thecodecampus</>

W11K

JavaScript von Morgen schon heute

- Ausgangssituation
- Setup
- Features

ÜBER MICH

- Philipp Burgmer
  - Software-Entwickler, Trainer
  - Fokus: Frontend, Web-Technologien
  - burgmer@w11k.de
- w11k GmbH
  - Software Design, Entwicklung & Wartung
  - Consulting, Schulungen & Projekt Kickoff
  - Web Anwendungen mit AngularJS
  - Native Rich Clients mit Eclipse RCP

AUSGANGSSITUATION

ECMASCRIPT 5

- Standard seit 2009
- Kompabilität
  - Aktuelle Browser 100%
  - IE8 11% ohne Polyfill
  - IE9 97% (nur kein strict-Mode)
  - Node 100%
  - io.JS 100%

Quellen:

- ES5 Compat Table

AUSGANGSSITUATION

ECMASCRIPT 6

- Standard wird Juni 2015 verabschieded
- Aktuelle Implementierungen gegen Entwurf
- Kompabilität
  - IE10 4%
  - Chrome 43 41%
  - Firefox 38 60%
  - Edge 63%
  - Node 19%
  - io.JS 39%

Quellen:

- ES6 Compat Table

AUSGANGSSITUATION

ECMASCRIPT 7

- Ferne Zukunft? Auch ES 2016 genannt!
- Entwürfe und Vorschläge
- Interessante neue Features
  - Pattern Matching
  - Traits
  - Guards und Trademarks
  - Object.observe

PROBLEME

- Alte Umgebungen / Browser
  - IE8 immer noch relevant
  - IE9 wird uns noch Jahre begleiten (derzeit > IE10 [1])
  - Node 0.10?
- Nicht alle Engines implementieren alles gleich
  Verschiedene Versionen von Proposals und Drafts

Quellen:

- [1] https://www.netmarketshare.com/browser-market-share.aspx?qprid=2&qpcustomd=0

NEUE FEATURES IN ES

- Neu in Standard-Biblothek (Promises, Maps, Sets, ...)
- Syntactic Sugar (Klassen, Module, ...)
- Erweiterungen der Sprache und Engine
  - Typed Arrays
  - Proxies
  - Erben von vordefinierten Konstruktor-Funktionen
  - ...

SHIMS / POLYFILLS

- Rüsten fehlende Teile der Standard-Biblothek nach

- es-shims
  - es5-shim (44%)
  - es6-shim (17%)
  - es7-shim (33%)
- core-js (*48%* von ES6)
  - *Modular compact standard library for JavaScript*
  - Shims für ES5, ES6 und ES7 proposals
  - dictionaries, extended partial application, console cap, date formatting
- ...

Quellen:

- ES Compat Table
- core-js (ohne Babel) Wert über eigene Erweiterung von *ES6 Compat Table*

TRANSPILER

- Übersetzen *Syntactic Sugar* von ES6 in ES5 Code
- Bringen meist Polyfills gleich mit

- Google Traceur
- Babel (früher 6to5)
- Microsoft TypeScript

TypeScript

- Optionale Typen, Generics
- Classes, Interfaces, Modules
- Mixins

- Viele nicht ES6 Features
- Interoperabel da Superset von ES5
- Bewegt sich mit 1.5 und 1.6 mehr zu ES6
- Für neues Projekt sehr zu empfehlen
- Migration teilweise schwierig
  - Nicht jeder ES5 Code ist gültiger TypeScript Code (Type Inference)
  - Aber jeder ES5 Code kann in TypeScript eingebunden werden (separate Datei mit Typ-Deklarationen notwendig)

SETUP

SETUP

PACKAGE MANAGER

- Optional!
- Läd Abhängigkeiten von externen Quellen
  - Für Entwicklung auf lokalen Rechner
  - Für Betrieb auf Server
- Verwaltet Versionsnummern und Konflikte
- Bower, NPM, JSPM, ...

SETUP

MODULE LOADER

- Optional!
- Läuft im Browser/Client
- Läd benötigten Code vom Server in den Client
- Nicht Teil von ES6, extra Standard/Entwurf [1]
- RequireJS, Browserify, SystemJS, WebPack, ...
- Bundling nicht vergessen

Quellen:

- [1] ES6-Module-Loader Polyfill

SETUP

TRANSPILER

- Optional!
- Übersetzt ES6 in ES5
- Dynamisch im Client, on demand
- Kann in Module-Loader integriert werden
- Statisch im Build-Prozess

SETUP

DEMO

ES6 FEATURES

UND WIE SIE UNSER LEBEN VEREINFACHEN

PROMISES

- Besserer Umgang mit Asynchronität
- Callback-Hell vermeiden
- Weit verbreitet als Biblotheken

"Pyramid of Doom"

```
step1(function (value1) {
  step2(value1, function(value2) {
    step3(value2, function(value3) {
      // Do something with value3
    });
  });
});
```

Flache Struktur mit Promises

```
promisedStep1()
.then(promisedStep2)
.then(promisedStep3)
.then(function (value3) {
    // Do something with value3
})
```

PROMISES

KRIS KOWAL'S Q

- Weit verbreitete Biblothek (z.B. in AngularJS)
- Andere API als ES6 Promises: Promise & Deferred
- In AngularJS 1.3 `$q` als Factory-Funktion, ähnlich wie ES6

Deferred API of Q

```
function asyncFunc() {
  var deferred = Q.defer();
  setTimeout(function () {
    deferred.resolve('result'); // or deferred.reject('error');
  }, 500);

  return deferred.promise;
}
```

Factory API of $q in AngularJS

```
function asyncFunc() {
  return $q(function(resolve, reject) {
    $timeout(function () { resolve(); /* or reject(); */ }, 500);
}
```

PROMISES

ES6

- Keine Biblothek notwendig, Teil der Standard-Bibliothek
- Vordefinierter Konstruktor

```
function asyncFunc() {
  return new Promise(function(resolve, reject) {
    setTimeout(function () {
      resolve(); // or reject();
  }, 500);
}
```

MODULE

- In ES5 kein native Modul-System, daher AMD und CommonJS
- In ES6 natives Modul-System (Beste aus beiden)
  - Dynamisches, on demand Laden von Code
  - Scope für Variablen, keine globalen Variablen mehr
  - Ein Modul pro Datei, eine Datei pro Modul
  - Struktur auf Datei Ebene
  - Abhängigkeiten zwischen Dateien/Modulen im Code

Modul mit Import und Named-Export

```
1 import $ from 'jquery';
2 import 'another-lib';
3
4 export class Point {}
```

PROBLEME

MIT MODULEN

- Verschieden Module Systeme, da bisher kein Standard
- Viele Module-Loader (RequireJS, Browserify, SystemJS, Webpack, ...)
- Nicht jede Bibliothek für jedes Modul-System vorhanden
- Viele Bibliotheken noch nicht als ES6 verfügbar

MODULE

UND ANGULARJS

- AngularJS 1.x hat eigenes Modul-System
- Hat praktisch keinen Mehrwert (mehr) [1]
- Wird man nicht los, zumindest nicht ganz
- Zwei Möglichkeiten
  - Modul-System über Modul-System
  - AngularJS Module-System weitestgehend ignorieren

Infos:

- [1] This AngularJS modules/dependencies thing is a lie

MODULE

UND ANGULARJS

ES6 Modul-System über AngularJS Module-System

```
1  import angular from 'angular';
2  import 'angular-ui-router';
3
4  import {homeIndexModule} from './index/home-index';
5  import {authRequiredModule} from 'app/routes/auth-required';
6
7  export var homeModule = angular.module('app.route.home', [
8    'ui.router',
9    authRequiredModule.name,
10   homeIndexModule.name
11 ]);
12 homeModule.config(function homeRoute($stateProvider) {
13     $stateProvider.state('app.auth-required.home', {
14       abstract: true,
15       template: '<ui-view></ui-view>'
16     });
17   }
18 );
```

Quellen:

- gocardless/es6-angularjs

MODULE

UND ANGULARJS

ES6 Modul-System statt AngularJS Module-System

```
1  import angular from 'angular';
2  import 'angular-ui-router';
3
4  import './index/home-index';
5  import 'app/routes/auth-required';
6
7  var appModule = angular.module('app');
8  appModule.config(function homeRoute($stateProvider) {
9      $stateProvider.state('app.auth-required.home', {
10       abstract: true,
11       template: '<ui-view></ui-view>'
12     });
13   }
14 );
```

KLASSEN

- Andere Schreibweise für existierende Funktionalität
- Kann fast vollständig in ES5 übersetzt werden
- Vererbung per Transpiler nicht zu 100% möglich

Klasse in ES6

```
class Point {
  constructor(x, y) { this.x = x; this.y = y; }
  toString() { return `(${this.x}, ${this.y})`; }
}
```

Klasse übersetzt in ES5

```
var Point = (function () {
  function Point(x, y) { this.x = x; this.y = y; }
  Point.prototype.toString = function () {
    return "(" + this.x + ", " + this.y + ")";
  };
  return Point;
})();
```

SERVICES IN ANGULARJS

OHNE KLASSEN

- Services in AngularJS: allgemeine Objekte, die Funktionalität kapseln
- Werden vom Framework instanziiert und verwaltet

Module#service mit Konstruktor-Funktion

```
1 angular.module("myModule").service("myService", function ($http) {
2   this.doSomething = function () { };
3 });
4 // service = new myService();
```

Module#factory mit Factory-Funktion

```
1 angular.module("myModule").factory("myService", function ($http) {
2   var service = {}; // var service = function () {};
3   service.doSomething = function () { };
4   return service;
5 });
6 // service = myService();
```

SERVICES IN ANGULARJS

MIT KLASSEN

Service in AngularJS mit ES6 Klasse

```
1  var http = Symbol();
2  class MyService {
3
4    constructor($http) {
5      this[http] = $http; // pseudo private
6    }
7
8    doSomething() {}
9  }
10 MyService.$inject = ["$http"];
11 angular.module("myModule").service("myService", MyService);
```

ARROW FUNCTIONS

- Kurzschreibweise für Funktionen
- `(param) => { statement; }` statt `function (param) { statement; }`
- Aber: kein eigener lexikalischer Kontext, this vom äußeren Kontext

Funktion mit gleichem Kontext

```
1  class Person {
2    constructor(name, friends) { this.name = name; this.friends = friends; }
3    printFriends() {
4      this.friends.forEach(friend =>
5        console.log(this.name + " knows " + friend));
6    }
7  }
```

ARROW FUNCTIONS

ANWENDUNGSFÄLLE

- Promise Handler
- Event Handler

Funktion mit gleichem Kontext

```
1  function TestCtrl ($rootScope, DataService) {
2    // no more var that = this;
3    this.data = []; this.lastEvent = undefined;
4    DataService.get().then(data => { this.data = data; });
5    $rootScope.$on('eventName', event => { this.lastEvent = event.name});
6  }
7  angular.module('myModule').controller('TestCtrl', TestCtrl)
```

NOCH MEHR DAZU ...

- Exploring ES6 von Axel Rauschmayer
- Deploying ES6 und Using ES6 today
- ES6 Featues (kurz und bündig)

Philipp Burgmer
burgmer@w11k.de

www.w11k.de
www.thecodecampus.de