thecodecampus</>

WMK

ng1 zu ng2+ migrieren

TypeScript

Über mich

Philipp Burgmer
<> burgmer@w11k.de | @philippburgmer
Software-Entwickler, Trainer
Web-Technologien, Sicherheit
TypeScript, Angular </>

Über uns

W11K GmbH - The Web Engineers

<> Gegründet 2000
Auftrags-Entwicklung / Consulting
Web / Java
Esslingen / Siegburg </>

theCodeCampus.de - Weiter.Entwickeln.

<> Schulungen seit 2007
theCodeCampus seit 2013
Projekt-Kickoffs & Unterstützung im Projekt </>

Überblick

<> Historie & Motivation

<> Strategien

<> Schritt für Schritt

Historie AngularJS

<> **AngularJS = Angular 1 = ng1**

<> **Erstes Release: 2009 (IE8)**

- Mehr für Designer als für Entwickler

<> **Probleme**

- Performance bei großen Anwendungen
- Keine Struktur auf Code Ebene
- Schlechte Steuerung des Datenflusses

Historie Angular

<> **Angular = Angular 2+ = ng2 = ng**
- Kein Angular 3
- Aktuell: Angular 4.1

<> **Release 2.0 im September 2016**

<> **Moderne Konzepte umsetzten, aktuelle Standards nutzen**
- Natives Modul System
- Komponentenorientiertes Design
- Vereinheitlichtes Templating
- Deutliche Steigerung der Performance
- Verbessertes Tooling

Motivation

<> **Danach ist alles besser**

- Sauberer Code
- Super Performance
- Zukunftssicher
- Nie wieder `$scope.$apply()`

<> **Während dessen ist aber auch manches schlimmer**

Konzepte

<> **Grundlegendes gleich**
- SPA
- Data-Binding
- Dependency-Injection

<> **AngularJS → Angular 2+**
- Service & Factory → Service
- Filters → Pipes
- Directive - Template → Directive
- Directive + Template → Components
- Controllers → Components
- Module → NgModule
- Scope →

Strategien

Big Bang

<> **Anwendung neu schreiben**

<> **Fachliche Details bekannt**

- Umsetzung deutlich schneller als Erstentwicklung

<> **Angular 2+ Wissen benötigt**

- Keine Zeit für seichten Einstieg

Big Bang - Vorteile

<> Danach kein Legacy-Code mehr

<> Chance technische Schulden abzubauen

<> Kein zusätzlicher Aufwand für Schritweise Migration
- Upgrade & Downgrade
- Build

<> Danach hoch optimierte Anwendung

Big Bang - Nachteile

<> Oft zu aufwändig / zu teuer

<> Weiterentwicklung stockt

<> Paralelle Weiterentwicklung muss nachgezogen werden

Adaptieren

<> **Anwendung bleibt rein AngularJS**

<> **Konzepte und Best-Practises aufgreifen**

- Neuer Code mit neuen Konzeten
- Alter Code schrittweise angepasst
- z.B. Smart-vs-Dump-Components

<> **Little Bang am Ende**

- Umstellung von AngularJS auf Angular

Adaptieren - Vorteile

<> Schrittweise neue Konzepte lernen & einbauen

<> Weiterentwicklung paralell möglich

<> AngularJS kann auch sehr komfortabel sein
- TypeScript
- ES Module
- Komponenten

Adaptieren - Nachteile

<> **Tooling / Build**

<> **Kein echtes Angular 2+**

- Performance
- Features (z.B. Style-Encapsulation)

<> **Little Bang: sämtlichen Code anfassen**

Hybrid

<> **Anwendung bleibt AngularJS**
- Angular 2+ wird zusätzlich eingebunden

<> **Neuer Code wird für Angular geschrieben**

<> **Alter Code kann Schrittweise migriert werden**

<> **Komponenten und Services können gegenseitig verwendet werden (Upgrade und Downgrade)**

Hybrid - Vorteile

<> Schrittweise neue Konzepte lernen & einbauen

<> Weiterentwicklung profitiert sofort von Angular 2+

<> Alten Code Schritt für Schritt migrieren

<> Tiny Pop: Leichter Wechsel zu Angular am Ende

Hybrid - Nachteile

<> **Angular wird als Bibliothek eingebunden**
- Recht schwergewichtig (ohne Optimierungen des CLI)
- Für Mobile eher ungeeignet

<> **Tooling / Build**

<> **Verstehen von Upgrade und Downgrade**

<> **Nicht alles lässt sich up- bzw. downgraden**
- Pipes bzw. Filter
- Nicht-Komponenten-Direktiven

Die Wahrheit

liegt irgendwo dazwischen

Die Wahrheit

<> Erst Adaptieren

<> Dann Hybrid

<> Kurz vor Ende vielleicht doch Rest-Big-Bang

Schritt für Schritt

zur Angular Anwendung

Schritt 1

AngularJS aktualisieren

AngularJS aktualisieren

<> **Für Adaptieren**
- Neue Featues verwenden können
- Angular 2+ Konzepte aufgreifen können

<> **Für Hybrid**
- `UpgradeAdapter` setzt AngularJS 1.5.3 voraus

<> **Bugfixes**

<> **Performance**

- <> Animationen
- <> Feature-Module
- <> ngTouch

- <> Kein IE8 Support
- <> strict-DI
- <> ngModelOptions
- <> ngModel Validator Pipeline
- <> ngMessages
- <> One-Time-Bindings

AngularJS 1.4

<> **Keine neuen Features**

<> **Viele Detail-Verbesserungen**
- Teilweise Breaking-Changes
- Animationen, $http, jQuery, Cookies, ngMessages

<> **Performance: Expression-Parser, Compiling, Watching**

<> **Multi-Slot-Transclusion**

<> **Annäherung an Angular 2**

- `module.component` als Alternative zu `module.directive`
- Lifecycle-Hooks für Direktiven (ab 1.5.3)
  `$onInit()`, `$onChanges()`, `$onDestroy()` und `$postLink()`

- `ngModelOptions` können vererbt werden
- Nicht standardisierte Promise-Methoden `success` und `error` an Rückgabe von `$http` entfernt
- Breaking Change für Komponenten-Bindings: keine Zuweisungen mehr vor Controller-Instanziierung

Schritt 2

TypeScript einführen

TypeScript einführen

<> **Warum?**

- Typisierung
- Code-Strukturierung
- ES6+ Features

<> **Wie?**

- `*.js` zu `*.ts` umbenennen
- Tooling einbauen (z.B. WebPack)

ES-Module

<> **TypeScript + Module-Loader**

<> **Wichtiges Feature für große Anwendungen**

<> **Ein Einstiegspunkt, danach nur über import-Statements**
- Pro Software-Komponente eine Datei (ein Modul)
- Erleichtert Navigation im Code

<> **Bibliotheken müssen per NPM installiert werden**

AngularJS Module

<> **Module-System von AngularJS ist Schrott**
- Man kann aber nicht ohne

<> **Trotzdem: Feature-Module**
- Erleichtert Wechsel zu Angular 2+ Modulen später
- Ordner-Struktur = Feature-Struktur

<> **Angular-Module exportieren (eigene Datei)**

<> **Angular-Module importieren und über Name einbinden**
- AppModule bindet Feature-Module ein

Typen für AngularJS

<> **@types stellt Typisierung für JS Bibliotheken bereit**

- AngularJS in JS entwickelt, keine statische Typisierung
- Community stellt Declaration-Files bereit
- Declaration-Files beinhalten nur Typ-Informationen

<> **Abhängigkeiten in `package.json`**

- `angular` als `dependency`
- `@types/angular` als `devDependency`

Schritt 3

An Style-Guide anpassen

Style-Guide

<> **Versuchen sich an Style-Guide und Best Practices zu halten**

<> John Papa's Style-Guide
- Feature-Module und Ordner
- Direktiven mit isolated Scope
- Kein Zugriff auf fremde Daten (Controller-As)
- Routing
- ...

<> **Viele Angular 2+ Konzept und Best Practices bauen darauf auf**

ES6+ Features

<> **Service & Controller als Klassen**
- Service mit `module.service` registrieren
- Controller wie gewohnt

<> **ES6+ Features nutzen**
- `Promise` statt `$q`
- `for..of` statt `angular.forEach`
- `let` Variablen

Schritt 4

Angular 2+ einbinden

Bootstrapping

<> **AngularJS Bootstrapping**
- Automatisch über `ng-app` Attribute im Markup
- Manuell über `angular.bootstrap` Aufruf im Code

<> **Bei Verwendung von `UpgradeAdapter` nur manuelles Bootstrapping möglich**
- Angular 2+ Module anlegen
- Angular 2+ bootstrappen (zumindest fast)
- AngularJS bootstrappen

<> **Kompliziert, aber ein mal Copy & Paste**

Bootstrapping

```
@NgModule({
  imports: [BrowserModule, UpgradeModule]
})
export class AppModule {
  ngDoBootstrap() {}
}

platformBrowserDynamic().bootstrapModule(AppModule).then(platformRef => {
  const upgrade = platformRef.injector.get(UpgradeModule) as UpgradeModule;
  upgrade.bootstrap(document.body, ['app'], {strictDi: true});
});
```

Schritt 5

Service migrieren

Services migrieren

<> **Service schon als Klasse implementiert**
- Kleine Anpassungen für DI

<> **Registrierung ändern: Angular 2+ UND AngularJS Module**

<> **Downgrade wenn in AngularJS Code verwendet**

<> **Abhängigkeiten upgraden oder ersetzen**

Service downgraden

```
import {downgradeInjectable} from '@angular/upgrade/static';

angular.module('app')
.factory('MyService', downgradeInjectable(MyService));
```

```
@NgModule({
  providers: [
    MyService,
    otherServiceProvider
  ]
})
export class AppModule {}
```

Service upgraden

```
import {OpaqueToken} from "@angular/core";
export const OtherService = new OpaqueToken("OtherService");

export function otherServiceFactory(i: any) {
  return i.get('otherService');
}
export const otherServiceProvider = {
  provide: OtherService,
  useFactory: otherServiceFactory,
  deps: ['$injector']
};
```

```
export class MyService {
  constructor(@Inject(OtherService) private other) {}
}
```

Schritt 6

Komponenten migrieren

Komponenten migrieren

<> **Controller schon als Klasse implementiert**
- Kleine Anpassungen für DI
- Wird zu Komponenten-Klasse

<> **Template muss Angular 2+ Expression Language verwenden**

<> **Registrierung ändern: Angular 2+ UND AngularJS Module**
- Downgrade wenn in AngularJS Code verwendet
- `declaration` und `entryComponent` an Angular 2+ Module

<> **Verwendete Komponenten upgraden oder ersetzen**

Komponente downgraden

```
1  import {downgradeComponent} from "@angular/upgrade/static";
2
3  @Component({
4    selector: 'app-list',
5    template: '<div></div>'
6  })
7  export class ListComponent {}
8
9  angular.module('app')
10   .directive('list', downgradeComponent({
11     component: ListComponent
12   }) as angular.IDirectiveFactory);
```

Schritt 7

AngularJS entfernen

Routing

<> **Alle Komponenten auf Angular 2+ umgestellt**

<> **Routen sind nur noch sehr dünne Wrapper**

- Verweisen auf Komponente

<> **Einfach zu migrieren**

- UI-Router zu Angular Router

Bootstrapping

<> Nur noch Angular 2+ starten

<> Alle Import-Statements zu AngularJS entfernen

<> EntryComponents kontrollieren

Tooling

<> **Tooling auf Angular CLI umstellen**
- Von hoch spezialisiertem Build profitieren
- AOT Compile
- Test Ausführung

<> **Neues Projekt mit CLI anlegen**

<> **Projekte zusammenführen**

Zusammenfassung

<> **Migration mittlerweile sehr gut dokumentiert**
- angular.io / Upgarding from AngularJS mit PhoneCat Tutorial
- NG-Conf 2017 Vortrag von Asim Hussain mit Contacts App

<> **UpgradeAdapter erspart Boilerplate**
- Nicht gescheit dokumentiert

<> **Upgrade Pfad klar ersichtlich**
- Code nicht immer ganz einfach

<> **Nicht immer Upgrade zwingend**
- TypeScript, Module, RxJS
- gut ohne Angular 2+ möglich

Philipp Burgmer
burgmer@w11k.de
Twitter: @philippburgmer
GitHub: pburgmer

www.thecodecampus.de
@theCodeCampus