



the**code**campus</> w1k



JavaScript von Morgen  
schon heute

- Ausgangssituation
- Setup um ES6 heute zu verwenden
- Ein paar ES6 Neuerungen

# ÜBER MICH

- Philipp Burgmer
  - Software-Entwickler, Trainer
  - Fokus: Frontend, Web-Technologien
  - [burgmer@w11k.de](mailto: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 verabschiedet
- 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-Bibliothek (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-Bibliothek 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
  - symbols, collections, iterators, promises, setImmediate, array generics 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
- Kann im Build oder im Browser passieren
- 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

DEMO

# ES6 FEATURES

UND WIE SIE UNSER LEBEN VEREINFACHEN

# PROMISES

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

## *Pyramid of Doom*

```
1 step1(function (value1) {  
2   step2(value1, function(value2) {  
3     step3(value2, function(value3) {  
4       // Do something with value3  
5     });  
6   });  
7 });
```

## *Flache Struktur mit Promises*

```
1 promisedStep1()  
2 .then(promisedStep2)  
3 .then(promisedStep3)  
4 .then(function (value3) {  
5   // Do something with value3  
6 })
```

# PROMISES

## KRIS KOWAL'S Q

- Weit verbreitete Bibliothek (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*

```
1 function asyncFunc() {  
2   var deferred = Q.defer();  
3   setTimeout(function () {  
4     deferred.resolve('result'); // or deferred.reject('error');  
5   }, 500);  
6  
7   return deferred.promise;  
8 }
```

### *Factory API of \$q in AngularJS*

```
1 function asyncFunc() {  
2   return $q(function(resolve, reject) {  
3     $timeout(function () { resolve(); /* or reject(); */ }, 500);  
4   })  
}
```

# PROMISES

## ES6

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

```
1 function asyncFunc() {  
2   return new Promise(function(resolve, reject) {  
3     setTimeout(function () {  
4       resolve(); // or reject();  
5     }, 500);  
6   })  
}
```



# KLASSEN

- Andere Schreibweise für existierende Funktionalität
- Kann fast vollständig in ES5 übersetzt werden
- Vererbung bei Transpiler schwierig

## *Klasse in ES6*

```
1 class Point {  
2   constructor(x, y) { this.x = x; this.y = y; }  
3   toString() { return `(${this.x}, ${this.y})`; }  
4 }  
5 var point = new Point();
```

## *Klasse übersetzt in ES5*

```
1 var Point = (function () {  
2   function Point(x, y) { this.x = x; this.y = y; }  
3   Point.prototype.toString = function () {  
4     return "(" + this.x + ", " + this.y + ")";  
5   };  
6   return Point;  
7 })();  
8 var point = new 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)
```

# MODULE

- Grundgedanke wie AMD und CommonJS, Unterschiede im Detail
- Dynamisches, on demand Laden von Code
- Ein Modul pro Datei, eine Datei pro Modul
- Struktur auf Datei Ebene
- Abhängigkeiten zwischen Dateien/Modulen ersichtlich
- Differenzieren zwischen Module-Loader und Package Manager

## *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
- Wird man nicht los, zumindest nicht ganz
- Zwei Möglichkeiten
  - Modul-System über Modul-System
  - AngularJS Module-System weitestgehend ignorieren



# 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](https://github.com/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 );
```

## 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](http://www.w11k.de)  
[www.thecodecampus.de](http://www.thecodecampus.de)