Code-Organisation in JavaScript

Webworker-Stammtisch NRW

10. Januar 2013

Frederic Hemberger

Codekapselung

What happens in Vegas, stays in Vegas.

Warum sollte man Code kapseln?

Wir haben immer mehr Scripte auf unseren Webseiten:

- JavaScript-Frameworks
- Plug-ins
- Banner, Adwords, Affiliate-Scripts
- Social Media Plug-ins
- Webtracking

Warum sollte man Code kapseln?

„Lass nicht überall dein Zeug rumliegen!“ – „Jaaa, Mama.“

JAVASCRIPT

```
// script1.js
function doStuff() { console.log("I do stuff"); }
```

JAVASCRIPT

```
// script2.js
function doStuff() { console.log("Ha, I don't!"); }
doStuff();
```

Codekapselung

```javascript
(function() {
  // Mein Code …
})();
```

Codekapselung

Immediately Invoked Function Expression (IIFE)

JAVASCRIPT

```
// Function
(function() {
  // Mein Code …
})();
```

JAVASCRIPT

```
// Expression
(function() {
  // Mein Code …
})();
```

JAVASCRIPT

```
// Unmittelbare Ausführung
(function() {
  // Mein Code …
})();
```

Modularisierung

Codekapselung und Sichtbarkeit

Definierte Variablen und Methoden nach außen sichtbar machen:

JAVASCRIPT

```
(function(exports) {
  var privateVar        = "foo";
  var exports.publicVar = "bar";

  exports.myFunction = function() {
    console.log("Hello global namespace!");
  }
})(window);


  console.log(privateVar);
  // => ReferenceError: privateVar is not defined


  console.log(publicVar);
  // => "bar"


  myFunction();
  // => "Hello global namespace!"
```

Codekapselung und Sichtbarkeit

Warum „**exports**“?

JAVASCRIPT

```
(function(exports) {
  exports.myFunction = function() {
    console.log("Hello global namespace!");
  }
})(…);
```

Der globale Namespace kann sich beliebig ändern:

- Browser: `window`
- Node.js: `exports`
- Applikation: z.B. `myApp`

Der gekapselte Code selbst muss nicht mehr angepasst werden.

Hinweis: "public" und "export" sind reservierte Wörter in JavaScript, daher "exports"

Module Pattern

```javascript
var myApp = (function(exports) {
  exports.myFunction = function() {
    console.log("Hello global namespace!");
  }

  return exports;
})(myApp || {});
```

JAVASCRIPT

Vorteile:

- Zentraler Namespace nach außen (`myApp`)
- Codekapselung und Sichtbarkeit (wie bei der IIFE)
- Code kann so über mehrere Dateien verteilt werden

Addy Osmani: Learning JavaScript Design Patterns

Enge und lose Kopplung

Wer bin ich, und wenn ja, wie viele?

Enge Kopplung

`modul2.js` kann Methoden aus `modul1.js` aufrufen:

```javascript
// modul2.js
myApp.functionFromModule1();
```

JAVASCRIPT

- solange `modul2.js` von der Existenz von `modul1.js` weiß …
- … und beide in der richtigen Reihenfolge eingebunden sind.

```html
<!-- index.html -->
<script src="modul1.js"></script>
<script src="modul2.js"></script>
```

HTML

Enge Kopplung

Was bei enger Kopplung beachtet werden muss:

- Welches Modul verwendet welche anderen Module?
- In welcher Reihenfolge müssen die Module geladen werden?
- Was ist, wenn ich weitere Module hinzufügen möchte …
- … oder andere weglassen/austauschen?

Kommunikation über Events

Publish-Subscribe-Prinzip (PubSub)

Kommunikation über Events

Publish-Subscribe-Prinzip (PubSub)

- Kommunikation erfolgt asynchron über Events
- Funktioniert analog zu Events im DOM (z.B. `click`)
- Module müssen nicht voneinander wissen
- Module interagieren nicht direkt miteinander
- Einfach erweiterbar

Kommunikation über Events

Publish-Subscribe-Prinzip (PubSub)

JAVASCRIPT

```
$.subscribe('myevent', function(event, param1, param2) {
  console.log('Mein Event wurde ausgelöst:', param1, param2);
});
// Entspricht $(document).on(…)
```

JAVASCRIPT

```
$.publish('myevent', ['Erster Wert', 'Zweiter Wert']);
// Entspricht $(document).trigger(…)
```

Beispiel mit „jQuery Tiny Pub/Sub“: https://gist.github.com/661855

Beispiel

Darstellung auf einer Shop-Seite

JAVASCRIPT

```
// shoppingcart.js
$.subscribe('quantity:change', function(event, article, quantity) {
  redrawShoppingCart(article, quantity);
});

// article-overview.js
$.subscribe('quantity:change', function(event, article, quantity) {
  checkForDiscount(article, quantity);
});

// order-familypack.js
$.publish('quantity:change', ['Product 1', 5]);
```

Abhängigkeiten definieren

Modul-Standards in JavaScript

Abhängigkeiten definieren

Modul-Standards in JavaScript

- **CommonJS**
  meist serverseitig (z.B. in Node.js):
  ```
  module = require('modulename');
  ```
- **Asyncronous Module Definition (AMD)**
  clientseitig, z.B. mit Require.js (requirejs.org)
- künftig: native Unterstützung (ECMAScript 6)

AMD-Module mit Require.js

- Lädt Module per Ajax (**Asyncronous** Module Definition)
- Verwaltet die Abhängigkeiten untereinander
- Kann auch Modul-Assets laden (z.B. Templates, CSS)
- Kann Module für Produktivbetrieb kompilieren
  (Auflösung von Abhängigkeiten, Zusammenfassen & Minifizieren)

AMD-Module mit Require.js

HTML

```
<!-- index.html -->
<script src="require.js" data-main="app.js"></script>
```

JAVASCRIPT

```
// app.js
require(['lib/jquery', 'modul1'], function($, modul1) {
  $(function() {
    modul1.myFunction();
  }
});
```

AMD-Module mit Require.js

Abhängigkeiten innerhalb von Modulen

JAVASCRIPT

```
// modul1.js
define(function() {
  return {
    myFunction: function() {
      console.log('Hello Require.js!');
    }
  }
});
```

JAVASCRIPT

```
define(['modul2', 'include/modul3'], function(module2, module3) {
  // …
});
```

Deployment?

Grunt. ;-)

<Thank You!>

twitter @fhemberger
www frederic-hemberger.de