
 = function(a){
  this.foo = this.foo + a;
}
var myNewObject1 = new MyObject();
myNewObject1.add(2);
```

Schwächen der Sprache (Beispiele)

- Globale Variablen
  - Implizite Deklaration
- Scopes von lokalen Variablen
  - Gilt von Begin der Methode
- Semikolon Einsetzung
  - Es wird versucht diese hinzuzufügen, wenn diese nicht vorhanden waren
- typeof
  - typeof null → object !?
- parseInt
  - Arbeitet stillschweigend bis zum 1. nicht Ziffer
  - Arbeitet mit führender „0“ im oktalen System



Schwächen der Sprache (Beispiele)

- Vergleiche

  " == '0' //false

  0 == " // true

  0 == '0' //true

  false == 'false' //false

  false == '0' //true

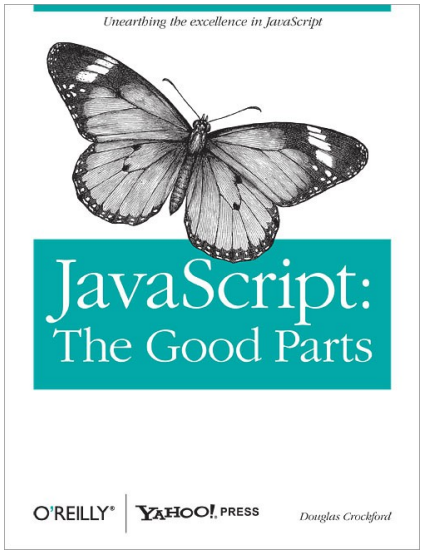
  false == undefined //false

  false == null //flase

  null == undefined // true

  ' \t\r\n' == 0 //true

  → Nur „===“ und „!==“ verwenden



JSLint

- Definiert ein stabiles und professionell nutzbares Subset von JavaScript
- Automatische Prüfung möglich
- http://www.jslint.com

Coffeescript

- Javascript:
  - Hat schwerwiegende Designfehler!
  - Kann nicht repariert werden!

  → Entwicklung einer Alternative
- Compiliert zu Javascript
- Compiler ist in JS geschrieben
  - Online-Compilierung bei der Entwicklung möglich
- Übersetzt instabile JS-Konstrukte in stabile
- Sehr kompakte Sprache
  - Für meinen Geschmack teilweise leider zu viele Optionen das gleiche zu machen
- Lässt sich in Verbindung mit jQuery und anderen JS-Frameworks nutzen

Die $-Funktion (jQuery, ...)

- Wird in allen aktuellen Frameworks genutzt
  - Erster Einsatz in Prototype
- Bietet einfache Zugriffe auf den DOM mit CSS-Selektoren
- Bietet einheitliches, Browser-übergreifende Möglichkeiten mit DOM-Objekten zu arbeiten
  - z.B. `addClass(<styleClass>)`, `removeClass(<styleClass>)`, `val()`, `attr(<dom-Attributname>, <Wert>)`

„Unobtrusive“ HTML-Einbettung

- Unaufdringliches Einbinden
  - Keine Vermischung von HTML und JavaScript
    - Auslagerung
    - Binden an HTML-Elemente aus Javascript heraus
      - `$('#tabs').click(someCallbackFunction1)`
  - Kollisionsvermeidung

    → Namensräume

JavaScript Grundlagen – Namensräume

```
var tk_hildebrandt = {
  foo = 1;
  bar = 2;
  addFooBar = function(){
    return foo+bar;
  }
}
```

- Ggf. prüfung auf bereits bestehenden Namensraum einbauen
- Über Verschachtelungen und Closures ist Data-Hiding möglich

Struktur?

- Viele Projekte setzen auf jQuery auf
  - Bietet kaum Struktur für große Projekte
    → Diese muss durch andere Frameworks geboten werden
  - Beispiele
    - Batman.js (Morgen 14h)
    - AngularJS
    - extJS ab 4.0
    - Ember.js
    - Knockout.js
    - Google Closure
    - GWT
    - Javascript-MCV
    - viele weitere, da stark im Fluss

Javascript-MVC

- Wurde ins leben gerufen um große JS-Anwendungen evolvierbar entwickeln zu können
  - Struktur für die Anwendung
    - devide and conqueror
  - Dependency Management
  - Build-System inkl. Testausführung
- Bestandteile
  - jQueryMX
  - StealJS
  - FuncUnit (qunit)
  - DocumentJS

JQuery MX

- Erweiterung von jQuery mit Strukturelementen
  - Class
    - Ermöglicht die Erstellung von Klassen inkl. Vererbung und Konstruktoren
  - Model
    - Stellt die Basis für Zugriffe auf JSON-Rest-Services zur Verfügung
  - View
    - Templating zum (Re-)Rendern von Daten
  - Controller
    - Event-Binding und Event-Handling

JQuery MX – Class

```
$.Class('Monster',
/* @static */ { count: 0 },
/* @prototype */ {
  init: function( name ) {
    this.name = name; this.health = 10;
    this.constructor.count++;
  },
  eat: function( smallChildren ){
    this.health += smallChildren;
  },
  fight: function() {
    this.health -= 2;
  }
});
```

JQuery MX – Class

```
hydra = new Monster('hydra');
dragon = new Monster('dragon');
hydra.name          // -> hydra
Monster.count       // -> 2
Monster.shortName // -> 'Monster'
hydra.eat(2);       // health = 12
dragon.fight();     // health = 8
```

JQuery MX – Class – Inheritance

```
Monster("SeaMonster",{
  eat: function( smallChildren ) {
    this._super(smallChildren / 2);
  },
  fight: function() {
    this.health -= 1;
  }
});
lockNess = new SeaMonster('Lock Ness');
lockNess.eat(4);   //health = 12
lockNess.fight();  //health = 11
```

JQuery MX – Controller und Binding

- Deklaration:

```
$.Controller("TaskList", {
  defaults : {}
},{
  init : function(){  },
  "li click" : function(){  },
  "li a click" : function(el, ev)
 {  }
- })
```

- Instanzierung und Bindung

```
$('ul#tasks').task_list()
```

- Alterativ:

```
new TaskList($('ul#tasks'),{});
```

JQuery MX – Controller-Interaktion

- Aufruf eines Controllers aus einem anderen
  ```
  $('ul#tasks').task_list('doFoo')
  $('ul#tasks').task_list('doFoo',
    'With Bar')
  ```
- Alle Controller an einem Element
  ```
  $('ul#tasks').controllers()
  ```
- Alle Controller von einem Typ (inkl. abgeleitete)
  ```
  $('ul#tasks').controllers(TaskList)
  ```

JQuery MX – Controller-Eventing

```
$.Controller("TaskDetails", {
    defaults:{},
    listensTo:["taskSelected"]
}, {
    init:function () {
       this.element
       .addClass('taskSelectedListener');
    },
    taskSelected:function () {
       //do checks hier
    }
});
```

JQuery MX – Controller-Eventing

```
$.Controller("TaskList", {
    defaults:{}
}, {
    'li click':function (el) {
        …
    $('.taskSelectedListener')
     .trigger('taskSelected', el);
     }
});
```

Beispiel 3

- Erstellung von ValidationRules Controllern
  - Required
  - GLN
- Erstellung eines Form-Controllers
  - Überprüfung des gesamten Formulars und Unterbindung des submits
- Verbindung der Controller

Beispiel 3

- Siehe Code-Beispiel

jQuery MX – Model

- Kapselung von JSON-REST-Service aufrufen
- Events bei geladenen/geänderten Daten

```
$.Model('Todo',{
  findAll: 'GET /todos.json',
  findOne: 'GET /todos/{id}.json',
  create:  'POST /todos.json',
  update:  'PUT /todos/{id}.json',
  destroy: 'DELETE /todos/{id}.json'
},{});
```

jQuery MX – Model

- Neue Instanz erstellen

```
var todo = new Todo({name: "do the
 dishes"})
```

- Speichern einer Instanz

```
todo.save();
```

- Aus einem Controller heraus mit Callback

```
newTodo(el.formParams())
    .save(this.callback('saved'));
```

JQuery MX – Fixtures

- Fixtures sind ein Dummy Model
- Verhält sich wie der REST-Service
  - Einbindung in Unit-Tests
  - Einbindung während der Entwicklung

JQuery MX – Fixtures

```
$.fixture.make("inventory_entry", 5, function(i,
  inventory_entry){
     var descriptions = ["grill fish", "make
     ice", "cut onions"]
     return {
        name: "inventory_entry "+i,
        type: "Kategorie "+i,
        count: i,
        location: i,
        description: $.fixture.rand( descriptions
        , 1)[0]
     }
})
```

jQuery MX – Templates

- Templating ist Notwendig, um neue View-Bestandteile für Komponenten hinzufügen zu können
  - z.B. Aufbau von Listen
- Die Komponente View unterstützt unterschiedliche Template-Engines
  - EmbeddedJS (ejs)
  - JAML
  - Micro
  - Jquery.Tmpl

jQuery MX – Templates

- Embeded JS

```
<%for(var i = 0; i < this.length ; i++){%>
  <li <%= this[i]%>>
    <h3><%= name %> <a href='javascript://'
    class='destroy'>X</a></h3>
    <p><label>Kategorie:</label> <%= type %></p>
    <p><label>Anzahl:</label> <%= count %> </p>
    <p><label>Ort:</label> <%= location %> </p>
    <p><label>Beschreibung:</label> <%=
    description %></p>
  </li>
<%}%>
```

steal

- Dependency Management
  - js, css, Templates
- Build-Management

Beispiel 4

- Großes Beispiel Javascript-MVC

Zum Schluss

- Weitere wichtige Aspekte bei der JS-Entwicklung in weiteren Vorträgen hier auf dem HC
  - Testausführung im Build und CI
  - Zusammenfassung und Komprimierung
  - Dependency Management

Noch Fragen?

Vielen Dank!

Stefan Hildebrandt

consulting.hildebrandt.tk

consulting.hildebrandt.tk

- Beratung, Coaching und Projektunterstützung
  - Java EE
    - CDI, EJB3.1, JSF2, JPA
    - Spring, Hibernate, JpaSecurity
    - Tomcat, JBoss, TomEE
    - Maven, Gradle, ant
    - Troubleshooting (Performance, Speicher)
  - Singepage Applications mit REST und JavaScript
  - Testautomatisierung
  - Testen in agilen Projekten

consulting.hildebrandt.tk
@hildebrandttk