Aktuelle JavaScript Techniken

Pascal Hertleif

6. Oktober 2014

Contents

Pascal Hertleif

- Informatik-Student
- Web-Entwickler
  - "Reguläre" Websites, Typo3
  - Web-Applikationen, u.a. mit angular.js
  - Build-Tools, REST APIs mit Node.js
- pascalhertleif.de

JavaScript — Was bisher geschah

- Seit 1995: Skriptsprache für dynamisches HTML
- Seit 2000: AJAX, Browser-Apps
- Seit etwa 2008 eine der schnellsten Script-Sprachen [2]
- 2008: *JavaScript: The Good Parts* [3]

- Erfunden von Brendan Eich
- ECMAScript
- AJAX: Client Side Views

- Node.js [4]
  - Basiert auf V8 (Chrome's JS Engine)
  - System-Schnittstellen
  - Event Loop
  - Ideal für JavaScript auf dem Server

Dilemma

- Immer größere Applikation in JavaScript
- Sehr dynamische, asynchrone Sprache, wenige Sicherheiten
- Ausweg: Struktur, Konventionen, Programmiertechniken

Module

Ohne Module

```
<script src="js/jquery.js"></script>
<script src="js/jquery.lightbox.js"></script>
<script src="js/effects.js"></script>
```

- Das sind 3 HTTP-Requests
- `jquery.lightbox.js`: Anscheinend ein jQuery-Plugin
  - Welche Version von jQuery?
- `effects.js` benötigt vermutlich jQuery und das Lightbox-Plugin. Diese werden vom Browser vorher geladen, und sind *nur* deshalb verfügbar.

Ziel von Modulen

- Code aufteilen
- Abhängigkeiten abbilden
- Code-Teile wiederverwenden können

CommonJS Module [5]

- von node.js verwendet
- explizite Abhängigkeiten
- Nur eins von vielen Modul-Systemen

Modul-Aufbau

- 1 Datei = 1 Modul
- `require` lädt andere Module
- `module` repräsentiert das aktuelle Modul
- `module.exports` ist, was andere Module von `require` bekommen

Beispiel

```
// src/helper.js
var fs = require('fs');
module.exports = {
  saveTheQueen: function (name) {
    fs.writeFile(name+".txt", name+" was saved.");
  }
}


// src/main.js
var helper = require('./helper');
helper.saveTheQueen("Elizabeth");
```

Pakete

- (wiederverwendbare) Code-Einheiten
- Beschrieben durch
  - Name
  - Einstiegspunkt
  - externe Abhängigkeiten

NPM

- npmjs.org, "Node Packaged Modules"
- *Registry* mit 96.608 Paketen (26. Sep 2014)
- löst Abhängigkeiten automatisch auf
- `npm install express`

- Pakete können zentral gespeichert werden (müssen aber nicht)
- Ein Grund, warum node.js so viel Anklang findet

Module im Browser

- Code kann nicht direkt eingebunden werden
- Tools, um CommonJS zu verarbeiten:
  - browserify [6]
  - webpack [7]

Browserify Beispiel

```
$ browserify src/main.js -o dist/bundle.js
```

- Beginnt bei `src/main.js` an
- Löst rekursive alle Abhängigkeiten auf
- Schreibt gesamten Code in `dist/bundle.js`

Webpack [7]

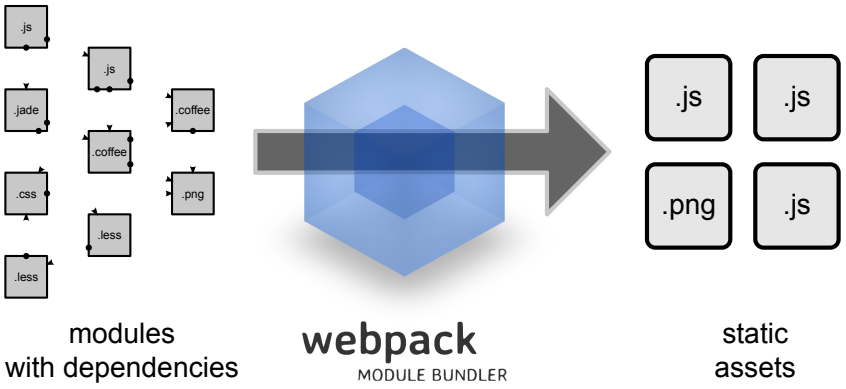

Figure 1:

Alternative: Require.js/AMD [8]

```
// scripts/fooify.js
define(function () {
  // ...
});
```

```
// scripts/main.js
require(['jquery', 'fooify'], function ($, fooify) {
  // ...
});

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

Entwicklungsvorgehen

- Require.js:
  - bindet einzelne Dateien direkt ein (*development*)
  - oder kombiniert sie im Vorhinein (*production*)
- Browserify/webpack
  - muss immer ausgeführt werden
  - kann performant nebenher laufen (*watch*)
- beide Systeme
  - externen Code übernehmen (*shim*)
  - Code via Plugins verarbeiten und minifizieren ## Callbacks

Anonyme Funktionen

- *First-order functions*, *closures*
- Grundlegend für funktionale Programmierung

```
function forEach (list, fn) {
  for (i = 0; i < list.length; i++) {
    fn(list[i]);
  }
}

var user = ['Hans', 'Paul'];

forEach(users, function (user) {
  console.log(user);
});
```

Was sind Callbacks?

- Beim Aufruf einer Funktion wird eine “Rückgabe-Funktion” mitgegeben

- Reagieren auf (externe) Ereignisse
- werden in bestehendem Kontext ausgeführt
- Code soll ausgeführt werden, sobald z.B.
  - User klickt
  - Datei geschrieben wurde

Synchroner Code

```
var fs = require('fs');
fs.writeFileSync('example.txt', "Lorem Ipsum");
console.log("File written");
```

- `writeFileSync` blockiert den Prozess, bis Datei geschrieben wurde
- danach wird auf die Konsole geschrieben

Asynchroner Code

```
var fs = require('fs');

function log (err, res) {
  if (!err) {
    console.log("File written");
  }
}

fs.writeFile('example1.txt', "One", log);
fs.writeFile('example2.txt', "Two", log);
```

- `log` ist ein Callback
- wird ausgeführt, nachdem Datei geschrieben wurde
- welche Datei zuerst geschrieben wurde, wissen wir nicht
- auch zu beachten: Fehler-Parameter

Verwendet jQuery.

```
var currentUser = {name: "Pascal"};
jQuery.getJSON('/me', function (result) {
  currentUser = result.data;
});
```

- Ajax Event: “Request fertig”
- Callback wird im Fehlerfall nicht ausgeführt (Sonderfall)

Node.js Callback Conventionen

- Asynchrone Funktionen verlangen als letztes Argument eine Callback-Funktion
- Callbacks werden mit Fehler als erstes Argument aufgerufen

```
async(param1, param2, function (err, result) {});
```

Verschachtelte Callbacks

- Beispiel:
  - Config-Datei lesen
  - dann 3 Netzwerk-Abfragen
  - dann User-Eingabe abwarten
  - dann Ergebnis in Datei schreiben
- viele verschachtelte Funktionen, viele `if (error)`
- "Callback-Hell"

Promises

Definition

- Ein *Promise* repräsentiert einen zukünftigen Wert.
- Es kann erfüllt (*fulfilled*) oder abgelehnt (*rejected*) werden.
- Es können Callbacks angehangen werden, die ausgeführt werden, sobald der Wert bekannt ist.

Genauer: Promises/A+ [9]

Beispiel (ajax)

```
var currentUser = {name: "Pascal"};

jQuery.getJSON('/me')
.then(function (result) {
  currentUser = result.data;
}, function (err) {
  console.error(err);
});
```

Figure 2: Promises/A+ Logo [9]

Beispiel (node)

```
requests = readConfig()
.then(function (config) {
  return Promise.all(urls, function (url) {
    return request(config, url);
  });
})
.then(userInput)
.then(writeFile('result.txt'))
.catch(function (error) {
  console.error(error);
})
```

Promise erstellen

```
var Promise = require('bluebird');
var maybe = new Promise(function (resolve, reject) {
  var x = Math.random();
  setTimeout(function () {
    if (x > 0.5) {
      resolve("yay");
    } else {
      reject("nope");
    }
  }, 1000);
});
```

Events

Asynchrone Ereignisse

- User klickt
- Antwort von Netzwerk

Wiederholbar

- 5 Klicks auf den selben Knopf
- Teil einer Datei gelesen (bis Ende erreicht)

Beispiel (jQuery)

```
var count = 1;
jQuery('button').on('click', function (event) {
  console.log('Clicked', count, 'times');
  count = count + 1;
});
```

Beispiel (node.js)

```
var fs = require('fs');
var file = fs.createReadStream('test.csv');
file.on('data', logProgress);
file.on('error', logError);
file.on('end', logSuccess);
```

- Auch: Event bei neuer Verbindung zu Server.
- System-Events

Events erstellen

```
jQuery('#bar').on('click', function () {
  jQuery('#foo').trigger('magic', {answer: 42});
});
jQuery('#foo').on('magic', function (event, data) {
  console.log("Answer is", data.answer);
});
```

Streams

Event & Data Streams

- “Ereignis-Fluß”
- Daten werden in Buffer gespeichert (Push/Pull)
- Functional Reactive Programming

Streams in Node

```
var request = require("request");
var zlib = require("zlib");
var fs = require("fs");
```

```
// HTTP GET Request
request("http://nodestreams.com/input/people.json.gz")
    // Un-Gzip
    .pipe(zlib.createGunzip())
    // Write File
    .pipe(fs.createWriteStream("output/people.json"));
```

Vergleiche auch Beispiele auf nodestreams.com.

Fragen?

Danke für die Aufmerksamkeit.

Quellen

[1] S. Powers, *Learning node*. O'Reilly Media, Inc., 2012.

[2] Google, „V8 JavaScript Engine: Changelog", 2008. https://code.google.com/p/v8/source/browse/trunk/ChangeLog. (Zugegriffen: 26. September 2014)

[3] D. Crockford, *JavaScript: The Good Parts*. O'Reilly Media, Inc., 2008.

[4] D. Ryan, „Node.js", 2009. http://nodejs.org/. (Zugegriffen: 26. September 2014)

[5] CommonJS group, „CommonJS Modules 1.1.1". 23. März 2010. http://wiki.commonjs.org/wiki/Modules/1.1.1. (Zugegriffen: 28. September 2014)

[6] Browserling Inc., „Browserify website", 2012. http://browserify.org/. (Zugegriffen: 22. September 2014)

[7] T. Koppers, „webpack module bundler", 4. März 2012. http://webpack.github.io/. (Zugegriffen: 21. September 2014)

[8] J. Burke, „Require.js", 8. September 2014. http://requirejs.org/. (Zugegriffen: 18. Oktober 2014)

[9] B. Cavalier und D. Denicola, „Promises/A+". 6. Dezember 2012. http://promisesaplus.com/. (Zugegriffen: 26. September 2014)