

Frontend Entwicklung mit AngularJS

Rene Behring & Damian Poddebniak

7. Januar 2015



INHALTSVERZEICHNIS

① Einführung

② Konzepte

③ Web-Services

④ Praktikum

EINFÜHRUNG IN JAVASCRIPT

GESCHICHTE

1995 Veröffentlicht

LiveScript von *Netscape*

Überprüfen von Formulardaten

Erstellen von dynamischem HTML (clientseitig)

1996 *JScript* von *Microsoft*

1997 Standardisierung (*ECMAScript*)

Zunehmende Verwendung

- Serverseite (*NodeJS*), Gnome-Shell, PDF Dokumente, ...

EIGENSCHAFTEN

Paradigmen

- Objektorientiert
- Prozedural
- Funktional

Eigenschaften

- “C-ähnliche” Syntax
- Dynamische Typisierung
- Duck-Typing

PROGRAMMIERUNG

JAVASCRIPT LIVE

- `'use strict';`
- Typisierung
- Klassen inkl. (`class` Keyword)
- Weitere Anpassungen an andere Programmiersprachen

EINFÜHRUNG IN ANGULARJS

WAS IST EIN FRONTEND

Aufgaben

- Schnittstelle zwischen Benutzer und Backend
- Ermöglicht Interaktion mit lokaler Anwendung, Server, ...
- Kommando-basiert (CLI), Grafisch (GUI), ...

Web-basierte GUI

- HTML, CSS, JavaScript, ...

GENERIERUNG EINER WEBSEITE

Serverseitige Interpolation

- Server bettet lokale Daten in eine HTML Vorlage ein
- Beantwortung einer Anfragen mit fertigem HTML Dokument

Clientseitige Interpolation

- Beantwortung einer Anfrage mit codierten Daten
- Client aktualisiert lokale Darstellung

≈ Fließender Übergang

WAS IST ANGULARJS?



“Superheroic JavaScript MVW Framework”

- Single Page Application (SPA) Framework
- Erleichtert Entwicklung von Web-Frontends

UNTERSCHIEDE ZU ANDEREN FRAMEWORKS

Client-side Templates

- Vollständige Web-Anwendungen ohne Daten
- Kein Server-seitiges zusammensetzen von HTML

Zwei getrennte Komponenten

- Entkopplung von Front- und Backend
- Basiert auf Kommunikation mit Web-Services

VOR- UND NACHTEILE

Vorteile

- Substitution der Web-Anwendung möglich
- Anbindung nativer Anwendungen durch Webservices
- Keine Aufwendung von Rechenzeit für Interpolation

Nachteile?

- Mehrere Komponenten notwendig
- Benötigt JavaScript und “aktuellen” Browser
- “Single Page” & Referenzierbarkeit

KONZEPTE

MODULE

- Aufteilung in Module (“Java packages”)
 - LoginModule
 - BlogModule
 - AdminModule
 - ...
- Vollständig und wiederverwendbar (auch in anderen Modulen)

Da im wesentlichen selber Aufbau \Rightarrow Fokus auf ein Modul

MODULE

Bestandteile

- HTML Templates (wenn nötig)
- JavaScript für Funktionalität
- Konfiguration (Adressen, Verknüpfungen, ...)

Zusätzlich...

- Direktiven, Filter, Services, ...

MODULE

```
1 // Create Module
2 var blogModule =
3     angular.module('demoApp.blog', // Name
4                     ['ngRoute']);   // Dependencies
5
6 // Configuration
7 blogModule.config([function ()
8 {
9 }]);
```

Listing 1: Blog Modul

ROUTING

Konfiguration

```
1 $routeProvider.when('/blog',  
2 {  
3     templateUrl: 'blog/blog.html',  
4     controller: 'BlogController as ctrl'  
5 });
```

Listing 2: Routing Einstellungen

Form von Adressen

- `http://localhost/index.html#/blog`
- HTML5 Modus ohne Hashtag

ROUTING

```
1 <html ng-app="demoApp">
2   <head>
3     <script src="angular.js"></script>
4   </head>
5
6   <body>
7     <div ng-view/>
8
9     <script src="app.js"></script>
10    <script src="blog/blog.js"></script>
11    <script src="blog/blog-controller.js"></script>
12  </body>
13 </html>
```

Listing 3: Startseite (index.html)

DIREKTIVEN

- Verschiedene Direktiven mitgeliefert
 - “Curly-braces” syntax `{{ }}`
 - `ng-bind`
 - `ng-model`
 - `ng-click`
 - ...
- Ändern das Verhalten von DOM Elementen

DIREKTIVEN

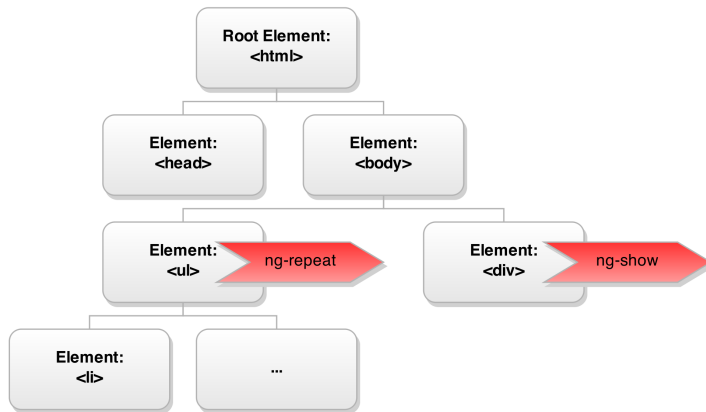


Abbildung: AngularJS Direktiven

DIREKTIVEN

```
1 <div ng-repeat="i in ctrl.list | filter: o">
2   <b>{{ i.title | uppercase}}</b>
3   <p>{{ i.content }}</p>
4 </div>
```

Listing 4: Nutzung von Direktiven

Implementierung eigener Direktiven

⇒ Einzige Stelle an der manuelle DOM Manipulation erlaubt ist

MODEL-VIEW-CONTROLLER

Model JavaScript Objekte und Properties
⇒ Single Source of Truth (SSOT)

View Spezielle HTML Templates
⇒ View wird aus Template generiert

Controller JavaScript Funktionen
⇒ Implementierte Funktionalität

MODEL-VIEW-CONTROLLER

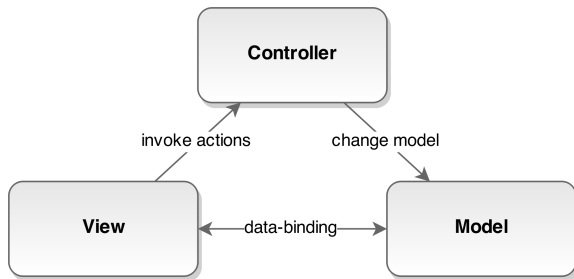


Abbildung: Umsetzung von MVC in AngularJS [4]

MODEL-VIEW-CONTROLLER

```
1 <div>
2     {{ ctrl.name }}s Blog
3     <input type="text" ng-model="ctrl.name"/>
4 </div>
```

Listing 5: Blog Template

```
1 demoApp.controller('BlogController', [function ()
2 {
3     this.name = "Hugo";
4 }]);
```

Listing 6: Blog Controller

MODEL-VIEW-CONTROLLER

Hugos Blog Hugo

Abbildung: Resultat im Browser

MODEL-VIEW-CONTROLLER

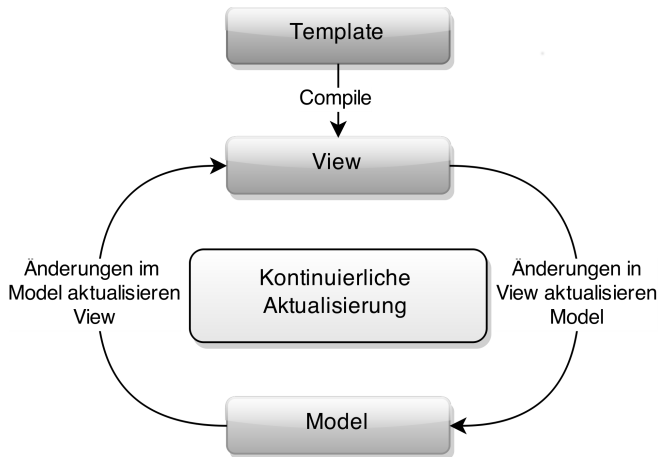


Abbildung: Two way data-binding [4]

MODEL-VIEW-CONTROLLER

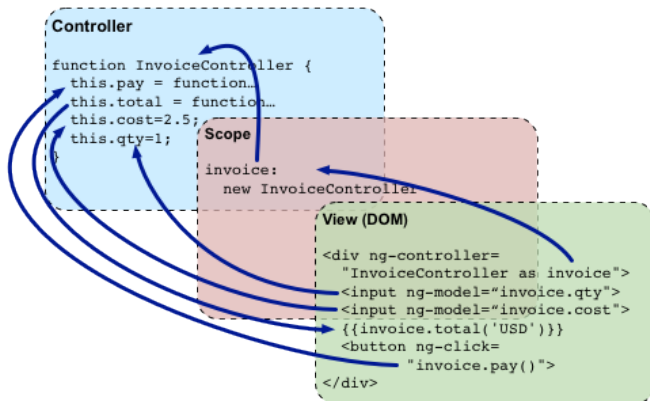


Abbildung: Verknüpfung zwischen Controller und View [4]

MODEL-VIEW-CONTROLLER

Vorteile

- Controller hat keine Referenz auf die View
- Automatische Aktualisierung durch Two way data-binding
- Ermöglicht isoliertes testen von Komponenten
- Keine DOM Manipulation

DEPENDENCY INJECTION

Prinzip

- “Frage” nach Objekten, anstatt diese anzulegen
- “Inversion of Control”

Vorteile

- Ermöglicht isoliertes Testen von Komponenten
- Einfache Nutzung von Mock-Objekten
- Einfacher Austausch von Funktionalität

DEPENDENCY INJECTION (VERALTETES BEISPIEL)

```
1  demoApp.controller('BlogController',
2      [
3          "$scope",
4
5          function ($scope)
6          {
7              $scope.name = "Hugo";
8          }
9      ]
10 );
```

Listing 7: Injection von \$scope

DEPENDENCY INJECTION (LOGGING SERVICE)

```
1  demoApp.controller('BlogController',
2      [
3          "$log",
4
5          function ($log)
6          {
7              $log.info("Initialisiere BlogController");
8          }
9      ]
10 );
```

Listing 8: Injection des \$log Services

WEB-SERVICES

SERVICES

- Stellen Daten Controller-Übergreifend zur Verfügung
- Können in jedem Controller injected werden
- Mehrere Arten von Services in AngularJS

SERVICES

```
1  demoApp.factory('MathService', function()
2  {
3      return
4      {
5          pi: function()
6          {
7              return Math.PI;
8          },
9          e: function()
10         {
11             return Math.E;
12         }
13     };
14 });
```

Listing 9: Service für mathematische Konstanten

ANGULARS \$HTTP SERVICE

```
1  $http.get('/students/' + '13')
2
3      .success(function(data,
4                      status,
5                      headers,
6                      config)
7      {
8          // onSuccess
9      })
10     .error(function(data,
11                    status,
12                    headers,
13                    config)
14     {
15         // onError
16     });
```

Listing 10: Anfrage über \$http

ANGULARS \$RESOURCE SERVICE

- Wrapper über Angulars \$http Service
- Erleichtert den Umgang mit RESTful Ressourcen
- Konfigurierbar

\$RESOURCE SERVICE

```
1  demoApp.factory('Blog', ['$resource',  
2    function ($resource)  
3    {  
4        return $resource  
5        (  
6            // parametrized URL  
7            'http://localhost/blog/:id',  
8  
9            // default values for parameters  
10           { "id": "@id" }  
11        );  
12    }]);
```

Listing 11: Anfrage über \$resource

ASYNCHRONE ABFRAGEN

```
1 var current = Student.get({"id": 12345});  
2  
3 $log.debug(current.firstName);
```

Listing 12: Anfrage über \$resource

Fallstricke

- Abfragen blockieren nicht - Programmfluss stoppt nicht
- Weiterverarbeitung der Daten wird vermutlich fehlschlagen

VERSPRECHEN: ABFRAGEN VON RESSOURCEN

```
1  Student.query().$promise.then(  
2      function (data)  
3      {  
4          /* onSuccess */  
5      },  
6      function (error)  
7      {  
8          /* onFailure */  
9      }  
10 );  
11  
12 Student.get({ "id": 12345 }).$promise.then(  
13     [...]  
14 );
```

Listing 13: Query & Get

VERSPRECHEN: ARBEITEN MIT RESSOURCEN

```
1  var item = new Student();
2
3  item.$save().then(
4      // callbacks
5  );
6
7  item.$update().then(
8      // callbacks
9  );
10
11 item.$remove().then(
12     // callbacks
13 );
```

Listing 14: Create, Update & Destroy

TESTEN MIT ANGULARJS

Test-driven development (TDD)


- Framework ist auf TDD ausgelegt
- Unit-Tests mit *Karma*
- End-To-End Tests mit *Protractor*

⇒ Wichtiger Bestandteil moderner Web-Anwendungen

Leider nicht Teil des Praktikums

- Benutzung weiterer Tools notwendig
- Lernen einer weiteren “Sprache”
- Zu viel Zeitaufwand für ein Praktikum

BÜCHER & DOKUMENTATION

-  Douglas Crockford. *JavaScript: The Good Parts - The Good Parts*. Sebastopol: O'Reilly Media, Inc., 2008. ISBN: 978-0-596-55487-3.
-  Brad Green und Shyam Seshadri. *AngularJS*. 1. Aufl. Sebastopol: O'Reilly Media, Inc., 2013. ISBN: 978-1-449-35588-3.
-  Brad Green und Shyam Seshadri. *AngularJS: Up and Running - Enhanced Productivity with Structured Web Apps*. 1. Aufl. Sebastopol: O'Reilly Media, Inc., 2014. ISBN: 978-1-491-90192-2.
-  *AngularJS - Superheroic JavaScript MVW Framework*. URL: <https://angularjs.org/> (besucht am 12.12.2014).

PRAKTIKUM

VIELEN DANK