Kostenloses eBook

LERNEN AngularJS

Free unaffiliated eBook created from
Stack Overflow contributors.

#angularjs

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angularjs

It is an unofficial and free AngularJS ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official AngularJS.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit AngularJS

Bemerkungen

AngularJS ist ein Webanwendungs-Framework, das die Entwicklung komplexer clientseitiger Anwendungen vereinfacht. Diese Dokumentation bezieht sich auf Angular 1.x , den Vorgänger des moderneren Angular 2 oder in der Stack Overflow-Dokumentation für Angular 2 .

Versionen

Ausführung	Veröffentlichungsdatum
1.6.5	2017-07-03
1.6.4	2017-03-31
1.6.3	2017-03-08
1.6.2	2017-02-07
1.5.11	2017-01-13
1.6.1	2016-12-23
1.5.10	2016-12-15
1.6.0	2016-12-08
1.6.0-rc.2	2016-11-24
1.5.9	2016-11-24
1.6.0-rc.1	2016-11-21
1.6.0-rc.0	2016-10-26
1.2.32	2016-10-11
1.4.13	2016-10-10
1.2.31	2016-10-10
1.5.8	2016-07-22
1.2.30	2016-07-21
1.5.7	2016-06-15

Ausführung	Veröffentlichungsdatum
1.4.12	2016-06-15
1.5.6	2016-05-27
1.4.11	2016-05-27
1.5.5	2016-04-18
1.5.4	2016-04-14
1.5.3	2016-03-25
1.5.2	2016-03-19
1.4.10	2016-03-16
1.5.1	2016-03-16
1.5.0	2016-02-05
1,5,0-rc,2	2016-01-28
1.4.9	2016-01-21
1.5.0-rc.1	2016-01-16
1,5,0-rc,0	2015-12-09
1.4.8	2015-11-20
1.5.0-beta.2	2015-11-18
1.4.7	2015-09-30
1.3.20	2015-09-30
1.2.29	2015-09-30
1.5.0-beta.1	2015-09-30
1.5.0-beta.0	2015-09-17
1.4.6	2015-09-17
1.3.19	2015-09-17
1.4.5	2015-08-28
1.3.18	2015-08-19

Ausführung	Veröffentlichungsdatum
1.4.4	2015-08-13
1.4.3	2015-07-15
1.3.17	2015-07-07
1.4.2	2015-07-07
1.4.1	2015-06-16
1.3.16	2015-06-06
1.4.0	2015-05-27
1.4.0-rc.2	2015-05-12
1.4.0-rc.1	2015-04-24
1.4.0-rc.0	2015-04-10
1.3.15	2015-03-17
1.4.0-beta.6	2015-03-17
1.4.0-beta.5	2015-02-24
1.3.14	2015-02-24
1.4.0-beta.4	09.02.2015
1.3.13	09.02.2015
1.3.12	2015-02-03
1.4.0-beta.3	2015-02-03
1.3.11	2015-01-27
1.4.0-beta.2	2015-01-27
1.4.0-beta.1	2015-01-20
1.3.10	2015-01-20
1.3.9	2015-01-15
1.4.0-beta.0	2015-01-14
1.3.8	2014-12-19

Ausführung	Veröffentlichungsdatum
1.2.28	2014-12-16
1.3.7	2014-12-15
1.3.6	2014-12-09
1.3.5	2014-12-02
1.3.4	2014-11-25
1.2.27	2014-11-21
1.3.3	2014-11-18
1.3.2	2014-11-07
1.3.1	2014-10-31
1.3.0	2014-10-14
1,3,0-rc,5	2014-10-09
1.2.26	2014-10-03
1.3.0-rc.4	2014-10-02
1.3.0-rc.3	2014-09-24
1.2.25	2014-09-17
1.3.0-rc.2	2014-09-17
1.2.24	2014-09-10
1.3.0-rc.1	2014-09-10
1,3,0-rc,0	2014-08-30
1.2.23	2014-08-23
1.3.0-beta.19	2014-08-23
1.2.22	2014-08-12
1.3.0-beta.18	2014-08-12
1.2.21	2014-07-25
1.3.0-beta.17	2014-07-25

Ausführung	Veröffentlichungsdatum
1.3.0-beta.16	2014-07-18
1.2.20	2014-07-11
1.3.0-beta.15	2014-07-11
1.2.19	2014-07-01
1.3.0-beta.14	2014-07-01
1.3.0-beta.13	2014-06-16
1.3.0-beta.12	2014-06-14
1.2.18	2014-06-14
1.3.0-beta.11	2014-06-06
1.2.17	2014-06-06
1.3.0-beta.10	2014-05-24
1.3.0-beta.9	2014-05-17
1.3.0-beta.8	2014-05-09
1.3.0-beta.7	2014-04-26
1.3.0-beta.6	2014-04-22
1.2.16	2014-04-04
1.3.0-beta.5	2014-04-04
1.3.0-beta.4	2014-03-28
1.2.15	2014-03-22
1.3.0-beta.3	2014-03-21
1.3.0-beta.2	2014-03-15
1.3.0-beta.1	2014-03-08
1.2.14	2014-03-01
1.2.13	2014-02-15
1.2.12	2014-02-08

Ausführung	Veröffentlichungsdatum
1.2.11	2014-02-03
1.2.10	2014-01-25
1.2.9	2014-01-15
1.2.8	2014-01-10
1.2.7	2014-01-03
1.2.6	2013-12-20
1.2.5	2013-12-13
1.2.4	2013-12-06
1.2.3	2013-11-27
1.2.2	2013-11-22
1.2.1	2013-11-15
1.2.0	2013-11-08
1,2,0-rc,3	2013-10-14
1,2,0-rc,2	2013-09-04
1,0,8	2013-08-22
1.2.0rc1	2013-08-13
1,0,7	2013-05-22
1.1.5	2013-05-22
1.0.6	2013-04-04
1.1.4	2013-04-04
1,0,5	2013-02-20
1.1.3	2013-02-20
1.0.4	2013-01-23
1.1.2	2013-01-23
1.1.1	2012-11-27

Ausführung	Veröffentlichungsdatum
1.0.3	2012-11-27
1.1.0	2012-09-04
1.0.2	2012-09-04
1.0.1	2012-06-25
1.0.0	2012-06-14
v1.0.0rc12	2012-06-12
v1.0.0rc11	2012-06-11
v1.0.0rc10	2012-05-24
v1.0.0rc9	2012-05-15
v1.0.0rc8	2012-05-07
v1.0.0rc7	2012-05-01
v1.0.0rc6	2012-04-21
v1.0.0rc5	2012-04-12
v1.0.0rc4	2012-04-05
v1.0.0rc3	2012-03-30
v1.0.0rc2	2012-03-21
g3-v1.0.0rc1	2012-03-14
g3-v1.0.0-rc2	2012-03-16
1.0.0rc1	2012-03-14
0.10.6	2012-01-17
0.10.5	2011-11-08
0.10.4	2011-10-23
0.10.3	2011-10-14
0.10.2	2011-10-08
0.10.1	2011-09-09

Ausführung	Veröffentlichungsdatum
0,10,0	2011-09-02
0,9,19	2011-08-21
0.9.18	2011-07-30
0,9,17	2011-06-30
0.9.16	2011-06-08
0,9,15	2011-04-12
0.9.14	2011-04-01
0,9,13	2011-03-14
0.9.12	2011-03-04
0,9.11	2011-02-09
0,9,10	2011-01-27
0,9,9	2011-01-14
0,9,7	2010-12-11
0,9,6	2010-12-07
0,9,5	2010-11-25
0.9.4	2010-11-19
0,9,3	2010-11-11
0,9,2	2010-11-03
0,9,1	2010-10-27
0,9,0	2010-10-21

Examples

Fertig machen

Erstellen Sie eine neue HTML-Datei und fügen Sie den folgenden Inhalt ein:

```
<!DOCTYPE html>
<html ng-app>
```

```
<head>
  <title>Hello, Angular</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
</head>
<body ng-init="name='World'">
  <label>Name</label>
  <input ng-model="name" />
  <span>Hello, {{ name }}!</span>
  <p ng-bind="name"></p>
</body>
</html>
```

Live-Demo

Wenn Sie die Datei mit einem Browser öffnen, wird ein Eingabefeld gefolgt von dem Text `Hello, World!` . Durch Bearbeiten des Werts in der Eingabe wird der Text in Echtzeit aktualisiert, ohne dass die gesamte Seite aktualisiert werden muss.

Erläuterung:

1. Laden Sie das Angular-Framework aus einem Content Delivery Network.

   ```
   <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
   ```

2. Definieren Sie das HTML-Dokument als Angular-Anwendung mit der Direktive `ng-app`

   ```
   <html ng-app>
   ```

3. Initialisieren der `name` Variable `ng-init`

   ```
   <body ng-init=" name = 'World' ">
   ```

   Beachten Sie, dass ng-init nur zu Demonstrations- und Testzwecken verwendet werden sollte. Beim Erstellen einer tatsächlichen Anwendung sollten Controller die Daten initialisieren.

4. Binden Sie Daten aus dem Modell an die Ansicht in HTML-Steuerelementen. Binden Sie mit `ng-model` eine `<input>` an die `name ng-model`

   ```
   <input ng-model="name" />
   ```

5. Anzeigen von Inhalten aus dem Modell mit doppelten geschweiften Klammern `{{ }}`

   ```
   <span>Hello, {{ name }}</span>
   ```

6. Eine andere Methode zum Binden der `name` Eigenschaft ist die Verwendung von `ng-bind` anstelle von Lenkern `"{{ }}"`

   ```
    <span ng-bind="name"></span>
   ```

Die letzten drei Schritte stellen die *bidirektionale Datenbindung her* . Durch Änderungen an der Eingabe wird das *Modell* aktualisiert, was in der *Ansicht angezeigt wird* .

Es gibt einen Unterschied zwischen der Verwendung von Lenker und `ng-bind` . Wenn Sie eine Lenkstange verwenden, wird möglicherweise das eigentliche `Hello, {{name}}` während die Seite geladen wird, bevor der Ausdruck aufgelöst wird (bevor die Daten geladen werden). Wenn Sie `ng-bind` , werden die Daten nur beim Namen angezeigt ist gelöst. Alternativ kann die Direktive `ng-cloak` verwendet werden, um zu verhindern, dass Lenker angezeigt werden, bevor er kompiliert wird.

Zeigt alle gängigen Winkelkonstrukte

Das folgende Beispiel zeigt gängige AngularJS-Konstrukte in einer Datei:

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <style>.started { background: gold; }</style>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>
      function MyDataService() {
        return {
          getWorlds: function getWorlds() {
            return ["this world", "another world"];
          }
        };
      }

      function DemoController(worldsService) {
        var vm = this;
        vm.messages = worldsService.getWorlds().map(function(w) {
          return "Hello, " + w + "!";
        });
      }

      function startup($rootScope, $window) {
        $window.alert("Hello, user! Loading worlds...");
        $rootScope.hasStarted = true;
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .service("worldsService", [MyDataService])
        .controller("demoController", ["worldsService", DemoController])
        .config(function() {
          console.log('configuring application');
        })
        .run(["$rootScope", "$window", startup]);
    </script>
  </head>
  <body ng-class="{ 'started': hasStarted }" ng-cloak>
    <div ng-controller="demoController as vm">
      <ul>
        <li ng-repeat="msg in vm.messages">{{ msg }}</li>
      </ul>
    </div>
  </body>
</html>
```

Jede Zeile der Datei wird unten erklärt:

Live Demo

1. `ng-app="myDemoApp"` , die ngApp-Direktive , die die Anwendung bootstraps und winklig angibt , dass ein DOM-Element von einem bestimmten `angular.module` Namen `"myDemoApp"` gesteuert wird.
2. `<script src="angular.min.js">` ist der erste Schritt beim Bootstrapping der AngularJS-Bibliothek .

Es werden drei Funktionen (`MyDataService` , `DemoController` und `startup`) deklariert, die im Folgenden verwendet (und erläutert) werden.

3. `angular.module(...)` das mit einem Array als zweites Argument verwendet wird, erstellt ein neues Modul. Dieses Array wird verwendet, um eine Liste von Modulabhängigkeiten bereitzustellen. In diesem Beispiel verketten wir Aufrufe des Ergebnisses der `module(...)` .

4. `.service(...)` erstellt einen Angular Service und `.service(...)` das Modul zur Verkettung zurück.

5. `.controller(...)` erstellt einen Angular Controller und `.controller(...)` das Modul zur Verkettung zurück;

6. `.config(...)` Verwenden Sie diese Methode, um die Arbeit zu registrieren, die beim Laden des Moduls ausgeführt werden muss.

7. `.run(...)` stellt sicher, dass der Code zur Startzeit ausgeführt wird und ein Array von Elementen als Parameter verwendet. Verwenden Sie diese Methode, um die Arbeit zu registrieren, die ausgeführt werden soll, wenn der Injektor alle Module geladen hat.

    - das erste Element ist die Vermietung Angular weiß , dass die `startup` erfordert den Einbau- `$rootScope` Dienst als Argument zu injizierenden;
    - Der zweite Punkt ist, dass Angular informiert wird, dass für die `startup` der integrierte `$window` Dienst als Argument eingefügt werden muss.
    - Das *letzte* Element im Array, `startup` , ist die eigentliche Funktion, die beim Start ausgeführt wird.

8. `ng-class` ist die ngClass-Direktive zum Festlegen einer dynamischen `class` . In diesem Beispiel wird `hasStarted` für `$rootScope` dynamisch verwendet

9. `ng-cloak` ist eine Direktive , die verhindert, dass die nicht gerenderte Angular-HTML-Vorlage (z. B. " `{{ msg }}` ") kurz angezeigt wird, bevor Angular die Anwendung vollständig geladen hat.

10. `ng-controller` ist die Anweisung , mit der Angular aufgefordert wird, einen neuen Controller mit einem bestimmten Namen zu instanziieren, um diesen Teil des DOM zu orchestrieren.

11. `ng-repeat` ist die Direktive, mit der Angular über eine Collection iteriert und eine DOM-Vorlage für jedes Element geklont wird.

12. `{{ msg }}` zeigt Interpolation : Rendern eines Teils des Bereichs oder Controllers vor Ort;

Die Bedeutung des Umfangs

Da Angular HTML verwendet, um eine Webseite zu erweitern und Javascript einfach um Logik zu erweitern, ist es einfach, eine Webseite mit **ng-app** , **ng-controller** und einigen integrierten Anweisungen wie **ng-if** , **ng-repeat** usw. zu erstellen Mit der neuen **controllerAs-** Syntax können Neulinge bei Angular-Benutzern Funktionen und Daten an ihren Controller anfügen, anstatt `$scope` .

Doch früher oder später, ist es wichtig zu verstehen , was genau diese `$scope` Sache. Es wird immer wieder in Beispielen auftauchen, daher ist es wichtig, etwas Verständnis zu haben.

Die gute Nachricht ist, dass es sich um ein einfaches, aber leistungsfähiges Konzept handelt.

Wenn Sie Folgendes erstellen:

```
<div ng-app="myApp">
 <h1>Hello {{ name }}</h1>
</div>
```

Wo wohnt der **Name** ?

Die Antwort ist, dass Angular ein `$rootScope` Objekt erstellt. Dies ist einfach ein `$rootScope` Javascript-Objekt, und **name** ist eine Eigenschaft für das `$rootScope` Objekt:

```
angular.module("myApp", [])
  .run(function($rootScope) {
    $rootScope.name = "World!";
  });
```

Und genau wie bei globalem Gültigkeitsbereich in Javascript ist es normalerweise keine gute Idee, dem globalen Gültigkeitsbereich oder `$rootScope` Elemente `$rootScope` .

Natürlich erstellen wir die meiste Zeit einen Controller und fügen unsere erforderliche Funktionalität in diesen Controller ein. Wenn wir jedoch einen Controller erstellen, macht Angular seine Magie und erstellt ein `$scope` Objekt für diesen Controller. Dies wird manchmal als **lokaler Bereich bezeichnet** .

So erstellen Sie den folgenden Controller:

```
<div ng-app="myApp">
  <div ng-controller="MyController">
    <h1>Hello {{ name }}</h1>
  </div>
</div>
```

würde den lokalen Bereich über den Parameter `$scope` zugänglich machen.

```
angular.module("myApp", [])
```

```
  .controller("MyController", function($scope) {
    $scope.name = "Mr Local!";
  });
```

Ein Controller ohne einen `$scope` Parameter benötigt ihn aus irgendeinem Grund möglicherweise nicht. Es ist jedoch wichtig zu wissen, dass der lokale Geltungsbereich **auch mit der controllerAs-Syntax** vorhanden ist.

Da `$scope` ein JavaScript-Objekt ist, wird es von Angular auf magische Weise so eingestellt, dass es von `$rootScope` prototypisch erbt. Und wie Sie sich vorstellen können, kann es eine Reihe von Bereichen geben. Sie können beispielsweise ein Modell in einem übergeordneten Controller erstellen und als `$scope.model` mit dem Bereich des übergeordneten Controllers `$scope.model` .

Über die Prototypkette kann ein `$scope.model` Controller mit `$scope.model` lokal auf dasselbe Modell `$scope.model` .

Nichts davon ist anfangs offensichtlich, da es nur Angular ist, das im Hintergrund seine Magie ausübt. Das Verständnis von `$scope` ist jedoch ein wichtiger Schritt, um zu erfahren, wie Angular funktioniert.

Die einfachste mögliche winklige Hallo Welt.

Angular 1 ist im Kern ein DOM-Compiler. Wir können HTML entweder als Vorlage oder einfach als normale Webseite übergeben und dann eine App kompilieren lassen.

Wir können Angular anweisen, einen Bereich der Seite als *Ausdruck* mit der `{{ }}` Lenkersymbolsyntax zu behandeln. Alles zwischen den geschweiften Klammern wird wie folgt zusammengestellt:

```
{{ 'Hello' + 'World' }}
```

Dies wird ausgegeben:

```
HelloWorld
```

ng-app

Wir teilen Angular mit, welcher Teil unseres DOMs als Master-Vorlage mit der `ng-app` *Direktive behandelt werden soll* . Eine Direktive ist ein benutzerdefiniertes Attribut oder Element, mit dem der Angular-Vorlagen-Compiler umgehen kann. Fügen wir jetzt eine ng-app-Direktive hinzu:

```
<html>
  <head>
    <script src="/angular.js"></script>
  </head>
  <body ng-app>
    {{ 'Hello' + 'World' }}
  </body>
</html>
```

Ich habe jetzt dem Body-Element gesagt, dass es die Root-Vorlage sein soll. Alles, was sich darin befindet, wird kompiliert.

Richtlinien

Direktiven sind Compiler-Direktiven. Sie erweitern die Fähigkeiten des Angular DOM-Compilers. Deshalb beschreibt **Misko** , der Schöpfer von Angular, Angular als:

> "Was wäre ein Webbrowser gewesen, wenn er für Webanwendungen erstellt worden wäre.

Wir erstellen buchstäblich neue HTML-Attribute und -Elemente und lassen diese von Angular in eine App kompilieren. `ng-app` ist eine Direktive, die den Compiler einfach aktiviert. Andere Richtlinien beinhalten:

- `ng-click` , wodurch ein Click-Handler hinzugefügt wird,
- `ng-hide` , das ein Element bedingt verbirgt, und
- `<form>` , wodurch einem Standard-HTML-Formularelement zusätzliches Verhalten hinzugefügt wird.

Angular verfügt über rund 100 integrierte Anweisungen, mit denen Sie die häufigsten Aufgaben ausführen können. Wir können auch eigene schreiben, und diese werden wie die eingebauten Richtlinien behandelt.

Wir erstellen eine Angular-App aus einer Reihe von Anweisungen, die mit HTML verbunden sind.

Minification in Angular

Was ist Minifizierung?

Dabei werden alle nicht benötigten Zeichen aus dem Quellcode entfernt, ohne die Funktionalität zu ändern.

Normale Syntax

Wenn wir zum Schreiben eines Controllers die normale Winkelsyntax verwenden, wird nach der Minimierung unserer Dateien unsere Funktionalität beeinträchtigt.

Controller (vor der Minifizierung):

```
var app = angular.module('mainApp', []);
app.controller('FirstController', function($scope) {
    $scope.name= 'Hello World !';
});
```

Nach der Verwendung des Minifikations-Tools wird es wie unten beschrieben minimiert.

```
var app=angular.module("mainApp",[]);app.controller("FirstController",function(e){e.name=
'Hello World !'})
```

Hier hat die Minifizierung unnötige Leerzeichen und die Variable $ scope aus dem Code entfernt. Wenn wir also diesen reduzierten Code verwenden, wird nichts gedruckt. Denn $ scope ist ein entscheidender Teil zwischen Controller und View, der nun durch die kleine 'e'-Variable ersetzt wird. Wenn Sie also die Anwendung ausführen, wird der Abhängigkeitsfehler des unbekannten Anbieters angezeigt.

Es gibt zwei Möglichkeiten, Ihren Code mit Informationen zum Servicenamen zu versehen, die für die Minifizierung sicher sind:

Inline-Anmerkungssyntax

```
var app = angular.module('mainApp', []);
app.controller('FirstController', ['$scope', function($scope) {
    $scope.message = 'Hello World !';
}]);
```

$ inject-Eigenschaftsanmerkungen-Syntax

```
FirstController.$inject = ['$scope'];
var FirstController = function($scope) {
    $scope.message = 'Hello World !';
}

var app = angular.module('mainApp', []);
app.controller('FirstController', FirstController);
```

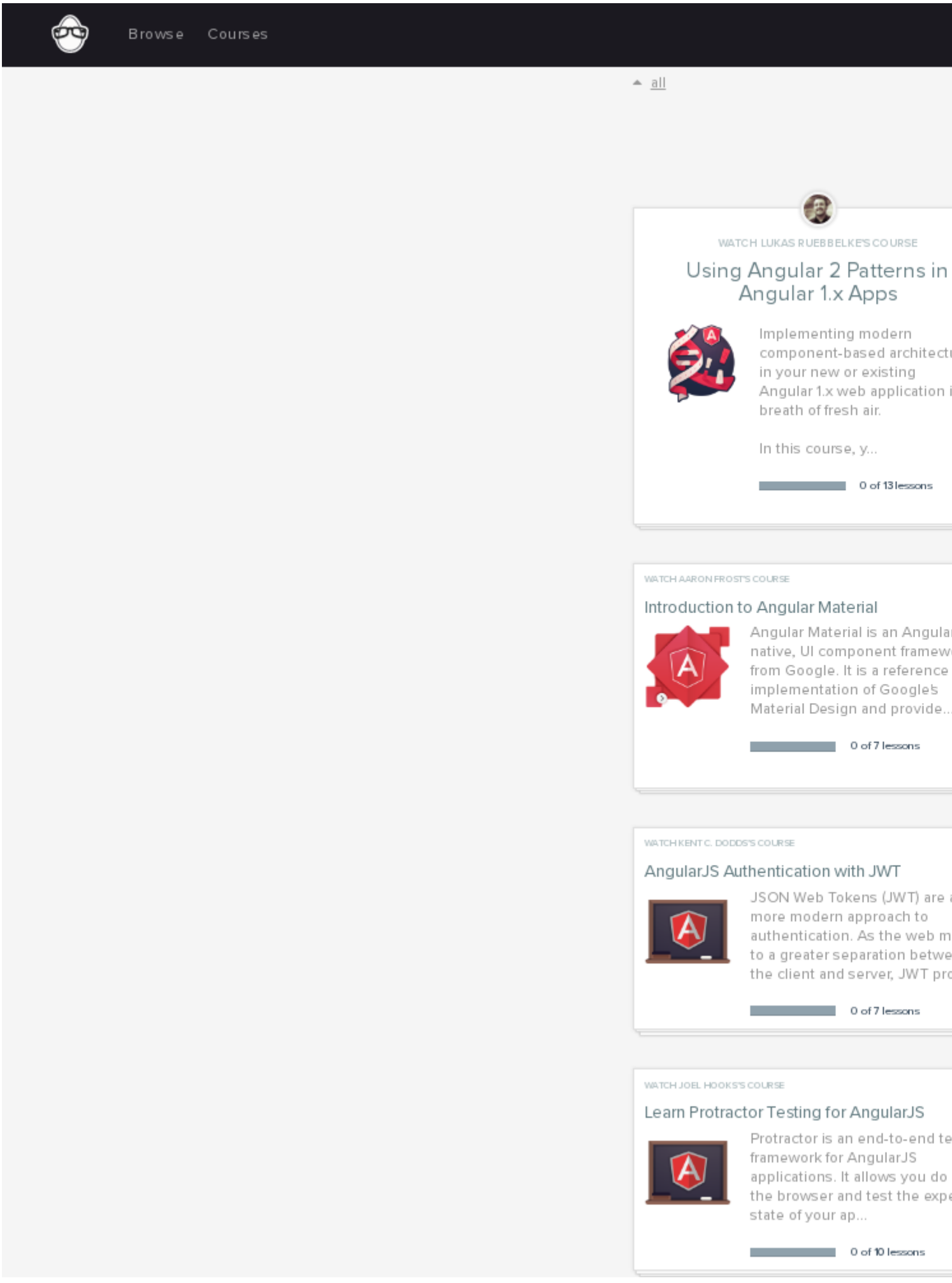
Nach der Minifizierung wird dieser Code sein

```
var
app=angular.module("mainApp",[]);app.controller("FirstController",["$scope",function(a){a.message="Hel
World !"}]);
```

In diesem Fall wird die Variable 'a' als 'scope' behandelt und die Ausgabe wird als 'Hello World!' Angezeigt.

AngularJS Erste Schritte Video-Tutorials

Es gibt viele gute Video-Tutorials für das AngularJS-Framework auf egghead.io



- https://egghead.io/courses/angularjs-app-from-scratch-getting-started

- https://egghead.io/courses/angularjs-application-architecture
- https://egghead.io/courses/angular-material-introduction
- https://egghead.io/courses/building-anangular-1-x-ionic-application
- https://egghead.io/courses/angular-and-webpack-für- modulare anwendungen
- https://egghead.io/courses/angularjs-authentication-with-jwt
- https://egghead.io/courses/angularjs-data-modeling
- https://egghead.io/courses/angular-automation-with-gulp
- https://egghead.io/courses/learn-protractor-testing-for-angularjs
- https://egghead.io/courses/ionic-quickstart-for-windows
- https://egghead.io/courses/build-angular-1-x-apps-with-redux
- https://egghead.io/courses/using-angular-2-patterns-in-angular-1-x-apps

Erste Schritte mit AngularJS online lesen: https://riptutorial.com/de/angularjs/topic/295/erste-schritte-mit-angularjs

Kapitel 2: $ http-Anfrage

Examples

$ Http in einem Controller verwenden

Der `$http` Dienst ist eine Funktion, die eine HTTP-Anforderung generiert und ein Versprechen zurückgibt.

Allgemeine Verwendung

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
  }, function errorCallback(response) {
    // called asynchronously if an error occurs
    // or server returns response with an error status.
  });
```

Verwendung im Controller

```
appName.controller('controllerName',
    ['$http', function($http){

    // Simple GET request example:
    $http({
        method: 'GET',
        url: '/someUrl'
    }).then(function successCallback(response) {
        // this callback will be called asynchronously
        // when the response is available
    }, function errorCallback(response) {
        // called asynchronously if an error occurs
        // or server returns response with an error status.
    });
}])
```

Verknüpfungsmethoden

`$http` Dienst verfügt auch über Verknüpfungsmethoden. Lesen Sie hier die http-Methoden

Syntax

```
$http.get('/someUrl', config).then(successCallback, errorCallback);
$http.post('/someUrl', data, config).then(successCallback, errorCallback);
```

Verknüpfungsmethoden

- $ http.get
- $ http.head
- $ http.post
- $ http.put
- $ http.delete
- $ http.jsonp
- $ http.patch

$ Http-Anfrage in einem Dienst verwenden

HTTP-Anforderungen werden in allen Webanwendungen häufig verwendet. Daher ist es ratsam, für jede allgemeine Anforderung eine Methode zu schreiben und sie an mehreren Stellen in der App zu verwenden.

Erstellen Sie eine `httpRequestsService.js`

httpRequestsService.js

```
appName.service('httpRequestsService', function($q, $http){

    return {
        // function that performs a basic get request
        getName: function(){
            // make sure $http is injected
            return $http.get("/someAPI/names")
                .then(function(response) {
                    // return the result as a promise
                    return response;
                }, function(response) {
                    // defer the promise
                    return $q.reject(response.data);
                });
        },

        // add functions for other requests made by your app
        addName: function(){
            // some code...
        }
    }
})
```

Der obige Dienst führt eine Abrufanforderung innerhalb des Dienstes aus. Dies steht allen Controllern zur Verfügung, in die der Service injiziert wurde.

Verwendungsbeispiel

```
appName.controller('controllerName',
    ['httpRequestsService', function(httpRequestsService){

        // we injected httpRequestsService service on this controller
        // that made the getName() function available to use.
        httpRequestsService.getName()
            .then(function(response){
                // success
```

```
            }, function(error){
                // do something with the error
            })
    }])
```

Mit diesem Ansatz können wir nun **httpRequestsService.js** jederzeit und in jedem Controller verwenden.

Zeitpunkt einer $ http-Anfrage

Die $ http-Anforderungen erfordern eine Zeit, die je nach Server unterschiedlich ist. Einige dauern einige Millisekunden, andere einige Sekunden. Oft ist die Zeit, die zum Abrufen der Daten aus einer Anforderung erforderlich ist, kritisch. Angenommen, der Antwortwert ist ein Array von Namen, betrachten Sie das folgende Beispiel:

Falsch

```
$scope.names = [];

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });

alert("The first name is: " + $scope.names[0]);
```

Beim Zugriff auf `$scope.names[0]` direkt unter der $ http-Anforderung wird häufig ein Fehler `$scope.names[0]` Diese Codezeile wird ausgeführt, bevor die Antwort vom Server empfangen wird.

Richtig

```
$scope.names = [];

$scope.$watch('names', function(newVal, oldVal) {
    if(!(newVal.length == 0)) {
        alert("The first name is: " + $scope.names[0]);
    }
});

$http({
    method: 'GET',
    url: '/someURL'
}).then(function successCallback(response) {
            $scope.names = response.data;
        },
        function errorCallback(response) {
            alert(response.status);
        });
```

Mit dem $ watch- Dienst greifen wir nur dann auf das Array `$scope.names` wenn die Antwort empfangen wird. Während der Initialisierung wird die Funktion auch dann aufgerufen, wenn `$scope.names` zuvor initialisiert wurde. Daher muss geprüft werden, ob `newVal.length` 0 `newVal.length` . Seien Sie sich bewusst - alle Änderungen, die Sie an `$scope.names` , lösen die `$scope.names` .

$ http-Anfrage online lesen: https://riptutorial.com/de/angularjs/topic/3620/--http-anfrage

Kapitel 3: Abhängigkeitsspritze

Syntax

- myApp.controller ('MyController', Funktion ($ scope) {...}); // nicht reduzierter Code
- myApp.controller ('MyController', ['$ scope', Funktion ($ scope) {...}]); // Minification unterstützen
- Funktion MyController () {}

  MyController. $ Inject = ['$ scope'];

  myApp.controller ('MyController', MyController); // $ Annotation einfügen
- $ injector.get ('injizierbar'); // dynamische / Laufzeitinjektion

Bemerkungen

Anbieter können nicht in `run` eingefügt werden.

Dienste oder Werte können nicht in `config` werden.

Achten Sie darauf, Ihre Injektionen mit Anmerkungen zu versehen, damit der Code bei der Minifizierung nicht beschädigt wird.

Examples

Injektionen

Das einfachste Beispiel für eine Injektion in einer Angular-App - wobei `$scope` in einen Angular `Controller` eingefügt wird:

```
angular.module('myModule', [])
.controller('myController', ['$scope', function($scope) {
    $scope.members = ['Alice', 'Bob'];
    ...
}])
```

Das Obige veranschaulicht eine Injektion eines `$scope` in einen `controller` , es ist jedoch dasselbe, ob Sie ein Modul in ein anderes injizieren. Der Prozess ist der gleiche.

Angulars System ist für die Auflösung von Abhängigkeiten für Sie zuständig. Wenn Sie beispielsweise einen Dienst erstellen, können Sie ihn wie im obigen Beispiel auflisten und für Sie verfügbar machen.

Sie können DI - Dependency Injection verwenden, wo immer Sie eine Komponente definieren.

Beachten Sie, dass wir im obigen Beispiel die sogenannte "Inline Array Annotation" verwenden. Das heißt, wir schreiben explizit als Namen die Namen unserer Abhängigkeiten. Wir tun dies, um zu verhindern, dass die Anwendung beschädigt wird, wenn der Code für die Produktion minimiert wird. Durch die Codeminifizierung werden die Namen der Variablen geändert (jedoch keine Zeichenfolgen), wodurch die Injektion unterbrochen wird. Durch die Verwendung von Strings weiß Angular, welche Abhängigkeiten wir wollen.

Sehr wichtig - die Reihenfolge der Zeichenfolgennamen muss mit den Parametern in der Funktion übereinstimmen.

Es gibt Tools, die diesen Prozess automatisieren und für Sie erledigen.

Dynamische Injektionen

Es gibt auch eine Option zum dynamischen Anfordern von Komponenten. Sie können dies mit dem `$injector` Dienst tun:

```
myModule.controller('myController', ['$injector', function($injector) {
    var myService = $injector.get('myService');
}]);
```

Hinweis: Diese Methode kann zwar verwendet werden, um das zirkuläre Abhängigkeitsproblem zu verhindern, durch das Ihre App möglicherweise beschädigt wird, es wird jedoch nicht als bewährte Methode angesehen, das Problem durch Verwendung zu umgehen. Eine kreisförmige Abhängigkeit weist normalerweise darauf hin, dass in der Architektur Ihrer Anwendung ein Fehler vorliegt, und Sie sollten stattdessen darauf eingehen.

$ inject Eigenschaftsanmerkung

Gleichermaßen können wir die `$inject` Eigenschaftsanmerkung verwenden, um dasselbe wie oben zu erreichen:

```
var MyController = function($scope) {
  // ...
}
MyController.$inject = ['$scope'];
myModule.controller('MyController', MyController);
```

Laden Sie den AngularJS-Dienst dynamisch in JavaScript

Sie können AngularJS-Dienste in Vanilla-JavaScript mithilfe der AngularJS `injector()` -Methode laden. Jedes Element jqLite abgerufen Aufruf `angular.element()` hat eine Methode `injector()` , die verwendet werden , kann der Injektor abzurufen.

```
var service;
var serviceName = 'myService';

var ngAppElement = angular.element(document.querySelector('[ng-app],[data-ng-app]') ||
document);
```

```
var injector = ngAppElement.injector();

if(injector && injector.has(serviceNameToInject)) {
    service = injector.get(serviceNameToInject);
}
```

Im obigen Beispiel versuchen wir, das jqLite-Element abzurufen, das die Wurzel der AngularJS-Anwendung (`ngAppElement`) enthält. Dazu verwenden wir die `angular.element()` -Methode, um nach einem DOM-Element zu suchen, das das Attribut " `ng-app` oder " `data-ng-app` . Wenn dies nicht der Fall ist, greifen wir auf das `document` . Wir verwenden `ngAppElement` , um die Injektorinstanz abzurufen (mit `ngAppElement.injector()`). Die Injector-Instanz wird verwendet, um zu prüfen, ob der zu injizierende Dienst vorhanden ist (mit `injector.has()`) und anschließend den Dienst (mit `injector.get()`) in der `service` Variablen zu laden.

Abhängigkeitsspritze online lesen:
https://riptutorial.com/de/angularjs/topic/1582/abhangigkeitsspritze

Kapitel 4: Anbieter

Syntax

- Konstante (Name, Wert);
- Wert (Name, Wert);
- factory (name, $ getFn);
- Dienst (Name, Konstruktor);
- Anbieter (Name, Anbieter);

Bemerkungen

Provider sind Einzelobjekte, die zum Beispiel in andere Dienste, Controller und Anweisungen eingefügt werden können. Alle Anbieter werden mit unterschiedlichen "Rezepten" registriert, wobei `Provider` der flexibelste ist. Alle möglichen Rezepte sind:

- Konstante
- Wert
- Fabrik
- Bedienung
- Anbieter

Dienste, Fabriken und Anbieter werden nur langsam initialisiert. Die Komponente wird nur dann initialisiert, wenn die Anwendung davon abhängig ist.

Dekorateure sind eng mit Anbietern verbunden. Dekorateure dienen dazu, Service- oder Fabrikerstellungen abzufangen, um ihr Verhalten zu ändern oder (Teile davon) zu überschreiben.

Examples

Konstante

`Constant` ist sowohl in der Konfigurations- als auch in der Ausführungsphase verfügbar.

```
angular.module('app',[])
  .constant('endpoint', 'http://some.rest.endpoint') // define
  .config(function(endpoint) {
    // do something with endpoint
    // available in both config- and run phases
  })
  .controller('MainCtrl', function(endpoint) {       // inject
    var vm = this;
    vm.endpoint = endpoint;                          // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
```

```
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

Endpunkt = http: //some.rest.endpoint

Wert

`Value` ist sowohl in der Konfigurations- als auch in der Ausführungsphase verfügbar.

```
angular.module('app',[])
  .value('endpoint', 'http://some.rest.endpoint') // define
  .run(function(endpoint) {
    // do something with endpoint
    // only available in run phase
  })
  .controller('MainCtrl', function(endpoint) {    // inject
    var vm = this;
    vm.endpoint = endpoint;                       // usage
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{ ::vm.endpoint }}</div>
</body>
```

Endpunkt = http: //some.rest.endpoint

Fabrik

`Factory` ist in der Run-Phase verfügbar.

> Das Factory-Rezept erstellt einen neuen Service unter Verwendung einer Funktion mit null oder mehr Argumenten (dies sind Abhängigkeiten von anderen Services). Der Rückgabewert dieser Funktion ist die Serviceinstanz, die mit diesem Rezept erstellt wurde.
>
> Factory kann einen Dienst eines beliebigen Typs erstellen, sei es ein Primitiv, ein Objektliteral, eine Funktion oder sogar eine Instanz eines benutzerdefinierten Typs.

```
angular.module('app',[])
  .factory('endpointFactory', function() {
    return {
      get: function() {
        return 'http://some.rest.endpoint';
      }
    };
  })
  .controller('MainCtrl', function(endpointFactory) {
    var vm = this;
    vm.endpoint = endpointFactory.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

Endpunkt = http: //some.rest.endpoint

Bedienung

`Service` ist in der Run-Phase verfügbar.

Das Service-Rezept erzeugt einen Service genauso wie die Value- oder Factory-Rezepte, jedoch durch *Aufruf eines Konstruktors mit dem neuen Operator* . Der Konstruktor kann null oder mehr Argumente annehmen, die die von der Instanz dieses Typs benötigten Abhängigkeiten darstellen.

```
angular.module('app',[])
  .service('endpointService', function() {
    this.get = function() {
      return 'http://some.rest.endpoint';
    };
  })
  .controller('MainCtrl', function(endpointService) {
    var vm = this;
    vm.endpoint = endpointService.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

Endpunkt = http: //some.rest.endpoint

Anbieter

`Provider` ist sowohl in der Konfigurations- als auch in der Ausführungsphase verfügbar.

Das Provider-Rezept ist syntaktisch als benutzerdefinierter Typ definiert, der eine `$get` Methode implementiert.

Sie sollten das Anbieterrezept nur verwenden, wenn Sie eine API für die anwendungsweite Konfiguration verfügbar machen möchten, die vor dem Start der Anwendung erstellt werden muss. Dies ist normalerweise nur für wiederverwendbare Dienste interessant, deren Verhalten zwischen den Anwendungen möglicherweise geringfügig variieren muss.

```
angular.module('app',[])
  .provider('endpointProvider', function() {
    var uri = 'n/a';
```

```
    this.set = function(value) {
      uri = value;
    };

    this.$get = function() {
      return {
        get: function() {
          return uri;
        }
      };
    };
  })
  .config(function(endpointProviderProvider) {
    endpointProviderProvider.set('http://some.rest.endpoint');
  })
  .controller('MainCtrl', function(endpointProvider) {
    var vm = this;
    vm.endpoint = endpointProvider.get();
  });
```

```
<body ng-controller="MainCtrl as vm">
  <div>endpoint = {{::vm.endpoint }}</div>
</body>
```

Endpunkt = http: //some.rest.endpoint

Ohne `config` wäre das Ergebnis

Endpunkt = n / a

Anbieter online lesen: https://riptutorial.com/de/angularjs/topic/5169/anbieter

Kapitel 5: Angular MVC

Einführung

In **AngularJS ist** das **MVC-** Muster in JavaScript und HTML implementiert. Die Ansicht ist in HTML definiert, während das Modell und der Controller in JavaScript implementiert sind. Es gibt verschiedene Möglichkeiten, diese Komponenten in AngularJS zusammenzusetzen, aber die einfachste Form beginnt mit der Ansicht.

Examples

Die statische Ansicht mit Controller

MVC Demo

Hallo Welt

Controller-Funktionsdefinition

```
var indexController = myApp.controller("indexController", function ($scope) {
    // Application logic goes here
});
```

Informationen zum Modell hinzufügen

```
var indexController = myApp.controller("indexController", function ($scope) {
    // controller logic goes here
    $scope.message = "Hello Hacking World"
});
```

Angular MVC online lesen: https://riptutorial.com/de/angularjs/topic/8667/angular-mvc

Kapitel 6: AngularJS Fallstiche und Fallen

Examples

Die bidirektionale Datenbindung funktioniert nicht mehr

Man sollte bedenken, dass:

1. Angulars Datenbindung beruht auf der prototypischen Vererbung von JavaScript und unterliegt daher variablen Spiegelungen .
2. Ein untergeordneter Bereich erbt normalerweise prototypisch von seinem übergeordneten Bereich. Eine Ausnahme von dieser Regel ist eine Direktive, die einen isolierten Geltungsbereich hat, da sie nicht prototypisch erbt.
3. Es gibt einige Anweisungen, die einen neuen untergeordneten Bereich erstellen: `ng-repeat` , `ng-switch` , `ng-view` , `ng-if` , `ng-controller` , `ng-include` usw.

Dies bedeutet, dass beim Versuch, Daten in zwei Richtungen an ein Primitiv zu binden, das sich in einem untergeordneten Bereich (oder umgekehrt) befindet, die Dinge möglicherweise nicht wie erwartet funktionieren. Hier ist ein Beispiel, wie einfach es ist, AngularJS zu *"brechen"* .

Dieses Problem kann leicht durch folgende Schritte vermieden werden:

1. Haben eine "." in Ihrer HTML-Vorlage, wenn Sie Daten binden
2. Verwenden Sie die `controllerAs` Syntax, da die Verwendung der Bindung an ein "gepunktetes" Objekt gefördert wird
3. $ parent kann verwendet werden, um auf übergeordnete `scope` statt auf untergeordneten `scope` zuzugreifen. wie in `ng-if` wir `ng-model="$parent.foo"` .

Eine Alternative für das Obige besteht darin, `ngModel` an eine Getter / Setter-Funktion zu binden, die die zwischengespeicherte Version des Modells aktualisiert, wenn sie mit Argumenten aufgerufen wird, oder sie zurückgeben, wenn sie ohne Argumente aufgerufen wird. Um eine Getter / Setter-Funktion zu verwenden, müssen Sie dem Element mit dem Attribut `ngModal ng-model-options="{ getterSetter: true }" ngModal` und die Getter-Funktion aufrufen, wenn Sie ihren Wert in expression anzeigen möchten (Arbeitsbeispiel).

Beispiel

Aussicht:

```
<div ng-app="myApp" ng-controller="MainCtrl">
    <input type="text" ng-model="foo" ng-model-options="{ getterSetter: true }">
    <div ng-if="truthyValue">
        <!-- I'm a child scope (inside ng-if), but i'm synced with changes from the outside
scope -->
        <input type="text" ng-model="foo">
    </div>
```

```
    <div>$scope.foo: {{ foo() }}</div>
</div>
```

Regler:

```
angular.module('myApp', []).controller('MainCtrl', ['$scope', function($scope) {
    $scope.truthyValue = true;

    var _foo = 'hello'; // this will be used to cache/represent the value of the 'foo' model

    $scope.foo = function(val) {
        // the function return the the internal '_foo' varibale when called with zero
arguments,
        // and update the internal `_foo` when called with an argument
        return arguments.length ? (_foo = val) : _foo;
    };
}]);
```

Bewährte Methode : Es ist am besten, Getter schnell zu halten, da Angular diese wahrscheinlich häufiger als andere Teile Ihres Codes (Referenz) nennt.

Dinge, die bei der Verwendung von html5Mode zu tun sind

Bei Verwendung von `html5Mode([mode])` ist `html5Mode([mode])` erforderlich:

1. Sie geben die Basis-URL für die Anwendung mit einem `<base href="">` im Kopf Ihrer `index.html` .
2. Es ist wichtig, dass das `base` Tag vor allen Tags mit URL-Anforderungen steht. Andernfalls kann dies zu diesem Fehler führen - `"Resource interpreted as stylesheet but transferred with MIME type text/html"` . Zum Beispiel:

   ```
   <head>
       <meta charset="utf-8">
       <title>Job Seeker</title>

       <base href="/">

       <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.css" />
       <link rel="stylesheet" href="/styles/main.css">
   </head>
   ```

3. Wenn Sie kein `base` Tag angeben möchten, konfigurieren Sie `$locationProvider` , dass kein `base` Tag erforderlich ist, indem Sie ein Definitionsobjekt mit `requireBase:false` an `$locationProvider.html5Mode()` wie `requireBase:false` :

   ```
   $locationProvider.html5Mode({
       enabled: true,
       requireBase: false
   });
   ```

4. Um das direkte Laden von HTML5-URLs zu unterstützen, müssen Sie das serverseitige

URL-Umschreiben aktivieren. Aus AngularJS / Developer Guide / Verwenden von $ location

> In diesem Modus müssen Sie die URL auf dem Server neu schreiben. Grundsätzlich müssen Sie alle Links zum Einstiegspunkt Ihrer Anwendung (z. B. `index.html`) neu schreiben. Das Erfordern eines `<base>` -Tags ist auch für diesen Fall wichtig, da Angular zwischen dem Teil der URL, der die Anwendungsbasis ist, und dem Pfad unterscheidet, der von der Anwendung verarbeitet werden soll.

Eine hervorragende Ressource für das Umschreiben von Beispielen für verschiedene HTTP-Serverimplementierungen finden Sie in den häufig gestellten Fragen - Anleitung: Vorgehensweise: Konfigurieren Sie Ihren Server für die Verwendung von html5Mode . Zum Beispiel Apache

```
RewriteEngine on

# Don't rewrite files or directories
RewriteCond %{REQUEST_FILENAME} -f [OR]
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^ - [L]

# Rewrite everything else to index.html to allow html5 state links
RewriteRule ^ index.html [L]
```

nginx

```
server {
    server_name my-app;

    root /path/to/app;

    location / {
        try_files $uri $uri/ /index.html;
    }
}
```

ausdrücken

```
var express = require('express');
var app = express();

app.use('/js', express.static(__dirname + '/js'));
app.use('/dist', express.static(__dirname + '/../dist'));
app.use('/css', express.static(__dirname + '/css'));
app.use('/partials', express.static(__dirname + '/partials'));

app.all('/*', function(req, res, next) {
    // Just send the index.html for other files to support HTML5Mode
    res.sendFile('index.html', { root: __dirname });
});

app.listen(3006); //the port you want to use
```

7 Todsünden von AngularJS

Nachfolgend finden Sie eine Liste einiger Fehler, die Entwickler häufig während der Verwendung von AngularJS-Funktionen machen, einige erlernte Lektionen und Lösungen für sie.

1. Manipulieren von DOM durch die Steuerung

Es ist legal, muss aber vermieden werden. Controller sind die Stellen, an denen Sie Ihre Abhängigkeiten definieren, Ihre Daten an die Sicht binden und weitere Geschäftslogik erstellen. Sie können das DOM in einem Controller technisch manipulieren. Wenn Sie jedoch dieselbe oder eine ähnliche Manipulation in einem anderen Teil Ihrer App benötigen, ist ein anderer Controller erforderlich. Die bewährte Methode dieses Ansatzes ist also das Erstellen einer Direktive, die alle Manipulationen und die Verwendung der Direktive in Ihrer App umfasst. Daher lässt der Controller die Ansicht intakt und erledigt ihre Aufgabe. In einer Direktive ist die Verknüpfungsfunktion der beste Ort, um das DOM zu bearbeiten. Sie hat vollen Zugriff auf den Geltungsbereich und das Element. Mit einer Direktive können Sie außerdem die Wiederverwendbarkeit nutzen.

```
link: function($scope, element, attrs) {
    //The best place to manipulate DOM
}
```

Sie können auf DOM-Elemente in der Verknüpfungsfunktion auf verschiedene Arten zugreifen, z. B. über den Parameter `element` , die Methode `angular.element()` oder reines Javascript.

2. Datenbindung in Transclusion

AngularJS ist bekannt für seine bidirektionale Datenbindung. Es kann jedoch vorkommen, dass Ihre Daten nur in einer Richtung in Anweisungen gebunden sind. Stoppen Sie dort, AngularJS ist nicht falsch, aber wahrscheinlich Sie. Richtlinien sind ein wenig gefährlicher Ort, da Kinder- und isolierte Bereiche betroffen sind. Angenommen, Sie haben die folgende Anweisung mit einer Transklusion

```
<my-dir>
    <my-transclusion>
    </my-transclusion>
</my-dir>
```

In my-transclusion haben Sie einige Elemente, die an die Daten im äußeren Bereich gebunden sind.

```
<my-dir>
    <my-transclusion>
        <input ng-model="name">
    </my-transclusion>
</my-dir>
```

Der obige Code wird nicht korrekt funktionieren. Hier wird durch Transklusion ein untergeordneter Bereich erstellt, und Sie können die Namensvariable richtig abrufen. Die Änderung an dieser Variablen bleibt jedoch erhalten. Sie können also wirklich auf diese Variable als **$ parent.name zugreifen** . Diese Verwendung ist jedoch möglicherweise nicht die beste Methode. Ein besserer Ansatz wäre das Umschließen der Variablen in ein Objekt. Zum Beispiel können Sie im Controller

Folgendes erstellen:

```
$scope.data = {
    name: 'someName'
}
```

Dann können Sie in der Transklusion über das Datenobjekt auf diese Variable zugreifen und sehen, dass die bidirektionale Bindung perfekt funktioniert!

```
<input ng-model="data.name">
```

Nicht nur bei Transclusions, sondern in der gesamten App ist es eine gute Idee, die Punktnotation zu verwenden.

3. Mehrere Anweisungen zusammen

Es ist eigentlich legal, zwei Anweisungen zusammen innerhalb desselben Elements zu verwenden, sofern Sie sich an die Regel halten: Zwei isolierte Bereiche können nicht auf demselben Element existieren. Im Allgemeinen weisen Sie beim Erstellen einer neuen benutzerdefinierten Direktive einen isolierten Bereich für die einfache Parameterübergabe zu. Wenn man davon ausgeht, dass die Direktiven myDirA und myDirB Geltungsbereiche isoliert haben und myDirC nicht, werden die folgenden Elemente gültig:

```
<input my-dir-a my-dirc>
```

Während das folgende Element einen Konsolenfehler verursacht:

```
<input my-dir-a my-dir-b>
```

Daher müssen Richtlinien weise verwendet werden, wobei die Anwendungsbereiche zu berücksichtigen sind.

4. Missbrauch von $ emit

$ emit, $ broadcast und $ on, diese arbeiten nach dem Sender-Empfänger-Prinzip. Mit anderen Worten, sie sind ein Kommunikationsmittel zwischen Steuerungen. Beispielsweise gibt die folgende Zeile das "someEvent" von Controller A aus, das vom betreffenden Controller B abgefangen werden soll.

```
$scope.$emit('someEvent', args);
```

Und die folgende Zeile fängt das "someEvent"

```
$scope.$on('someEvent', function(){});
```

Bisher scheint alles perfekt zu sein. Denken Sie jedoch daran, dass das Ereignis nicht erfasst wird, wenn der Controller B noch nicht aufgerufen wurde. Dies bedeutet, dass sowohl Sender- als auch Empfänger-Controller aufgerufen werden müssen, damit dies funktioniert. Wenn Sie sich

also nicht sicher sind, ob Sie $ emit unbedingt verwenden müssen, scheint der Aufbau eines Services der bessere Weg.

5. Missbrauch von $ scope. $ Watch

$ scope. $ watch wird zum Überwachen einer Variablenänderung verwendet. Immer wenn sich eine Variable geändert hat, wird diese Methode aufgerufen. Ein häufiger Fehler ist jedoch das Ändern der Variablen in $ scope. $ Watch. Dies führt zu Unstimmigkeiten und einer unendlichen $ Digest-Schleife.

```
$scope.$watch('myCtrl.myVariable', function(newVal) {
    this.myVariable++;
});
```

Stellen Sie in der obigen Funktion sicher, dass Sie keine Operationen mit myVariable und newVal ausführen.

6. Bindungsmethoden für Ansichten

Dies ist eine der tödlichsten Sünden. AngularJS hat eine bidirektionale Bindung. Wenn sich etwas ändert, werden die Ansichten viele Male aktualisiert. Wenn Sie also eine Methode an ein Attribut einer Sicht binden, wird diese Methode möglicherweise hundertmal aufgerufen, was Sie auch beim Debuggen verrückt macht. Es gibt jedoch nur einige Attribute, die für die Methodenbindung erstellt wurden, z. B. ng-click, ng-blur, ng-on-change usw., die Methoden als Parameter erwarten. Angenommen, Sie haben die folgende Ansicht in Ihrem Markup:

```
<input ng-disabled="myCtrl.isDisabled()" ng-model="myCtrl.name">
```

Hier überprüfen Sie den deaktivierten Status der Ansicht über die Methode isDisabled. In der Steuerung myCtrl haben Sie:

```
vm.isDisabled = function(){
    if(someCondition)
        return true;
    else
        return false;
}
```

Theoretisch mag es richtig erscheinen, technisch wird dies jedoch zu einer Überlastung führen, da die Methode unzählige Male ausgeführt wird. Um dies zu beheben, sollten Sie eine Variable binden. In Ihrem Controller muss die folgende Variable vorhanden sein:

```
vm.isDisabled
```

Sie können diese Variable bei der Aktivierung des Controllers erneut initiieren

```
if(someCondition)
    vm.isDisabled = true
else
    vm.isDisabled = false
```

Wenn die Bedingung nicht stabil ist, können Sie diese an ein anderes Ereignis binden. Dann sollten Sie diese Variable an die Sicht binden:

```
<input ng-disabled="myCtrl.isDisabled" ng-model="myCtrl.name">
```

Nun haben alle Attribute der Ansicht das, was sie erwarten, und die Methoden werden nur dann ausgeführt, wenn sie benötigt werden.

7. Angulars Funktionen nicht verwenden

AngularJS bietet einige Funktionalitäten, die den Code nicht nur vereinfachen, sondern auch effizienter machen. Einige dieser Funktionen sind unten aufgeführt:

1. **angle.forEach** für die Schleifen (Achtung, Sie können es nicht "brechen"; Sie können nur verhindern, dass Sie in den Körper gelangen, betrachten Sie hier die Leistung.)
2. **Winkelelement** für DOM-Selektoren
3. **angle.copy** : Verwenden Sie dies, wenn Sie das Hauptobjekt nicht ändern **möchten**
4. **Formularvalidierungen** sind bereits großartig. Verwenden Sie schmutzige, makellose, berührte, gültige, erforderliche und so weiter.
5. Verwenden Sie neben dem Chrome-Debugger auch das **Remote-Debugging** für die mobile Entwicklung.
6. Und stellen Sie sicher, dass Sie **Batarang verwenden** . Es ist eine kostenlose Chrome-Erweiterung, mit der Sie Bereiche einfach überprüfen können
   .

AngularJS Fallstiche und Fallen online lesen:
https://riptutorial.com/de/angularjs/topic/3208/angularjs-fallstiche-und-fallen

Kapitel 7: AngularJS-Bindungsoptionen (`=`, @, `&` usw.)

Bemerkungen

Verwenden Sie diesen Plunker , um mit Beispielen zu spielen.

Examples

@ einseitige Bindung, Attributbindung.

Übergeben Sie einen Literalwert (kein Objekt), z. B. eine Zeichenfolge oder eine Zahl.

Der untergeordnete Bereich erhält seinen eigenen Wert. Wenn er den Wert aktualisiert, hat der übergeordnete Bereich seinen eigenen alten Wert (der untergeordnete Bereich kann den Wert des Parens-Bereichs nicht ändern). Wenn der übergeordnete Bereichswert geändert wird, wird auch der untergeordnete Bereichswert geändert. Alle Interpolationen werden jedes Mal beim Digest-Aufruf angezeigt, nicht nur beim Erstellen von Anweisungen.

```
<one-way text="Simple text." <!-- 'Simple text.' -->
         simple-value="123" <!-- '123' Note, is actually a string object. -->
         interpolated-value="{{parentScopeValue}}" <!-- Some value from parent scope. You
can't change parent scope value, only child scope value. Note, is actually a string object. --
>
         interpolated-function-value="{{parentScopeFunction()}}" <!-- Executes parent scope
function and takes a value. -->

         <!-- Unexpected usage. -->
         object-item="{{objectItem}}" <!-- Converts object|date to string. Result might be:
'{"a":5,"b":"text"}'. -->
         function-item="{{parentScopeFunction}}"> <!-- Will be an empty string. -->
</one-way>
```

= wechselseitige Bindung.

Wenn Sie einen Wert als Referenz übergeben, möchten Sie den Wert für beide Bereiche freigeben und von beiden Bereichen aus bearbeiten. Sie sollten {{...}} nicht für die Interpolation verwenden.

```
<two-way text="'Simple text.'" <!-- 'Simple text.' -->
         simple-value="123" <!-- 123 Note, is actually a number now. -->
         interpolated-value="parentScopeValue" <!-- Some value from parent scope. You may
change it in one scope and have updated value in another. -->
         object-item="objectItem" <!-- Some object from parent scope. You may change object
properties in one scope and have updated properties in another. -->

         <!-- Unexpected usage. -->
         interpolated-function-value="parentScopeFunction()" <!-- Will raise an error. -->
```

```
         function-item="incrementInterpolated"> <!-- Pass the function by reference and you
may use it in child scope. -->
</two-way>
```

Das Übergeben einer Funktion als Referenz ist eine schlechte Idee: Um die Definition einer Funktion ändern zu können und zwei unnötige Watcher werden erstellt, müssen Sie die Anzahl der Beobachter minimieren.

& Funktionsbindung, Ausdrucksbindung.

Übergeben Sie eine Methode in eine Direktive. Es bietet eine Möglichkeit, einen Ausdruck im Kontext des übergeordneten Bereichs auszuführen. Die Methode wird im Geltungsbereich des übergeordneten Elements ausgeführt. Sie können dort einige Parameter aus dem untergeordneten Gültigkeitsbereich übergeben. Sie sollten {{...}} nicht für die Interpolation verwenden. Wenn Sie & in einer Direktive verwenden, wird eine Funktion generiert, die den Wert des Ausdrucks zurückgibt, der für den übergeordneten Bereich ausgewertet wurde (nicht dasselbe wie =, wenn Sie nur eine Referenz übergeben).

```
<expression-binding interpolated-function-value="incrementInterpolated(param)" <!--
interpolatedFunctionValue({param: 'Hey'}) will call passed function with an argument. -->
                    function-item="incrementInterpolated" <!-- functionItem({param: 'Hey'})()
will call passed function, but with no possibility set up a parameter. -->
                    text="'Simple text.'"  <!-- text() == 'Simple text.'-->
                    simple-value="123" <!-- simpleValue() == 123 -->
                    interpolated-value="parentScopeValue" <!-- interpolatedValue() == Some
value from parent scope. -->
                    object-item="objectItem"> <!-- objectItem() == Object item from parent
scope. -->
</expression-binding>
```

Alle Parameter werden in Funktionen eingebunden.

Verfügbare Bindung durch ein einfaches Muster

```
angular.component("SampleComponent", {
  bindings: {
     title: '@',
     movies: '<',
     reservation: "=",
     processReservation: "&"
  }
});
```

Hier haben wir alle verbindlichen Elemente.

@ zeigt an, dass wir eine sehr **grundlegende Bindung** benötigen, vom übergeordneten Bereich zum untergeordneten Bereich, ohne einen Beobachter, in irgendeiner Weise. Jede Aktualisierung im übergeordneten Bereich bleibt im übergeordneten Bereich, und keine Aktualisierung des untergeordneten Bereichs wird dem übergeordneten Bereich nicht mitgeteilt.

< zeigt eine **einseitige Bindung an** . Aktualisierungen im übergeordneten Bereich werden an den

untergeordneten Bereich weitergegeben, jedoch werden Aktualisierungen im untergeordneten Bereich nicht auf den übergeordneten Bereich angewendet.

= ist bereits als wechselseitige Bindung bekannt. Jede Aktualisierung des übergeordneten Bereichs wird auf die untergeordneten angewendet, und jede untergeordnete Aktualisierung wird auf den übergeordneten Bereich angewendet.

& wird jetzt für eine Ausgabebindung verwendet. Gemäß der Komponentendokumentation sollte sie verwendet werden, um auf die übergeordnete Bereichsmethode zu verweisen. Anstatt den untergeordneten Bereich zu ändern, rufen Sie einfach die übergeordnete Methode mit den aktualisierten Daten auf!

Binden Sie optionales Attribut

```
bindings: {
   mandatory: '='
   optional: '=?',
   foo: '=?bar'
}
```

Optionale Attribute sollten mit Fragezeichen markiert werden: `=?` oder `=?bar` . `($compile:nonassign)` ist ein Schutz für eine Ausnahme `($compile:nonassign)` .

AngularJS-Bindungsoptionen (`=`, @, `&` usw.) online lesen:
https://riptutorial.com/de/angularjs/topic/6149/angularjs-bindungsoptionen----------amp---usw--

Kapitel 8: Anweisungen, die ngModelController verwenden

Examples

Eine einfache Kontrolle: Bewertung

Lassen Sie uns ein einfaches Steuerelement, ein Bewertungs-Widget, erstellen, das wie folgt verwendet werden soll:

```
<rating min="0" max="5" nullifier="true" ng-model="data.rating"></rating>
```

Kein fancy CSS für jetzt; das würde als:

```
0 1 2 3 4 5 x
```

Durch Klicken auf eine Zahl wird diese Bewertung ausgewählt. Durch Klicken auf "x" wird die Bewertung auf null gesetzt.

```
app.directive('rating', function() {

    function RatingController() {
        this._ngModel = null;
        this.rating = null;
        this.options = null;
        this.min = typeof this.min === 'number' ? this.min : 1;
        this.max = typeof this.max === 'number' ? this.max : 5;
    }

    RatingController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 1
            ngModel.$render = this._render.bind(this);
        }
    };

    RatingController.prototype._render = function() {
        this.rating = this._ngModel.$viewValue != null ? this._ngModel.$viewValue : -
Number.MAX_VALUE;
    };

    RatingController.prototype._calculateOptions = function() {
        if( this.min == null || this.max == null ) {
            this.options = [];
        }
        else {
            this.options = new Array(this.max - this.min + 1);
            for( var i=0; i < this.options.length; i++ ) {
                this.options[i] = this.min + i;
```

```
            }
        }
    };

    RatingController.prototype.setValue = function(val) {
        this.rating = val;
        // KEY POINT 2
        this._ngModel.$setViewValue(val);
    };

    // KEY POINT 3
    Object.defineProperty(RatingController.prototype, 'min', {
        get: function() {
            return this._min;
        },
        set: function(val) {
            this._min = val;
            this._calculateOptions();
        }
    });

    Object.defineProperty(RatingController.prototype, 'max', {
        get: function() {
            return this._max;
        },
        set: function(val) {
            this._max = val;
            this._calculateOptions();
        }
    });

    return {
        restrict: 'E',
        scope: {
            // KEY POINT 3
            min: '<?',
            max: '<?',
            nullifier: '<?'
        },
        bindToController: true,
        controllerAs: 'ctrl',
        controller: RatingController,
        require: ['rating', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<span ng-repeat="o in ctrl.options" href="#" class="rating-option" ng-
class="{\'rating-option-active\': o <= ctrl.rating}" ng-click="ctrl.setValue(o)">{{ o
}}</span>' +
            '<span ng-if="ctrl.nullifier" ng-click="ctrl.setValue(null)" class="rating-
nullifier">&#10006;</span>'
    };
});
```

Schlüsselpunkte:

1. Implementieren Sie `ngModel.$render` , um den *Ansichtswert* des Modells in Ihre Ansicht zu übertragen.
2. Rufen Sie `ngModel.$setViewValue()` wenn Sie der Ansicht sind, dass der Ansichtswert

aktualisiert werden soll.
3. Die Steuerung kann natürlich parametriert werden; Verwenden Sie `'<'` Gültigkeitsbereichsbindungen für Parameter, wenn in Angular> = 1,5, um die Eingabe eindeutig anzugeben. Wenn Sie Maßnahmen ergreifen müssen, wenn sich ein Parameter ändert, können Sie eine JavaScript-Eigenschaft verwenden (siehe `Object.defineProperty()`), um einige Uhren zu speichern.

Hinweis 1: Um die Implementierung nicht zu `ctrl.options` , werden die Bewertungswerte in ein Array eingefügt - die `ctrl.options` . Dies ist nicht erforderlich. Eine effizientere, aber auch komplexere Implementierung könnte die DOM-Manipulation zum Einfügen / Entfernen von Bewertungen verwenden, wenn sich die `min` / `max` Änderung ändert.

Hinweis 2: Mit Ausnahme der Gültigkeitsbereichsbindungen `'<'` kann dieses Beispiel in Angular <1.5 verwendet werden. Wenn Sie sich auf Angular> = 1.5 befinden, sollten Sie dies in eine Komponente umwandeln und den Lebenszyklus-Hook `$onInit()` , um `min` und `max` zu initialisieren, anstatt dies im Konstruktor des Controllers zu tun.

Und eine notwendige Geige: https://jsfiddle.net/h81mgxma/

Einige komplexe Steuerelemente: Bearbeiten Sie ein vollständiges Objekt

Eine benutzerdefinierte Steuerung muss sich nicht auf banale Dinge wie Primitive beschränken. Es kann interessantere Dinge bearbeiten. Hier stellen wir zwei Arten von benutzerdefinierten Steuerelementen vor, eines zum Bearbeiten von Personen und eines zum Bearbeiten von Adressen. Die Adresssteuerung wird verwendet, um die Adresse der Person zu bearbeiten. Ein Beispiel für die Verwendung wäre:

```
<input-person ng-model="data.thePerson"></input-person>
<input-address ng-model="data.thePerson.address"></input-address>
```

Das Modell für dieses Beispiel ist bewusst vereinfachend:

```
function Person(data) {
  data = data || {};
  this.name = data.name;
  this.address = data.address ? new Address(data.address) : null;
}

function Address(data) {
  data = data || {};
  this.street = data.street;
  this.number = data.number;
}
```

Der Adresseditor:

```
app.directive('inputAddress', function() {

    InputAddressController.$inject = ['$scope'];
    function InputAddressController($scope) {
        this.$scope = $scope;
```

```
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }

    InputAddressController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            // KEY POINT 3
            ngModel.$render = this._render.bind(this);
        }
    };

    InputAddressController.prototype._makeWatch = function() {
        // KEY POINT 1
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Address(newval) : null);
                }
            }).bind(this)
        );
    };

    InputAddressController.prototype._render = function() {
        // KEY POINT 2
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Address(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
        controller: InputAddressController,
        require: ['inputAddress', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Street:</span><input type="text" ng-model="ctrl.value.street"
/></label>' +
                '<label><span>Number:</span><input type="text" ng-model="ctrl.value.number"
/></label>' +
            '</div>'
    };
});
```

Schlüsselpunkte:

1. Wir bearbeiten ein Objekt. Wir möchten das Objekt, das wir von unseren Eltern erhalten haben, nicht direkt ändern (wir möchten, dass unser Modell mit dem Unveränderlichkeitsprinzip kompatibel ist). Daher erstellen wir eine flache `$setViewValue()` für

das zu bearbeitende Objekt und aktualisieren das Modell mit `$setViewValue()` wenn sich eine Eigenschaft ändert. Wir geben eine *Kopie* an unsere Eltern weiter.
2. Wenn das Modell von außen geändert wird, kopieren wir es und speichern die Kopie in unserem Gültigkeitsbereich. Unveränderlichkeitsprinzipien wieder, obwohl die interne Kopie nicht unveränderlich ist, könnte die externe Kopie sehr wohl sein. Außerdem bauen wir die Uhr neu auf (`this_unwatch();this._makeWatch();`), um zu vermeiden, dass der Beobachter für Änderungen ausgelöst wird, die vom Modell an uns `this_unwatch();this._makeWatch();` . (Wir möchten nur, dass die Uhr für Änderungen an der Benutzeroberfläche ausgelöst wird.)
3. `ngModel.$render()` den oben genannten Punkten implementieren wir `ngModel.$render()` und rufen `ngModel.$setViewValue()` wie ein einfaches Steuerelement (siehe Bewertungsbeispiel).

Der Code für die Personensteuerungsfunktion ist nahezu identisch. Die Vorlage verwendet die `<input-address>` . In einer fortgeschritteneren Implementierung könnten wir die Controller in einem wiederverwendbaren Modul extrahieren.

```
app.directive('inputPerson', function() {

    InputPersonController.$inject = ['$scope'];
    function InputPersonController($scope) {
        this.$scope = $scope;
        this._ngModel = null;
        this.value = null;
        this._unwatch = angular.noop;
    }

    InputPersonController.prototype.setNgModel = function(ngModel) {
        this._ngModel = ngModel;

        if( ngModel ) {
            ngModel.$render = this._render.bind(this);
        }
    };

    InputPersonController.prototype._makeWatch = function() {
        this._unwatch = this.$scope.$watchCollection(
            (function() {
                return this.value;
            }).bind(this),
            (function(newval, oldval) {
                if( newval !== oldval ) { // skip the initial trigger
                    this._ngModel.$setViewValue(newval !== null ? new Person(newval) : null);
                }
            }).bind(this)
        );
    };

    InputPersonController.prototype._render = function() {
        this._unwatch();
        this.value = this._ngModel.$viewValue ? new Person(this._ngModel.$viewValue) : null;
        this._makeWatch();
    };

    return {
        restrict: 'E',
        scope: {},
        bindToController: true,
        controllerAs: 'ctrl',
```

```
        controller: InputPersonController,
        require: ['inputPerson', 'ngModel'],
        link: function(scope, elem, attrs, ctrls) {
            ctrls[0].setNgModel(ctrls[1]);
        },
        template:
            '<div>' +
                '<label><span>Name:</span><input type="text" ng-model="ctrl.value.name"
/></label>' +
                '<input-address ng-model="ctrl.value.address"></input-address>' +
            '</div>'
    };
});
```

Hinweis: Hier werden die Objekte typisiert, dh sie verfügen über korrekte Konstruktoren. Dies ist nicht obligatorisch. Das Modell kann ein einfaches JSON-Objekt sein. Verwenden `angular.copy()` in diesem Fall einfach `angular.copy()` anstelle der Konstruktoren. Ein zusätzlicher Vorteil ist, dass der Controller für die beiden Steuerungen identisch ist und leicht in ein gemeinsames Modul extrahiert werden kann.

Die Geige: https://jsfiddle.net/3tzyqfko/2/

Zwei Versionen der Geige haben den allgemeinen Code der Controller extrahiert: https://jsfiddle.net/agj4cp0e/ und https://jsfiddle.net/ugb6Lw8b/

Anweisungen, die ngModelController verwenden online lesen: https://riptutorial.com/de/angularjs/topic/2438/anweisungen--die-ngmodelcontroller-verwenden

Kapitel 9: Benutzerdefinierte Filter

Examples

Einfaches Filterbeispiel

Filter formatieren den Wert eines Ausdrucks zur Anzeige für den Benutzer. Sie können in Ansichtsvorlagen, Controllern oder Services verwendet werden. In diesem Beispiel wird ein Filter (`addZ`) erstellt und dann in einer Ansicht verwendet. Dieser Filter fügt am Ende der Zeichenfolge ein "Z" hinzu.

beispiel.js

```
angular.module('main', [])
    .filter('addZ', function() {
            return function(value) {
                return value + "Z";
            }
     })
    .controller('MyController', ['$scope', function($scope) {
        $scope.sample = "hello";
    }])
```

example.html

In der Ansicht wird der Filter mit der folgenden Syntax angewendet: `{ variable | filter}` . In diesem Fall wird die von uns im Controller definierte Variable `sample` durch den von uns erstellten Filter `addZ` .

```
<div ng-controller="MyController">
   <span>{{sample | addZ}}</span>
</div>
```

Erwartete Ausgabe

```
helloZ
```

Verwenden Sie einen Filter in einer Steuerung, einem Dienst oder einem Filter

Sie müssen `$filter` injizieren:

```
angular
  .module('filters', [])
  .filter('percentage', function($filter) {
    return function (input) {
      return $filter('number')(input * 100) + ' %';
```

```
    };
  });
```

Erstellen Sie einen Filter mit Parametern

Standardmäßig hat ein Filter einen einzigen Parameter: die Variable, auf die er angewendet wird. Sie können jedoch weitere Parameter an die Funktion übergeben:

```
angular
  .module('app', [])
  .controller('MyController', function($scope) {
    $scope.example = 0.098152;
  })
  .filter('percentage', function($filter) {
    return function (input, decimals) {
      return $filter('number')(input * 100, decimals) + ' %';
    };
  });
```

Jetzt können Sie eine Genauigkeit auf den geben `percentage`

```
<span ng-controller="MyController">{{ example | percentage: 2 }}</span>
=> "9.81 %"
```

... aber andere Parameter sind optional, Sie können jedoch den Standardfilter verwenden:

```
<span ng-controller="MyController">{{ example | percentage }}</span>
=> "9.8152 %"
```

Benutzerdefinierte Filter online lesen:
https://riptutorial.com/de/angularjs/topic/2552/benutzerdefinierte-filter

Kapitel 10: Benutzerdefinierte Filter mit ES6

Examples

FileSize-Filter mit ES6

Wir haben hier einen Dateigrößenfilter, um zu beschreiben, wie Sie einem vorhandenen Modul einen Costum-Filter hinzufügen:

```
let fileSize=function (size,unit,fixedDigit) {
return size.toFixed(fixedDigit) + ' '+unit;
};

  let fileSizeFilter=function () {
        return function (size) {
            if (isNaN(size))
                size = 0;

            if (size < 1024)
                return size + ' octets';

            size /= 1024;

            if (size < 1024)
               return fileSize(size,'Ko',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Mo',2);

            size /= 1024;

            if (size < 1024)
               return  fileSize(size,'Go',2);

            size /= 1024;
            return  fileSize(size,'To',2);
        };
    };
export default fileSizeFilter;
```

Der Filteraufruf in das Modul:

```
import fileSizeFilter from 'path...';
let myMainModule =
    angular.module('mainApp', [])
   .filter('fileSize', fileSizeFilter);
```

Der HTML-Code, in dem wir den Filter aufrufen:

```
<div ng-app="mainApp">
```

```
    <div>
        <input type="text" ng-model="size" />
    </div>
    <div>
        <h3>Output:</h3>
        <p>{{size| Filesize}}</p>
    </div>
</div>
```

Benutzerdefinierte Filter mit ES6 online lesen:
https://riptutorial.com/de/angularjs/topic/9421/benutzerdefinierte-filter-mit-es6

Kapitel 11: Benutzerdefinierte Richtlinien

Einführung

Hier erfahren Sie mehr über die Direktive-Funktion von AngularJS. Nachfolgend finden Sie Informationen zu Richtlinien und grundlegende und erweiterte Anwendungsbeispiele.

Parameter

Parameter	Einzelheiten
Umfang	Eigenschaft, um den Geltungsbereich der Richtlinie festzulegen. Es kann als falsch, wahr oder als isolierter Bereich festgelegt werden: {@, =, <, &}.
Geltungsbereich: Fälschung	Richtlinie verwendet den Geltungsbereich der Muttergesellschaft. Kein Bereich für Direktive erstellt.
Geltungsbereich: wahr	Die Richtlinie erbt den Elternbereich prototypisch als neuen untergeordneten Bereich. Wenn für dasselbe Element mehrere Anweisungen vorhanden sind, die einen neuen Gültigkeitsbereich anfordern, teilen sie sich einen neuen Gültigkeitsbereich.
Umfang: { @ }	Einwegbindung einer Direktive-Bereichseigenschaft an einen DOM-Attributwert. Da der Attributwert an das übergeordnete Element gebunden ist, ändert sich dieser im Direktive-Bereich.
Umfang: {=}	Bidirektionale Attributbindung, die das Attribut im übergeordneten Element ändert, wenn sich das Direktive-Attribut ändert und umgekehrt.
Umfang: {<}	Einwegbindung einer Bereichsrichtlinie und eines DOM-Attributausdrucks. Der Ausdruck wird im übergeordneten Element ausgewertet. Dadurch wird die Identität des übergeordneten Werts überwacht, sodass Änderungen an einer Objekteigenschaft im übergeordneten Wert nicht in der Direktive angezeigt werden. Änderungen an einer Objekteigenschaft in einer Direktive werden im übergeordneten Objekt angezeigt, da beide auf dasselbe Objekt verweisen
Umfang: { & }	Ermöglicht der Direktive, Daten an einen Ausdruck zu übergeben, der im übergeordneten Element ausgewertet werden soll.
kompilieren: Funktion	Diese Funktion wird verwendet, um die DOM-Umwandlung in der Direktivenvorlage auszuführen, bevor die Verknüpfungsfunktion ausgeführt wird. Es akzeptiert `tElement` (die Direktivenvorlage) und `tAttr` (Liste der in der Direktive deklarierten Attribute). Es hat keinen Zugriff auf

Parameter	Einzelheiten
	den Geltungsbereich. Es kann eine Funktion zurück , die als registriert wird `post-link` - Funktion oder es kann ein Objekt mit Rück `pre` und `post` - Eigenschaften mit wird als registriert seine `pre-link` und `post-link` - Funktionen.
Link: Funktion / Objekt	Die Link-Eigenschaft kann als Funktion oder Objekt konfiguriert werden. Es kann die folgenden Argumente empfangen: Bereich (Direktive-Bereich), iElement (DOM-Element, auf das die Direktive angewendet wird), iAttrs (Sammlung von DOM-Elementattributen), Controller (Array von Controllern, die von der Direktive benötigt werden), transcludeFn. Es wird hauptsächlich verwendet, um DOM-Listener einzurichten, Modelleigenschaften auf Änderungen zu prüfen und das DOM zu aktualisieren. Es wird ausgeführt, nachdem die Vorlage geklont wurde. Sie wird unabhängig konfiguriert, wenn keine Kompilierfunktion vorhanden ist.
Pre-Link-Funktion	Verknüpfungsfunktion, die vor untergeordneten Verknüpfungsfunktionen ausgeführt wird. Standardmäßig werden untergeordnete Direktive-Verknüpfungsfunktionen vor übergeordneten Direktive-Verknüpfungsfunktionen ausgeführt. Mit der Pre-Link-Funktion kann der übergeordnete Link zuerst verknüpft werden. Ein Anwendungsfall ist, wenn das Kind Daten vom Elternteil benötigt.
Post-Link-Funktion	Verknüpfungsfunktion, bei der Führungskräfte nach untergeordneten Elementen mit übergeordneten Elementen verknüpft werden. Sie wird häufig für das Anhängen von Ereignishandlern und für den Zugriff auf untergeordnete Direktiven verwendet. Die für die untergeordnete Direktive erforderlichen Daten sollten hier jedoch nicht festgelegt werden, da die untergeordnete Direktive bereits verknüpft wurde.
einschränken: string	Definiert, wie die Direktive aus dem DOM aufgerufen wird. Mögliche Werte (Angenommen, unser Direktionsname ist `demoDirective`): `E` - Elementname (`<demo-directive></demo-directive>`), `A` - Attribut (`<div demo-directive></div>`), `C` - Übereinstimmende Klasse (`<div class="demo-directive"></div>`), `M` - Durch Kommentar (`<!-- directive: demo-directive -->`). Die `restrict` kann auch mehrere Optionen unterstützen, z. B. - `restrict: "AC"` beschränkt die Anweisung auf *Attribut* ODER *Klasse* . Wenn nicht angegeben, ist der **Standardwert** `"EA"` (Element oder Attribut).
erfordern: 'demoDirective'	Suchen Sie den Controller von demoDirective auf dem aktuellen Element und fügen Sie den Controller als viertes Argument in die Verknüpfungsfunktion ein. Fehler melden, falls nicht gefunden.
erfordern: '? demoDirective'	Versuchen Sie, den Controller der demoDirective zu finden, oder übergeben Sie null an den Link fn, falls dieser nicht gefunden wird.

Parameter	Einzelheiten
erfordern: '^ demoDirective'	Suchen Sie den Controller von demoDirective, indem Sie das Element und seine übergeordneten Elemente durchsuchen. Fehler melden, falls nicht gefunden.
erfordern: '^^ demoDirective'	Suchen Sie den Controller von demoDirective, indem Sie die übergeordneten Elemente des Elements durchsuchen. Fehler melden, falls nicht gefunden.
erfordern: '? ^ demoDirective'	Versuchen Sie, den Controller der demoDirective zu finden, indem Sie das Element und seine übergeordneten Elemente durchsuchen oder den Link fn mit Null belegen, wenn er nicht gefunden wird.
erfordern: '? ^^ demoDirective'	Versuchen Sie, den Controller von demoDirective zu finden, indem Sie die übergeordneten Elemente des Elements durchsuchen, oder übergeben Sie null an den Link fn, falls dieser nicht gefunden wird.

Examples

Benutzerdefinierte Anweisungen erstellen und verwenden

Direktiven sind eine der leistungsstärksten Funktionen von anglejs. Benutzerdefinierte winkls-Direktiven werden verwendet, um die Funktionalität von HTML zu erweitern, indem neue HTML-Elemente oder benutzerdefinierte Attribute erstellt werden, um einem HTML-Tag ein bestimmtes Verhalten zu verleihen.

Direktive.js

```
// Create the App module if you haven't created it yet
var demoApp= angular.module("demoApp", []);

// If you already have the app module created, comment the above line and create a reference
of the app module
var demoApp = angular.module("demoApp");


// Create a directive using the below syntax
// Directives are used to extend the capabilities of html element
// You can either create it as an Element/Attribute/class
// We are creating a directive named demoDirective. Notice it is in CamelCase when we are
defining the directive just like ngModel
// This directive will be activated as soon as any this element is encountered in html

demoApp.directive('demoDirective', function () {

  // This returns a directive definition object
  // A directive definition object is a simple JavaScript object used for configuring the
directive’s behaviour,template..etc
  return {
    // restrict: 'AE', signifies that directive is Element/Attribute directive,
    // "E" is for element, "A" is for attribute, "C" is for class, and "M" is for comment.
```

```
    // Attributes are going to be the main ones as far as adding behaviors that get used the
most.
    // If you don't specify the restrict property it will default to "A"
    restrict :'AE',

    // The values of scope property decides how the actual scope is created and used inside a
directive. These values can be either “false”, “true” or “{}”. This creates an isolate scope
for the directive.
    // '@' binding is for passing strings. These strings support {{}} expressions for
interpolated values.
    // '=' binding is for two-way model binding. The model in parent scope is linked to the
model in the directive's isolated scope.
    // '&' binding is for passing a method into your directive's scope so that it can be
called within your directive.
    // The method is pre-bound to the directive's parent scope, and supports arguments.
    scope: {
      name: "@",  // Always use small casing here even if it's a mix of 2-3 words
    },

    // template replaces the complete element with its text.
    template: "<div>Hello {{name}}!</div>",

    // compile is called during application initialization. AngularJS calls it once when html
page is loaded.
    compile: function(element, attributes) {
      element.css("border", "1px solid #cccccc");

      // linkFunction is linked with each element with scope to get the element specific data.
      var linkFunction = function($scope, element, attributes) {
        element.html("Name: <b>"+$scope.name +"</b>");
        element.css("background-color", "#ff00ff");
      };
      return linkFunction;
    }
  };
});
```

Diese Direktive kann dann in App verwendet werden als:

```
<html>

   <head>
      <title>Angular JS Directives</title>
   </head>
   <body>
   <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
   <script src="directive.js"></script>
    <div ng-app = "demoApp">
       <!-- Notice we are using Spinal Casing here -->
       <demo-directive name="World"></demo-directive>

      </div>
   </body>
</html>
```

Richtliniendefinitionsobjektvorlage

```
demoApp.directive('demoDirective', function () {
  var directiveDefinitionObject = {
    multiElement:
    priority:
    terminal:
    scope: {},
    bindToController: {},
    controller:
    controllerAs:
    require:
    restrict:
    templateNamespace:
    template:
    templateUrl:
    transclude:
    compile:
    link: function(){}
  };
  return directiveDefinitionObject;
});
```

1. **`multiElement`** - auf true gesetzt, und alle DOM-Knoten zwischen Anfang und Ende des Direktivenamens werden gesammelt und als Direktiveelemente zusammengefasst
2. **`priority`** - ermöglicht die Angabe der Reihenfolge zum Anwenden von Anweisungen, wenn mehrere Anweisungen für ein einzelnes DOM-Element definiert sind. Richtlinien mit höheren Nummern werden zuerst erstellt.
3. **`terminal`** - auf true gesetzt, und die aktuelle Priorität ist die letzte auszuführende Direktive
4. **`scope`** - legt den Geltungsbereich der Richtlinie fest
5. **`bind to controller`** - bindet die Scope-Eigenschaften direkt an den Direktive Controller
6. **`controller`** Controller-Konstruktorfunktion
7. **`require`** - erfordert eine andere Direktive und fügt ihren Controller als viertes Argument der Verknüpfungsfunktion hinzu
8. **`controllerAs`** - Namensreferenz auf den Controller im Direktive-Bereich, damit der Controller aus der Direktive-Vorlage referenziert werden kann.
9. **`restrict`** - beschränkt die Anweisung auf Element, Attribut, Klasse oder Kommentar
10. **`templateNameSpace`** - Legt den von der Vorlage **`templateNameSpace`** verwendeten Dokumenttyp fest: html, svg oder math. html ist die Standardeinstellung
11. **`template`** - html-Markup, das standardmäßig den Inhalt des Direktive-Elements ersetzt oder den Inhalt des Direktive-Elements umschließt, wenn transclude den Wert true hat
12. **`templateUrl`** - URL, die asynchron für die Vorlage bereitgestellt wird
13. **`transclude`** - Extrahieren Sie den Inhalt des Elements, in dem die Direktive angezeigt wird, und stellen Sie es der Direktive zur Verfügung. Die Inhalte werden kompiliert und der Richtlinie als Transclusionsfunktion zur Verfügung gestellt.
14. **`compile`** - Funktion zum Umwandeln des Vorlagen-DOMs
15. **`link`** - Wird nur verwendet, wenn die Compile-Eigenschaft nicht definiert ist. Die Link-Funktion ist für die Registrierung der DOM-Listener sowie für die Aktualisierung des DOM verantwortlich. Sie wird ausgeführt, nachdem die Vorlage geklont wurde.

Beispiel für eine Grundrichtlinie

Superman-Direktive.js

```
angular.module('myApp', [])
  .directive('superman', function() {
    return {
      // restricts how the directive can be used
      restrict: 'E',
      templateUrl: 'superman-template.html',
      controller: function() {
        this.message = "I'm superman!"
      },
      controllerAs: 'supermanCtrl',
      // Executed after Angular's initialization. Use commonly
      // for adding event handlers and DOM manipulation
      link: function(scope, element, attributes) {
        element.on('click', function() {
          alert('I am superman!')
        });
      }
    }
  });
```

superman-template.html

```
<h2>{{supermanCtrl.message}}</h2>
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.0/angular.js"></script>
  <script src="superman-directive.js"><script/>
</head>
<body>
<div ng-app="myApp">
  <superman></superman>
</div>
</body>
</html>
```

Weitere Informationen zu den `restrict` und `link` Direktive finden Sie in der offiziellen Dokumentation von AngularJS zu Direktiven

So erstellen Sie eine wiederverwendbare Komponente mithilfe der Direktive

AngularJS-Direktiven steuern das Rendern von HTML in einer AngularJS-Anwendung. Sie können ein HTML-Element, ein Attribut, eine Klasse oder ein Kommentar sein. Direktiven werden verwendet, um das DOM zu bearbeiten, HTML-Elementen neues Verhalten zu verleihen, Daten zu binden und vieles mehr. Beispiele für Anweisungen, die von angle bereitgestellt werden, sind ng-model, ng-hide, ng-if.

Ebenso kann man seine eigene benutzerdefinierte Direktive erstellen und diese wieder herstellen. Zum Erstellen von benutzerdefinierten Anweisungen Referenz . Der Sinn beim Erstellen

wiederverwendbarer Direktiven besteht darin, eine Reihe von Direktiven / Komponenten zu erstellen, die von Ihnen geschrieben werden, so wie es von anglejs bei der Verwendung von angle.js bereitgestellt wird. Diese wiederverwendbaren Anweisungen können besonders hilfreich sein, wenn Sie über eine Suite von Anwendungen / Anwendungen verfügen, für die ein konsistentes Verhalten und Erscheinungsbild erforderlich ist. Ein Beispiel für eine solche wiederverwendbare Komponente kann eine einfache Symbolleiste sein, die Sie möglicherweise in Ihrer Anwendung oder in anderen Anwendungen verwenden möchten. Sie soll sich jedoch gleich verhalten oder gleich aussehen.

Erstellen Sie zunächst einen Ordner mit dem Namen resuableComponents in Ihrem App-Ordner, und erstellen Sie reusableModuleApp.js

reusableModuleApp.js:

```
(function(){

 var reusableModuleApp = angular.module('resuableModuleApp', ['ngSanitize']);

 //Remember whatever dependencies you have in here should be injected in the app module where
it is intended to be used or it's scripts should be included in your main app
                               //We will be injecting ng-sanitize

 resubaleModuleApp.directive('toolbar', toolbar)

 toolbar.$inject=['$sce'];

 function toolbar($sce){

     return{
     restrict :'AE',
        //Defining below isolate scope actually provides window for the directive to take data
from app that will be using this.
        scope : {
                value1: '=',
                value2: '=',
                },

         }
     template : '<ul>  <li><a ng-click="Add()" href="">{{value1}}</a></li>  <li><a ng-
click="Edit()" href="#">{{value2}}</a></li> </ul> ',
         link : function(scope, element, attrs){

               //Handle's Add function
               scope.Add = function(){


               };

               //Handle's Edit function
               scope.Edit = function(){

               };
     }
  }

});
```

mainApp.js:

```
(function(){
   var mainApp = angular.module('mainApp', ['reusableModuleApp']); //Inject resuableModuleApp
in your application where you want to use toolbar component

   mainApp.controller('mainAppController', function($scope){
      $scope.value1 = "Add";
      $scope.value2 = "Edit";

   });

 });
```

index.html:

```
 <!doctype html>
 <html ng-app="mainApp">
 <head>
  <title> Demo Making a reusable component
 <head>
  <body ng-controller="mainAppController">

     <!-- We are providing data to toolbar directive using mainApp'controller -->
     <toolbar value1="value1" value2="value2"></toolbar>

    <!-- We need to add the dependent js files on both apps here -->
     <script src="js/angular.js"></script>
     <script src="js/angular-sanitize.js"></script>

     <!-- your mainApp.js should be added afterwards --->
      <script src="mainApp.js"></script>

     <!-- Add your reusable component js files here -->
      <script src="resuableComponents/reusableModuleApp.js"></script>

  </body>
 </html>
```

Direktive sind standardmäßig wiederverwendbare Komponenten. Wenn Sie Anweisungen in einem separaten Winkelmodul erstellen, ist es tatsächlich exportierbar und für verschiedene Winkel-J-Anwendungen wiederverwendbar. Neue Anweisungen können einfach in reusableModuleApp.js hinzugefügt werden, und reusableModuleApp kann über einen eigenen Controller, eigene Services und ein DDO-Objekt in der Direktive verfügen, um das Verhalten zu definieren.

Grundlegende Direktive mit Vorlage und isoliertem Geltungsbereich

Durch das Erstellen einer benutzerdefinierten Direktive mit *isoliertem Bereich* wird der Gültigkeitsbereich **innerhalb** der Direktive von dem **externen** Gültigkeitsbereich getrennt, um zu verhindern, dass unsere Direktive die Daten im übergeordneten Bereich versehentlich ändert und verhindert, dass private Daten aus dem übergeordneten Bereich gelesen werden.

Um einen isolierten Bereich zu erstellen und immer noch unsere eigene Richtlinie ermöglichen ,

mit dem Außenumfang zu kommunizieren, können wir die Verwendung `scope` Option , die beschreibt , wie die Bindungen der Richtlinien inneren Umfangs **abzubilden** mit dem Außenumfang.

Die eigentlichen Bindungen werden mit zusätzlichen **Attributen erstellt,** die an die Direktive angehängt werden. Die Bindungseinstellungen werden mit der `scope` und einem Objekt mit Schlüssel-Wert-Paaren definiert:

- Ein **Schlüssel** , der dem isolierten Geltungsbereich unserer Richtlinie entsprach
- Ein **Wert** , der Angular mitteilt, wie der innere Gültigkeitsbereich der Anweisung an ein übereinstimmendes **Attribut** gebunden **wird**

Einfaches Beispiel für eine Direktive mit isoliertem Geltungsbereich:

```
var ProgressBar = function() {
  return {
    scope: { // This is how we define an isolated scope
      current: '=', // Create a REQUIRED bidirectional binding by using the 'current'
attribute
      full: '=?maxValue' // Create an OPTIONAL (Note the '?'): bidirectional binding using
'max-value' attribute to the `full` property in our directive isolated scope
    }
    template: '<div class="progress-back">' +
              '  <div class="progress-bar"' +
              '       ng-style="{width: getProgress()}">' +
              '  </div>' +
              '</div>',
    link: function(scope, el, attrs) {
      if (scope.full === undefined) {
        scope.full = 100;
      }
      scope.getProgress = function() {
        return (scope.current / scope.size * 100) + '%';
      }
    }
  }
}

ProgressBar.$inject = [];
angular.module('app').directive('progressBar', ProgressBar);
```

Beispiel für die Verwendung dieser Direktive und das Binden von Daten aus dem Controller-Bereich an den internen Geltungsbereich der Direktive:

Regler:

```
angular.module('app').controller('myCtrl', function($scope) {
    $scope.currentProgressValue = 39;
    $scope.maxProgressBarValue = 50;
});
```

Aussicht:

```
<div ng-controller="myCtrl">
    <progress-bar current="currentProgressValue"></progress-bar>
    <progress-bar current="currentProgressValue" max-value="maxProgressBarValue"></progress-
bar>
</div>
```

Eine wiederverwendbare Komponente erstellen

Mithilfe von Direktiven können wiederverwendbare Komponenten erstellt werden. Hier ist ein Beispiel einer "User Box" Komponente:

userBox.js

```
angular.module('simpleDirective', []).directive('userBox', function() {
  return {
    scope: {
        username: '=username',
        reputation: '=reputation'
    },
    templateUrl: '/path/to/app/directives/user-box.html'
  };
});
```

Controller.js

```
var myApp = angular.module('myApp', ['simpleDirective']);

myApp.controller('Controller', function($scope) {

    $scope.user = "John Doe";
    $scope.rep = 1250;

    $scope.user2 = "Andrew";
    $scope.rep2 = 2850;

});
```

myPage.js

```
<html lang="en" ng-app="myApp">
  <head>
    <script src="/path/to/app/angular.min.js"></script>
    <script src="/path/to/app/js/controllers/Controller.js"></script>
    <script src="/path/to/app/js/directives/userBox.js"></script>
  </head>

  <body>

    <div ng-controller="Controller">
        <user-box username="user" reputation="rep"></user-box>
        <user-box username="user2" reputation="rep2"></user-box>
    </div>

  </body>
</html>
```

user-box.html

```
<div>{{username}}</div>
<div>{{reputation}} reputation</div>
```

Das Ergebnis wird sein:

```
John Doe
1250 reputation
Andrew
2850 reputation
```

Direktive Dekorateur

In manchen Fällen benötigen Sie möglicherweise zusätzliche Funktionen aus einer Direktive. Anstatt die Direktive neu zu schreiben (zu kopieren), können Sie das Verhalten der Direktive ändern.

Der Dekorator wird während der $ Injektionsphase ausgeführt.

Stellen Sie dazu eine .config für Ihr Modul bereit. Die Direktive heißt myDirective, daher müssen Sie myDirectiveDirective konfigurieren. (dies in einer Winkelkonvention [über Anbieter lesen]).

In diesem Beispiel wird die templateUrl der Direktive geändert:

```
angular.module('myApp').config(function($provide){
        $provide.decorator('myDirectiveDirective', function($delegate){
             var directive = $delegate[0]; // this is the actual delegated, your directive
             directive.templateUrl = 'newTemplate.html'; // you change the directive template
        return $delegate;
    })
});
```

In diesem Beispiel fügen Sie dem Direktive-Element ein OnClick-Ereignis hinzu, wenn Sie darauf klicken. Dies geschieht während der Kompilierungsphase.

```
angular.module('myApp').config(function ($provide) {
        $provide.decorator('myDirectiveTwoDirective', function ($delegate) {
            var directive = $delegate[0];
            var link = directive.link; // this is directive link phase
            directive.compile = function () { // change the compile of that directive
                return function (scope, element, attrs) {
                    link.apply(this, arguments); // apply this at the link phase
                    element.on('click', function(){ // when add an onclick that log hello when
the directive is clicked.
                            console.log('hello!');
                    });
                };
            };
            return $delegate;
        });

    });
```

Ein ähnlicher Ansatz kann sowohl für Anbieter als auch für Dienste verwendet werden.

Vererbung und Interoperabilität der Richtlinie

Angular js-Direktiven können verschachtelt oder interoperabel gemacht werden.

In diesem Beispiel macht die Direktive Adir der Direktive Bdir den Controller $ scope aus, da Bdir Adir benötigt.

```
angular.module('myApp',[]).directive('Adir', function () {
        return {
            restrict: 'AE',
            controller: ['$scope', function ($scope) {
                    $scope.logFn = function (val) {
                            console.log(val);
                        }
                  }]
            }
    })
```

Stellen Sie sicher, dass Require gesetzt ist: '^ Adir' (sehen Sie sich die Winkeldokumentation an, einige Versionen benötigen kein ^ Zeichen).

```
.directive('Bdir', function () {
        return {
            restrict: 'AE',
            require: '^Adir', // Bdir require Adir
            link: function (scope, elem, attr, Parent) {
            // Parent is Adir but can be an array of required directives.
                elem.on('click', function ($event) {
                    Parent.logFn("Hello!"); // will log "Hello! at parent dir scope
                    scope.$apply(); // apply to parent scope.
                });
            }
        }
    }]);
```

Sie können Ihre Direktive auf diese Weise verschachteln:

```
<div a-dir><span b-dir></span></div>
<a-dir><b-dir></b-dir> </a-dir>
```

Es ist nicht erforderlich, dass Anweisungen in Ihrem HTML-Code verschachtelt sind.

Benutzerdefinierte Richtlinien online lesen:
https://riptutorial.com/de/angularjs/topic/965/benutzerdefinierte-richtlinien

Kapitel 12: Bereiten Sie sich auf die Produktion vor - Grunt

Examples

Vorladen anzeigen

Wenn die erste Ansicht angefordert wird, fordert Angular normalerweise eine `XHR` Anforderung an, um diese Ansicht zu erhalten. Für mittelgroße Projekte kann die Anzahl der Ansichten erheblich sein und die Reaktionsfähigkeit der Anwendung verlangsamen.

Es **empfiehlt sich,** alle Ansichten gleichzeitig für kleine und mittlere Projekte **vorab zu laden** . Bei größeren Projekten ist es sinnvoll, sie auch in bedeutungsvollen Bänden zusammenzufassen, aber andere Methoden können hilfreich sein, um die Last aufzuteilen. Um diese Aufgabe zu automatisieren, ist es praktisch, Grunt- oder Gulp-Aufgaben zu verwenden.

Um die Ansichten vorab zu laden, können wir das `$templateCache` Objekt verwenden. Das ist ein Objekt, in dem Angular jede empfangene Ansicht vom Server speichert.

Es ist möglich, das `html2js` Modul zu verwenden, das alle unsere Ansichten in eine modul-js-Datei konvertiert. Dann müssen wir dieses Modul in unsere Anwendung einspritzen und das war's.

Um eine verkettete Datei aller Ansichten zu erstellen, können Sie diese Aufgabe verwenden

```
module.exports = function (grunt) {
 //set up the location of your views here
 var viewLocation = ['app/views/**.html'];

 grunt.initConfig({
        pkg: require('./package.json'),
            //section that sets up the settings for concatenation of the html files into one
file
            html2js: {
                        options: {
                            base: '',
                            module: 'app.templates', //new module name
                            singleModule: true,
                            useStrict: true,
                            htmlmin: {
                                collapseBooleanAttributes: true,
                                collapseWhitespace: true
                            }
                        },
                        main: {
                            src: viewLocation,
                            dest: 'build/app.templates.js'
                        }
                    },
            //this section is watching for changes in view files, and if there was a change,
it will regenerate the production file. This task can be handy during development.
            watch: {
```

```
                views:{
                    files: viewLocation,
                    tasks: ['buildHTML']
                },
            }
        });

        //to automatically generate one view file
        grunt.loadNpmTasks('grunt-html2js');

        //to watch for changes and if the file has been changed, regenerate the file
        grunt.loadNpmTasks('grunt-contrib-watch');

        //just a task with friendly name to reference in watch
        grunt.registerTask('buildHTML', ['html2js']);
};
```

Um diese Art der Verkatenung zu verwenden, müssen Sie zwei Änderungen vornehmen: In Ihrer `index.html` Datei müssen Sie auf die verkettete Ansichtsdatei verweisen

```
<script src="build/app.templates.js"></script>
```

In der Datei, in der Sie Ihre App deklarieren, müssen Sie die Abhängigkeit einfügen

```
angular.module('app', ['app.templates'])
```

Wenn Sie gängige Router wie `ui-router` , ändert sich die Art und Weise, wie Sie auf Vorlagen verweisen

```
    .state('home', {
        url: '/home',
        views: {
            "@": {
                controller: 'homeController',
                //this will be picked up from $templateCache
                templateUrl: 'app/views/home.html'
            },
        }

    })
```

Skriptoptimierung

Es ist empfehlenswert **, JS-Dateien zusammenzufassen** und zu minimieren. Bei größeren Projekten können hunderte von JS-Dateien vorhanden sein, was zu unnötiger Latenz führt, um jede Datei getrennt vom Server zu laden.

Für die Winkelminifizierung müssen alle Funktionen kommentiert werden. Das ist für die Angular-Abhängigkeitsinjektion die richtige Minificaiton. (Während der Minifizierung werden Funktionsnamen und -variablen umbenannt und die Abhängigkeitsinjektion wird unterbrochen, wenn keine zusätzlichen Aktionen ausgeführt werden.)

Während der `myService` Variablen `$scope` und `myService` durch einige andere Werte ersetzt. Angular

Dependency Injection arbeitet auf der Grundlage des Namens, daher sollten sich diese Namen nicht ändern

```
.controller('myController', function($scope, myService){
})
```

Angular wird die Array-Notation verstehen, da die Minifizierung keine String-Literale ersetzt.

```
.controller('myController', ['$scope','myService', function($scope, myService){
}])
```

- Erstens werden wir alle Dateien von Anfang bis Ende zusammenfassen.
- Zweitens verwenden wir das `ng-annotate` Modul, das den Code für die Minifizierung vorbereitet
- Zum Schluss wenden wir das `uglify` Modul an.

module.exports = function (grunt) {// Hier können Sie den Ort Ihrer Skripte für die Wiederverwendung im Code einrichten var varlocation = ['app / scripts / *. js'];

```
 grunt.initConfig({
        pkg: require('./package.json'),
            //add necessary annotations for safe minification
         ngAnnotate: {
            angular: {
                src: ['staging/concatenated.js'],
                dest: 'staging/anotated.js'
            }
        },
        //combines all the files into one file
        concat: {
                js: {
                    src: scriptLocation,
                    dest: 'staging/concatenated.js'
                }
            },
        //final uglifying
        uglify: {
            options: {
                report: 'min',
                mangle: false,
                sourceMap:true
            },
            my_target: {
                files: {
                    'build/app.min.js': ['staging/anotated.js']
                }
            }
        },

        //this section is watching for changes in JS files, and if there was a change, it will
regenerate the production file. You can choose not to do it, but I like to keep concatenated
version up to date
        watch: {
            scripts: {
                files: scriptLocation,
                tasks: ['buildJS']
```

```
            }
        }

});

    //module to make files less readable
    grunt.loadNpmTasks('grunt-contrib-uglify');

    //mdule to concatenate files together
    grunt.loadNpmTasks('grunt-contrib-concat');

    //module to make angularJS files ready for minification
    grunt.loadNpmTasks('grunt-ng-annotate');

    //to watch for changes and if the file has been changed, regenerate the file
    grunt.loadNpmTasks('grunt-contrib-watch');

    //task that sequentially executes all steps to prepare JS file for production
    //concatinate all JS files
    //annotate JS file (prepare for minification
    //uglify file
     grunt.registerTask('buildJS', ['concat:js', 'ngAnnotate', 'uglify']);
};
```

Bereiten Sie sich auf die Produktion vor - Grunt online lesen:
https://riptutorial.com/de/angularjs/topic/4434/bereiten-sie-sich-auf-die-produktion-vor---grunt

Kapitel 13: Controller

Syntax

- <htmlElement ng-controller = "controllerName"> ... </ htmlElement>
- <script> app.controller ('controllerName', controllerFunction); </ script>

Examples

Dein erster Controller

Ein Controller ist eine grundlegende Struktur, die in Angular verwendet wird, um den Umfang zu erhalten und bestimmte Aktionen auf einer Seite auszuführen. Jeder Controller ist mit einer HTML-Ansicht gekoppelt.

Nachfolgend finden Sie eine Grundversion für eine Angular-App:

```
<!DOCTYPE html>

<html lang="en" ng-app='MyFirstApp'>
    <head>
        <title>My First App</title>

        <!-- angular source -->
        <script src="https://code.angularjs.org/1.5.3/angular.min.js"></script>

        <!-- Your custom controller code -->
        <script src="js/controllers.js"></script>
    </head>
    <body>
        <div ng-controller="MyController as mc">
            <h1>{{ mc.title }}</h1>
            <p>{{ mc.description }}</p>
            <button ng-click="mc.clicked()">
                Click Me!
            </button>
        </div>
    </body>
</html>
```

Hier sind einige Dinge zu beachten:

```
<html ng-app='MyFirstApp'>
```

Wenn Sie den Anwendungsnamen mit `ng-app` festlegen, können Sie auf die Anwendung in einer externen Javascript-Datei zugreifen, auf die weiter unten eingegangen wird.

```
<script src="js/controllers.js"></script>
```

Wir benötigen eine Javascript-Datei, in der Sie Ihre Controller und ihre Aktionen / Daten definieren.

```
<div ng-controller="MyController as mc">
```

Das Attribut `ng-controller` legt den Controller für dieses DOM-Element und alle untergeordneten Elemente (rekursiv) fest.

Sie können mehrere Controller desselben `MyController` (in diesem Fall `MyController`) haben, indem Sie sagen, dass `... as mc` dieser Instanz des Controllers ein Alias `MyController` wird.

```
<h1>{{ mc.title }}</h1>
```

Die `{{ ... }}` Notation ist ein Winkelausdruck. In diesem Fall wird der innere Text dieses `<h1>` -Elements auf den Wert von `mc.title` .

Hinweis: Angular verwendet eine `mc.title` Datenbindung, dh unabhängig davon, wie Sie den `mc.title` Wert aktualisieren, wird er sowohl im Controller als auch auf der Seite `mc.title` .

Beachten Sie auch , dass Angular Ausdrücke *keinen* Controller verweisen müssen. Ein Angular-Ausdruck kann so einfach wie `{{ 1 + 2 }}` oder `{{ "Hello " + "World" }}` .

```
<button ng-click="mc.clicked()">
```

`ng-click` ist eine Angular-Direktive. In diesem Fall wird das Click-Ereignis für die Schaltfläche `MyController` die `MyController` `clicked()` Funktion der `MyController` Instanz `MyController` .

In diesem Sinne schreiben wir eine Implementierung des `MyController` Controllers. Mit dem obigen Beispiel würden Sie diesen Code in `js/controller.js` schreiben.

Zunächst müssen Sie die Angular-App in Ihrem Javascript instanziieren.

```
var app = angular.module("MyFirstApp", []);
```

Beachten Sie, dass der Name, den wir hier übergeben, mit dem Namen übereinstimmt, den Sie in Ihrem HTML-Code mit der Direktive `ng-app` .

Nun, da wir das App-Objekt haben, können wir damit Controller erstellen.

```
app.controller('MyController', function(){
    var ctrl = this;

    ctrl.title = "My First Angular App";
    ctrl.description = "This is my first Angular app!";

    ctrl.clicked = function(){
        alert("MyController.clicked()");
    };
});
```

Hinweis: Für alle Elemente, die Teil der Controller-Instanz sein möchten, verwenden wir das Schlüsselwort `this` .

Dies ist alles, was erforderlich ist, um eine einfache Steuerung aufzubauen.

Controller erstellen

```
angular
    .module('app')
    .controller('SampleController', SampleController)

SampleController.$inject = ['$log', '$scope'];
function SampleController($log, $scope){
    $log.debug('*****SampleController******');

    /* Your code below */
}
```

Hinweis: Der `.$inject` sicher, dass Ihre Abhängigkeiten nach der Minifizierung nicht durcheinander geraten. Stellen Sie außerdem sicher, dass die angegebene Funktion in Ordnung ist.

Erstellen von Controllern, Minification-Safe

Es gibt verschiedene Möglichkeiten, die Controller-Erstellung vor Minification zu schützen.

Die erste heißt Inline-Array-Annotation. Es sieht wie folgt aus:

```
var app = angular.module('app');
app.controller('sampleController', ['$scope', '$http', function(a, b){
    //logic here
}]);
```

Der zweite Parameter der Controller-Methode kann ein Array von Abhängigkeiten akzeptieren. Wie Sie sehen können, habe ich `$scope` und `$http` die den Parametern der Controller-Funktion entsprechen sollten, in denen `a` der `$scope` , und `b` wäre `$http` . Beachten Sie, dass das letzte Element im Array die Controller-Funktion sein sollte.

Die zweite Option verwendet die `$inject` Eigenschaft. Es sieht wie folgt aus:

```
var app = angular.module('app');
app.controller('sampleController', sampleController);
sampleController.$inject = ['$scope', '$http'];
function sampleController(a, b) {
    //logic here
}
```

Dies macht dasselbe wie die Inline-Array-Annotation, bietet jedoch ein anderes Design für diejenigen, die eine Option der anderen vorziehen.

Die Reihenfolge der injizierten Abhängigkeiten ist wichtig

Stellen Sie beim Einfügen von Abhängigkeiten mit dem Array-Formular sicher, dass die Liste der Abhängigkeiten mit der entsprechenden Liste der an die Controller-Funktion übergebenen Argumente übereinstimmt.

Beachten Sie, dass im folgenden Beispiel `$scope` und `$http` umgekehrt sind. Dies führt zu einem Problem im Code.

```
// Intentional Bug: injected dependencies are reversed which will cause a problem
app.controller('sampleController', ['$scope', '$http',function($http, $scope) {
    $http.get('sample.json');
}]);
```

Verwenden von ControllerAs in Angular JS

In Angular `$scope` ist die Verbindung zwischen Controller und View, die alle unsere Datenbindungsanforderungen unterstützt. Controller As ist eine andere Möglichkeit, Controller und View zu binden und wird meistens empfohlen. Grundsätzlich sind dies die beiden Controller-Konstrukte in Angular (dh $ scope und Controller As).

Verschiedene Arten der Verwendung von Controller As are -

controllerAs View-Syntax

```
<div ng-controller="CustomerController as customer">
    {{ customer.name }}
</div>
```

controllerAs Controller-Syntax

```
function CustomerController() {
    this.name = {};
    this.sendMessage = function() { };
}
```

ControllerAs mit vm

```
function CustomerController() {
    /*jshint validthis: true */
    var vm = this;
    vm.name = {};
    vm.sendMessage = function() { };
}
```

`controllerAs` ist syntaktischer Zucker über `$scope` . Sie können immer noch an die View binden und auf `$scope` Methoden zugreifen. Die Verwendung von `controllerAs` ist eine der Best Practices, die das Winkel-Kernteam vorschlägt. Dafür gibt es viele Gründe, wenige davon -

- `$scope` zeigt die Mitglieder der Steuerung über ein Zwischenobjekt an der Ansicht an. Mit `this.*` Einstellung `this.*` können wir genau das zeigen, was wir vom Controller für die Ansicht verfügbar machen möchten. Es folgt auch der Standard-JavaScript-Methode, um

dies zu verwenden.

- Wenn Sie die `controllerAs` Syntax verwenden, verfügen wir über besser lesbaren Code. Auf die übergeordnete Eigenschaft kann über den Aliasnamen des übergeordneten Controllers zugegriffen werden, anstatt die `$parent` Syntax zu verwenden.
- Es fördert die Verwendung der Bindung für ein "gepunktetes" Objekt in der Ansicht (z. B. customer.name anstelle des Namens), das kontextbezogener, lesbarer ist und alle Referenzprobleme vermeidet, die ohne "Dotting" auftreten können.
- Verhindert die Verwendung von `$parent` Aufrufen in Ansichten mit verschachtelten Controllern.
- Verwenden Sie dazu eine Erfassungsvariable, wenn Sie die `controllerAs` Syntax verwenden. Wählen Sie einen konsistenten Variablennamen wie `vm` , der für ViewModel steht. `this` Schlüsselwort ist kontextabhängig und kann bei Verwendung innerhalb einer Funktion in einem Controller seinen Kontext ändern. Durch das Erfassen des Kontextes wird vermieden, dass dieses Problem auftritt.

ANMERKUNG: Verwenden Sie die `controllerAs` Syntax, um sie dem aktuellen `controllerAs` hinzuzufügen, so dass er als Feld verfügbar ist

```
<div ng-controller="Controller as vm>...</div>
```

`vm` ist als `$scope.vm` verfügbar.

Erstellen von Minification-Safe-Winkelreglern

Um minification sichere Winkel Controller zu erstellen, wird die Änderung `controller` Funktionsparameter.

Das zweite Argument in der `module.controller` Funktion sollte ein **Array übergeben werden** , wobei der **letzte Parameter** die **Controller-Funktion** ist und jeder Parameter davor der **Name** jedes injizierten Werts ist.

Dies unterscheidet sich vom normalen Paradigma; das übernimmt die **Controller-Funktion** mit den eingegebenen Argumenten.

Gegeben:

```
var app = angular.module('myApp');
```

Der Controller sollte so aussehen:

```
app.controller('ctrlInject',
    [
        /* Injected Parameters */
        '$Injectable1',
        '$Injectable2',
        /* Controller Function */
```

```
        function($injectable1Instance, $injectable2Instance) {
            /* Controller Content */
        }
    ]
);
```

Hinweis: *Die Namen der injizierten Parameter müssen nicht übereinstimmen, werden jedoch in der Reihenfolge gebunden.*

Dies wird auf etwas Ähnliches reduziert:

```
var
a=angular.module('myApp');a.controller('ctrlInject',['$Injectable1','$Injectable2',function(b,c){/*
Controller Content */}]);
```

Der Minifizierungsprozess ersetzt jede Instanz der `app` durch `a` , jede Instanz von `$Injectable1Instance` durch `b` und jede Instanz von `$Injectable2Instance` durch `c` .

Verschachtelte Controller

Verschachtelungscontroller ketten auch den `$scope` . Wenn Sie eine `$scope` Variable im verschachtelten Controller ändern, wird dieselbe `$scope` Variable im übergeordneten Controller geändert.

```
.controller('parentController', function ($scope) {
    $scope.parentVariable = "I'm the parent";
});

.controller('childController', function ($scope) {
    $scope.childVariable = "I'm the child";

    $scope.childFunction = function () {
        $scope.parentVariable = "I'm overriding you";
    };
});
```

Versuchen wir nun, beide zu behandeln, verschachtelt.

```
<body ng-controller="parentController">
    What controller am I? {{parentVariable}}
    <div ng-controller="childController">
        What controller am I? {{childVariable}}
        <button ng-click="childFunction()"> Click me to override! </button>
    </div>
</body>
```

Das Schachteln von Controllern mag seine Vorteile haben, aber dabei muss eines bedacht werden. Durch Aufrufen der `ngController` Direktive wird eine neue Instanz des Controllers erstellt. `ngController` kann häufig zu Verwirrung und unerwarteten Ergebnissen führen.

Controller online lesen: https://riptutorial.com/de/angularjs/topic/601/controller

Kapitel 14: Controller mit ES6

Examples

Regler

es ist sehr einfach, einen angleJS-Controller mit ES6 zu schreiben, wenn Sie mit der **objektorientierten Programmierung** vertraut sind

```
class exampleContoller{

       constructor(service1,service2,...serviceN){
          let ctrl=this;
              ctrl.service1=service1;
              ctrl.service2=service2;
               .
               .
               .
              ctrl.service1=service1;
              ctrl.controllerName = 'Example Controller';
              ctrl.method1(controllerName)

       }

       method1(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
              ctrl.scopeName=param;
       }
       .
       .
       .
       methodN(param){
          let ctrl=this;
               ctrl.service1.serviceFunction();
              .
              .
       }


   }
   exampleContoller.$inject = ['service1','service2',...,'serviceN'];
   export default exampleContoller;
```

Controller mit ES6 online lesen: https://riptutorial.com/de/angularjs/topic/9419/controller-mit-es6

Kapitel 15: Das Selbst oder diese Variable in einem Controller

Einführung

Dies ist eine Erklärung für ein allgemeines Muster und allgemein als bewährte Methode, die Sie im AngularJS-Code finden.

Examples

Den Zweck der Variable "Selbst" verstehen

Wenn Sie "Controller als Syntax" verwenden, geben Sie Ihrem Controller bei Verwendung der Anweisung ng-controller einen Aliasnamen in der HTML-Datei.

```
<div ng-controller="MainCtrl as main">
</div>
```

Sie können dann auf Eigenschaften und Methoden von der ***Hauptvariable,*** die unsere Controller - Instanz darstellt. Lassen Sie uns beispielsweise auf die ***Begrüßungseigenschaft*** unseres Controllers zugreifen und diese auf dem Bildschirm anzeigen:

```
<div ng-controller="MainCtrl as main">
    {{ main.greeting }}
</div>
```

Jetzt müssen wir in unserem Controller einen Wert für die Begrüßungseigenschaft unserer Controller-Instanz festlegen (im Gegensatz zu $ scope oder etwas anderem):

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";
})
```

Um die HTML-Anzeige korrekt anzeigen zu können, mussten wir die Begrüßungseigenschaft innerhalb ***dieses*** Controllerkörpers festlegen. Ich erstelle eine Zwischenvariable namens ***self*** , die einen Verweis darauf enthält. Warum? Betrachten Sie diesen Code:

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;
```

```
  self.greeting = "Hello World";

  function itsLate () {
    this.greeting = "Goodnight";
  }

})
```

In diesem Code können Sie erwarten, dass der Text auf dem Bildschirm aktualisiert wird, wenn die Methode *itsLate* aufgerufen wird. *Dies* ist jedoch nicht der Fall. JavaScript verwendet Regeln auf Funktionsebene, sodass das "this" in itsLate auf etwas anderes als "this" außerhalb des Methodenkörpers verweist. Das gewünschte Ergebnis können wir jedoch erhalten, wenn wir die Variable ***self verwenden*** :

```
 angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
  var self = this;

  self.greeting = "Hello World";

  function itsLate () {
    self.greeting = "Goodnight";
  }

})
```

Das ist die Schönheit der Verwendung einer "self" -Variable in Ihren Controllern. Sie können überall auf Ihre Controller zugreifen und immer sicher sein, dass sie auf Ihre Controller-Instanz verweist.

Das Selbst oder diese Variable in einem Controller online lesen:
https://riptutorial.com/de/angularjs/topic/8867/das-selbst-oder-diese-variable-in-einem-controller

Kapitel 16: Daten teilen

Bemerkungen

Eine sehr häufige Frage bei der Arbeit mit Angular ist, wie Daten zwischen Controllern gemeinsam genutzt werden. Die Verwendung eines Dienstes ist die häufigste Antwort. Dies ist ein einfaches Beispiel, das ein Fabrikmuster zeigt , mit dem alle Arten von Datenobjekten zwischen zwei oder mehr Steuerungen gemeinsam genutzt werden können. Da es sich um eine gemeinsame Objektreferenz handelt, ist eine Aktualisierung in einem Controller sofort in allen anderen Controllern verfügbar, die den Dienst verwenden. Beachten Sie, dass sowohl Service als auch Factory und beide Provider .

Examples

Verwenden von ngStorage zum Freigeben von Daten

Fügen Sie zunächst die Quelle **ngStorage** in Ihre index.html ein.

Ein Beispiel für das `ngStorage` src wäre:

```
<head>
    <title>Angular JS ngStorage</title>
    <script src =
"http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js"></script>
    <script src="https://rawgithub.com/gsklee/ngStorage/master/ngStorage.js"></script>
</head>
```

`ngStorage` bietet Ihnen 2 Speicher, nämlich `$localStorage` und `$sessionStorage` . Sie müssen die Dienste ngStorage und Inject anfordern.

Angenommen, wenn `ng-app="myApp"` , würden Sie `ngStorage` wie folgt `ngStorage` :

```
var app = angular.module('myApp', ['ngStorage']);
        app.controller('controllerOne', function($localStorage,$sessionStorage) {
        // an object to share
        var sampleObject = {
            name: 'angularjs',
            value: 1
        };
       $localStorage.valueToShare = sampleObject;
       $sessionStorage.valueToShare = sampleObject;
    })
.controller('controllerTwo', function($localStorage,$sessionStorage) {
    console.log('localStorage: '+ $localStorage +'sessionStorage: '+$sessionStorage);
})
```

`$localStorage` und `$sessionStorage` über jeden Controller global `$sessionStorage` solange Sie diese Dienste in den Controller `$sessionStorage` .

Sie können auch `localStorage` und `sessionStorage` von `HTML5` . Bei der Verwendung von `HTML5` `localStorage` müssen Sie Ihre Objekte jedoch vor dem Verwenden oder Speichern serialisieren und deserialisieren.

Zum Beispiel:

```
var myObj = {
    firstname: "Nic",
    lastname: "Raboy",
    website: "https://www.google.com"
}
//if you wanted to save into localStorage, serialize it
window.localStorage.set("saved", JSON.stringify(myObj));

//unserialize to get object
var myObj = JSON.parse(window.localStorage.get("saved"));
```

Freigeben von Daten von einem Controller zu einem anderen über den Dienst

Wir können einen erstellen `service` zu `set` und `get` die Daten zwischen den `controllers` und dann diesen Dienst in der Reglerfunktion injizieren , wo wir sie verwenden möchten.

Bedienung :

```
app.service('setGetData', function() {
  var data = '';
    getData: function() { return data; },
    setData: function(requestData) { data = requestData; }
});
```

Controller:

```
app.controller('myCtrl1', ['setGetData',function(setGetData) {

  // To set the data from the one controller
  var data = 'Hello World !!';
  setGetData.setData(data);

}]);

app.controller('myCtrl2', ['setGetData',function(setGetData) {

  // To get the data from the another controller
  var res = setGetData.getData();
  console.log(res); // Hello World !!

}]);
```

Hier können wir sehen , dass `myCtrl1` für verwendet wird , `setting` der Daten und `myCtrl2` wird verwendet für `getting` die Daten. So können wir die Daten von einem Controller zu einem anderen Controller wie diesem weitergeben.

Daten teilen online lesen: https://riptutorial.com/de/angularjs/topic/1923/daten-teilen

Kapitel 17: Debuggen

Examples

Grundlegendes Debugging in Markup

Umfangstest und Ausgabe des Modells

```
<div ng-app="demoApp" ng-controller="mainController as ctrl">
    {{$id}}
    <ul>
        <li ng-repeat="item in ctrl.items">
            {{$id}}<br/>
            {{item.text}}
        </li>
    </ul>
    {{$id}}
    <pre>
       {{ctrl.items | json : 2}}
    </pre>
</div>


angular.module('demoApp', [])
.controller('mainController', MainController);

function MainController() {
    var vm = this;
    vm.items = [{
        id: 0,
        text: 'first'
    },
    {
        id: 1,
        text: 'second'
    },
    {
        id: 2,
        text: 'third'
    }];
}
```

Manchmal kann es hilfreich sein zu sehen, ob es einen neuen Spielraum gibt, um Probleme mit dem Umfang zu beheben. `$scope.$id` kann überall in Ihrem Markup in einem Ausdruck verwendet werden, um festzustellen, ob es einen neuen $ scope gibt.

Im Beispiel sehen Sie, dass sich außerhalb des ul-Tags derselbe Gültigkeitsbereich ($ id = 2) befindet, und innerhalb der `ng-repeat` gibt es für jede Iteration neue untergeordnete Gültigkeitsbereiche.

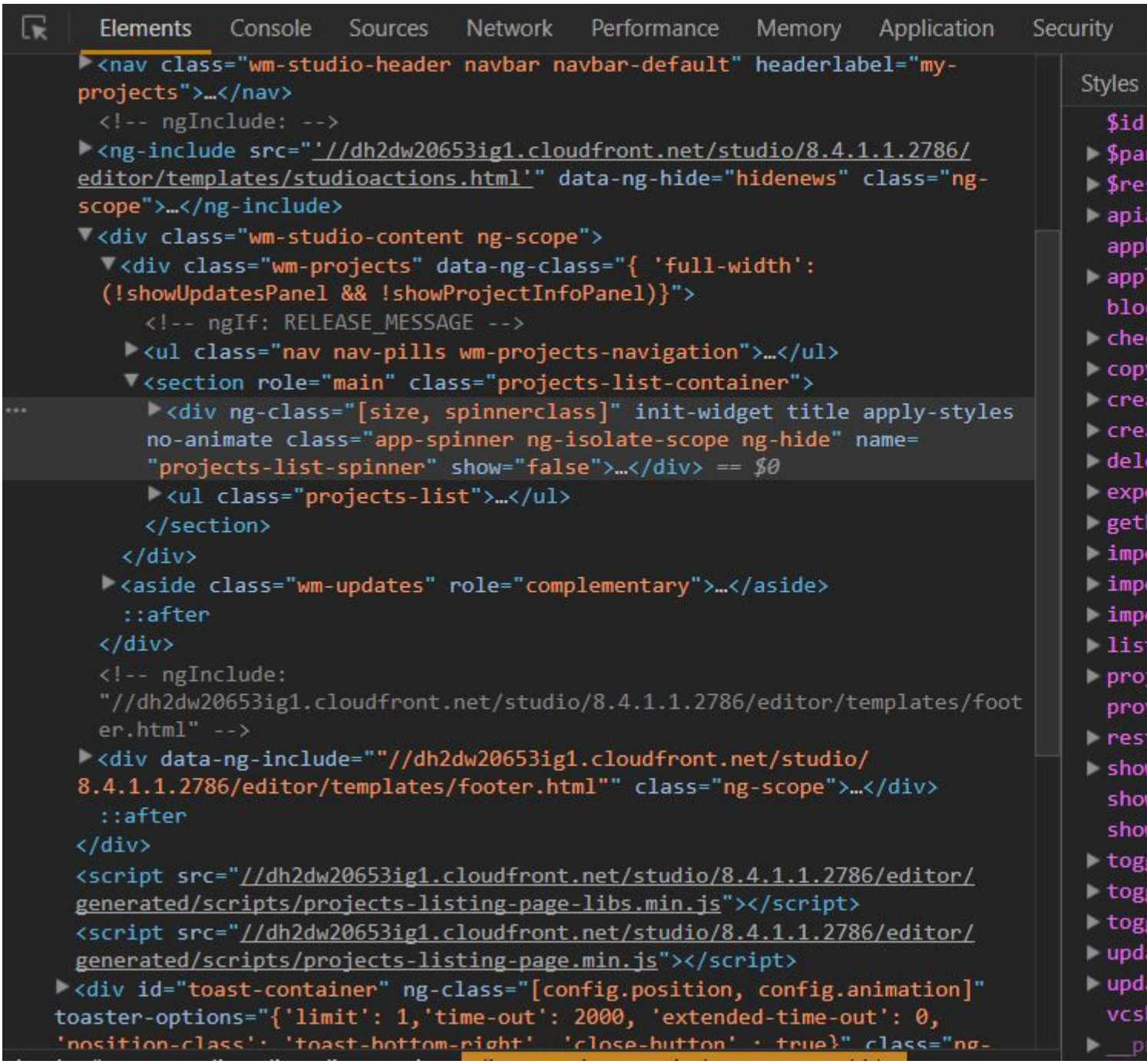
Eine Ausgabe des Modells in einem Pre-Tag ist hilfreich, um die aktuellen Daten Ihres Modells anzuzeigen. Der `json` Filter erzeugt eine gut aussehende formatierte Ausgabe. Das Pre-Tag wird verwendet, da innerhalb dieses Tags alle neuen Zeilen `\n` korrekt angezeigt werden.

Demo

Mit ng-inspect Chromverlängerung

ng-inspect ist eine leichte Chrome-Erweiterung zum Debuggen von AngularJS-Anwendungen.

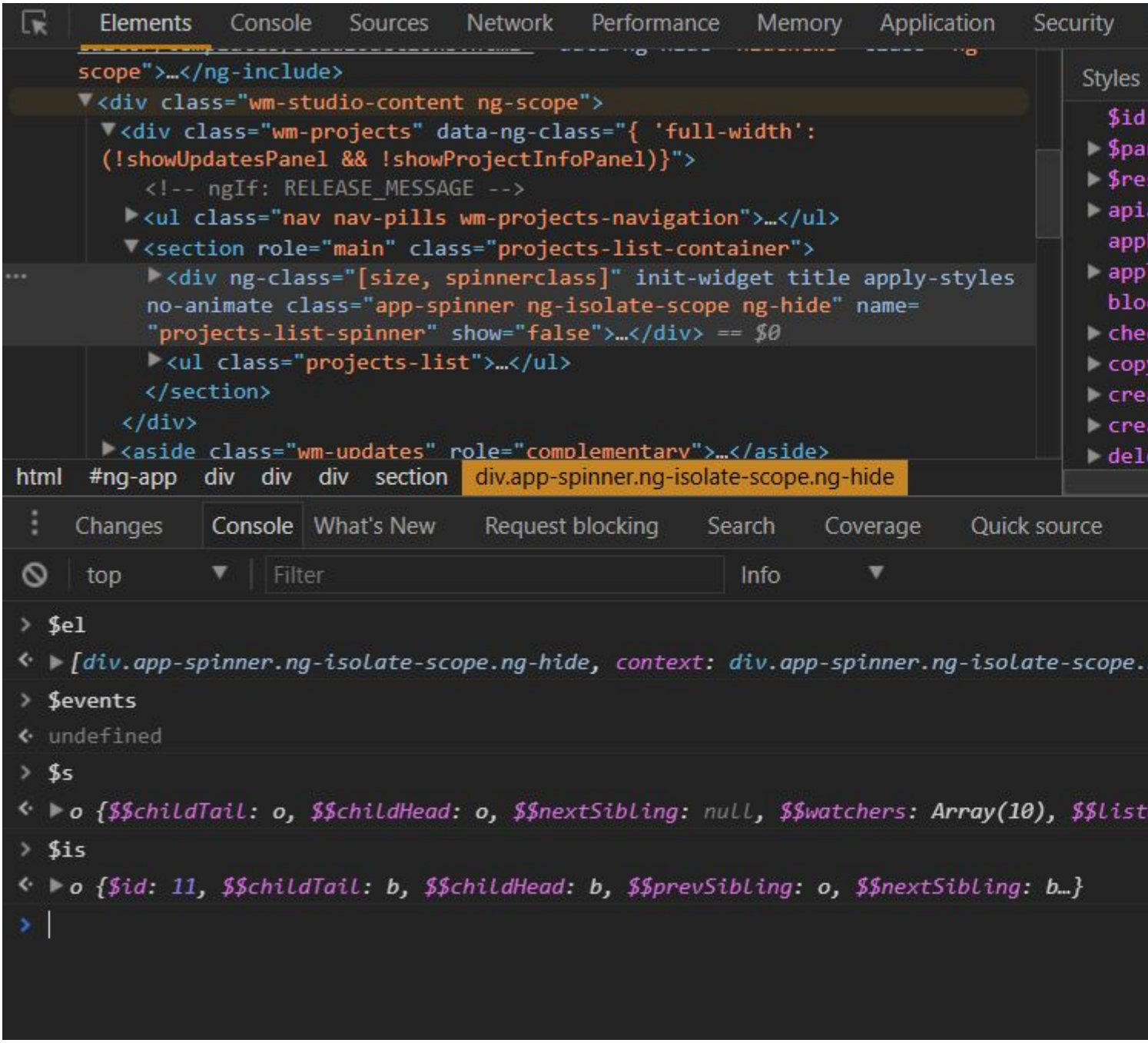
Wenn ein Knoten im Elementbereich ausgewählt wird, werden die Informationen zum Bereich im Bereich ng-inspect angezeigt.



Stellt einige globale Variablen für den schnellen Zugriff auf `scope/isolateScope` .

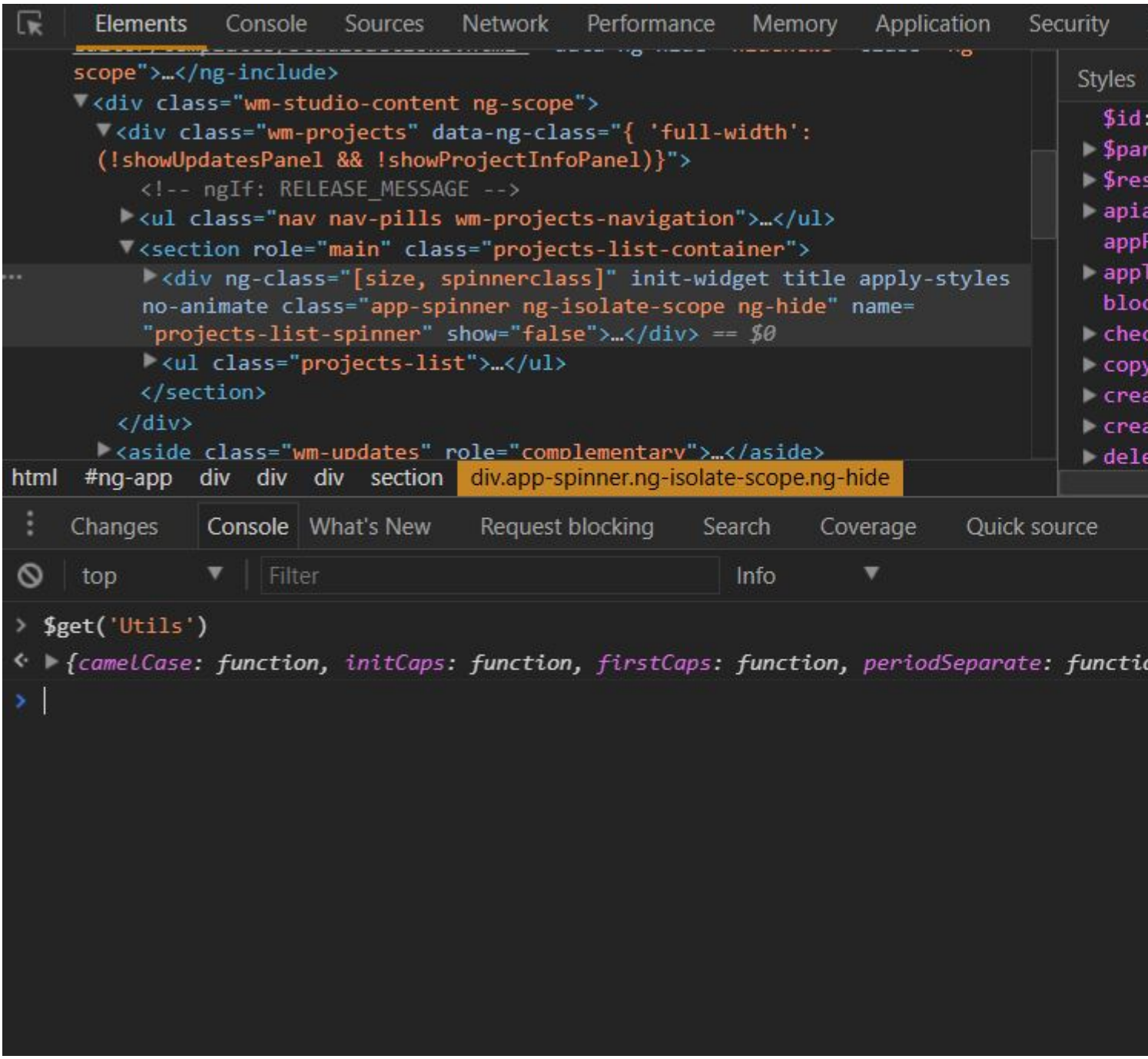
```
$s      -- scope of the selected node
$is     -- isolateScope of the selected node
$el     -- jQuery element reference of the selected node (requiers jQuery)
```

```
$events -- events present on the selected node (requires jQuery)
```



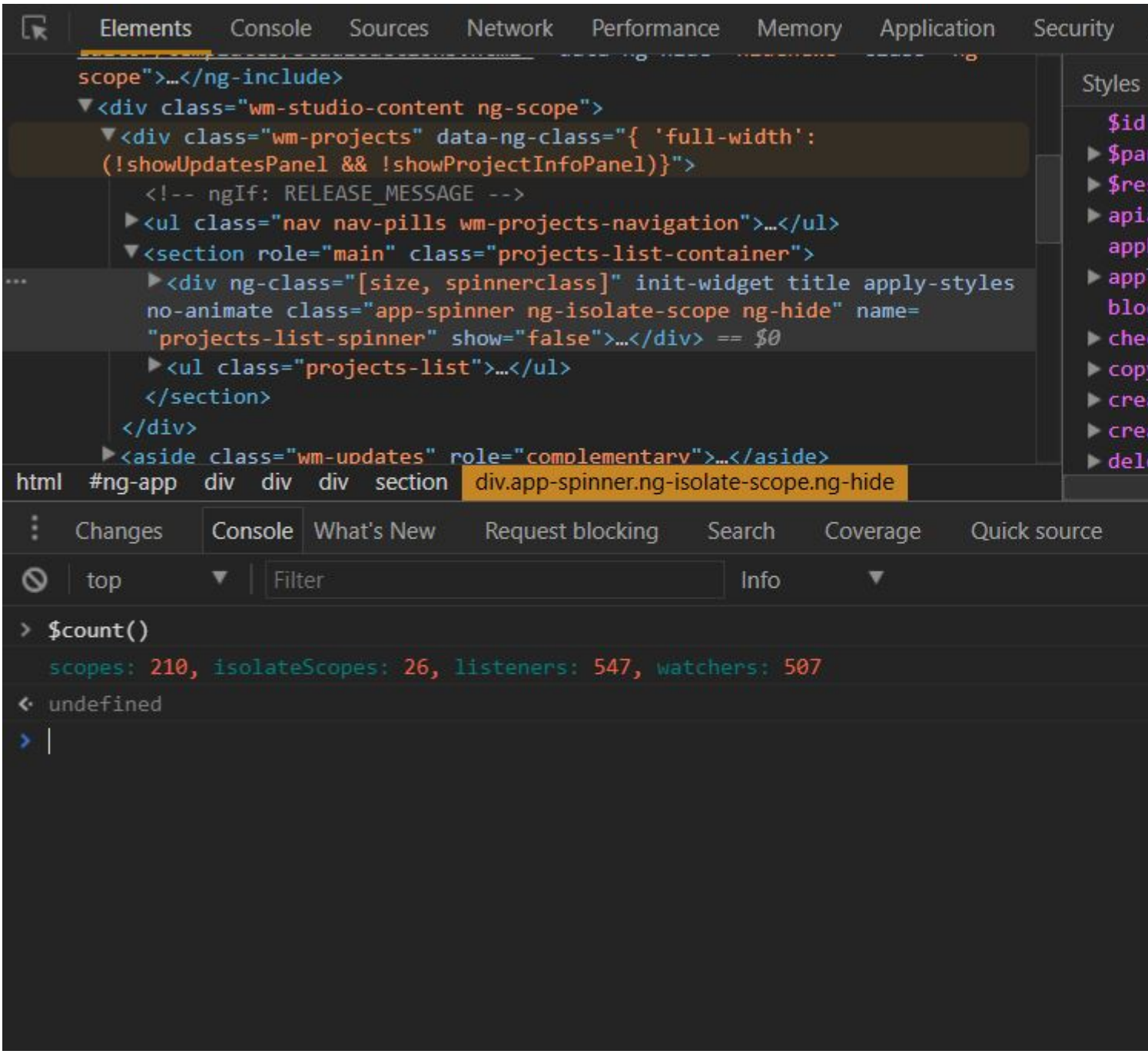
Bietet einfachen Zugriff auf Services / Factories.

Verwenden Sie `$get()` , um die Instanz eines Service / Factory nach Namen abzurufen.



Die Leistung der Anwendung kann überwacht werden, indem die Anzahl der Bereiche, IsolateScopes, Beobachter und Listener in der Anwendung gezählt wird.

Verwenden Sie `$count()` , um die Anzahl der Bereiche, isolateScopes, Beobachter und Zuhörer zu ermitteln.



Hinweis: Diese Erweiterung funktioniert nur, wenn DebugInfo aktiviert ist.

Download ng-inspizieren hier

Den Umfang des Elements erhalten

In einer Winkel-App geht alles um den Umfang. Wenn wir einen Element-Bereich erhalten könnten, ist es einfach, die Winkel-App zu debuggen. So greifen Sie auf den Umfang des Elements zu:

```
angular.element(myDomElement).scope();
e.g.
angular.element(document.getElementById('yourElementId')).scope()  //accessing by ID
```

Den Umfang des Controllers erhalten: -

```
angular.element('[ng-controller=ctrl]').scope()
```

Eine andere einfache Möglichkeit, auf ein DOM-Element über die Konsole zuzugreifen (wie von jm erwähnt), ist das Klicken auf das Register "Elemente". Es wird automatisch als $ 0 gespeichert.

```
angular.element($0).scope();
```

Debuggen online lesen: https://riptutorial.com/de/angularjs/topic/4761/debuggen

Kapitel 18: Dekorateure

Syntax

- Dekorateur (Name, Dekorateur);

Bemerkungen

> **Decorator** ist eine Funktion, mit der ein Dienst , eine Fabrik , eine Anweisung oder ein Filter vor seiner Verwendung geändert werden kann. Decorator wird verwendet, um das Verhalten des Dienstes zu überschreiben oder zu ändern. Der Rückgabewert der Decorator-Funktion kann der ursprüngliche Dienst sein oder ein neuer Dienst, der den ursprünglichen Dienst ersetzt oder an diesen delegiert.

Jede Dekoration ***muss*** in Winkel Anwendung erfolgen `config` - Phase durch die Injektion von `$provide` und verwendet es ist `$provide.decorator` Funktion.

> Die Decorator-Funktion verfügt über ein `$delegate` Objekt, das den Zugriff auf den Dienst ermöglicht, der mit dem Selektor im Decorator übereinstimmt. Dieser `$delegate` wird der Service sein, den Sie schmücken. Der Rückgabewert der Funktion, die dem Dekorateur zur Verfügung gestellt wird, wird für den Dienst, die Direktive oder den Filter vorgenommen, die dekoriert werden.

Der Dekorateur sollte nur in Betracht gezogen werden, wenn ein anderer Ansatz nicht angemessen ist oder sich als zu langweilig erweist. Wenn eine große Anwendung denselben Dienst verwendet und ein Teil das Dienstverhalten ändert, können leicht Verwirrung und / oder Fehler im Prozess auftreten.

Ein typischer Anwendungsfall wäre, wenn Sie eine Drittanbieter-Abhängigkeit haben, die Sie nicht aktualisieren können, aber etwas anders funktionieren oder erweitern müssen.

Examples

Service schmücken, Fabrik

Im Folgenden finden Sie Beispiel für Service - Dekorateur, zwingende `null` nach Dienst zurückgegebene Datum.

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myService', function($delegate) {
      $delegate.getDate = function() { // override with actual date object
        return new Date();
      };
```

```
      return $delegate;
    });
  })
  .service('myService', function() {
    this.getDate = function() {
      return null; // w/o decoration we'll be returning null
    };
  })
  .controller('myController', function(myService) {
    var vm = this;
    vm.date = myService.getDate();
  });
```

```
<body ng-controller="myController as vm">
  <div ng-bind="vm.date | date:'fullDate'"></div>
</body>
```

> Saturday, August 6, 2016

Direktive dekorieren

Direktiven können genauso wie Dienste dekoriert werden, und wir können jede ihrer Funktionen ändern oder ersetzen. Beachten Sie, dass auf die Direktive selbst an Position 0 im Delegatenarray $ zugegriffen wird, und der Name-Parameter in decorator muss das Suffix `Directive` (Groß- / Kleinschreibung) enthalten.

Wenn die Direktive `myDate` heißt, kann mit `myDateDirective` mit `$delegate[0]` darauf zugegriffen werden.

Nachfolgend finden Sie ein einfaches Beispiel, in dem die Direktive die aktuelle Uhrzeit anzeigt. Wir werden es dekorieren, um die aktuelle Uhrzeit im Sekundentakt zu aktualisieren. Ohne Dekoration wird es immer gleich angezeigt.

```
<body>
  <my-date></my-date>
</body>
```

```
angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('myDateDirective', function($delegate, $interval) {
      var directive = $delegate[0]; // access directive

      directive.compile = function() { // modify compile fn
        return function(scope) {
          directive.link.apply(this, arguments);
          $interval(function() {
            scope.date = new Date(); // update date every second
          }, 1000);
        };
      };
```

```
      return $delegate;
    });
  })
  .directive('myDate', function() {
    return {
      restrict: 'E',
      template: '<span>Current time is {{ date | date:\'MM:ss\' }}</span>',
      link: function(scope) {
        scope.date = new Date(); // get current date
      }
    };
  });
```

> Current time is 08:33

Filter dekorieren

Beim Dekorieren von Filtern muss der Name-Parameter das Suffix " `Filter` (Groß- und Kleinschreibung beachten) enthalten. Wenn der Filter " `repeat` , lautet der Parameter "decorator" " `repeatFilter` . Im Folgenden werden wir einen benutzerdefinierten Filter dekorieren, der eine bestimmte Zeichenfolge *n*- mal wiederholt, sodass das Ergebnis umgekehrt wird. Sie können auch die eingebauten Filter von Eawn auf dieselbe Weise dekorieren, dies wird jedoch nicht empfohlen, da dies die Funktionalität des Frameworks beeinflussen kann.

```
<body>
  <div ng-bind="'i can haz cheeseburger ' | repeat:2"></div>
</body>

angular.module('app', [])
  .config(function($provide) {
    $provide.decorator('repeatFilter', function($delegate) {
      return function reverse(input, count) {
        // reverse repeated string
        return ($delegate(input, count)).split('').reverse().join('');
      };
    });
  })
  .filter('repeat', function() {
    return function(input, count) {
      // repeat string n times
      return (input || '').repeat(count || 1);
    };
  });
```

> i can haz cheeseburger i can haz cheeseburger

> regrubeseehc zah nac i regrubeseehc zah nac i

Dekorateure online lesen: https://riptutorial.com/de/angularjs/topic/5255/dekorateure

Kapitel 19: Dienstleistungen

Examples

Wie erstelle ich einen Service?

```
angular.module("app")
  .service("counterService", function(){

    var service = {
        number: 0
    };

    return service;
   });
```

Wie benutze ich einen Dienst?

```
    angular.module("app")

        // Custom services are injected just like Angular's built-in services
        .controller("step1Controller", ['counterService', '$scope', function(counterService,
$scope) {
            counterService.number++;
            // bind to object (by reference), not to value, for automatic sync
            $scope.counter = counterService;
        })
```

In die Vorlage mit diesem Controller würden Sie dann schreiben:

```
// editable
<input ng-model="counter.number" />
```

oder

```
// read-only
<span ng-bind="counter.number"></span>
```

In echtem Code würden Sie natürlich mit dem Dienst über Methoden auf dem Controller interagieren, die wiederum an den Dienst delegieren. Das obige Beispiel erhöht den Zählerwert einfach jedes Mal, wenn der Controller in einer Vorlage verwendet wird.

Dienstleistungen in Angularjs sind Singletons:

Services sind Singleton-Objekte, die nur einmal pro App instanziiert werden (durch den $ injector) und faul geladen (nur bei Bedarf erstellt) werden.

> Ein Singleton ist eine Klasse, die nur die Erstellung einer Instanz von sich selbst

zulässt - und einen einfachen und einfachen Zugriff auf diese Instanz ermöglicht. Wie hier angegeben

Erstellen eines Dienstes mithilfe von angle.factory

Definieren Sie zuerst den Dienst (in diesem Fall wird das Werksmuster verwendet):

```
.factory('dataService', function() {
    var dataObject = {};
    var service = {
        // define the getter method
        get data() {
            return dataObject;
        },
        // define the setter method
        set data(value) {
            dataObject = value || {};
        }
    };
    // return the "service" object to expose the getter/setter
    return service;
})
```

Jetzt können Sie den Dienst verwenden, um Daten zwischen Controllern gemeinsam zu nutzen:

```
.controller('controllerOne', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // create an object to store
    var someObject = {
        name: 'SomeObject',
        value: 1
    };
    // store the object
    this.dataService.data = someObject;
})

.controller('controllerTwo', function(dataService) {
    // create a local reference to the dataService
    this.dataService = dataService;
    // this will automatically update with any changes to the shared data object
    this.objectFromControllerOne = this.dataService.data;
})
```

$ sce - Bereinigen und Rendern von Inhalten und Ressourcen in Vorlagen

$ sce ("Strict Contextual Escaping") ist ein integrierter Winkeldienst, der automatisch Inhalte und interne Quellen in Vorlagen bereinigt.

Das Einfügen von **externen** Quellen und **rohem HTML-** `$sce` in die Vorlage erfordert ein manuelles Wrapping von `$sce` .

In diesem Beispiel erstellen wir einen einfachen $ sce-Sanitationsfilter: `.

Demo

```
.filter('sanitizer', ['$sce', [function($sce) {
     return function(content) {
          return $sce.trustAsResourceUrl(content);
      };
}]);
```

Verwendung in der Vorlage

```
<div ng-repeat="item in items">

    // Sanitize external sources
    <ifrmae ng-src="{{item.youtube_url | sanitizer}}">

    // Sanitaize and render HTML
    <div ng-bind-html="{{item.raw_html_content| sanitizer}}"></div>

</div>
```

So erstellen Sie einen Dienst mit Abhängigkeiten unter Verwendung der 'Array-Syntax'

```
angular.module("app")
  .service("counterService", ["fooService", "barService", function(anotherService,
barService){

    var service = {
        number: 0,
        foo: function () {
            return fooService.bazMethod(); // Use of 'fooService'
        },
        bar: function () {
            return barService.bazMethod(); // Use of 'barService'
        }
    };

    return service;
   }]);
```

Einen Service registrieren

Die gebräuchlichste und flexibelste Art, einen Dienst zu erstellen, verwendet die API-Factory von angle.module:

```
angular.module('myApp.services', []).factory('githubService', function() {
   var serviceInstance = {};
   // Our first service
   return serviceInstance;
});
```

Die Service Factory-Funktion kann entweder eine Funktion oder ein Array sein, genauso wie wir Controller erstellen:

```
// Creating the factory through using the
```

```
// bracket notation
angular.module('myApp.services', [])
.factory('githubService', [function($http) {
}]);
```

Um eine Methode für unseren Service verfügbar zu machen, können wir sie als Attribut für das Serviceobjekt platzieren.

```
angular.module('myApp.services', [])
    .factory('githubService', function($http) {
        var githubUrl = 'https://api.github.com';
        var runUserRequest = function(username, path) {
        // Return the promise from the $http service
        // that calls the Github API using JSONP
        return $http({
            method: 'JSONP',
            url: githubUrl + '/users/' +
            username + '/' +
            path + '?callback=JSON_CALLBACK'
        });
    }
// Return the service object with a single function
// events
return {
    events: function(username) {
    return runUserRequest(username, 'events');
    }
};
```

Unterschied zwischen Service und Werk

1) Dienstleistungen

Ein Service ist eine `constructor` , die zur Laufzeit einmalig mit `new` aufgerufen wird, genau wie das, was wir mit einfachem Javascript tun würden, mit dem Unterschied, dass `AngularJs` das `new` hinter den Kulissen aufruft.

Es gibt eine Daumenregel, die Sie bei Services beachten sollten

1. Dienste sind Konstruktoren, die mit `new` aufgerufen werden

Sehen wir uns ein einfaches Beispiel an, bei dem wir einen Dienst registrieren, der den `$http` Dienst zum Abrufen von Studentendaten verwendet, und ihn im Controller verwenden

```
function StudentDetailsService($http) {
  this.getStudentDetails = function getStudentDetails() {
    return $http.get('/details');
  };
}

angular.module('myapp').service('StudentDetailsService', StudentDetailsService);
```

Wir injizieren diesen Dienst einfach in die Steuerung

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

Wann verwenden?

Verwenden Sie `.service()` wo immer Sie einen Konstruktor verwenden möchten. Sie wird normalerweise zum Erstellen öffentlicher APIs wie bei `getStudentDetails()` . Wenn Sie jedoch keinen Konstruktor verwenden möchten und stattdessen ein einfaches API-Muster verwenden möchten, gibt es in `.service()` keine große Flexibilität.

2) Fabrik

Obwohl wir alle Dinge mit `.factory()` was wir mit `.services()` machen würden, macht es `.factory()` "gleich" `.service()` . Es ist viel leistungsfähiger und flexibler als `.service()`

Eine `.factory()` ist ein Entwurfsmuster, mit dem ein Wert zurückgegeben wird.

Für Fabriken gibt es zwei Daumenregeln, die Sie beachten sollten

1. Fabriken geben Werte zurück
2. Fabriken (können) Objekte erstellen (Beliebiges Objekt)

`.factory()` wir uns einige Beispiele an, was wir mit `.factory()`

Rückgabe von Objekten Literalen

Sehen wir uns ein Beispiel an, in dem Factory verwendet wird, um ein Objekt unter Verwendung eines grundlegenden Revealing-Modulmusters zurückzugeben

```
function StudentDetailsService($http) {
  function getStudentDetails() {
    return $http.get('/details');
  }
  return {
    getStudentDetails: getStudentDetails
  };
}

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Verwendung in einem Controller

```
function StudentController(StudentDetailsService) {
  StudentDetailsService.getStudentDetails().then(function (response) {
      // handle response
    });
}
angular.module('app').controller('StudentController', StudentController);
```

Rückgabe von Verschlüssen

Was ist eine Schließung?

Closures sind Funktionen, die auf lokal verwendete Variablen verweisen, die jedoch in einem einschließenden Bereich definiert sind.

Folgendes ist ein Beispiel für eine Schließung

```
function closureFunction(name) {
  function innerClosureFunction(age) { // innerClosureFunction() is the inner function, a
closure
    // Here you can manipulate 'age' AND 'name' variables both
  };
};
```

Der *"wundervolle"* Teil ist, dass er auf den `name` zugreifen kann, der sich im übergeordneten Bereich befindet.

Verwenden `.factory()` das obige Schließungsbeispiel in `.factory()`

```
function StudentDetailsService($http) {
  function closureFunction(name) {
  function innerClosureFunction(age) {
    // Here you can manipulate 'age' AND 'name' variables
    };
  };
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Verwendung in einem Controller

```
function StudentController(StudentDetailsService) {
  var myClosure = StudentDetailsService('Student Name'); // This now HAS the
innerClosureFunction()
  var callMyClosure = myClosure(24); // This calls the innerClosureFunction()
};

angular.module('app').controller('StudentController', StudentController);
```

Konstruktoren / Instanzen erstellen

`.service()` erstellt Konstruktoren mit einem Aufruf von `new` wie oben. `.factory()` kann auch Konstruktoren mit einem Aufruf von `new` erstellen

Sehen wir uns ein Beispiel dafür an

```
function StudentDetailsService($http) {
  function Student() {
    this.age = function () {
        return 'This is my age';
    };
  }
```

```
  Student.prototype.address = function () {
        return 'This is my address';
  };
  return Student;
};

angular.module('myapp').factory('StudentDetailsService', StudentDetailsService);
```

Verwendung in einem Controller

```
function StudentController(StudentDetailsService) {
  var newStudent = new StudentDetailsService();

  //Now the instance has been created. Its properties can be accessed.

 newStudent.age();
 newStudent.address();

};

angular.module('app').controller('StudentController', StudentController);
```

Dienstleistungen online lesen: https://riptutorial.com/de/angularjs/topic/1486/dienstleistungen

Kapitel 20: Digest Loop Komplettlösung

Syntax

- $ scope. $ watch (watchExpression, Rückruf, [Deep Compare])
- $ scope. $ digest ()
- $ scope. $ apply ([exp])

Examples

Zweiwege-Datenbindung

Angular hat etwas Magie unter der Haube. Es ermöglicht das Binden von DOM an reale js-Variablen.

Angular verwendet eine Schleife mit dem Namen " *Digest Loop* ", die nach jeder Änderung einer Variablen aufgerufen wird - Callbacks, die das DOM aktualisieren.

Zum Beispiel fügt die Direktive `ng-model` dieser Eingabe einen `keyup` eventListener hinzu :

```
<input ng-model="variable" />
```

Jedes Mal, `keyup` das `keyup` Ereignis `keyup` wird, beginnt die *Digest-Schleife* .

Irgendwann durchläuft die *Digest-Schleife* einen Rückruf, der den Inhalt dieses Bereichs aktualisiert:

```
<span>{{variable}}</span>
```

Der grundlegende Lebenszyklus dieses Beispiels fasst (sehr schematisch) die Wirkungsweise von Winkel zusammen:

1. Angular scannt HTML
    - `ng-model` Direktive `ng-model` erstellt bei der Eingabe einen `keyup` Listener
    - `expression` innerhalb des Bereichs fügt dem *Digest-Zyklus* einen Rückruf hinzu
2. Der Benutzer interagiert mit der Eingabe
    - `keyup` listener startet den *Digest-Zyklus*
    - *Digest-Zyklus* ruft den Rückruf auf
    - Callback-Aktualisierungen umfassen den Inhalt

$ Digest und $ Watch

Die Implementierung der bidirektionalen Datenbindung zum Erreichen des Ergebnisses aus dem vorherigen Beispiel könnte mit zwei Kernfunktionen erfolgen:

- **$ digest** wird nach einer Benutzerinteraktion aufgerufen (bindendes DOM => variable)
- **$ watch** setzt einen Callback, der nach Variablenänderungen aufgerufen wird (Bindungsvariable => DOM)

Hinweis: Bei diesem Beispiel handelt es sich um eine Demonstration, nicht um den tatsächlichen Winkelcode

---

```
<input id="input"/>
<span id="span"></span>
```

Die zwei Funktionen, die wir brauchen:

```
var $watches = [];
function $digest(){
    $watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
            $w.prevVal = val;
        }
    })
}
function $watch(val, callback){
    $watches.push({val:val, callback:callback, prevVal: val() })
}
```

Jetzt können wir diese Funktionen verwenden, um eine Variable an das DOM anzuschließen.

```
var realVar;
//this is usually done by ng-model directive
input1.addEventListener('keyup',function(e){
    realVar=e.target.value;
    $digest()
}, true);

//this is usually done with {{expressions}} or ng-bind directive
$watch(function(){return realVar},function(val){
    span1.innerHTML = val;
});
```

Natürlich sind die realen Implementierungen komplexer und unterstützen Parameter wie **das** zu bindende **Element** und die zu verwendende **Variable**

Ein laufendes Beispiel finden Sie hier: https://jsfiddle.net/azofxd4j/

der $ scope-Baum

Das vorige Beispiel ist gut genug, wenn wir ein einzelnes HTML-Element an eine einzelne Variable binden müssen.

In der Realität müssen viele Elemente an viele Variablen gebunden werden:

```
<span ng-repeat="number in [1,2,3,4,5]">{{number}}</span>
```

Diese `ng-repeat` bindet 5 Elemente an 5 Variablen, die als `number` , und für jeden von ihnen einen anderen Wert!

Die Art und Weise, wie Winkel dieses Verhalten erreicht, verwendet einen separaten Kontext für jedes Element, das separate Variablen benötigt. Dieser Kontext wird als Bereich bezeichnet.

Jeder Bereich enthält Eigenschaften, dh die an das DOM gebundenen Variablen. Die Funktionen `$digest` und `$watch` werden als Methoden des Bereichs implementiert.

Das DOM ist eine Baumstruktur, und Variablen müssen in verschiedenen Ebenen der Baumstruktur verwendet werden:

```
<div>
    <input ng-model="person.name" />
    <span ng-repeat="number in [1,2,3,4,5]">{{number}} {{person.name}}</span>
</div>
```

Wie wir jedoch sahen, unterscheidet sich der Kontext (oder Geltungsbereich) von Variablen in `ng-repeat` von dem darüber liegenden Kontext. Um dies zu lösen, werden Winkelbereiche als Baum implementiert.

Jeder Bereich verfügt über ein Array von `$digest` Elementen. Wenn Sie seine `$digest` Methode aufrufen, werden alle deren `$digest` Methoden ausgeführt.

Auf diese Weise wird - nach Änderung der Eingabe - `$digest` für den Gültigkeitsbereich der div aufgerufen, der dann den `$digest` für seine 5 Kinder ausführt, wodurch der Inhalt aktualisiert wird.

Eine einfache Implementierung für einen Bereich könnte folgendermaßen aussehen:

```
function $scope(){
    this.$children = [];
    this.$watches = [];
}

$scope.prototype.$digest = function(){
    this.$watches.forEach(function($w){
        var val = $w.val();
        if($w.prevVal !== val){
            $w.callback(val, $w.prevVal);
          $w.prevVal = val;
        }
    });
    this.$children.forEach(function(c){
        c.$digest();
    });
}

$scope.prototype.$watch = function(val, callback){
    this.$watches.push({val:val, callback:callback, prevVal: val() })
}
```

Hinweis: Bei diesem Beispiel handelt es sich um eine Demonstration, nicht um den tatsächlichen Winkelcode

Digest Loop Komplettlösung online lesen: https://riptutorial.com/de/angularjs/topic/3156/digest-loop-komplettlosung

Kapitel 21: Drucken

Bemerkungen

Erstellen Sie eine Ng-Hide-Klasse in der CSS-Datei. ng-show / hide funktioniert nicht ohne die Klasse.

Mehr Details

Examples

Druckservice

Bedienung:

```
angular.module('core').factory('print_service', ['$rootScope', '$compile', '$http',
'$timeout','$q',
    function($rootScope, $compile, $http, $timeout,$q) {

        var printHtml = function (html) {
            var deferred = $q.defer();
            var hiddenFrame = $('<iframe style="display:
none"></iframe>').appendTo('body')[0];

            hiddenFrame.contentWindow.printAndRemove = function() {
                hiddenFrame.contentWindow.print();
                $(hiddenFrame).remove();
                deferred.resolve();
            };

            var htmlContent =   "<!doctype html>"+
                                "<html>"+
                                    '<head><link rel="stylesheet" type="text/css"
href="/style/css/print.css"/></head>'+
                                    '<body onload="printAndRemove();">' +
                                        html +
                                    '</body>'+
                                "</html>";

            var doc = hiddenFrame.contentWindow.document.open("text/html", "replace");
            doc.write(htmlContent);
            doc.close();
            return deferred.promise;
        };

        var openNewWindow = function (html) {
            var newWindow = window.open("debugPrint.html");
            newWindow.addEventListener('load', function(){
                $(newWindow.document.body).html(html);
            }, false);
        };

        var print = function (templateUrl, data) {
```

```
            $rootScope.isBeingPrinted = true;

            $http.get(templateUrl).success(function(template){
                var printScope = $rootScope.$new()
                angular.extend(printScope, data);
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if(printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint, 1000);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html());
                        printScope.$destroy();
                    }
                };
                waitForRenderAndPrint();
            });
        };

        var printFromScope = function (templateUrl, scope, afterPrint) {
            $rootScope.isBeingPrinted = true;
            $http.get(templateUrl).then(function(response){
                var template = response.data;
                var printScope = scope;
                var element = $compile($('<div>' + template + '</div>'))(printScope);
                var waitForRenderAndPrint = function() {
                    if (printScope.$$phase || $http.pendingRequests.length) {
                        $timeout(waitForRenderAndPrint);
                    } else {
                        // Replace printHtml with openNewWindow for debugging
                        printHtml(element.html()).then(function() {
                            $rootScope.isBeingPrinted = false;
                            if (afterPrint) {
                                afterPrint();
                            }
                        });
                    }
                };
                waitForRenderAndPrint();
            });
        };

        return {
            print : print,
            printFromScope : printFromScope
        }
    }
]);
```

Controller:

```
var template_url = '/views/print.client.view.html';
print_service.printFromScope(template_url,$scope,function(){
      // Print Completed
});
```

Drucken online lesen: https://riptutorial.com/de/angularjs/topic/6750/drucken

Kapitel 22: Eingebaute Hilfsfunktionen

Examples

eckig

Die `angular.equals` Funktion vergleicht und ermittelt, ob 2 Objekte oder Werte gleich sind. `angular.equals` führt einen tiefen Vergleich durch und gibt true zurück, wenn nur eine der folgenden Bedingungen erfüllt ist.

> Winkelgleichungen (Wert1, Wert2)

1. Wenn die Objekte oder Werte den `===` Vergleich bestehen
2. Wenn beide Objekte oder Werte vom gleichen Typ sind und alle ihre Eigenschaften auch gleich sind, verwenden Sie `angular.equals`
3. Beide Werte sind gleich `NaN`
4. Beide Werte repräsentieren das Ergebnis des gleichen regulären Ausdrucks.

Diese Funktion ist hilfreich, wenn Sie Objekte oder Arrays nicht nur nach Referenzen, sondern nach Werten oder Ergebnissen vergleichen müssen.

Beispiele

```
angular.equals(1, 1) // true
angular.equals(1, 2) // false
angular.equals({}, {}) // true, note that {}==={} is false
angular.equals({a: 1}, {a: 1}) // true
angular.equals({a: 1}, {a: 2}) // false
angular.equals(NaN, NaN) // true
```

angle.isString

Die Funktion `angular.isString` gibt true zurück, wenn das angegebene Objekt oder der Wert dem Typ `string`

> angle.isString (Wert1)

Beispiele

```
angular.isString("hello") // true
angular.isString([1, 2]) // false
angular.isString(42) // false
```

Dies entspricht der Leistung

```
typeof someValue === "string"
```

eckig.isArray

Die `angular.isArray` Funktion gibt nur dann true zurück, wenn das an sie übergebene Objekt oder Wert vom Typ `Array` .

angle.isArray (Wert)

Beispiele

```
angular.isArray([]) // true
angular.isArray([2, 3]) // true
angular.isArray({}) // false
angular.isArray(17) // false
```

Es ist das Äquivalent von

```
Array.isArray(someValue)
```

eckig

Die Funktion angle.merge übernimmt alle aufzählbaren Eigenschaften des Quellobjekts, um das Zielobjekt tief zu erweitern.

Die Funktion gibt eine Referenz auf das jetzt erweiterte Zielobjekt zurück

angle.merge (Ziel, Quelle)

Beispiele

```
angular.merge({}, {}) // {}
angular.merge({name: "king roland"}, {password: "12345"})
// {name: "king roland", password: "12345"}
angular.merge({a: 1}, [4, 5, 6]) // {0: 4, 1: 5, 2: 6, a: 1}
angular.merge({a: 1}, {b: {c: {d: 2}}}) // {"a":1,"b":{"c":{"d":2}}}
```

angle.isDefiniert und angle.isUndefined

Die Funktion `angular.isDefined` testet einen Wert, wenn er definiert ist

angle.isDefined (someValue)

Dies entspricht der Leistung

```
value !== undefined; // will evaluate to true is value is defined
```

Beispiele

```
angular.isDefined(42) // true
angular.isDefined([1, 2]) // true
angular.isDefined(undefined) // false
```

```
angular.isDefined(null) // true
```

Die Funktion `angular.isUndefined` prüft, ob ein Wert undefiniert ist (er ist tatsächlich das Gegenteil von `angular.isDefined`).

    angle.isUndefined (someValue)

Dies entspricht der Leistung

```
value === undefined; // will evaluate to true is value is undefined
```

Oder nur

```
!angular.isDefined(value)
```

Beispiele

```
angular.isUndefined(42) // false
angular.isUndefined(undefined) // true
```

angle.isDate

Die `angular.isDate` Funktion gibt nur dann true zurück, wenn das übergebene Objekt vom Typ Date ist.

    angle.isDate (Wert)

Beispiele

```
angular.isDate("lone star") // false
angular.isDate(new Date()) // true
```

eckig.istNummer

Die `angular.isNumber` Funktion gibt nur dann true zurück, wenn das übergebene Objekt oder der Wert dem Typ Number entspricht. Dazu gehören + Infinity, -Infinity und NaN

    angle.isNumber (Wert)

Diese Funktion führt nicht zu einer Typumwandlung wie

```
"23" == 23 // true
```

Beispiele

```
angular.isNumber("23") // false
angular.isNumber(23) // true
angular.isNumber(NaN) // true
```

```
angular.isNumber(Infinity) // true
```

Diese Funktion führt nicht zu einer Typumwandlung wie

```
"23" == 23 // true
```

eckig.isFunktion

Die Funktion `angular.isFunction` ermittelt und gibt true zurück, wenn und nur wenn der übergebene Wert eine Referenz auf eine Funktion ist.

Die Funktion gibt eine Referenz auf das jetzt erweiterte Zielobjekt zurück

    angle.isFunction (fn)

Beispiele

```
var onClick = function(e) {return e};
angular.isFunction(onClick); // true

var someArray = ["pizza", "the", "hut"];
angular.isFunction(someArray ); // false
```

winklig.toJson

Die Funktion `angular.toJson` nimmt ein Objekt und serialisiert es in eine JSON-formatierte Zeichenfolge.

Im Gegensatz zur `JSON.stringify` Funktion `JSON.stringify` diese Funktion alle Eigenschaften, die mit `$$` (da Winkel normalerweise interne Eigenschaften mit `$$`).

```
angular.toJson(object)
```

Da Daten vor dem Durchlauf durch ein Netzwerk serialisiert werden müssen, ist diese Funktion nützlich, um alle Daten, die Sie übertragen möchten, in JSON umzuwandeln.

Diese Funktion ist auch für das Debugging hilfreich, da sie ähnlich funktioniert wie eine `.toString` Methode.

Beispiele:

```
angular.toJson({name: "barf", occupation: "mog", $$somebizzareproperty: 42})
// "{"name":"barf","occupation":"mog"}"
angular.toJson(42)
// "42"
angular.toJson([1, "2", 3, "4"])
// "[1,"2",3,"4"]"
var fn = function(value) {return value}
angular.toJson(fn)
// undefined, functions have no representation in JSON
```

eckig.von Json

Die Funktion `angular.fromJson` deserialisiert eine gültige JSON-Zeichenfolge und gibt ein Object oder ein Array zurück.

angle.fromJson (string | object)

Beachten Sie, dass diese Funktion nicht nur auf Strings beschränkt ist, sondern eine Repräsentation aller übergebenen Objekte ausgibt.

Beispiele:

```
angular.fromJson("{\"yogurt\": \"strawberries\"}")
// Object {yogurt: "strawberries"}
angular.fromJson('{jam: "raspberries"}')
// will throw an exception as the string is not a valid JSON
angular.fromJson(this)
// Window {external: Object, chrome: Object, _gaq: Y, angular: Object, ng339: 3…}
angular.fromJson([1, 2])
// [1, 2]
typeof angular.fromJson(new Date())
// "object"
```

eckig

Die `angular.noop` ist eine Funktion, die keine Operationen ausführt. Sie übergeben `angular.noop` wenn Sie ein Funktionsargument `angular.noop` müssen, das nichts `angular.noop` .

angle.noop ()

Eine übliche Verwendung für `angular.noop` kann darin bestehen, einer Funktion einen leeren Rückruf bereitzustellen, der andernfalls einen Fehler `angular.noop` wenn eine andere Funktion als eine Funktion an sie übergeben wird.

Beispiel:

```
$scope.onSomeChange = function(model, callback) {
    updateTheModel(model);
    if (angular.isFunction(callback)) {
        callback();
    } else {
        throw new Error("error: callback is not a function!");
    }
};

$scope.onSomeChange(42, function() {console.log("hello callback")});
// will update the model and print 'hello callback'
$scope.onSomeChange(42, angular.noop);
// will update the model
```

Zusätzliche Beispiele:

```
angular.noop() // undefined
```

```
angular.isFunction(angular.noop) // true
```

Winkel.istObject

Das `angular.isObject` true zurück, wenn und nur wenn das an ihn übergebene Argument ein Objekt ist, diese Funktion auch true für ein Array und false für `null typeof null` obwohl `typeof null` ein `object` .

angle.isObject (Wert)

Diese Funktion ist nützlich für die Typüberprüfung, wenn Sie ein definiertes Objekt zur Verarbeitung benötigen.

Beispiele:

```
angular.isObject({name: "skroob", job: "president"})
// true
angular.isObject(null)
// false
angular.isObject([null])
// true
angular.isObject(new Date())
// true
angular.isObject(undefined)
// false
```

angle.isElement

Das `angular.isElement` gibt true zurück, wenn das übergebene Argument ein DOM-Element oder ein mit jQuery umschlossenes Element ist.

angle.isElement (elem)

Diese Funktion ist nützlich, um zu prüfen, ob ein übergebenes Argument ein Element ist, bevor es als solches verarbeitet wird.

Beispiele:

```
angular.isElement(document.querySelector("body"))
// true
angular.isElement(document.querySelector("#some_id"))
// false if "some_id" is not using as an id inside the selected DOM
angular.isElement("<div></div>")
// false
```

eckige.Kopie

Die `angular.copy` Funktion nimmt ein Objekt, ein Array oder einen Wert und erstellt eine tiefe Kopie davon.

eckige.Kopie ()

Beispiel:

Objekte:

```
let obj = {name: "vespa", occupation: "princess"};
let cpy = angular.copy(obj);
cpy.name = "yogurt"
// obj = {name: "vespa", occupation: "princess"}
// cpy = {name: "yogurt", occupation: "princess"}
```

Arrays:

```
var w = [a, [b, [c, [d]]]];
var q = angular.copy(w);
// q = [a, [b, [c, [d]]]]
```

Im obigen Beispiel wird `angular.equals(w, q)` als wahr ausgewertet, da `.equals` Gleichheit nach Wert testet. `w === q` wird jedoch als falsch ausgewertet, da ein strenger Vergleich zwischen Objekten und Arrays durch Referenz erfolgt.

winkel.identität

Die `angular.identity` Funktion gibt das erste übergebene Argument zurück.

winkel.identität (argument)

Diese Funktion ist nützlich für die Funktionsprogrammierung. Sie können diese Funktion als Standardwert angeben, falls eine erwartete Funktion nicht übergeben wurde.

Beispiele:

```
angular.identity(42) // 42
```

```
var mutate = function(fn, num) {
    return angular.isFunction(fn) ? fn(num) : angular.identity(num)
}

mutate(function(value) {return value-7}, 42) // 35
mutate(null, 42) // 42
mutate("mount. rushmore", 42) // 42
```

eckig

Das `angular.forEach` akzeptiert ein Objekt und eine Iteratorfunktion. Anschließend wird die Iteratorfunktion für jede aufzählbare Eigenschaft / jeden Wert des Objekts ausgeführt. Diese Funktion funktioniert auch bei Arrays.

Wie die JS-Version von `Array.prototype.forEach` Die Funktion `Array.prototype.forEach` keine geerbten Eigenschaften (Prototypeneigenschaften). Die Funktion versucht jedoch nicht, einen `null` oder einen `undefined` Wert zu verarbeiten und gibt ihn nur zurück.

angle.forEach (Objekt, Funktion (Wert, Schlüssel) {// Funktion});

Beispiele:

```
angular.forEach({"a": 12, "b": 34}, (value, key) => console.log("key: " + key + ", value: " +
value))
// key: a, value: 12
// key: b, value: 34
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(key))
// will print the array indices: 1, 2, 3, 4, 5
angular.forEach([2, 4, 6, 8, 10], (value, key) => console.log(value))
// will print the array values: 2, 4, 6, 7, 10
angular.forEach(undefined, (value, key) => console.log("key: " + key + ", value: " + value))
// undefined
```

Eingebaute Hilfsfunktionen online lesen: https://riptutorial.com/de/angularjs/topic/3032/eingebaute-hilfsfunktionen

Kapitel 23: Eingebaute Richtlinien

Examples

Winkelausdrücke - Text vs. Nummer

Dieses Beispiel zeigt, wie Angular-Ausdrücke ausgewertet werden, wenn `type="text"` und `type="number"` für das Eingabeelement verwendet werden. Betrachten Sie den folgenden Controller und die folgende Ansicht:

Regler

```
var app = angular.module('app', []);

app.controller('ctrl', function($scope) {
    $scope.textInput = {
        value: '5'
    };
    $scope.numberInput = {
        value: 5
    };
});
```

Aussicht

```
<div ng-app="app" ng-controller="ctrl">
    <input type="text" ng-model="textInput.value">
    {{ textInput.value + 5 }}
    <input type="number" ng-model="numberInput.value">
    {{ numberInput.value + 5 }}
</div>
```

- Bei der Verwendung von `+` in einem Ausdruck gebunden eingegeben *Text,* wird der Betreiber die Saiten (erstes Beispiel) **verketten,** 55 auf dem Bildschirm angezeigt wird `*` .
- Bei der Verwendung von `+` in einem Ausdruck zu *Nummerneingabe* gebunden, kehrt die Bedienperson die **Summe** der Zahlen (zweites Beispiel), 10 auf dem Bildschirm angezeigt wird `*` .

`*` - Bis der Benutzer den Wert im Eingabefeld ändert, ändert sich die Anzeige danach entsprechend.

Arbeitsbeispiel

ngRepeat

`ng-repeat` ist eine integrierte Anweisung in Angular, mit der Sie ein Array oder ein Objekt iterieren und ein Element für jedes Element in der Auflistung einmal wiederholen können.

Wiederholen Sie ein Array

```
<ul>
    <li ng-repeat="item in itemCollection">
       {{item.Name}}
    </li>
</ul>
```

Woher:
item = einzelner Artikel in der Sammlung
itemCollection = Das Array, das Sie durchlaufen

Wiederholen Sie ein Objekt

```
<ul>
    <li ng-repeat="(key, value) in myObject">
       {{key}} : {{value}}
    </li>
</ul>
```

Woher:
key = der Name der Eigenschaft
value = der Wert der Eigenschaft
myObject = das Objekt, das Sie durchlaufen

Filtern Sie Ihren Wiederholungsvorgang nach Benutzereingaben

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText">
       {{string}}
    </li>
</ul>
```

Woher:
searchText = Der Text, nach dem der Benutzer die Liste filtern möchte
stringArray = ein Array von Strings, zB `['string', 'array']`

Sie können die gefilterten Elemente auch an anderer Stelle anzeigen oder referenzieren, indem Sie der Filterausgabe einen Alias mit `as aliasName` .

```
<input type="text" ng-model="searchText">
<ul>
    <li ng-repeat="string in stringArray | filter:searchText as filteredStrings">
       {{string}}
    </li>
</ul>
<p>There are {{filteredStrings.length}} matching results</p>
```

ng-repeat-start und ng-repeat-end

Um mehrere DOM-Elemente durch Definieren eines Start- und Endpunkts zu `ng-repeat-start` können Sie die Direktiven `ng-repeat-start` und `ng-repeat-end` .

```
<ul>
    <li ng-repeat-start="item in [{a: 1, b: 2}, {a: 3, b:4}]">
        {{item.a}}
    </li>
    <li ng-repeat-end>
        {{item.b}}
    </li>
</ul>
```

Ausgabe:

- 1
- 2
- 3
- 4

`ng-repeat-start` mit `ng-repeat-end` .

Variablen

`ng-repeat` macht diese Variablen auch innerhalb des Ausdrucks verfügbar

Variable	Art	Einzelheiten
`$index`	Nummer	Entspricht dem Index der aktuellen Iteration ($ index ==== 0 wird beim ersten iterierten Element als wahr ausgewertet; siehe `$first`).
`$first`	Boolean	Wird beim ersten iterierten Element als wahr ausgewertet
`$last`	Boolean	Bewertet am letzten iterierten Element als wahr
`$middle`	Boolean	Wird als wahr ausgewertet, wenn das Element zwischen `$first` und `$last`
`$even`	Boolean	Wertet bei einer geradzahligen Iteration den `$index%2===0` true (entspricht `$index%2===0`).
`$odd`	Boolean	Wird bei einer ungeradzahligen Iteration als wahr `$index%2===1` (entspricht `$index%2===1`)

Überlegungen zur Leistung

Das Rendern von `ngRepeat` kann langsam werden, besonders wenn Sie große Sammlungen verwenden.

Wenn die Objekte in der Auflistung über eine Bezeichner-Eigenschaft verfügen, sollten Sie immer

`track by` dem Bezeichner statt nach dem gesamten Objekt, der Standardfunktion, nachverfolgen. Wenn keine Kennung vorhanden ist, können Sie immer den integrierten `$index` .

```
<div ng-repeat="item in itemCollection track by item.id">
<div ng-repeat="item in itemCollection track by $index">
```

Umfang von ngRepeat

`ngRepeat` immer einen isolierten `ngRepeat` muss sorgfältig darauf geachtet werden, ob auf den übergeordneten Bereich innerhalb der Wiederholung zugegriffen werden muss.

Hier ein einfaches Beispiel, das zeigt, wie Sie einen Wert in Ihrem übergeordneten Bereich von einem Click-Ereignis in `ngRepeat` .

```
scope val:  {{val}}<br/>
ctrlAs val: {{ctrl.val}}
<ul>
    <li ng-repeat="item in itemCollection">
        <a href="#" ng-click="$parent.val=item.value; ctrl.val=item.value;">
            {{item.label}} {{item.value}}
        </a>
    </li>
</ul>

$scope.val = 0;
this.val = 0;

$scope.itemCollection = [{
    id: 0,
    value: 4.99,
    label: 'Football'
},
{
    id: 1,
    value: 6.99,
    label: 'Baseball'
},
{
    id: 2,
    value: 9.99,
    label: 'Basketball'
}];
```

Wenn bei `ng-click` nur `val = item.value` vorhanden war, wird der `val` im übergeordneten Bereich aufgrund des isolierten Bereichs nicht aktualisiert. Deshalb wird auf den übergeordneten Bereich mit der `$parent` Referenz oder mit der `controllerAs` Syntax zugegriffen (z. B. `ng-controller="mainController as ctrl"`).

Verschachtelte ng-Wiederholung

Sie können auch verschachtelte Wiederholungen verwenden.

```
<div ng-repeat="values in test">
    <div ng-repeat="i in values">
      [{{$parent.$index}},{{$index}}] {{i}}
```

```
    </div>
</div>

var app = angular.module("myApp", []);
app.controller("ctrl", function($scope) {
  $scope.test = [
    ['a', 'b', 'c'],
    ['d', 'e', 'f']
  ];
});
```

Um auf den Index des übergeordneten ng-repeat innerhalb des untergeordneten ng-repeat zuzugreifen, können Sie `$parent.$index` .

ngShow und ngHide

Die Direktive `ng-show` zeigt oder versteckt das HTML-Element basierend darauf, ob der übergebene Ausdruck wahr oder falsch ist. Wenn der Wert des Ausdrucks falsch ist, wird er ausgeblendet. Wenn es wahr ist, wird es sich zeigen.

Die `ng-hide` Anweisung ist ähnlich. Wenn der Wert jedoch falsch ist, wird das HTML-Element angezeigt. Wenn der Ausdruck wahr ist, wird er verborgen.

Arbeitsbeispiel für JSBin

Controller :

```
var app = angular.module('app', []);

angular.module('app')
  .controller('ExampleController', ExampleController);

function ExampleController() {

  var vm = this;

  //Binding the username to HTML element
  vm.username = '';

  //A taken username
  vm.taken_username = 'StackOverflow';

}
```

Aussicht

```
<section ng-controller="ExampleController as main">

    <p>Enter Password</p>
    <input ng-model="main.username" type="text">

    <hr>

    <!-- Will always show as long as StackOverflow is not typed in -->
    <!-- The expression is always true when it is not StackOverflow -->
```

```
    <div style="color:green;" ng-show="main.username != main.taken_username">
      Your username is free to use!
    </div>

    <!-- Will only show when StackOverflow is typed in -->
    <!-- The expression value becomes falsy -->
    <div style="color:red;" ng-hide="main.username != main.taken_username">
      Your username is taken!
    </div>

    <p>Enter 'StackOverflow' in username field to show ngHide directive.</p>

 </section>
```

ngOptions

`ngOptions` ist eine Direktive, die das Erstellen einer HTML-Dropdown-Box für die Auswahl eines Elements aus einem Array vereinfacht, das in einem Modell gespeichert wird. Das ngOptions-Attribut wird verwendet, um dynamisch eine Liste von `<option>` -Elementen für das `<select>` -Element mithilfe des Arrays oder Objekts zu erstellen, das durch Auswertung des ngOptions-Verständnisausdrucks erhalten wird.

Mit `ng-options` das Markup auf ein select-Tag reduziert werden, und die Direktive erstellt das gleiche select:

```
<select ng-model="selectedFruitNgOptions"
        ng-options="curFruit as curFruit.label for curFruit in fruit">
</select>
```

Es gibt anthere Weg zur Schaffung von `select` unter Verwendung `ng-repeat` , aber es wird nicht empfohlen , zu verwenden `ng-repeat` , wie es vor allem für allgemeine Zwecke verwendet wird , wie die `forEach` nur Schleife. Während sich `ng-options` speziell für das Erstellen von `select` Tag `ng-options` eignet.

Das obige Beispiel wäre `ng-repeat`

```
<select ng-model="selectedFruit">
    <option ng-repeat="curFruit in fruit" value="{{curFruit}}">
        {{curFruit.label}}
    </option>
</select>
```

VOLLES BEISPIEL

Schauen wir uns das obige Beispiel auch im Detail an.

Datenmodell für das Beispiel:

```
$scope.fruit = [
    { label: "Apples", value: 4, id: 2 },
    { label: "Oranges", value: 2, id: 1 },
    { label: "Limes", value: 4, id: 4 },
```

```
    { label: "Lemons", value: 5, id: 3 }
];
```

```
<!-- label for value in array -->
<select ng-options="f.label for f in fruit" ng-model="selectedFruit"></select>
```

Bei Auswahl erzeugtes Options-Tag:

```
 <option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Auswirkungen:

`f.label` ist das Label der `<option>` und der Wert enthält das gesamte Objekt.

VOLLES BEISPIEL

```
<!-- select as label for value in array -->
<select ng-options="f.value as f.label for f in fruit" ng-model="selectedFruit"></select>
```

Bei Auswahl erzeugtes Options-Tag:

```
 <option value="4"> Apples </option>
```

Auswirkungen:

`f.value` (4) ist in diesem Fall der Wert, wenn die Beschriftung noch gleich ist.

VOLLES BEISPIEL

```
<!-- label group by group for value in array -->
<select ng-options="f.label group by f.value for f in fruit" ng-
model="selectedFruit"></select>
```

Bei Auswahl erzeugtes Options-Tag:

```
<option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Auswirkungen:

Optionen werden nach ihrem `value` gruppiert. Optionen mit demselben `value` fallen unter eine Kategorie

VOLLES BEISPIEL

```
<!-- label disable when disable for value in array -->
<select ng-options="f.label disable when f.value == 4 for f in fruit" ng-
model="selectedFruit"></select>
```

Bei Auswahl erzeugtes Options-Tag:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Auswirkungen:

"Äpfel" und "Limes" werden deaktiviert (kann nicht ausgewählt werden), weil die Bedingung `disable when f.value==4` . Alle Optionen mit dem `value=4` müssen deaktiviert werden

VOLLES BEISPIEL

```
<!-- label group by group for value in array track by trackexpr -->
<select ng-options="f.value as f.label group by f.value for f in fruit track by f.id" ng-
model="selectedFruit"></select>
```

Bei Auswahl erzeugtes Options-Tag:

```
<option value="4"> Apples </option>
```

Auswirkungen:

Es gibt nicht sichtbare Änderung bei der Verwendung von `trackBy` , aber Angular wird durch die Änderungen erkennen `id` statt durch Bezugnahme , die die meisten ist immer eine bessere Lösung.

VOLLES BEISPIEL

```
<!-- label for value in array | orderBy:orderexpr track by trackexpr -->
<select ng-options="f.label for f in fruit | orderBy:'id' track by f.id" ng-
model="selectedFruit"></select>
```

Bei Auswahl erzeugtes Options-Tag:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Auswirkungen:

`orderBy` ist ein AngularJS-Standardfilter, der die Optionen in aufsteigender Reihenfolge (standardmäßig) anordnet, so dass "Oranges" in diesem ersten Platz erscheint, da seine `id` = 1 ist.

VOLLES BEISPIEL

Alle `<select>` mit `ng-options` müssen mit einem `ng-model` verbunden sein.

ngModel

Mit ng-model können Sie eine Variable an jeden Eingabefeldtyp binden. Sie können die Variable mit doppelten geschweiften Klammern anzeigen, z. B. `{{myAge}}` .

```
<input type="text" ng-model="myName">
<p>{{myName}}</p>
```

Wenn Sie das Eingabefeld eingeben oder auf irgendeine Weise ändern, wird der Wert im Absatz sofort aktualisiert.

In diesem Fall steht die Variable ng-model in Ihrem Controller als `$scope.myName` . Wenn Sie die `controllerAs` Syntax verwenden:

```
<div ng-controller="myCtrl as mc">
    <input type="text" ng-model="mc.myName">
    <p>{{mc.myName}}</p>
</div>
```

Sie müssen auf den Bereich des Controllers verweisen, indem Sie den im Attribut ng-controller definierten Alias des Controllers an die Variable ng-model anhängen. Auf diese Weise müssen Sie nicht `$scope` in Ihren Controller injizieren, um Ihre ng-model-Variable zu referenzieren. Die Variable steht als `this.myName` in der Controller-Funktion zur Verfügung.

ngClass

Nehmen wir an, Sie müssen den Status eines Benutzers anzeigen und haben mehrere mögliche CSS-Klassen, die verwendet werden können. Angular macht es sehr einfach, aus einer Liste mit mehreren möglichen Klassen auszuwählen, mit denen Sie eine Objektliste mit Bedingungen angeben können. Angular kann die richtige Klasse basierend auf der Wahrhaftigkeit der Bedingungen verwenden.

Ihr Objekt sollte Schlüssel / Wert-Paare enthalten. Der Schlüssel ist ein Klassenname, der angewendet wird, wenn der Wert (bedingt) als wahr ausgewertet wird.

```
<style>
    .active { background-color: green; color: white; }
    .inactive { background-color: gray; color: white; }
    .adminUser { font-weight: bold; color: yellow; }
    .regularUser { color: white; }
</style>

<span ng-class="{
    active: user.active,
    inactive: !user.active,
    adminUser: user.level === 1,
    regularUser: user.level === 2
}">John Smith</span>
```

Angular das überprüfen `$scope.user` Objekt das zu sehen , `active` Status und die `level` Nummer. Abhängig von den Werten in diesen Variablen wendet Angular den passenden Stil auf den `<span>` .

ngIf

`ng-if` ist eine Anweisung ähnlich wie `ng-show` , fügt das Element jedoch aus dem DOM ein oder entfernt es, anstatt es einfach auszublenden. In Angular 1.1.5 wurde die ng-If-Direktive eingeführt. Sie können die ng-if-Direktive ab Version 1.1.5 verwenden. Dies ist nützlich, da Angular keine Digests für Elemente innerhalb einer entfernten `ng-if` Angular insbesondere für komplexe Datenbindungen weniger Workload benötigt.

Im Gegensatz zu `ng-show` die Direktive `ng-if` einen untergeordneten Bereich, der prototypische Vererbung verwendet. Das bedeutet, dass das Festlegen eines primitiven Werts für den untergeordneten Bereich nicht für den übergeordneten Bereich gilt. Um ein Grundelement für den übergeordneten Bereich festzulegen, muss die `$parent` Eigenschaft für den untergeordneten Bereich verwendet werden.

JavaScript

```
angular.module('MyApp', []);

angular.module('MyApp').controller('myController', ['$scope', '$window', function
myController($scope, $window) {
    $scope.currentUser= $window.localStorage.getItem('userName');
}]);
```

Aussicht

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

DOM Wenn `currentUser` nicht `currentUser` ist

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <!-- ng-if: !currentUser -->
</div>
```

DOM Wenn `currentUser` ist

```
<div ng-controller="myController">
```

```
    <!-- ng-if: currentUser -->
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

Arbeitsbeispiel

Funktionsversprechen

Die ngIf-Direktive akzeptiert auch Funktionen, die logisch die Rückgabe von true oder false erfordern.

```
<div ng-if="myFunction()">
    <span>Span text</span>
</div>
```

Der Bereichstext wird nur angezeigt, wenn die Funktion true zurückgibt.

```
$scope.myFunction = function() {
    var result = false;
    // Code to determine the boolean value of result
    return result;
};
```

Als Angular-Ausdruck akzeptiert die Funktion jede Art von Variablen.

ngMouseenter und ngMouseleave

Die Direktiven `ng-mouseenter` und `ng-mouseleave` sind nützlich, um Ereignisse auszuführen und das CSS-Styling anzuwenden, wenn Sie sich in Ihren DOM-Elementen befinden.

Die `ng-mouseenter` Direktive führt einen Ausdruck aus, `ng-mouseenter` ein Mausereignis eintritt (wenn der Benutzer den Mauszeiger über das DOM-Element bewegt, in dem sich diese Direktive befindet).

HTML

```
<div ng-mouseenter="applyStyle = true" ng-class="{'active': applyStyle}">
```

Wenn der Benutzer im obigen Beispiel mit der Maus über das `div` , wird `applyStyle` auf `true` , wodurch wiederum die `.active` CSS-Klasse auf die `ng-class` `.active` .

Die `ng-mouseleave` führt einen Ausdruck aus, der ein Mouse-Exit-Ereignis ist.

HTML

```
<div ng-mouseenter="applyStyle = true" ng-mouseleaver="applyStyle = false" ng-
```

```
class="{'active': applyStyle}">
```

Wenn Sie das erste Beispiel erneut verwenden, wird die `.active` Klasse jetzt entfernt, wenn der Benutzer den Mauszeiger aus dem div entfernt.

ngDisabled

Diese Direktive ist nützlich, um Eingabeereignisse basierend auf bestimmten vorhandenen Bedingungen zu begrenzen.

Die Direktive `ng-disabled` akzeptiert einen Ausdruck, der entweder einen Wahrheitswert oder einen Falschwert ergeben soll.

`ng-disabled` wird verwendet, um das `disabled` Attribut bedingt auf ein `input` anzuwenden.

HTML

```
<input type="text" ng-model="vm.name">

<button ng-disabled="vm.name.length===0" ng-click="vm.submitMe">Submit</button>
```

`vm.name.length===0` wird als "true" ausgewertet, wenn die Länge der `input vm.name.length===0` beträgt. `vm.name.length===0` , dass die Schaltfläche deaktiviert wird, sodass der Benutzer das Klickereignis von `ng-click`

ngDblclick

Die Anweisung `ng-dblclick` ist hilfreich, wenn Sie ein Doppelklickereignis in Ihre DOM-Elemente `ng-dblclick` möchten.

Diese Direktive akzeptiert einen Ausdruck

HTML

```
<input type="number" ng-model="num = num + 1" ng-init="num=0">

<button ng-dblclick="num++">Double click me</button>
```

Im obigen Beispiel wird der am `input` gehaltene Wert erhöht, wenn die Schaltfläche doppelt angeklickt wird.

Eingebaute Richtlinien-Spickzettel

`ng-app` Legt den AngularJS-Abschnitt fest.

`ng-init` Legt einen Standardvariablenwert fest.

`ng-bind` Alternative zur {{}} Vorlage.

`ng-bind-template`

Bindet mehrere Ausdrücke an die Ansicht.

`ng-non-bindable` dass die Daten nicht bindbar sind.

`ng-bind-html` Bindet die innere HTML-Eigenschaft eines HTML-Elements.

`ng-change` Wertet den angegebenen Ausdruck aus, wenn der Benutzer die Eingabe ändert.

`ng-checked` Legt das Kontrollkästchen fest.

`ng-class` Legt die CSS-Klasse dynamisch fest.

`ng-cloak` Verhindert die Anzeige des Inhalts, bis AngularJS die Kontrolle übernommen hat.

`ng-click` Führt eine Methode oder einen Ausdruck aus, wenn auf ein Element geklickt wird.

`ng-controller` Fügt der Ansicht eine Controller-Klasse hinzu.

`ng-disabled` Steuert die deaktivierte Eigenschaft des Formularelements

`ng-form` Legt ein Formular fest

`ng-href` Binde AngularJS-Variablen dynamisch an das href-Attribut.

`ng-include` Wird verwendet, um ein externes HTML-Fragment abzurufen, zu kompilieren und in Ihre Seite aufzunehmen.

`ng-if` Abhängig von einem Ausdruck ein Element im DOM entfernen oder neu erstellen

`ng-switch` Bedingtes Umschalten der Steuerung basierend auf übereinstimmenden Ausdrücken.

`ng-model` Bindet Elemente der Eingabe, Auswahl, Textbereiche usw. an die Modelleigenschaft.

`ng-readonly` Wird verwendet, um das readonly-Attribut für ein Element festzulegen.

`ng-repeat` Wird verwendet, um jedes Element in einer Sammlung zu durchlaufen, um eine neue Vorlage zu erstellen.

`ng-selected` Wird verwendet, um die ausgewählte Option im Element festzulegen.

`ng-show/ng-hide` Elemente auf Basis eines Ausdrucks anzeigen / ausblenden.

`ng-src` Binden Sie AngularJS-Variablen dynamisch an das Attribut src.

`ng-submit` submit Bindungswinkelausdrücke für Onsubmit-Ereignisse.

`ng-value` Bindet Winkelausdrücke an den Wert von.

`ng-required` Bindungswinkelausdrücke für Onsubmit-Ereignisse.

`ng-style` Legt den CSS-Stil für ein HTML-Element fest.

`ng-pattern` Fügt den Musterprüfer zu ngModel hinzu.

`ng-maxlength` Fügt den maxlength-Validator zu ngModel hinzu.

`ng-minlength` Fügt den Minlength-Validator zu ngModel hinzu.

`ng-classeven` Funktioniert in Verbindung mit ngRepeat und wird nur in ungeraden (geraden) Zeilen wirksam.

`ng-classodd` Funktioniert in Verbindung mit ngRepeat und wird nur für ungerade (gerade) Zeilen wirksam.

`ng-cut` Wird verwendet, um ein benutzerdefiniertes Verhalten beim Ausschneidereignis anzugeben.

`ng-copy` Wird verwendet, um ein benutzerdefiniertes Verhalten beim Kopierereignis anzugeben.

`ng-paste` Wird verwendet, um ein benutzerdefiniertes Verhalten beim Einfügeereignis anzugeben.

`ng-options` Wird verwendet, um eine Liste von Elementen für das Element dynamisch zu generieren.

`ng-list` Wird verwendet, um einen String in eine Liste umzuwandeln, die auf dem angegebenen Trennzeichen basiert.

`ng-open` Wird verwendet, um das Attribut open für das Element festzulegen, wenn der Ausdruck in ngOpen wahr ist.

Quelle (etwas bearbeitet)

ngClick

Die Anweisung `ng-click` fügt einem DOM-Element ein Ereignis click hinzu.

Mit `ng-click` Anweisung `ng-click` können Sie ein benutzerdefiniertes Verhalten angeben, wenn auf ein Element von DOM geklickt wird.

Dies ist nützlich, wenn Sie Klickereignisse an Schaltflächen anhängen und an Ihrem Controller behandeln möchten.

Diese Direktive akzeptiert einen Ausdruck mit dem als `$event` verfügbaren `$event`

HTML

```
<input ng-click="onClick($event)">Click me</input>
```

Regler

```
.controller("ctrl", function($scope) {
    $scope.onClick = function(evt) {
        console.debug("Hello click event: %o ",evt);
```

```
    }
})
```

HTML

```
<button ng-click="count = count + 1" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

HTML

```
<button ng-click="count()" ng-init="count=0">
  Increment
</button>
<span>
  count: {{count}}
</span>
```

Regler

```
...

$scope.count = function(){
    $scope.count = $scope.count + 1;
}
...
```

Wenn Sie auf die Schaltfläche `onClick` , wird beim `onClick` Funktion `onClick` "Hallo `onClick` " gefolgt vom Ereignisobjekt gedruckt.

ngRequired

`ng-required` fügt das `required` Validierungsattribut einem Element hinzu oder entfernt es. Dadurch wird der `require` Validierungsschlüssel für die `input` aktiviert und deaktiviert.

Es wird verwendet, um optional festzulegen, ob ein `input` einen nicht leeren Wert haben muss. Die Direktive ist hilfreich beim Entwerfen der Validierung komplexer HTML-Formulare.

HTML

```
<input type="checkbox" ng-model="someBooleanValue">
<input type="text" ng-model="username" ng-required="someBooleanValue">
```

ng-Modell-Optionen

`ng-model-options` können Sie das Standardverhalten von `ng-model` ändern. Mit dieser Anweisung können Sie Ereignisse registrieren, die ausgelöst werden, wenn das ng-model aktualisiert wird, und einen Entprellungseffekt hinzufügen.

Diese Direktive akzeptiert einen Ausdruck, der ein Definitionsobjekt oder einen Verweis auf einen Bereichswert auswertet.

Beispiel:

```
<input type="text" ng-model="myValue" ng-model-options="{'debounce': 500}">
```

Das obige Beispiel fügt einen Debounce-Effekt von 500 Millisekunden für `myValue` , wodurch das Modell 500 ms aktualisiert wird, nachdem der Benutzer die `input` (das heißt, wenn die Aktualisierung von `myValue` abgeschlossen ist).

Verfügbare Objekteigenschaften

1. `updateOn` : `updateOn` an, welches Ereignis an die Eingabe gebunden werden soll

   ```
   ng-model-options="{ updateOn: 'blur'}" // will update on blur
   ```

2. `debounce` : `debounce` eine Verzögerung von einigen Millisekunden gegenüber dem Modellupdate an

   ```
   ng-model-options="{'debounce': 500}" // will update the model after 1/2 second
   ```

3. `allowInvalid` : Ein boolesches Flag, das einen ungültigen Wert für das Modell ermöglicht, wodurch die Standard-Formularvalidierung umgangen wird. Standardmäßig werden diese Werte als `undefined` behandelt.

4. `getterSetter` : Ein boolesches Flag, das angibt, ob das `ng-model` als Getter / Setter-Funktion anstelle eines einfachen Modellwerts behandelt werden soll. Die Funktion wird dann ausgeführt und gibt den Modellwert zurück.

   Beispiel:

   ```
   <input type="text" ng-model="myFunc" ng-model-options="{'getterSetter': true}">

   $scope.myFunc = function() {return "value";}
   ```

5. `timezone` : Definiert die Zeitzone für das Modell, wenn `date` oder `time` eingegeben `time` . Typen

ngCloak

Die `ngCloak` Direktive wird verwendet, um zu verhindern, dass die Angular-HTML-Vorlage vom Browser kurz in ihrem rohen (nicht kompilierten) Formular angezeigt wird, während Ihre Anwendung geladen wird. - Quelle anzeigen

HTML

```
<div ng-cloak>
  <h1>Hello {{ name }}</h1>
</div>
```

`ngCloak` kann auf das body-Element angewendet werden. Die bevorzugte Anwendung besteht jedoch darin, mehrere ngCloak-Anweisungen auf kleine Bereiche der Seite anzuwenden, um eine progressive Wiedergabe der Browseransicht zu ermöglichen.

Die `ngCloak` Direktive enthält keine Parameter.

Siehe auch: Verhindern des Flimmerns

ngInclude

Mit ng-include können Sie die Steuerung eines Teils der Seite an einen bestimmten Controller delegieren. Möglicherweise möchten Sie dies tun, weil die Komplexität dieser Komponente so wird, dass Sie die gesamte Logik in einem dedizierten Controller kapseln möchten.

Ein Beispiel ist:

```
  <div ng-include
       src="'/gridview'"
       ng-controller='gridController as gc'>
  </div>
```

Beachten Sie, dass der `/gridview` vom Webserver als eindeutige und legitime URL `/gridview` werden muss.

Beachten Sie auch, dass das `src` -attribute einen Angular-Ausdruck akzeptiert. Dies kann beispielsweise eine Variable oder ein Funktionsaufruf sein oder, wie in diesem Beispiel, eine String-Konstante. In diesem Fall müssen Sie sicherstellen, dass **die Quell-URL in einfache Anführungszeichen gesetzt wird** , damit sie als String-Konstante ausgewertet wird. Dies ist eine häufige Verwirrung.

In der `/gridview` HTML- `/gridview` können Sie auf den `gridController` so verweisen, als ob er um die Seite gewickelt wäre, zB:

```
<div class="row">
  <button type="button" class="btn btn-default" ng-click="gc.doSomething()"></button>
</div>
```

ngSrc

Die Verwendung eines Angular-Markups wie `{{hash}}` in einem SRC-Attribut funktioniert nicht richtig. Der Browser ruft die URL mit dem wörtlichen Text `{{hash}}` bis Angular den Ausdruck in `{{hash}}` . `ng-src` Direktive `ng-src` überschreibt das ursprüngliche `src` Attribut für das Image-Tag-Element und löst das Problem

```
<div ng-init="pic = 'pic_angular.jpg'">
    <h1>Angular</h1>
    <img ng-src="{{pic}}">
</div>
```

ngPattern

Die Anweisung `ng-pattern` akzeptiert einen Ausdruck, der ein reguläres Ausdrucksmuster ergibt, und verwendet dieses Muster, um eine Texteingabe zu überprüfen.

Beispiel:

Nehmen wir an, wir möchten, dass ein `<input>` -Element gültig wird, wenn sein Wert (ng-model) eine gültige IP-Adresse ist.

Vorlage:

```
<input type="text" ng-model="ipAddr" ng-pattern="ipRegex" name="ip" required>
```

Regler:

```
$scope.ipRegex = /\b(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b/;
```

ngValue

Wird meist unter `ng-repeat` NgValue ist nützlich, wenn Sie Listen von Optionsfeldern mit ngRepeat dynamisch generieren

```
<script>
   angular.module('valueExample', [])
     .controller('ExampleController', ['$scope', function($scope) {
       $scope.names = ['pizza', 'unicorns', 'robots'];
       $scope.my = { favorite: 'unicorns' };
     }]);
</script>
 <form ng-controller="ExampleController">
   <h2>Which is your favorite?</h2>
     <label ng-repeat="name in names" for="{{name}}">
       {{name}}
       <input type="radio"
              ng-model="my.favorite"
              ng-value="name"
              id="{{name}}"
              name="favorite">
     </label>
   <div>You chose {{my.favorite}}</div>
 </form>
```

Arbeits plnkr

ngCopy

Die Direktive `ngCopy` gibt das Verhalten an, das für ein `ngCopy` werden soll.

Verhindern, dass ein Benutzer Daten kopiert

```
<p ng-copy="blockCopy($event)">This paragraph cannot be copied</p>
```

In der Steuerung

```
$scope.blockCopy = function(event) {
    event.preventDefault();
    console.log("Copying won't work");
}
```

ngPaste

Die Anweisung `ngPaste` gibt ein benutzerdefiniertes Verhalten an, das ausgeführt werden soll, wenn ein Benutzer Inhalt `ngPaste`

```
<input ng-paste="paste=true" ng-init="paste=false" placeholder='paste here'>
pasted: {{paste}}
```

ngHref

ngHref wird anstelle des href-Attributs verwendet, wenn im href-Wert ein Winkelausdruck vorliegt. Die Direktive ngHref überschreibt das ursprüngliche href-Attribut eines html-Tags mit dem href-Attribut, z. B. tag, tag usw.

Die Anweisung ngHref stellt sicher, dass die Verknüpfung nicht unterbrochen wird, auch wenn der Benutzer auf die Verknüpfung klickt, bevor AngularJS den Code ausgewertet hat.

Beispiel 1

```
<div ng-init="linkValue = 'http://stackoverflow.com'">
    <p>Go to <a ng-href="{{linkValue}}">{{linkValue}}</a>!</p>
</div>
```

Beispiel 2 In diesem Beispiel wird der Href-Wert dynamisch aus dem Eingabefeld abgerufen und als Href-Wert geladen.

```
<input ng-model="value" />
<a id="link" ng-href="{{value}}">link</a>
```

Beispiel 3

```
<script>
angular.module('angularDoc', [])
.controller('myController', function($scope) {
  // Set some scope value.
  // Here we set bootstrap version.
  $scope.bootstrap_version = '3.3.7';

  // Set the default layout value
  $scope.layout = 'normal';
});
</script>
```

```
<!-- Insert it into Angular Code -->
<link rel="stylesheet" ng-href="//maxcdn.bootstrapcdn.com/bootstrap/{{ bootstrap_version
}}/css/bootstrap.min.css">
<link rel="stylesheet" ng-href="layout-{{ layout }}.css">
```

ngList

Die `ng-list` Direktive wird verwendet, um eine begrenzte Zeichenfolge aus einer Texteingabe in ein String-Array oder umgekehrt zu konvertieren.

Die `ng-list` Direktive verwendet ein Standardtrennzeichen von `", "` (Komma-Leerzeichen).

Sie können das Trennzeichen manuell festlegen, indem Sie `ng-list` ein Trennzeichen wie dieses zuweisen. `ng-list="; "` .

In diesem Fall wird das Trennzeichen auf ein Semikolon gefolgt von einem Leerzeichen gesetzt.

Standardmäßig hat `ng-list` ein Attribut `ng-trim` das auf true gesetzt ist. `ng-trim` bei false wird der Leerraum in Ihrem Trennzeichen berücksichtigt. Standardmäßig berücksichtigt `ng-list` keine Leerräume, es sei denn, Sie legen `ng-trim="false"` .

Beispiel:

```
angular.module('test', [])
  .controller('ngListExample', ['$scope', function($scope) {
    $scope.list = ['angular', 'is', 'cool!'];
}]);
```

Ein Kundenbegrenzer wird gesetzt `;` . Das Modell des Eingabefelds wird auf das Array festgelegt, das für den Bereich erstellt wurde.

```
  <body ng-app="test" ng-controller="ngListExample">
    <input ng-model="list" ng-list="; " ng-trim="false">
  </body>
```

Das Eingabefeld wird mit dem Inhalt angezeigt: `angular; is; cool!`

Eingebaute Richtlinien online lesen: https://riptutorial.com/de/angularjs/topic/706/eingebaute-richtlinien

Kapitel 24: Faules Laden

Bemerkungen

1. Wenn Ihre Lazy-Load-Abhängigkeiten andere Lazy-Load-Abhängigkeiten erfordern, müssen Sie sie in der richtigen Reihenfolge laden!

```
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 {
  files: [
  'path/to/lazily/loaded/dependency1.js',
  'path/to/lazily/loaded/dependency2.js', //<--- requires lazily loaded dependency1
  'path/to/lazily/loaded/dependency.css'
  ],
  serie: true //Sequential load instead of parallel
 }
]);
```

Examples

Vorbereiten Ihres Projekts für das langsame Laden

Nachdem Sie `oclazyload.js` in Ihre Indexdatei aufgenommen haben, deklarieren Sie `ocLazyLoad` als Abhängigkeit in `app.js`

```
//Make sure you put the correct dependency! it is spelled different than the service!
angular.module('app', [
 'oc.lazyLoad',
 'ui-router'
])
```

Verwendungszweck

Um Dateien `$ocLazyLoad` zu laden, `$ocLazyLoad` den `$ocLazyLoad` Dienst in einen Controller oder einen anderen Dienst ein

```
.controller('someCtrl', function($ocLazyLoad) {
 $ocLazyLoad.load('path/to/file.js').then(...);
});
```

Winkelmodule werden automatisch in Winkel geladen.

Andere Variante:

```
$ocLazyLoad.load([
  'bower_components/bootstrap/dist/js/bootstrap.js',
```

```
  'bower_components/bootstrap/dist/css/bootstrap.css',
  'partials/template1.html'
]);
```

Eine vollständige Liste der Varianten finden Sie in der offiziellen Dokumentation

Verwendung mit Router

UI-Router:

```
.state('profile', {
 url: '/profile',
 controller: 'profileCtrl as vm'
 resolve: {
  module: function($ocLazyLoad) {
   return $ocLazyLoad.load([
    'path/to/profile/module.js',
    'path/to/profile/style.css'
   ]);
  }
 }
});
```

ngRoute:

```
.when('/profile', {
     controller: 'profileCtrl as vm'
     resolve: {
      module: function($ocLazyLoad) {
       return $ocLazyLoad.load([
        'path/to/profile/module.js',
        'path/to/profile/style.css'
       ]);
      }
     }
    });
```

Abhängigkeitsinjektion verwenden

Mit der folgenden Syntax können Sie bei der Verwendung des Diensts Abhängigkeiten in Ihrer `module.js` anstelle einer expliziten Angabe angeben

```
//lazy_module.js
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 [
  'path/to/lazily/loaded/dependency.js',
  'path/to/lazily/loaded/dependency.css'
 ]
]);
```

Hinweis : Diese Syntax funktioniert nur für langsam geladene Module!

Verwendung der Direktive

```
<div oc-lazy-load="['path/to/lazy/loaded/directive.js',
'path/to/lazy/loaded/directive.html']">

<!-- myDirective available here -->
<my-directive></my-directive>

</div>
```

Faules Laden online lesen: https://riptutorial.com/de/angularjs/topic/6400/faules-laden

Kapitel 25: Filter

Examples

Ihr erster Filter

Bei Filtern handelt es sich um eine spezielle Art von Funktion, mit der die Art und Weise geändert werden kann, wie etwas auf die Seite gedruckt wird, oder um ein Array oder eine `ng-repeat` zu filtern. Sie können einen Filter erstellen, indem Sie die `app.filter()` -Methode aufrufen und einen Namen und eine Funktion übergeben. In den Beispielen unten finden Sie Details zur Syntax.

Erstellen wir zum Beispiel einen Filter, der eine Zeichenfolge in Großbuchstaben ändert (im Wesentlichen ein Wrapper der Javascript-Funktion `.toUpperCase()`):

```
var app = angular.module("MyApp", []);

// just like making a controller, you must give the
// filter a unique name, in this case "toUppercase"
app.filter('toUppercase', function(){
    // all the filter does is return a function,
    // which acts as the "filtering" function
    return function(rawString){
        // The filter function takes in the value,
        // which we modify in some way, then return
        // back.
        return rawString.toUpperCase();
    };
});
```

Schauen wir uns das an, was oben passiert.

Zunächst erstellen wir einen Filter namens "toUppercase", der einem Controller ähnelt. `app.filter(...)` . Dann gibt die Funktion dieses Filters die tatsächliche Filterfunktion zurück. Diese Funktion benötigt ein einzelnes Objekt, das das zu filternde Objekt ist, und sollte die gefilterte Version des Objekts zurückgeben.

Hinweis: *In dieser Situation nehmen wir an, dass das Objekt, das an den Filter übergeben wird, eine Zeichenfolge ist, und weiß daher, dass der Filter immer nur für Zeichenfolgen verwendet wird. Eine weitere Verbesserung des Filters könnte jedoch vorgenommen werden, die das Objekt durchläuft (wenn es sich um ein Array handelt) und dann jedes Element, das eine Zeichenfolge ist, in Großbuchstaben setzt.*

Jetzt nutzen wir unseren neuen Filter in Aktion. Unser Filter kann auf zwei Arten verwendet werden, entweder in einer Winkelvorlage oder als Javascript-Funktion (als eingespritzter Winkelreferenz).

Javascript

Fügen Sie einfach das angle `$filter` Objekt in Ihren Controller ein, und rufen Sie dann die Filterfunktion mit ihrem Namen ab.

```
app.controller("MyController", function($scope, $filter){
    this.rawString = "Foo";
    this.capsString = $filter("toUppercase")(this.rawString);
});
```

HTML

Verwenden Sie für eine Winkeldirektive das Pipe-Symbol (`|`), gefolgt von dem Filternamen in der Direktive nach der eigentlichen Zeichenfolge. `MyController` wir beispielsweise an, wir haben einen Controller namens `MyController` , der als Element einen String namens `rawString` .

```
<div ng-controller="MyController as ctrl">
    <span>Capital rawString: {{ ctrl.rawString | toUppercase }}</span>
</div>
```

Anmerkung des Editors: *Angular verfügt über eine Reihe integrierter Filter, einschließlich "Großbuchstaben". Der Filter "toUppercase" ist nur als Demo gedacht, um die Funktionsweise von Filtern zu demonstrieren. Sie müssen* jedoch *keine eigene Großbuchstabenfunktion erstellen.*

Benutzerdefinierter Filter zum Entfernen von Werten

Ein typischer Anwendungsfall für einen Filter ist das Entfernen von Werten aus einem Array. In diesem Beispiel übergeben wir ein Array, entfernen alle darin gefundenen Nullen und geben das Array zurück.

```
function removeNulls() {
    return function(list) {
        for (var i = list.length - 1; i >= 0; i--) {
            if (typeof list[i] === 'undefined' ||
                    list[i] === null) {
                list.splice(i, 1);
            }
        }
        return list;
    };
}
```

Das würde gerne im HTML verwendet

```
{{listOfItems | removeNulls}}
```

oder in einem Controller wie

```
listOfItems = removeNullsFilter(listOfItems);
```

Benutzerdefinierter Filter zum Formatieren von Werten

Ein weiterer Anwendungsfall für Filter ist das Formatieren eines einzelnen Werts. In diesem Beispiel übergeben wir einen Wert, und es wird ein passender wahrer boolescher Wert zurückgegeben.

```
function convertToBooleanValue() {
    return function(input) {
        if (typeof input !== 'undefined' &&
                input !== null &&
                (input === true || input === 1 || input === '1' || input
                        .toString().toLowerCase() === 'true')) {
            return true;
        }
        return false;
    };
}
```

Welches im HTML würde so verwendet werden:

```
{{isAvailable | convertToBooleanValue}}
```

Oder in einem Controller wie:

```
var available = convertToBooleanValueFilter(isAvailable);
```

Filter in einem untergeordneten Array ausführen

In diesem Beispiel wurde gezeigt, wie Sie einen Tiefenfilter in einem *untergeordneten* Array ausführen können, ohne dass ein benutzerdefinierter Filter erforderlich ist.

Regler:

```
(function() {
  "use strict";
  angular
    .module('app', [])
    .controller('mainCtrl', mainCtrl);

  function mainCtrl() {
    var vm = this;

    vm.classifications = ["Saloons", "Sedans", "Commercial vehicle", "Sport car"];
    vm.cars = [
       {
          "name":"car1",
          "classifications":[
             {
                "name":"Saloons"
             },
             {
                "name":"Sedans"
             }
          ]
```

```
        },
        {
           "name":"car2",
           "classifications":[
              {
                 "name":"Saloons"
              },
              {
                 "name":"Commercial vehicle"
              }
           ]
        },
        {
           "name":"car3",
           "classifications":[
              {
                 "name":"Sport car"
              },
              {
                 "name":"Sedans"
              }
           ]
        }
    ];
  }
})();
```

Aussicht:

```
<body ng-app="app" ng-controller="mainCtrl as main">
  Filter car by classification:
  <select ng-model="classificationName"
          ng-options="classification for classification in main.classifications"></select>
  <br>
  <ul>
    <li ng-repeat="car in main.cars |
                   filter: { classifications: { name: classificationName } } track by $index"
                  ng-bind-template="{{car.name}} - {{car.classifications | json}}">
    </li>
  </ul>
</body>
```

Überprüfen Sie die vollständige DEMO .

Verwenden von Filtern in einer Steuerung oder einem Dienst

Durch das Injizieren von `$filter` kann jeder definierte Filter in Ihrem Angular-Modul in Controllern, Diensten, Anweisungen oder sogar anderen Filtern verwendet werden.

```
angular.module("app")
  .service("users", usersService)
  .controller("UsersController", UsersController);

function usersService () {
  this.getAll = function () {
    return [{
      id: 1,
```

```
      username: "john"
    }, {
      id: 2,
      username: "will"
    }, {
      id: 3,
      username: "jack"
    }];
  };
}

function UsersController ($filter, users) {
  var orderByFilter = $filter("orderBy");

  this.users = orderByFilter(users.getAll(), "username");
  // Now the users are ordered by their usernames: jack, john, will

  this.users = orderByFilter(users.getAll(), "username", true);
  // Now the users are ordered by their usernames, in reverse order: will, john, jack
}
```

Zugriff auf eine gefilterte Liste außerhalb einer Wiederholungswiederholung

Gelegentlich möchten Sie auf das Ergebnis Ihrer Filter außerhalb der `ng-repeat` zugreifen, um beispielsweise die Anzahl der ausgefilterten Elemente anzugeben. Sie können dies `as [variablename]` -Syntax für das `ng-repeat` .

```
<ul>
  <li ng-repeat="item in vm.listItems | filter:vm.myFilter as filtered">
    {{item.name}}
  </li>
</ul>
<span>Showing {{filtered.length}} of {{vm.listItems.length}}</span>
```

Filter online lesen: https://riptutorial.com/de/angularjs/topic/1401/filter

Kapitel 26: Formularüberprüfung

Examples

Grundlegende Formularvalidierung

Eine der Stärken von Angular ist die clientseitige Formularvalidierung.

Der Umgang mit herkömmlichen Formulareingaben und die Verwendung von abfragender jQuery-Verarbeitung ist zeitaufwändig und umständlich. Mit Angular können Sie relativ einfach professionelle *interaktive* Formulare erstellen.

Die **ng-model-** Direktive bietet eine **bidirektionale** Bindung mit Eingabefeldern. Normalerweise wird das Attribut **novalidate** auch im Formularelement platziert, um zu verhindern, dass der Browser die native Überprüfung durchführt.

Ein einfaches Formular würde also so aussehen:

```
<form name="form" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
</form>
```

Verwenden Sie für die Überprüfung der Eingaben von Angular genau die gleiche Syntax wie ein reguläres *Eingabeelement* , mit Ausnahme des Zusatzes **ng-model** , um anzugeben, an welche Variable im Bereich gebunden werden soll. E-Mail wird im vorherigen Beispiel gezeigt. Um eine Zahl zu bestätigen, würde die Syntax lauten:

```
<input type="number" name="postalcode" ng-model="zipcode" />
```

Die letzten Schritte zur grundlegenden Formularüberprüfung bestehen darin, eine Verbindung zu einer Formularübergabefunktion auf dem Controller mithilfe von **ng-submit** herzustellen, anstatt das Standardformular absenden zuzulassen. Dies ist nicht obligatorisch, wird jedoch in der Regel verwendet, da die Eingabevariablen bereits im Gültigkeitsbereich vorhanden sind und somit Ihrer Übermittlungsfunktion zur Verfügung stehen. In der Regel empfiehlt es sich, der Form einen Namen zu geben. Diese Änderungen würden zu folgender Syntax führen:

```
<form name="signup_form" ng-submit="submitFunc()" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
  <button type="submit">Signup</button>
</form>
```

Der obige Code ist funktionsfähig, aber es gibt andere Funktionen, die Angular bietet.

Der nächste Schritt besteht darin, zu verstehen, dass Angular Klassenattribute mit **ng-pristine** , **ng-dirty** , **ng-valid** und **ng-invalid** für die Formularverarbeitung anfügt. Wenn Sie diese Klassen

in Ihrer CSS-Datei verwenden, können Sie **gültige / ungültige** und **unverfälschte / fehlerhafte** Eingabefelder formatieren und so die Präsentation ändern, wenn der Benutzer Daten in das Formular eingibt.

Form- und Eingabezustände

Winkelformen und Eingaben weisen verschiedene Status auf, die bei der Überprüfung von Inhalten hilfreich sind

Eingabezustände

Zustand	Beschreibung
`$touched`	Feld wurde berührt
`$untouched`	Feld wurde nicht berührt
`$pristine`	Feld wurde nicht geändert
`$dirty`	Feld wurde geändert
`$valid`	Feldinhalt ist gültig
`$invalid`	Feldinhalt ist ungültig

Alle oben genannten Zustände sind boolesche Eigenschaften und können entweder wahr oder falsch sein.

Mit diesen ist es sehr einfach, einem Benutzer Nachrichten anzuzeigen.

```
<form name="myForm" novalidate>
    <input name="myName" ng-model="myName" required>
    <span ng-show="myForm.myName.$touched && myForm.myName.$invalid">This name is
invalid</span>
</form>
```

Hier verwenden wir die Anweisung `ng-show` , um einem Benutzer eine Nachricht anzuzeigen, wenn er ein Formular geändert hat, das aber ungültig ist.

CSS-Klassen

Angular stellt abhängig vom Status auch einige CSS-Klassen für Formulare und Eingaben bereit

Klasse	Beschreibung
`ng-touched`	Feld wurde berührt
`ng-untouched`	Feld wurde nicht berührt
`ng-pristine`	Feld wurde nicht geändert

Klasse	Beschreibung
`ng-dirty`	Feld wurde geändert
`ng-valid`	Feld ist gültig
`ng-invalid`	Feld ist ungültig

Sie können diese Klassen verwenden, um Ihren Formularen Formatvorlagen hinzuzufügen

```
input.ng-invalid {
  background-color: crimson;
}
input.ng-valid {
  background-color: green;
}
```

ngNachrichten

`ngMessages` wird verwendet, um den Stil für die Anzeige von Überprüfungsmeldungen in der Ansicht zu verbessern.

Traditioneller Ansatz

Vor `ngMessages` wir die Validierungsnachrichten normalerweise mit vordefinierten Anweisungen von Angular `ng-class` Dieser Ansatz war Wurf und eine sich `repetitive` Aufgabe.

Mit `ngMessages` wir jetzt eigene benutzerdefinierte Nachrichten erstellen.

Beispiel

Html:

```
<form name="ngMessagesDemo">
  <input name="firstname" type="text" ng-model="firstname" required>
  <div ng-messages="ngMessagesDemo.firstname.$error">
    <div ng-message="required">Firstname is required.</div>
  </div>
</form>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.3.16/angular-
messages.min.js"></script>
```

JS:

```
var app = angular.module('app', ['ngMessages']);

app.controller('mainCtrl', function ($scope) {
```

```
    $scope.firstname = "Rohit";
});
```

Benutzerdefinierte Formularüberprüfung

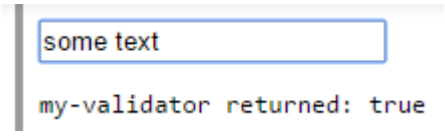
In einigen Fällen reicht die grundlegende Validierung nicht aus. Angular unterstützt die benutzerdefinierte Validierung, um dem `$validators` `ngModelController` Objekt auf dem `ngModelController` `$validators` `ngModelController` :

```
angular.module('app', [])
  .directive('myValidator', function() {
    return {
      // element must have ng-model attribute
      // or $validators does not work
      require: 'ngModel',
      link: function(scope, elm, attrs, ctrl) {
        ctrl.$validators.myValidator = function(modelValue, viewValue) {
            // validate viewValue with your custom logic
            var valid = (viewValue && viewValue.length > 0) || false;
            return valid;
        };
      }
    };
```

Der Validator ist als eine Direktive definiert, für die `ngModel` erforderlich ist. `ngModel` den Validator anwenden `ngModel` , fügen Sie einfach die benutzerdefinierte Direktive zum Eingabeformularsteuerelement hinzu.

```
<form name="form">
  <input type="text"
         ng-model="model"
         name="model"
         my-validator>
  <pre ng-bind="'my-validator returned: ' + form.model.$valid"></pre>
</form>
```

Und `my-validator` muss nicht auf die native Formularsteuerung angewendet werden. Es kann ein beliebiges Element sein, solange es in seinen Attributen ein `ng-model` . Dies ist nützlich, wenn Sie über eine benutzerdefinierte UI-Komponente verfügen.



Verschachtelte Formulare

Manchmal ist es wünschenswert, Formulare zu verschachteln, um Steuerelemente und Eingaben logisch auf der Seite zu gruppieren. HTML5-Formulare sollten jedoch nicht geschachtelt werden. Winkel liefert stattdessen `ng-form` .

```
<form name="myForm" noValidate>
```

```
  <!-- nested form can be referenced via 'myForm.myNestedForm' -->
  <ng-form name="myNestedForm" noValidate>
    <input name="myInput1" ng-minlength="1" ng-model="input1" required />
    <input name="myInput2" ng-minlength="1" ng-model="input2" required />
  </ng-form>

  <!-- show errors for the nested subform here -->
  <div ng-messages="myForm.myNestedForm.$error">
    <!-- note that this will show if either input does not meet the minimum -->
    <div ng-message="minlength">Length is not at least 1</div>
  </div>
</form>

<!-- status of the form -->
<p>Has any field on my form been edited? {{myForm.$dirty}}</p>
<p>Is my nested form valid? {{myForm.myNestedForm.$valid}}</p>
<p>Is myInput1 valid? {{myForm.myNestedForm.myInput1.$valid}}</p>
```

Jeder Teil des Formulars trägt zum Gesamtzustand des Formulars bei. Wenn also einer der Eingaben `myInput1` bearbeitet wurde und `$dirty` , ist das enthaltene Formular ebenfalls `$dirty` . `myNestedForm` dies zu jedem Formular führt, werden sowohl `myNestedForm` als auch `myForm` `$dirty` .

Async-Validatoren

Mit asynchronen Validatoren können Sie Formularinformationen gegen Ihr Backend überprüfen (mithilfe von $ http).

Diese Art von Validatoren wird benötigt, wenn Sie auf vom Server gespeicherte Informationen zugreifen müssen, die Sie aus verschiedenen Gründen nicht auf Ihrem Client haben können, z.

Um async-Validatoren zu verwenden, greifen Sie auf das `ng-model` Ihrer `input` und definieren Callback-Funktionen für die `$asyncValidators` Eigenschaft.

Beispiel:

Im folgenden Beispiel wird geprüft, ob ein angegebener Name bereits vorhanden ist. Das Backend gibt einen Status zurück, der das Versprechen ablehnt, wenn der Name bereits existiert oder nicht angegeben wurde. Wenn der Name nicht existiert, wird ein gelöstes Versprechen zurückgegeben.

```
ngModel.$asyncValidators.usernameValidate = function (name) {
     if (name) {
          return AuthenticationService.checkIfNameExists(name); // returns a promise
     } else {
          return $q.reject("This username is already taken!"); // rejected promise
     }
};
```

Bei jeder Änderung des `ng-model` der Eingabe wird diese Funktion ausgeführt und ein Versprechen mit dem Ergebnis zurückgegeben.

Formularüberprüfung online lesen:
https://riptutorial.com/de/angularjs/topic/3979/formularuberprufung

Kapitel 27: Grunzen Sie Aufgaben

Examples

Anwendung lokal ausführen

> Das folgende Beispiel setzt voraus , dass node.js installiert ist und npm verfügbar ist. Vollständiger Arbeitscode kann von GitHub @ https://github.com/mikkoviitala/angular-grunt-run-local abgeleitet werden

Normalerweise ist es eines der ersten Dinge, die Sie beim Entwickeln einer neuen Webanwendung tun möchten, wenn diese lokal ausgeführt wird.

Im Folgenden finden Sie ein vollständiges Beispiel, das genau dies erreicht, indem Sie grunt (Javascript Task Runner), npm (Node Package Manager) und Bower (noch einen weiteren Package Manager) verwenden.

Neben den eigentlichen Anwendungsdateien müssen Sie mit den oben genannten Tools einige Abhängigkeiten von Drittanbietern installieren. In Ihrem Projektverzeichnis, **vorzugsweise root** , benötigen Sie drei (3) Dateien.

- package.json (von npm verwaltete Abhängigkeiten)
- bower.json (von bower verwaltete Abhängigkeiten)
- gruntfile.js (grunzende Aufgaben)

So sieht Ihr Projektverzeichnis so aus:

---

package.json

Wir installieren **grunt** selbst, **matchdep** , um unser Leben einfacher zu machen, sodass wir Abhängigkeiten nach Namen filtern können. **Grunt-express** wird zum Starten des Express-Webservers über grunt und **grunt-open** zum Öffnen von URLs / Dateien aus einem Grunt-Task verwendet.

Bei diesen Paketen handelt es sich also um "Infrastruktur" und um Helfer, auf denen wir unsere Anwendung aufbauen werden.

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {},
```

```
  "devDependencies": {
    "grunt": "~0.4.1",
    "matchdep": "~0.1.2",
    "grunt-express": "~1.0.0-beta2",
    "grunt-open": "~0.2.1"
  },
  "scripts": {
    "postinstall": "bower install"
  }
}
```

bower.json

Bower ist (oder sollte es zumindest sein) über Front-End und wir werden es verwenden, um **eckig** zu installieren.

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {
    "angular": "~1.3.x"
  },
  "devDependencies": {}
}
```

gruntfile.js

In gruntfile.js haben wir die eigentliche "lokal ausgeführte Anwendung" -Magie, die unsere Anwendung in einem neuen Browserfenster öffnet und unter http: // localhost: 9000 / läuft .

```
'use strict';

// see http://rhumaric.com/2013/07/renewing-the-grunt-livereload-magic/

module.exports = function(grunt) {
  require('matchdep').filterDev('grunt-*').forEach(grunt.loadNpmTasks);

  grunt.initConfig({
    express: {
      all: {
        options: {
          port: 9000,
          hostname: 'localhost',
          bases: [__dirname]
        }
      }
    },

    open: {
      all: {
        path: 'http://localhost:<%= express.all.options.port%>'
      }
    }
  });

  grunt.registerTask('app', [
    'express',
```

```
    'open',
    'express-keepalive'
  ]);
};
```

Verwendungszweck

Speichern Sie die oben genannten Dateien im Stammverzeichnis des Projekts (alle leeren Ordner können dies tun), um Ihre Anwendung von Grund auf zu starten. Starten Sie dann die Konsole / Befehlszeile und geben Sie Folgendes ein, um alle erforderlichen Abhängigkeiten zu installieren.

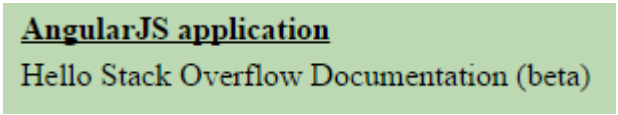
```
npm install -g grunt-cli bower
npm install
```

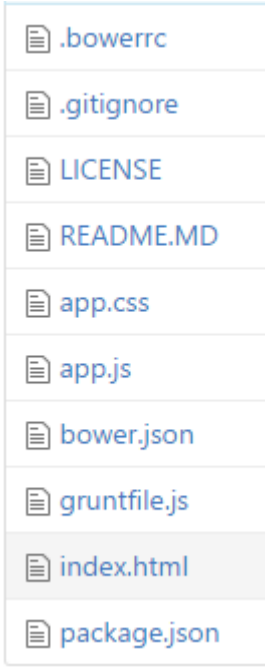
Führen Sie dann Ihre Anwendung mit aus

```
grunt app
```

Beachten Sie, dass Sie ja auch Ihre eigentlichen Anwendungsdateien benötigen.
Für ein fast minimales Beispiel durchsuchen Sie das GitHub-Repository, das zu Beginn dieses Beispiels erwähnt wurde.

Die Struktur ist nicht so anders. Es gibt nur eine `index.html` Vorlage, winkeligen Code in `app.js` und einige Stile in `app.css` . Andere Dateien sind für die Git- und Editor-Konfiguration und einige allgemeine Dinge. Versuche es!





Grunzen Sie Aufgaben online lesen: https://riptutorial.com/de/angularjs/topic/6077/grunzen-sie-

aufgaben

Kapitel 28: HTTP-Interceptor

Einführung

Der $ http-Dienst von AngularJS ermöglicht es uns, mit einem Backend zu kommunizieren und HTTP-Anforderungen zu stellen. Es gibt Fälle, in denen wir jede Anfrage erfassen und bearbeiten müssen, bevor sie an den Server gesendet wird. Zu anderen Zeiten möchten wir die Antwort erfassen und bearbeiten, bevor der Anruf abgeschlossen wird. Die globale http-Fehlerbehandlung kann ebenfalls ein gutes Beispiel für ein solches Bedürfnis sein. Abfangjäger werden genau für solche Fälle erstellt.

Examples

Fertig machen

Der integrierte `$http` Dienst von Angular ermöglicht das Senden von HTTP-Anforderungen. Oft besteht die Notwendigkeit, vor oder nach einer Anforderung etwas zu tun, z. B. zu jeder Anforderung ein Authentifizierungstoken hinzuzufügen oder eine generische Fehlerbehandlungslogik zu erstellen.

Generischer httpInterceptor Schritt für Schritt

Erstellen Sie eine HTML-Datei mit folgendem Inhalt:

```
<!DOCTYPE html>
<html>
<head>
  <title>Angular Interceptor Sample</title>
  <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
  <script src="app.js"></script>
  <script src="appController.js"></script>
  <script src="genericInterceptor.js"></script>
</head>
<body ng-app="interceptorApp">
    <div ng-controller="appController as vm">
        <button ng-click="vm.sendRequest()">Send a request</button>
    </div>
</body>
</html>
```

Fügen Sie eine JavaScript-Datei mit dem Namen 'app.js' hinzu:

```
var interceptorApp = angular.module('interceptorApp', []);

interceptorApp.config(function($httpProvider) {
        $httpProvider.interceptors.push('genericInterceptor');
});
```

Fügen Sie eine weitere mit dem Namen 'appController.js' hinzu:

```
(function() {
    'use strict';

    function appController($http) {
        var vm = this;

        vm.sendRequest = function(){
            $http.get('http://google.com').then(function(response){
                console.log(response);
            });
        };
    }

    angular.module('interceptorApp').controller('appController',['$http', appController]);
})();
```

Und schließlich die Datei, die den Interceptor selbst enthält: "genericInterceptor.js":

```
(function() {
    "use strict";

    function genericInterceptor($q) {
        this.responseError = function (response) {
            return $q.reject(response);
        };

        this.requestError = function(request){
            if (canRecover(rejection)) {
                return responseOrNewPromise
            }
            return $q.reject(rejection);
        };

        this.response = function(response){
            return response;
        };

        this.request = function(config){
            return config;
        }
    }

    angular.module('interceptorApp').service('genericInterceptor', genericInterceptor);
})();
```

Der 'genericInterceptor' umfasst die möglichen Funktionen, die wir außer Kraft setzen können, um der Anwendung zusätzliches Verhalten hinzuzufügen.

Flash-Nachricht bei Antwort mit http-Interceptor

In der Ansichtsdatei

Lassen Sie ein leeres div-Element in der Basis-HTML (index.html), in der wir normalerweise die Winkelskripte oder die in der App freigegebene HTML-Datei einschließen, die Flash-Nachrichten in diesem div-Element

```
<div class="flashmessage" ng-if="isVisible">
    {{flashMessage}}
</div>
```

Skriptdatei

Fügen Sie in der config-Methode des Winkelmoduls den httpProvider ein, der httpProvider verfügt über eine Interceptor-Array-Eigenschaft. Drücken Sie den benutzerdefinierten Interceptor. Im aktuellen Beispiel fängt der benutzerdefinierte Interceptor nur die Antwort ab und ruft eine an rootScope angehängte Methode auf.

```
var interceptorTest = angular.module('interceptorTest', []);

    interceptorTest.config(['$httpProvider',function ($httpProvider) {

        $httpProvider.interceptors.push(["$rootScope",function ($rootScope) {
            return {     //intercept only the response
                        'response': function (response)
                                    {

$rootScope.showFeedBack(response.status,response.data.message);

                                      return response;
                                    }
                    };
        }]);

    }])
```

Da nur Provider in die config-Methode eines Winkelmoduls eingefügt werden können (dh httpProvider und nicht der rootscope), müssen Sie die an rootscope angehängte Methode innerhalb der run-Methode des angle-Moduls deklarieren.

Zeigen Sie die Nachricht auch innerhalb von $ timeout an, damit die Nachricht die Flash-Eigenschaft besitzt, die nach einer Schwellenzeit verschwindet. In unserem Beispiel sind es 3000 ms.

```
interceptorTest.run(["$rootScope","$timeout",function($rootScope,$timeout){
     $rootScope.showFeedBack = function(status,message){

        $rootScope.isVisible = true;
        $rootScope.flashMessage = message;
        $timeout(function(){$rootScope.isVisible = false },3000)
    }
}]);
```

Häufige Fehler

Beim Versuch, **$ rootScope oder andere Dienste** in die **config-** Methode des Winkelmoduls einzuspeisen, lässt der Lebenszyklus der Winkel-App dies nicht zu, und ein unbekannter Anbieterfehler wird ausgelöst. In der **config-** Methode des Winkelmoduls können nur **Provider**

eingefügt werden

HTTP-Interceptor online lesen: https://riptutorial.com/de/angularjs/topic/6484/http-interceptor

Kapitel 29: Komponenten

Parameter

Parameter	Einzelheiten
=	Für die bidirektionale Datenbindung. Das bedeutet, wenn Sie diese Variable in Ihrem Komponentenbereich aktualisieren, wirkt sich die Änderung auf den übergeordneten Bereich aus.
<	Einwegbindungen, wenn wir einen Wert nur aus einem übergeordneten Bereich lesen und nicht aktualisieren möchten.
@	String-Parameter
&	Für Rückrufe, falls Ihre Komponente etwas in den übergeordneten Bereich ausgeben muss.
-	-
Lebenszyklus-Hooks	**Details** (erfordert angle.version> = 1.5.3)
$ onInit ()	Wird auf jedem Controller aufgerufen, nachdem alle Controller eines Elements erstellt wurden und deren Bindungen initialisiert wurden. Hier können Sie den Initialisierungscode für Ihren Controller eingeben.
$ onChanges (changesObj)	Wird aufgerufen, wenn Einwegbindungen aktualisiert werden. Die `changesObj` ist ein Hash, dessen Schlüssel die Namen der gebundenen Eigenschaften sind, die sich geändert haben, und die Werte sind ein Objekt der Form `{ currentValue, previousValue, isFirstChange() }`.
$ onDestroy ()	Wird von einem Controller aufgerufen, wenn der enthaltene Bereich zerstört wird. Verwenden Sie diesen Haken, um externe Ressourcen, Uhren und Event-Handler freizugeben.
$ postLink ()	Wird aufgerufen, nachdem das Element dieses Controllers und seine untergeordneten Elemente verknüpft wurden. Dieser Hook kann als analog zu den Hooks ngAfterViewInit und ngAfterContentInit in Angular 2 betrachtet werden.
$ doCheck ()	Wird in jeder Runde des Digest-Zyklus aufgerufen. Bietet die Möglichkeit, Änderungen zu erkennen und darauf zu reagieren. Alle Aktionen, die Sie als Reaktion auf die erkannten Änderungen durchführen möchten, müssen von diesem Hook aus aufgerufen werden. Die Implementierung hat keine Auswirkungen auf den Aufruf von $ onChanges.

Bemerkungen

Komponente ist eine spezielle Art von Direktive, die eine einfachere Konfiguration verwendet, die für eine komponentenbasierte Anwendungsstruktur geeignet ist. Komponenten wurden in Angular 1.5 eingeführt. Die Beispiele in diesem Abschnitt **funktionieren nicht** mit älteren AngularJS-Versionen.

Ein vollständiges Entwicklerhandbuch für Komponenten ist verfügbar unter https://docs.angularjs.org/guide/component

Examples

Grundkomponenten und Lebenszyklus-Hooks

Was ist eine Komponente?

- Eine Komponente ist im Grunde eine Direktive, die eine einfachere Konfiguration verwendet und sich für eine komponentenbasierte Architektur eignet. Angular 2 ist dies alles. Stellen Sie sich eine Komponente als Widget vor: Ein Stück HTML-Code, den Sie an verschiedenen Stellen in Ihrer Webanwendung wiederverwenden können.

Komponente

```
angular.module('myApp', [])
    .component('helloWorld', {
        template: '<span>Hello World!</span>'
    });
```

Markup

```
<div ng-app="myApp">
  <hello-world> </hello-world>
</div>
```

Live Demo

Verwenden externer Daten in Component:

Wir könnten einen Parameter hinzufügen, um unserer Komponente einen Namen zu übergeben, der wie folgt verwendet wird:

```
angular.module("myApp", [])
  .component("helloWorld",{
    template: '<span>Hello {{$ctrl.name}}!</span>',
    bindings: { name: '@' }
```

```
  });
```

Markup

```
<div ng-app="myApp">
  <hello-world name="'John'" > </hello-world>
</div>
```

Live Demo

Controller in Komponenten verwenden

Sehen wir uns an, wie Sie einen Controller hinzufügen.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.myName = 'Alain';
      }
  });
```

Markup

```
<div ng-app="myApp">
  <hello-world name="John"> </hello-world>
</div>
```

CodePen-Demo

Parameter, die an die Komponente übergeben werden, sind im Bereich des Controllers verfügbar, kurz bevor die `$onInit` Funktion von Angular aufgerufen wird. Betrachten Sie dieses Beispiel:

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.$onInit = function() {
          this.myName = "Mac" + this.name;
        }
      }
  });
```

In der Vorlage von oben würde dies "Hallo John, ich bin MacJohn!"

Beachten Sie, dass `$ctrl` der Angular-Standardwert für `controllerAs` wenn kein Wert angegeben wird.

Live Demo

Verwenden von "erfordern" als Objekt

In einigen Fällen müssen Sie möglicherweise auf Daten von einer übergeordneten Komponente in Ihrer Komponente zugreifen.

Dies kann erreicht werden, indem festgelegt wird, dass für unsere Komponente die übergeordnete Komponente erforderlich ist. Die Anforderung gibt uns einen Hinweis auf die erforderliche Komponentensteuerung, die dann in unserer Steuerung verwendet werden kann, wie im folgenden Beispiel gezeigt:

> Beachten Sie, dass die erforderlichen Controller garantiert erst nach dem Hook $ onInit bereit sind.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      require: {
        parent: '^parentComponent'
      },
      controller: function () {
        // here this.parent might not be initiated yet

        this.$onInit = function() {
           // after $onInit, use this.parent to access required controller
           this.parent.foo();
        }

      }
  });
```

Beachten Sie jedoch, dass dadurch eine enge Verbindung zwischen dem Kind und dem Elternteil entsteht.

Komponenten in Winkel JS

Die Komponenten in angleJS können als benutzerdefinierte Direktive dargestellt werden (<html> dies in einer HTML-Direktive, und so etwas wird eine benutzerdefinierte Direktive <ANYTHING> sein). Eine Komponente enthält eine Ansicht und einen Controller. Controller enthält die Geschäftslogik, die mit einer Ansicht verbunden ist, die der Benutzer sieht. Die Komponente unterscheidet sich von einer Winkeldirektive, da sie weniger Konfiguration enthält. Eine Winkelkomponente kann so definiert werden.

```
angular.module("myApp",[]).component("customer", {})
```

Komponenten werden auf den Winkelmodulen definiert. Sie enthalten zwei Argumente: Einer ist der Name der Komponente und der zweite ist ein Objekt, das ein Schlüsselwertpaar enthält, das

definiert, welche Ansicht und welcher Controller auf diese Weise verwendet wird.

```
angular.module("myApp",[]).component("customer", {
    templateUrl : "customer.html", // your view here
    controller: customerController, //your controller here
    controllerAs: "cust"        //alternate name for your controller
})
```

"myApp" ist der Name der App, die wir erstellen, und der Kunde ist der Name unserer Komponente. Nun, um es in der Haupt-HTML-Datei aufzurufen, werden wir es einfach so setzen

```
<customer></customer>
```

Jetzt wird diese Direktive durch die von Ihnen angegebene Sicht und die Geschäftslogik, die Sie in Ihre Steuerung geschrieben haben, ersetzt.

HINWEIS: Denken Sie daran, dass die Komponente ein Objekt als zweites Argument übernimmt, während die Direktive eine Factory-Funktion als Argument verwendet.

Komponenten online lesen: https://riptutorial.com/de/angularjs/topic/892/komponenten

Kapitel 30: Konstanten

Bemerkungen

UPPERCASE Ihre Konstante : Konstante in Kapital zu schreiben ist eine gängige bewährte **Methode,** die in vielen Sprachen verwendet wird. Es ist auch nützlich, die Art der eingespritzten Elemente eindeutig zu identifizieren:

Wenn Sie `.controller('MyController', function($scope, Profile, EVENT))` , wissen Sie sofort `.controller('MyController', function($scope, Profile, EVENT))` :

- `$scope` ist ein Winkelelement
- `Profile` ist eine benutzerdefinierte Dienstleistung oder eine Fabrik
- `EVENT` ist eine Winkelkonstante

Examples

Erstellen Sie Ihre erste Konstante

```
angular
  .module('MyApp', [])
  .constant('VERSION', 1.0);
```

Ihre Konstante ist jetzt deklariert und kann in einen Controller, einen Service, eine Factory, einen Provider und sogar in eine Config-Methode eingefügt werden:

```
angular
  .module('MyApp')
  .controller('FooterController', function(VERSION) {
    this.version = VERSION;
  });
```

```
<footer ng-controller="FooterController as Footer">{{ Footer.version }}</footer>
```

Anwendungsfälle

Hier gibt es keine Revolution, aber die Winkelkonstante kann insbesondere dann nützlich sein, wenn Ihre Anwendung und / oder Ihr Team zu wachsen beginnt ... oder wenn Sie einfach gerne schönen Code schreiben!

- **Refactor-Code** Beispiel mit Ereignisnamen. Wenn Sie in Ihrer Anwendung eine Vielzahl von Ereignissen verwenden, haben Sie die Ereignisnamen ein wenig überall. Wenn ein neuer Entwickler Ihrem Team beitritt, benennt er seine Ereignisse mit einer anderen Syntax. Sie können dies leicht verhindern, indem Sie die Namen Ihrer Ereignisse in einer Konstanten gruppieren:

```
angular
  .module('MyApp')
  .constant('EVENTS', {
    LOGIN_VALIDATE_FORM: 'login::click-validate',
    LOGIN_FORGOT_PASSWORD: 'login::click-forgot',
    LOGIN_ERROR: 'login::notify-error',
    ...
  });
```

```
angular
  .module('MyApp')
  .controller('LoginController', function($scope, EVENT) {
    $scope.$on(EVENT.LOGIN_VALIDATE_FORM, function() {
      ...
    });
  })
```

... und jetzt können die Namen Ihrer Veranstaltung von der automatischen Vervollständigung profitieren!

- **Konfiguration definieren** Finden Sie alle Ihre Konfiguration an einem Ort:

```
angular
  .module('MyApp')
  .constant('CONFIG', {
    BASE_URL: {
      APP: 'http://localhost:3000',
      API: 'http://localhost:3001'
    },
    STORAGE: 'S3',
    ...
  });
```

- **Teile isolieren.** Manchmal gibt es einige Dinge, auf die Sie nicht sehr stolz sind ... wie zum Beispiel ein fest codierter Wert. Anstatt sie in Ihren Hauptcode aufzunehmen, können Sie eine Winkelkonstante erstellen

```
angular
  .module('MyApp')
  .constant('HARDCODED', {
    KEY: 'KEY',
    RELATION: 'has_many',
    VAT: 19.6
  });
```

... und so etwas umgestalten

```
$scope.settings = {
  username: Profile.username,
  relation: 'has_many',
  vat: 19.6
}
```

zu

```
$scope.settings = {
  username: Profile.username,
  relation: HARDCODED.RELATION,
  vat: HARDCODED.VAT
}
```

Konstanten online lesen: https://riptutorial.com/de/angularjs/topic/3967/konstanten

Kapitel 31: Leistungsprofilierung

Examples

Alles über Profiling

Was ist Profiling?

Per Definition ist Profiling eine Form der dynamischen Programmanalyse, die beispielsweise den Speicherplatz (Speicher) oder die zeitliche Komplexität eines Programms, die Verwendung bestimmter Anweisungen oder die Häufigkeit und Dauer von Funktionsaufrufen misst.

Warum ist es notwendig?

Profiling ist wichtig, da Sie nicht effektiv optimieren können, wenn Sie nicht wissen, was Ihr Programm am meisten Zeit damit verbringt. Wenn Sie die Ausführungszeit Ihres Programms (Profiling) nicht messen, wissen Sie nicht, ob Sie es tatsächlich verbessert haben.

Werkzeuge und Techniken:

1. Integrierte Entwickler-Tools von Chrome

   Dazu gehört ein umfangreiches Set an Tools für die Profilerstellung. Sie können Engpässe in Ihrer Javascript-Datei, CSS-Dateien, Animationen, CPU-Auslastung, Speicherverluste, Netzwerksicherheit usw. ermitteln.

   Machen Sie eine Timeline- Aufnahme und suchen Sie nach verdächtig langen Evaluate Script-Ereignissen. Wenn Sie welche finden, können Sie den JS Profiler aktivieren und Ihre Aufnahme wiederholen, um detailliertere Informationen darüber zu erhalten, welche JS-Funktionen genau aufgerufen wurden und wie lange die einzelnen Funktionen dauerten. Weiterlesen...

2. FireBug (Verwendung mit Firefox)

3. Dynatrace (Verwendung mit IE)

4. Batarang (Verwendung mit Chrome)

   Es ist ein veraltetes Add-On für Chrome-Browser, obwohl es stabil ist und zur Überwachung von Modellen, Leistung und Abhängigkeiten für eine Winkelanwendung verwendet werden kann. Es eignet sich gut für kleine Anwendungen und kann Ihnen einen Einblick in die Gültigkeitsbereiche der Gültigkeitsbereichsvariablen auf verschiedenen Ebenen geben. Es informiert Sie über aktive Beobachter, Ausdrücke beobachten, Sammlungen in der App ansehen.

5. Watcher (Verwendung mit Chrome)

   Schöne und einfache Benutzeroberfläche, um die Anzahl der Beobachter in einer Angular-

App zu zählen.

6. Verwenden Sie den folgenden Code, um die Anzahl der Beobachter in Ihrer Winkel-App manuell herauszufinden (Verdienst @Words Like Jared Anzahl der Beobachter).

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
})();
```

7. Es stehen mehrere Online-Tools / Websites zur Verfügung, die eine Vielzahl von Funktionen ermöglichen, um ein Profil Ihrer Anwendung zu erstellen.

   Eine solche Site ist: https://www.webpagetest.org/

   Damit können Sie einen kostenlosen Website-Geschwindigkeitstest von mehreren Standorten auf der ganzen Welt mit realen Browsern (IE und Chrome) und mit echten Verbindungsgeschwindigkeiten für Konsumenten durchführen. Sie können einfache Tests oder erweiterte Tests durchführen, einschließlich Transaktionen in mehreren Schritten, Videoerfassung, Blockierung von Inhalten und vieles mehr.

Nächste Schritte:

Fertig mit Profiling. Es bringt Sie nur die Hälfte der Straße hinunter. Die nächste Aufgabe besteht darin, Ihre Ergebnisse tatsächlich in Aktionselemente umzuwandeln, um Ihre Anwendung zu optimieren. In dieser Dokumentation erfahren Sie, wie Sie mit einfachen Tricks die Leistung Ihrer Winkel-App verbessern können.

Viel Spaß beim Codieren :)

Leistungsprofilierung online lesen:

https://riptutorial.com/de/angularjs/topic/7033/leistungsprofilierung

Kapitel 32: Migration nach Angular 2+

Einführung

AngularJS wurde komplett mit der Sprache TypeScript umgeschrieben und in nur Angular umbenannt.

Mit einer AngularJS-App kann viel getan werden, um den Migrationsprozess zu vereinfachen. Wie in der offiziellen Upgrade-Anleitung angegeben , können mehrere "Vorbereitungsschritte" ausgeführt werden, um Ihre App zu überarbeiten und sie dem neuen Angular-Stil näher zu bringen.

Examples

Umwandlung Ihrer AngularJS-App in eine komponentenorientierte Struktur

Im neuen Angular-Framework sind **Komponenten** die Hauptbausteine, aus denen die Benutzeroberfläche besteht. Einer der ersten Schritte, der eine Migration der AngularJS-App auf den neuen Angular ermöglicht, besteht darin, sie in eine komponentenorientiertere Struktur umzuwandeln.

Komponenten wurden auch in der alten AngularJS ab Version **1.5+ eingeführt** . Durch die Verwendung von Komponenten in einer AngularJS-App wird die Struktur nicht nur dem neuen Angular 2+ angenähert, sondern auch modularer und einfacher zu warten.

Bevor Sie fortfahren, empfehle ich Ihnen, sich auf der offiziellen AngularJS-Dokumentationsseite über Komponenten zu informieren , wo ihre Vorteile und ihre Verwendung gut erklärt werden.

Ich möchte eher einige Tipps erwähnen, wie der alte `ng-controller` Code in den neuen `component` Stil konvertiert werden kann.

Brechen Sie Ihre App in Komponenten auf

Alle komponentenorientierten Apps haben normalerweise eine oder wenige Komponenten, die andere Unterkomponenten enthalten. Warum also nicht die erste Komponente erstellen, die einfach Ihre App (oder einen großen Teil davon) enthält.

Angenommen, wir haben einem Controller einen Code mit dem Namen `UserListController` , und wir möchten eine Komponente daraus `UserListComponent` , die wir `UserListComponent` .

aktuelles HTML:

```
<div ng-controller="UserListController as listctrl">
    <ul>
```

```
        <li ng-repeat="user in myUserList">
            {{ user }}
        </li>
    </ul>
</div>
```

aktuelles JavaScript:

```
app.controller("UserListController", function($scope, SomeService) {

    $scope.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

neues HTML:

```
<user-list></user-list>
```

neues JavaScript:

```
app.component("UserList", {
    templateUrl: 'user-list.html',
    controller: UserListController
});

function UserListController(SomeService) {

    this.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

Beachten Sie, wie wir `$scope` nicht mehr in die Controller-Funktion `this.myUserList` und deklarieren `this.myUserList` jetzt `this.myUserList` anstelle von `$scope.myUserList` .

neue Vorlagendatei `user-list.component.html` :

```
<ul>
    <li ng-repeat="user in $ctrl.myUserList">
        {{ user }}
    </li>
</ul>
```

Beachten Sie, wie wir uns jetzt auf die Variable `myUserList` , die zum Controller gehört, und `$ctrl.myUserList` aus dem HTML- `$scope.myUserList` anstelle von `$scope.myUserList` .

Wie Sie wahrscheinlich nach dem Lesen der Dokumentation herausgefunden haben, bezieht sich `$ctrl` in der Vorlage nun auf die Controller-Funktion.

Was ist mit Controllern und Routen?

Für den Fall, dass Ihr Controller über das Routing-System anstelle des `ng-controller` an die Vorlage gebunden war, sollten Sie also Folgendes haben:

```
$stateProvider
  .state('users', {
    url: '/users',
    templateUrl: 'user-list.html',
    controller: 'UserListController'
  })
  // ..
```

Sie können Ihre Staatserklärung einfach ändern in:

```
$stateProvider
    .state('users', {
      url: '/',
      template: '<user-list></user-list>'
    })
    // ..
```

Was kommt als nächstes?

Da Sie nun eine Komponente mit Ihrer App haben (unabhängig davon, ob die gesamte Anwendung oder ein Teil davon wie eine Ansicht enthalten ist), sollten Sie nun beginnen, Ihre Komponente in mehrere verschachtelte Komponenten zu unterteilen, indem Sie Teile davon in neue Unterkomponenten einbetten , und so weiter.

Sie sollten anfangen, die Komponentenfunktionen wie zu verwenden

- **Inputs** und **Outputs-** Bindungen
- **Lebenszyklus-Hooks** wie `$onInit()` , `$onChanges()` usw.

Nach dem Lesen Komponentendokumentation sollten Sie wissen bereits , wie alle diese Komponentenmerkmale verwenden, aber wenn Sie ein konkretes Beispiel für eine echte einfache Anwendung benötigen, können Sie überprüfen diese .

Wenn sich innerhalb des Controllers Ihrer Komponente einige Funktionen befinden, die viel Logikcode enthalten, kann es eine gute Idee sein, diese Logik in Dienste zu verschieben .

Fazit

Wenn Sie einen komponentenbasierten Ansatz wählen, wird Ihr AngularJS einen Schritt näher an das neue Angular-Framework heranrücken, es wird jedoch auch besser und viel modularer.

Natürlich gibt es noch viele weitere Schritte, um weiter in die neue Richtung von Angular 2+ zu gehen, die ich in den folgenden Beispielen aufführen werde.

Einführung in Webpack- und ES6-Module

Ein ***Modul*** - ***Lader*** wie durch die Verwendung Webpack können wir den Einbau-Modul - System in profitieren **ES6** (wie auch in **Typoskript).** Wir können dann die Import- und Exportfunktionen verwenden , mit denen wir festlegen können, welche Codeteile wir für verschiedene Teile der Anwendung freigeben können.

Wenn wir dann unsere Anwendungen in die Produktion aufnehmen, erleichtern die Modulladegeräte das Zusammenfassen der Pakete in Produktionspakete inklusive Batterien.

Migration nach Angular 2+ online lesen: https://riptutorial.com/de/angularjs/topic/9942/migration-nach-angular-2plus

Kapitel 33: Module

Examples

Module

Das Modul dient als Container für verschiedene Teile Ihrer App, z. B. Controller, Services, Filter, Anweisungen usw. Die Module können von anderen Modulen über den Abhängigkeitseinspritzmechanismus von Angular referenziert werden.

Ein Modul erstellen:

```
angular
    .module('app', []);
```

Bei Array `[]` das im obigen Beispiel übergeben wird, handelt es sich um die *Liste der Module, von denen* `app` abhängt. Wenn keine Abhängigkeiten vorhanden sind, übergeben wir Empty Array, dh `[]` .

Einfügen eines Moduls als Abhängigkeit eines anderen Moduls:

```
angular.module('app', [
    'app.auth',
    'app.dashboard'
]);
```

Verweis auf ein Modul:

```
angular
    .module('app');
```

Module

Das Modul ist ein Container für verschiedene Teile Ihrer Anwendungen - Controller, Services, Filter, Anweisungen usw.

Warum Module verwenden?
Die meisten Anwendungen verfügen über eine Hauptmethode, mit der die verschiedenen Teile der Anwendung instanziiert und miteinander verbunden werden.
Eckige Apps haben keine Hauptmethode.
In AngularJs ist der deklarative Prozess jedoch leicht verständlich und man kann Code als wiederverwendbare Module packen.
Module können in beliebiger Reihenfolge geladen werden, da Module die Ausführung verzögern.

ein Modul deklarieren

```
var app = angular.module('myApp', []);
// Empty array is list of modules myApp is depends on.
// if there are any required dependancies,
// then you can add in module, Like ['ngAnimate']

app.controller('myController', function() {

  // write your business logic here
});
```

Modul laden und Abhängigkeiten

1. Konfigurationsblöcke: - werden während der Provider- und Konfigurationsphase ausgeführt.

   ```
   angular.module('myModule', []).
   config(function(injectables) {
     // here you can only inject providers in to config blocks.
   });
   ```

2. Run Blocks: - werden ausgeführt, nachdem der Injektor erstellt wurde, und werden zum Starten der Anwendung verwendet.

   ```
   angular.module('myModule', []).
   run(function(injectables) {
     // here you can only inject instances in to config blocks.
   });
   ```

Module online lesen: https://riptutorial.com/de/angularjs/topic/844/module

Kapitel 34: ng-class Direktive

Examples

Drei Arten von NG-Klassenausdrücken

Angular unterstützt drei Arten von Ausdrücken in der Direktive `ng-class` .

1. String

```
<span ng-class="MyClass">Sample Text</span>
```

Durch die Angabe eines Ausdrucks, der eine Zeichenfolge ergibt, wird Angular aufgefordert, ihn als $ scope-Variable zu behandeln. Angular überprüft den $ scope und sucht nach einer Variablen namens "MyClass". Der in "MyClass" enthaltene Text wird zum tatsächlichen Klassennamen, der auf diesen `<span>` angewendet wird. Sie können mehrere Klassen angeben, indem Sie jede Klasse durch ein Leerzeichen trennen.

In Ihrem Controller haben Sie möglicherweise eine Definition, die wie folgt aussieht:

```
$scope.MyClass = "bold-red deleted error";
```

Angular wird den Ausdruck `MyClass` auswerten und die Definition des Bereichs $ finden. Es werden die drei Klassen "fett-rot", "gelöscht" und "Fehler" auf das `<span>` -Element angewendet.

Wenn Sie Klassen auf diese Weise angeben, können Sie die Klassendefinitionen in Ihrem Controller problemlos ändern. Beispielsweise müssen Sie möglicherweise die Klasse basierend auf anderen Benutzerinteraktionen oder neuen Daten ändern, die vom Server geladen werden. Wenn Sie viele Ausdrücke zum Auswerten haben, können Sie dies auch in einer Funktion tun, die die endgültige Liste der Klassen in einer `$scope` Variablen definiert. Dies kann einfacher sein, als zu versuchen, viele Auswertungen im `ng-class` Attribut in Ihrer HTML-Vorlage zusammenzufassen.

---

2. Objekt

Dies ist die am häufigsten verwendete Methode zum Definieren von Klassen mit `ng-class` da Sie leicht Auswertungen angeben können, die die zu verwendende Klasse bestimmen.

Geben Sie ein Objekt an, das Schlüsselwertpaare enthält. Der Schlüssel ist der Klassenname, der angewendet wird, wenn der Wert (eine Bedingung) als wahr ausgewertet wird.

```
<style>
    .red { color: red; font-weight: bold; }
```

```
    .blue { color: blue; }
    .green { color: green; }
    .highlighted { background-color: yellow; color: black; }
</style>

<span ng-class="{ red: ShowRed, blue: ShowBlue, green: ShowGreen, highlighted: IsHighlighted
}">Sample Text</span>

<div>Red: <input type="checkbox" ng-model="ShowRed"></div>
<div>Green: <input type="checkbox" ng-model="ShowGreen"></div>
<div>Blue: <input type="checkbox" ng-model="ShowBlue"></div>
<div>Highlight: <input type="checkbox" ng-model="IsHighlighted"></div>
```

---

3. Array

Mit einem Ausdruck, der zu einem Array ausgewertet wird, können Sie eine Kombination aus **Zeichenfolgen** (siehe Nr. 1 oben) und **bedingten Objekten** (Nr. 2 oben) verwenden.

```
<style>
    .bold { font-weight: bold; }
    .strike { text-decoration: line-through; }
    .orange { color: orange; }
</style>

<p ng-class="[ UserStyle, {orange: warning} ]">Array of Both Expression Types</p>
<input ng-model="UserStyle" placeholder="Type 'bold' and/or 'strike'"><br>
<label><input type="checkbox" ng-model="warning"> warning (apply "orange" class)</label>
```

Dadurch wird ein Texteingabefeld erstellt, das an die Gültigkeitsbereichsvariable `UserStyle` gebunden ist, wodurch der Benutzer beliebige Klassennamen `UserStyle` . Diese werden dynamisch als Benutzertypen auf das `<p>` -Element angewendet. Auch kann der Benutzer auf das Kontrollkästchen klicken , die an die Datengebunden ist `warning` Umfang variabel. Dies wird auch dynamisch auf das `<p>` -Element angewendet.

ng-class Direktive online lesen: https://riptutorial.com/de/angularjs/topic/2395/ng-class-direktive

Kapitel 35: ng-style

Einführung

Mit der Anweisung 'ngStyle' können Sie den CSS-Stil für ein HTML-Element bedingt festlegen. So wie wir *style*- Attribute für HTML-Elemente in Nicht-AngularJS-Projekten verwenden könnten, können wir `ng-style` in anglejs verwenden, um Stile auf der Grundlage einiger boolescher Bedingungen anzuwenden.

Syntax

- `<ANY ng-style="expression"></ANY >`
- `<ANY class="ng-style: expression;"> ... </ANY>`

Examples

Verwendung von ng-style

Das folgende Beispiel ändert die Deckkraft des Bildes auf der Grundlage des Parameters "status".

```
<img class="img-responsive"  ng-src="{{imagesrc}}"
    ng-style="{'opacity' : (status == 2) ? 1 : 0.5}">
```

ng-style online lesen: https://riptutorial.com/de/angularjs/topic/8773/ng-style

Kapitel 36: ng-view

Einführung

ng-view ist eine in-build-Direktive, die als Container zum Wechseln zwischen Ansichten verwendet wird. {info} ngRoute ist nicht mehr Bestandteil der Basisdatei angle.js, daher müssen Sie die Datei angle-route.js nach der Basis-Javascript-Datei einfügen. Wir können eine Route mithilfe der Funktion "when" des $ routeProvider konfigurieren. Wir müssen zuerst die Route angeben und dann in einem zweiten Parameter ein Objekt mit einer templateUrl-Eigenschaft und einer Controller-Eigenschaft angeben.

Examples

ng-view

`ng-view` ist eine Direktive, die mit `$route` zum Rendern einer Teilansicht im Hauptseitenlayout verwendet wird. In diesem Beispiel ist Index.html unsere Hauptdatei. Wenn der Benutzer auf der "/" - Route landet, wird die templateURL home.html in Index.html gerendert, wo `ng-view` erwähnt wird.

```
angular.module('ngApp', ['ngRoute'])

.config(function($routeProvider){
  $routeProvider.when("/",
    {
      templateUrl: "home.html",
      controller: "homeCtrl"
    }
  );
});

angular.module('ngApp').controller('homeCtrl',['$scope', function($scope) {
  $scope.welcome= "Welcome to stackoverflow!";
}]);

//Index.html
<body ng-app="ngApp">
    <div ng-view></div>
</body>

//Home Template URL or home.html
<div><h2>{{welcome}}</h2></div>
```

Registrierungsnavigation

1. Wir injizieren das Modul in die Anwendung

```
  var Registration=angular.module("myApp",["ngRoute"]);
```

2. jetzt verwenden wir $ routeProvider von "ngRoute"

```
    Registration.config(function($routeProvider) {

});
```

3. Schließlich integrieren wir die Route und definieren das Routing "/ add" für die Anwendung, falls die Anwendung "/ add" erhält, wenn sie an regi.htm umgeleitet wird

```
 Registration.config(function($routeProvider) {
    $routeProvider
    .when("/add", {
        templateUrl : "regi.htm"
    })
});
```

ng-view online lesen: https://riptutorial.com/de/angularjs/topic/8833/ng-view

Kapitel 37: Profiling und Leistung

Examples

7 Einfache Leistungsverbesserungen

1) Verwenden Sie ng-repeat sparsam

Die Verwendung von `ng-repeat` in Ansichten führt im Allgemeinen zu einer schlechten Leistung, insbesondere bei verschachtelten `ng-repeat` -Werten.

Das ist super langsam!

```
<div ng-repeat="user in userCollection">
  <div ng-repeat="details in user">
    {{details}}
  </div>
</div>
```

Versuchen Sie, geschachtelte Wiederholungen so weit wie möglich zu vermeiden. Eine Möglichkeit, die Leistung von `ng-repeat` zu verbessern `ng-repeat` ist die Verwendung von `track by $index` (oder einem anderen id-Feld). `ng-repeat` verfolgt standardmäßig das gesamte Objekt. Mit `track by` überwacht Angular das Objekt nur anhand des `$index` oder der Objekt-ID.

```
<div ng-repeat="user in userCollection track by $index">
  {{user.data}}
</div>
```

Verwenden Sie andere Ansätze wie Seitenumbruch , virtuelle Bildlauf , unendliche Bildlauf oder LimitToOo: Beginnen Sie wann immer möglich, um das Durchlaufen großer Sammlungen zu vermeiden.

---

2) Einmal binden

Angular hat eine bidirektionale Datenbindung. Es ist mit dem Preis verbunden, langsam zu sein, wenn zu viel verwendet wird.

Langsamere Leistung

```
<!-- Default data binding has a performance cost -->
<div>{{ my.data }}</div>
```

Schnellere Leistung (AngularJS> = 1,3)

```
<!-- Bind once is much faster -->
```

```
<div>{{ ::my.data }}</div>

<div ng-bind="::my.data"></div>

<!-- Use single binding notation in ng-repeat where only list display is needed  -->
<div ng-repeat="user in ::userCollection">
  {{::user.data}}
</div>
```

Wenn Sie die "einmal bindend" -Notation verwenden, wird Angular angewiesen, darauf zu warten, dass sich der Wert nach der ersten Serie von Digest-Zyklen stabilisiert. Angular verwendet diesen Wert im DOM und entfernt dann alle Beobachter, sodass dieser zu einem statischen Wert wird und nicht mehr an das Modell gebunden ist.

Das `{{}}` ist viel langsamer.

Dieses `ng-bind` ist eine Direktive und setzt einen Beobachter auf die übergebene Variable. Das `ng-bind` gilt also nur, wenn sich der übergebene Wert tatsächlich ändert.

Die Klammern auf der anderen Seite werden in jedem `$digest` überprüft, auch wenn dies nicht notwendig ist.

---

3) Scope-Funktionen und Filter benötigen Zeit

AngularJS hat eine Digest-Schleife. Alle Ihre Funktionen werden in einer Ansicht angezeigt und Filter werden jedes Mal ausgeführt, wenn der Digest-Zyklus ausgeführt wird. Die Digest-Schleife wird bei jeder Aktualisierung des Modells ausgeführt und kann Ihre App verlangsamen (der Filter kann mehrmals aufgerufen werden, bevor die Seite geladen wird).

Vermeiden dies:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Besserer Ansatz

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Wo kann der Controller sein:

```
app.controller('bigCalulations', function(valueService) {
    // bad, because this is called in every digest loop
    this.calculateMe = function() {
        var t = 0;
        for(i = 0; i < 1000; i++) {
```

```
            t += i;
        }
        return t;
    }
    // good, because this is executed just once and logic is separated in service to    keep
the controller light
    this.preCalulatedValue = valueService.valueCalculation(); // returns 499500
});
```

---

4 Beobachter

Beobachter sinken die Leistung enorm. Bei mehr Beobachtern dauert die Digest-Schleife länger und die Benutzeroberfläche wird langsamer. Wenn der Beobachter Änderungen erkennt, wird die Digest-Schleife ausgelöst und die Ansicht erneut gerendert.

Es gibt drei Möglichkeiten, die Winkeländerungen in Angular manuell zu überwachen.

`$watch()` - `$watch()` auf Wertänderungen

`$watchCollection()` - `$watchCollection()` Änderungen in der Sammlung (Uhren, die mehr als normale `$watch`)

`$watch(..., true)` - **Vermeiden Sie dies** so weit wie möglich, es führt "Deep Watch" aus und `watchCollection` die Leistung (Uhren mehr als `watchCollection`)

Wenn Sie Variablen in der Ansicht binden, erstellen Sie neue Uhren. Verwenden Sie `{{::variable}}` , um das Erstellen einer Uhr zu verhindern, insbesondere in Schleifen.

Daher müssen Sie nachverfolgen, wie viele Beobachter Sie verwenden. Sie können die Beobachter mit diesem Skript zählen (Verdienst @Words Like Jared Anzahl der Beobachter)

```
(function() {
    var root = angular.element(document.getElementsByTagName('body')),
        watchers = [],
        f = function(element) {
        angular.forEach(['$scope', '$isolateScope'], function(scopeProperty) {
            if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
                angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
                watchers.push(watcher);
                });
            }
        });

        angular.forEach(element.children(), function(childElement) {
            f(angular.element(childElement));
        });
    };

    f(root);

    // Remove duplicate watchers
    var watchersWithoutDuplicates = [];
    angular.forEach(watchers, function(item) {
        if(watchersWithoutDuplicates.indexOf(item) < 0) {
```

```
            watchersWithoutDuplicates.push(item);
        }
    });
    console.log(watchersWithoutDuplicates.length);
})();
```

---

5) ng-if / ng-show

Diese Funktionen sind im Verhalten sehr ähnlich. **`ng-if`** Elemente aus dem DOM entfernt, während mit **`ng-show`** nur die Elemente ausgeblendet werden, aber alle Handler beibehalten werden. Wenn Sie Teile des Codes haben, die Sie nicht anzeigen möchten, verwenden Sie `ng-if` .

Das hängt von der Art der Nutzung ab, aber oft ist einer besser geeignet als der andere.

- Wenn das Element nicht benötigt wird, verwenden Sie `ng-if`
- Um schnell ein- / auszuschalten, verwenden Sie `ng-show/ng-hide`

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special<!-- some complicated DOM --></p>
  <p ng-show="user.hasSubscribed">I am awesome<!-- switch this setting on and off --></p>
</div>
```

Im Zweifelsfall `ng-if` und testen!

---

6) Deaktivieren Sie das Debugging

Standardmäßig lassen Bindungsanweisungen und Bereiche zusätzliche Klassen und Markierungen im Code zu, um verschiedene Debugging-Tools zu unterstützen. Wenn Sie diese Option deaktivieren, werden diese verschiedenen Elemente während des Digest-Zyklus nicht mehr gerendert.

```
angular.module('exampleApp', []).config(['$compileProvider', function ($compileProvider) {
    $compileProvider.debugInfoEnabled(false);
}]);
```

---

7) Verwenden Sie Abhängigkeitsinjektion, um Ihre Ressourcen verfügbar zu machen

Abhängigkeitsinjektion ist ein Software-Entwurfsmuster, bei dem ein Objekt seine Abhängigkeiten erhält und nicht das Objekt, das es selbst erstellt. Es geht darum, die hartcodierten Abhängigkeiten zu entfernen und sie bei Bedarf jederzeit ändern zu können.

Sie werden sich vielleicht über die mit einer solchen String-Analyse aller injizierbaren Funktionen verbundenen Performance-Kosten wundern. Angular sorgt dafür, indem es die $ inject-

Eigenschaft nach dem ersten Mal zwischenspeichert. Das passiert also nicht immer, wenn eine Funktion aufgerufen werden muss.

PRO TIPP: Wenn Sie nach dem Ansatz mit der besten Leistung suchen, verwenden Sie den Annotationsansatz "$ inject property". Dieser Ansatz vermeidet die Analyse der Funktionsdefinition vollständig, da diese Logik innerhalb der folgenden Prüfung in der Anmerkungsfunktion eingeschlossen wird: if (! ($ Inject = fn. $ Inject)). Wenn $ inject bereits verfügbar ist, ist keine Analyse erforderlich!

```
var app = angular.module('DemoApp', []);

var DemoController = function (s, h) {
    h.get('https://api.github.com/users/angular/repos').success(function (repos) {
        s.repos = repos;
    });
}
// $inject property annotation
DemoController['$inject'] = ['$scope', '$http'];

app.controller('DemoController', DemoController);
```

PRO TIPP 2: Sie können eine `ng-strict-di` Direktive zu demselben Element wie `ng-app` hinzufügen, um den strikten DI-Modus zu aktivieren, der einen Fehler ausgibt, wenn ein Dienst implizite Annotationen verwendet. Beispiel:

```
<html ng-app="DemoApp" ng-strict-di>
```

Oder wenn Sie manuelles Bootstrapping verwenden:

```
angular.bootstrap(document, ['DemoApp'], {
    strictDi: true
});
```

Einmal binden

Angular hat den Ruf, eine hervorragende bidirektionale Datenbindung zu haben. Standardmäßig synchronisiert Angular fortlaufend Werte, die zwischen Modell- und Ansichtskomponenten gebunden sind, wenn sich Daten in der Modell- oder Ansichtskomponente ändern.

Dies ist mit etwas Kosten verbunden, wenn Sie zu viel verwendet werden. Dies hat einen größeren Performance-Erfolg:

Schlechte Leistung: `{{my.data}}`

Fügen Sie zwei Doppelpunkte `::` vor dem Variablennamen ein, um die einmalige Bindung zu verwenden. In diesem Fall wird der Wert nur aktualisiert, wenn my.data definiert ist. Sie weisen ausdrücklich darauf hin, nicht auf Datenänderungen zu achten. Angular führt keine Wertüberprüfungen durch, sodass bei jedem Digest-Zyklus weniger Ausdrücke ausgewertet werden.

Gute Leistungsbeispiele bei einmaliger Bindung

```
{{::my.data}}
<span ng-bind="::my.data"></span>
<span ng-if="::my.data"></span>
<span ng-repeat="item in ::my.data">{{item}}</span>
<span ng-class="::{ 'my-class': my.data }"></div>
```

Hinweis: Dadurch wird jedoch die bidirektionale Datenbindung für `my.data` . Wenn sich dieses Feld in Ihrer Anwendung ändert, wird dasselbe nicht automatisch in der Ansicht angezeigt. **Verwenden Sie es daher nur für Werte, die sich während der gesamten Lebensdauer Ihrer Anwendung nicht ändern** .

Bereichsfunktionen und Filter

AngularJS verfügt über eine Digest-Schleife und alle Ihre Funktionen in einer Ansicht. Filter werden jedes Mal ausgeführt, wenn der Digest-Zyklus ausgeführt wird. Die Digest-Schleife wird bei jeder Aktualisierung des Modells ausgeführt und kann Ihre App verlangsamen (der Filter kann mehrmals aufgerufen werden, bevor die Seite geladen wird).

Sie sollten dies vermeiden:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Besserer Ansatz

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Wo Controller-Probe ist:

```
.controller("bigCalulations", function(valueService) {
  // bad, because this is called in every digest loop
  this.calculateMe = function() {
    var t = 0;
    for(i = 0; i < 1000; i++) {
      t = t + i;
    }
    return t;
  }
  //good, because it is executed just once and logic is separated in service to    keep the
controller light
  this.preCalulatedValue = valueService.caluclateSumm(); // returns 499500
});
```

Beobachter

Beobachter müssen einen bestimmten Wert überwachen und feststellen, dass dieser Wert geändert wird.

Nach dem Aufruf `$watch()` oder `$watchCollection` neuer Watcher wird zur internen Watcher-Sammlung im aktuellen Bereich `$watchCollection` .

Was ist also ein Beobachter?

Watcher ist eine einfache Funktion, die bei jedem Digest-Zyklus aufgerufen wird und einen Wert zurückgibt. Angular prüft den zurückgegebenen Wert, falls er nicht derselbe ist wie beim vorherigen Aufruf. Ein Rückruf, der im zweiten Parameter an die Funktion `$watch()` oder `$watchCollection` wird ausgeführt.

```
(function() {
  angular.module("app", []).controller("ctrl", function($scope) {
    $scope.value = 10;
    $scope.$watch(
      function() { return $scope.value; },
      function() { console.log("value changed"); }
    );
  }
})();
```

Beobachter sind Performancekiller. Je mehr Beobachter Sie haben, desto länger dauert es, bis Sie eine Digest-Schleife erstellen, desto langsamer wird die Benutzeroberfläche. Wenn ein Beobachter Änderungen erkennt, startet er die Digest-Schleife (Neuberechnung auf allen Bildschirmen).

Es gibt drei Möglichkeiten zur manuellen Überwachung der variablen Änderungen in Angular.

`$watch()` - achtet nur auf Wertänderungen

`$watchCollection()` - `$watchCollection()` Änderungen in der Sammlung (Uhren, die mehr als normale $ watch sind)

`$watch(..., true)` - **Vermeiden Sie dies** so weit wie möglich, es wird "Deep Watch" ausgeführt und die Performance wird beendet (Uhren mehr als watchCollection)

Wenn Sie Variablen in der Ansicht binden, erstellen Sie neue Beobachter. Verwenden Sie `{{::variable}}` , um keinen Beobachter zu erstellen, insbesondere in Schleifen

Als Ergebnis müssen Sie nachverfolgen, wie viele Beobachter Sie verwenden. Sie können die Beobachter mit diesem Skript zählen (Verdienst @Words) Like Jared - Wie zählt man die Gesamtzahl der Uhren auf einer Seite?

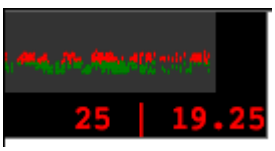
```
(function() {
  var root = angular.element(document.getElementsByTagName("body")),
      watchers = [];

  var f = function(element) {
```

```
    angular.forEach(["$scope", "$isolateScope"], function(scopeProperty) {
      if(element.data() && element.data().hasOwnProperty(scopeProperty)) {
        angular.forEach(element.data()[scopeProperty].$$watchers, function(watcher) {
          watchers.push(watcher);
        });
      }
    });

    angular.forEach(element.children(), function(childElement) {
      f(angular.element(childElement));
    });

  };

  f(root);

  // Remove duplicate watchers
  var watchersWithoutDuplicates = [];
  angular.forEach(watchers, function(item) {
    if(watchersWithoutDuplicates.indexOf(item) < 0) {
      watchersWithoutDuplicates.push(item);
    }
  });

  console.log(watchersWithoutDuplicates.length);

})();
```

Wenn Sie kein eigenes Skript erstellen möchten, gibt es ein Open Source-Dienstprogramm namens ng-stats , das ein Echtzeit-Diagramm verwendet, das in die Seite eingebettet ist, um Ihnen einen Einblick in die Anzahl der von Angular verwalteten Uhren sowie in die Häufigkeit und Dauer der Digest-Zyklen über die Zeit. Das Dienstprogramm macht eine globale Funktion mit dem Namen `showAngularStats` , die Sie aufrufen können, um zu konfigurieren, wie das Diagramm funktionieren soll.

```
showAngularStats({
  "position": "topleft",
  "digestTimeThreshold": 16,
  "autoload": true,
  "logDigest": true,
  "logWatches": true
});
```

Der obige Beispielcode zeigt die folgende Tabelle automatisch auf der Seite an (interaktive Demo).



ng-if vs ng-show

Diese Funktionen sind im Verhalten sehr ähnlich. Der Unterschied ist, dass `ng-if` Elemente aus dem DOM entfernt. Wenn große Teile des Codes nicht angezeigt werden, ist `ng-if` der richtige

Weg. `ng-show` blendet nur die Elemente aus, behält aber alle Handler bei.

ng-if

Die Direktive ngIf entfernt oder erstellt einen Teil der DOM-Struktur basierend auf einem Ausdruck. Wenn der Ausdruck ngIf einen falschen Wert ergibt, wird das Element aus dem DOM entfernt. Andernfalls wird ein Klon des Elements erneut in das DOM eingefügt.

ng-show

Die ngShow-Direktive zeigt oder versteckt das angegebene HTML-Element basierend auf dem Ausdruck, der dem ngShow-Attribut bereitgestellt wird. Das Element wird durch Entfernen oder Hinzufügen der CSS-Klasse ng-hide zum Element angezeigt oder ausgeblendet.

Beispiel

```
<div ng-repeat="user in userCollection">
  <p ng-if="user.hasTreeLegs">I am special
    <!-- some complicated DOM -->
  </p>
  <p ng-show="user.hasSubscribed">I am aweosme
    <!-- switch this setting on and off -->
  </p>
</div>
```

Fazit

Es hängt von der Art der Nutzung, aber oft ist besser geeignet ist als die andere (zB wenn 95% der Zeit das Element nicht benötigt wird, Verwendung `ng-if` , wenn Sie das DOM - Element Sichtbarkeit umschalten müssen, verwenden Sie `ng-show`).

Im Zweifelsfall `ng-if` und testen!

Hinweis : `ng-if` erstellt einen neuen isolierten Bereich, während `ng-show` und `ng-hide` nicht funktionieren. Verwenden Sie `$parent.property` wenn die übergeordnete Bereichseigenschaft darin nicht direkt `$parent.property` ist.

Enthüllen Sie Ihr Modell

```
<div ng-controller="ExampleController">
    <form name="userForm">
        Name:
        <input type="text" name="userName"
           ng-model="user.name"
           ng-model-options="{ debounce: 1000 }" />
```

```
            <button ng-click="userForm.userName.$rollbackViewValue();
user.name=''">Clear</button><br />
    </form>
    <pre>user.name = </pre>
</div>
```

Im obigen Beispiel setzen wir einen Entprellungswert von 1000 Millisekunden (1 Sekunde). Dies ist eine beträchtliche Verzögerung, verhindert jedoch, dass die Eingabe das `ng-model` wiederholt mit vielen `$digest` Zyklen `$digest` .

Durch die Verwendung von Entprellung in Ihren Eingabefeldern und überall dort, wo kein sofortiges Update erforderlich ist, können Sie die Leistung Ihrer Angular-Apps erheblich steigern. Sie können nicht nur zeitlich verzögern, sondern auch, wenn die Aktion ausgelöst wird. Wenn Sie Ihr ng-Modell nicht bei jedem Tastendruck aktualisieren möchten, können Sie auch die Unschärfe aktualisieren.

Heben Sie die Abmeldung der Listener immer auf andere Bereiche als den aktuellen Bereich auf

Sie müssen die Registrierung von Gültigkeitsbereichen außerhalb Ihres aktuellen Gültigkeitsbereichs wie folgt aufheben:

```
//always deregister these
$rootScope.$on(...);
$scope.$parent.$on(...);
```

Sie müssen keine Listener im aktuellen Umfang abmelden, da Angle sich darum kümmern würde:

```
//no need to deregister this
$scope.$on(...);
```

`$rootScope.$on` Listener bleibt im Speicher, wenn Sie zu einem anderen Controller navigieren. Dies führt zu einem Speicherverlust, wenn der Controller außerhalb des gültigen Bereichs liegt.

Nicht

```
angular.module('app').controller('badExampleController', badExample);
badExample.$inject = ['$scope', '$rootScope'];

function badExample($scope, $rootScope) {
    $rootScope.$on('post:created', function postCreated(event, data) {});
}
```

Tun

```
angular.module('app').controller('goodExampleController', goodExample);
goodExample.$inject = ['$scope', '$rootScope'];

function goodExample($scope, $rootScope) {
    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});
```

```
    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });
}
```

Profiling und Leistung online lesen: https://riptutorial.com/de/angularjs/topic/1921/profiling-und-leistung

Kapitel 38: Routing mit ngRoute

Bemerkungen

Das `ngRoute` ist ein `ngRoute` Modul, das Routing- und Deeplinking-Dienste und Anweisungen für eckige Apps bietet.

Die vollständige Dokumentation zu `ngRoute` Sie unter https://docs.angularjs.org/api/ngRoute

Examples

Grundlegendes Beispiel

In diesem Beispiel wird das Einrichten einer kleinen Anwendung mit 3 Routen mit jeweils eigener Ansicht und Controller unter Verwendung der `controllerAs` Syntax veranschaulicht.

Wir konfigurieren unseren Router mit der Funktion `.config`

1. Wir injizieren `$routeProvider` in `.config`
2. Wir definieren unsere `.when` bei der Methode `.when` mit einem `.when` .
3. Wir liefern die `.when` Methode mit einem Objekt, das unsere `template` oder `templateUrl` , `controller` und `controllerAs` angibt

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
```

```
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

Dann definieren wir in unserem HTML- `helloRoute` unsere Navigation mithilfe von `<a>` Elementen mit `href` . Für einen `helloRoute` von `helloRoute` wir als `<a href="#/helloRoute">My route</a>`

Wir bieten unserer Ansicht auch einen Container und die Direktive `ng-view` , um unsere Routen einzuschleusen.

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Beispiel für Routenparameter

Dieses Beispiel erweitert das grundlegende Beispiel, indem Parameter in der Route übergeben werden, um sie in der Steuerung zu verwenden

Dazu müssen wir:

1. Konfigurieren Sie die Position und den Namen des Parameters im Routennamen
2. `$routeParams` Dienst in unserem Controller

app.js

```
angular.module('myApp', ['ngRoute'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
```

```
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', ['$routeParams', function($routeParams) {
    var routeParam = $routeParams.paramName

    if ($routeParams.message) {
        // If a param called 'message' exists, we show it's value as the message
        this.message = $routeParams.message;
    } else {
        // If it doesn't exist, we show a default message
        this.message = 'Hello world from Controller Three!';
    }
  }])
  .config(function($routeProvider) {
    $routeProvider
    .when('/one', {
      templateUrl: 'view-one.html',
      controller: 'controllerOne',
      controllerAs: 'ctrlOne'
    })
    .when('/two', {
      templateUrl: 'view-two.html',
      controller: 'controllerTwo',
      controllerAs: 'ctrlTwo'
    })
    .when('/three', {
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    .when('/three/:message', { // We will pass a param called 'message' with this route
      templateUrl: 'view-three.html',
      controller: 'controllerThree',
      controllerAs: 'ctrlThree'
    })
    // redirect to here if no other routes match
    .otherwise({
      redirectTo: '/one'
    });
  });
```

Wenn Sie dann keine Änderungen an unseren Vorlagen vornehmen und nur einen neuen Link mit einer benutzerdefinierten Nachricht hinzufügen, wird die neue benutzerdefinierte Nachricht in unserer Ansicht angezeigt.

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a href="#/one">View One</a>
    <a href="#/two">View Two</a>
    <a href="#/three">View Three</a>
    <!-- New link with custom message -->
    <a href="#/three/This-is-a-message">View Three with "This-is-a-message" custom message</a>
  </nav>
  <!-- views will be injected here -->
  <div ng-view></div>
  <!-- templates can live in normal html files -->
```

```
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Definieren von benutzerdefiniertem Verhalten für einzelne Routen

Die einfachste Art, ein benutzerdefiniertes Verhalten für einzelne Routen zu definieren, ist ziemlich einfach.

In diesem Beispiel verwenden wir es, um einen Benutzer zu authentifizieren:

1) `routes.js` : Erstellen `requireAuth` für jede gewünschte Route eine neue Eigenschaft (wie " `requireAuth`)

```
angular.module('yourApp').config(['$routeProvider', function($routeProvider) {
    $routeProvider
        .when('/home', {
            templateUrl: 'templates/home.html',
            requireAuth: true
        })
        .when('/login', {
            templateUrl: 'templates/login.html',
        })
        .otherwise({
            redirectTo: '/home'
        });
}])
```

2) Überprüfen Sie in einem Top-Tier-Controller, der nicht an ein Element in der `ng-view` gebunden ist (um Konflikte mit dem `$routeProvider` zu vermeiden), ob die `requireAuth` Eigenschaft die Eigenschaft " `newUrl` hat, und `requireAuth` sich entsprechend

```
angular.module('YourApp').controller('YourController', ['$scope', 'session', '$location',
    function($scope, session, $location) {

        $scope.$on('$routeChangeStart', function(angularEvent, newUrl) {

            if (newUrl.requireAuth && !session.user) {
                // User isn't authenticated
                $location.path("/login");
            }

        });
    }
]);
```

Routing mit ngRoute online lesen: https://riptutorial.com/de/angularjs/topic/2391/routing-mit-ngroute

Kapitel 39: SignalR mit AngularJs

Einführung

In diesem Artikel konzentrieren wir uns auf "So erstellen Sie ein einfaches Projekt mit AngularJs And SignalR". In diesem Training müssen Sie wissen, wie "App mit AngleJs erstellen", "Service für AngularJs erstellen / verwenden" und Grundkenntnisse über SignalR "Dazu empfehlen wir https://www.codeproject.com/Tips/590660/Introduction-to-SignalR) .

Examples

SignalR und AngularJs [ChatProject]

Schritt 1: Projekt erstellen

```
- Application
    - app.js
    - Controllers
        - appController.js
    - Factories
        - SignalR-factory.js
- index.html
- Scripts
    - angular.js
    - jquery.js
    - jquery.signalR.min.js
- Hubs
```

Verwendung der SignalR-Version: SignalR-2.2.1

Schritt 2: Startup.cs und ChatHub.cs

Wechseln Sie in Ihr Verzeichnis *"/ Hubs"* und fügen Sie 2 Dateien hinzu *[Startup.cs, ChatHub.cs].*

Startup.cs

```
using Microsoft.Owin;
using Owin;
[assembly: OwinStartup(typeof(SignalR.Hubs.Startup))]

namespace SignalR.Hubs
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.MapSignalR();
        }
    }
}
```

ChatHub.cs

```
using Microsoft.AspNet.SignalR;

namespace SignalR.Hubs
{
    public class ChatHub : Hub
    {
        public void Send(string name, string message, string time)
        {
            Clients.All.broadcastMessage(name, message, time);
        }
    }
}
```

Schritt 3: Winkel-App erstellen

Wechseln Sie in Ihr Verzeichnis *"/ Application"* und fügen *Sie die* Datei *[app.js] hinzu*

app.js

```
var app = angular.module("app", []);
```

Schritt 4: SignalR Factory erstellen

Wechseln Sie in das Verzeichnis *"/ Application / Factories"* und fügen *Sie die* Datei *[SignalR-factory.js] hinzu*

SignalR-factory.js

```
app.factory("signalR", function () {
    var factory = {};

    factory.url = function (url) {
        $.connection.hub.url = url;
    }

    factory.setHubName = function (hubName) {
        factory.hub = hubName;
    }

    factory.connectToHub = function () {
        return $.connection[factory.hub];
    }

    factory.client = function () {
        var hub = factory.connectToHub();
        return hub.client;
    }

    factory.server = function () {
        var hub = factory.connectToHub();
        return hub.server;
    }

    factory.start = function (fn) {
```

```
        return $.connection.hub.start().done(fn);
    }

    return factory;
});
```

Schritt 5: Aktualisieren Sie app.js

```
var app = angular.module("app", []);

app.run(function(signalR) {
    signalR.url("http://localhost:21991/signalr");
});
```

localhost: 21991 / signalr | **Dies ist Ihre SignalR Hubs-URL**

Schritt 6: Controller hinzufügen

Wechseln Sie in das Verzeichnis *"/ Application / Controllers"* und fügen *Sie die* Datei *[appController.js] hinzu*

```
app.controller("ctrl", function ($scope, signalR) {
    $scope.messages = [];
    $scope.user = {};

    signalR.setHubName("chatHub");

    signalR.client().broadcastMessage = function (name, message, time) {
        var newChat = { name: name, message: message, time: time };

        $scope.$apply(function() {
            $scope.messages.push(newChat);
        });
    };

    signalR.start(function () {
        $scope.send = function () {
            var dt = new Date();
            var time = dt.getHours() + ":" + dt.getMinutes() + ":" + dt.getSeconds();

            signalR.server().send($scope.user.name, $scope.user.message, time);
        }
    });
});
```

signalR.setHubName ("chatHub") | **[ChatHub] (öffentliche Klasse)>** *ChatHub.cs*

Hinweis: *Fügen* Sie nicht *HubName* mit Großbuchstaben ein, der **erste Buchstabe** ist Kleinbuchstabe.

signalR.client () | Versuchen Sie, eine Verbindung zu Ihren Hubs herzustellen und alle Funktionen in den Hubs zu erhalten. In diesem Beispiel haben wir "chatHub", um die Funktion "broadcastMessage ()" zu erhalten.

Schritt 7: Fügen Sie index.html im Verzeichnispfad hinzu

index.html

```
<!DOCTYPE html>
<html ng-app="app" ng-controller="ctrl">
<head>
    <meta charset="utf-8" />
    <title>SignalR Simple Chat</title>
</head>
<body>
    <form>
        <input type="text" placeholder="name" ng-model="user.name" />
        <input type="text" placeholder="message" ng-model="user.message" />
        <button ng-click="send()">send</button>

        <ul>
            <li ng-repeat="item in messages">
                <b ng-bind="item.name"></b> <small ng-bind="item.time"></small> :
{{item.message}}
            </li>
        </ul>
    </form>

    <script src="Scripts/angular.min.js"></script>
    <script src="Scripts/jquery-1.6.4.min.js"></script>
    <script src="Scripts/jquery.signalR-2.2.1.min.js"></script>
    <script src="signalr/hubs"></script>
    <script src="app.js"></script>
    <script src="SignalR-factory.js"></script>
</body>
</html
```

Ergebnis mit Bild

Benutzer 1 (senden und empfangen)

Benutzer 2 (senden und empfangen)

SignalR mit AngularJs online lesen: https://riptutorial.com/de/angularjs/topic/9964/signalr-mit-angularjs

Kapitel 40: Sitzungsspeicher

Examples

Behandeln des Sitzungsspeichers über den Service mithilfe von "anglejs"

Sitzungsspeicherdienst:

Gemeinsamer Werksdienst, der die gespeicherten Sitzungsdaten auf der Grundlage des Schlüssels speichert und zurückgibt.

```
 'use strict';

/**
 * @ngdoc factory
 * @name app.factory:storageService
 * @description This function will communicate with HTML5 sessionStorage via Factory Service.
 */

app.factory('storageService', ['$rootScope', function($rootScope) {

    return {
        get: function(key) {
            return sessionStorage.getItem(key);
        },
        save: function(key, data) {
            sessionStorage.setItem(key, data);
        }
    };
}]);
```

Im Controller:

Fügen Sie die storageService-Abhängigkeit in den Controller ein, um die Daten aus dem Sitzungsspeicher festzulegen und abzurufen.

```
app.controller('myCtrl',['storageService',function(storageService) {

  // Save session data to storageService
  storageService.save('key', 'value');

  // Get saved session data from storageService
  var sessionData = storageService.get('key');

});
```

Sitzungsspeicher online lesen: https://riptutorial.com/de/angularjs/topic/8201/sitzungsspeicher

Kapitel 41: ui-router

Bemerkungen

Was ist ein `ui-router` ?

> Angular UI-Router ist ein clientseitiges Routing-Framework für Single Page Application für AngularJS.
>
> Routing-Frameworks für SPAs aktualisieren die Browser-URL, wenn der Benutzer durch die App navigiert. Umgekehrt können auf diese Weise Änderungen an der URL des Browsers vorgenommen werden, um die Navigation durch die App zu steuern. Auf diese Weise kann der Benutzer ein Lesezeichen an einer Stelle im SPA erstellen.
>
> UI-Router-Anwendungen werden als hierarchische Zustandsstruktur modelliert. Der UI-Router stellt eine Zustandsmaschine bereit, um die Übergänge zwischen diesen Anwendungszuständen transaktionsartig zu verwalten.

Nützliche Links

Die offizielle API-Dokumentation finden Sie hier . Bei Fragen zu `ui-router` VS `ng-router` können Sie eine ziemlich ausführliche Antwort finden hier . Denken Sie daran, dass `ng-router` bereits ein neues ng-router-Update namens `ngNewRouter` (kompatibel mit Angular 1.5 + / 2.0) veröffentlicht hat, das Zustände wie der ui-router unterstützt. Weitere `ngNewRouter` zu `ngNewRouter` hier .

Examples

Basisbeispiel

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .config(function($stateProvider, $urlRouterProvider) {
    $stateProvider
      .state('one', {
        url: "/one",
        templateUrl: "view-one.html",
        controller: 'controllerOne',
        controllerAs: 'ctrlOne'
      })
      .state('two', {
```

```
        url: "/two",
        templateUrl: "view-two.html",
        controller: 'controllerTwo',
        controllerAs: 'ctrlTwo'
      })
      .state('three', {
        url: "/three",
        templateUrl: "view-three.html",
        controller: 'controllerThree',
        controllerAs: 'ctrlThree'
      });

      $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">View One</a>
    <a ui-sref="two">View Two</a>
    <a ui-sref="three">View Three</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>
</div>
```

Mehrere Ansichten

app.js

```
angular.module('myApp', ['ui.router'])
  .controller('controllerOne', function() {
    this.message = 'Hello world from Controller One!';
  })
  .controller('controllerTwo', function() {
    this.message = 'Hello world from Controller Two!';
  })
  .controller('controllerThree', function() {
    this.message = 'Hello world from Controller Three!';
  })
  .controller('controllerFour', function() {
    this.message = 'Hello world from Controller Four!';
  })
  .config(function($stateProvider, $urlRouterProvider) {
```

```
    $stateProvider
      .state('one', {
        url: "/one",
        views: {
          "viewA": {
            templateUrl: "view-one.html",
            controller: 'controllerOne',
            controllerAs: 'ctrlOne'
          },
          "viewB": {
            templateUrl: "view-two.html",
            controller: 'controllerTwo',
            controllerAs: 'ctrlTwo'
          }
        }
      })
      .state('two', {
        url: "/two",
        views: {
          "viewA": {
            templateUrl: "view-three.html",
            controller: 'controllerThree',
            controllerAs: 'ctrlThree'
          },
          "viewB": {
            templateUrl: "view-four.html",
            controller: 'controllerFour',
            controllerAs: 'ctrlFour'
          }
        }
      });

    $urlRouterProvider.otherwise('/one');
  });
```

index.html

```
<div ng-app="myApp">
  <nav>
    <!-- links to switch routes -->
    <a ui-sref="one">Route One</a>
    <a ui-sref="two">Route Two</a>
  </nav>
  <!-- views will be injected here -->
  <div ui-view="viewA"></div>
  <div ui-view="viewB"></div>
  <!-- templates can live in normal html files -->
  <script type="text/ng-template" id="view-one.html">
    <h1>{{ctrlOne.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-two.html">
    <h1>{{ctrlTwo.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-three.html">
    <h1>{{ctrlThree.message}}</h1>
  </script>

  <script type="text/ng-template" id="view-four.html">
```

```
    <h1>{{ctrlFour.message}}</h1>
  </script>
</div>
```

Verwenden von Auflösungsfunktionen zum Laden von Daten

app.js

```
angular.module('myApp', ['ui.router'])
  .service('User', ['$http', function User ($http) {
    this.getProfile = function (id) {
      return $http.get(...) // method to load data from API
    };
  }])
  .controller('profileCtrl', ['profile', function profileCtrl (profile) {
    // inject resolved data under the name of the resolve function
    // data will already be returned and processed
    this.profile = profile;
  }])
  .config(['$stateProvider', '$urlRouterProvider', function ($stateProvider,
$urlRouterProvider) {
    $stateProvider
      .state('profile', {
        url: "/profile/:userId",
        templateUrl: "profile.html",
        controller: 'profileCtrl',
        controllerAs: 'vm',
        resolve: {
          profile: ['$stateParams', 'User', function ($stateParams, User) {
            // $stateParams will contain any parameter defined in your url
            return User.getProfile($stateParams.userId)
              // .then is only necessary if you need to process returned data
              .then(function (data) {
                return doSomeProcessing(data);
              });
          }]
        }
      }]);

      $urlRouterProvider.otherwise('/');
  });
```

profile.html

```
<ul>
  <li>Name: {{vm.profile.name}}</li>
  <li>Age: {{vm.profile.age}}</li>
  <li>Sex: {{vm.profile.sex}}</li>
</ul>
```

Hier können Sie den UI-Router-Wiki-Eintrag anzeigen .

Auflösungsfunktionen müssen aufgelöst werden, bevor das `$stateChangeSuccess` Ereignis ausgelöst wird. Das bedeutet, dass die Benutzeroberfläche nicht geladen wird, bis *alle* Auflösungsfunktionen für den Status abgeschlossen sind. Auf diese Weise können Sie sicherstellen, dass Daten für Ihren Controller und Ihre Benutzeroberfläche verfügbar sind. Sie können jedoch sehen, dass eine

Auflösungsfunktion schnell sein sollte, um das Hängen der Benutzeroberfläche zu vermeiden.

Verschachtelte Ansichten / Zustände

app.js

```
var app = angular.module('myApp',['ui.router']);

app.config(function($stateProvider,$urlRouterProvider) {

    $stateProvider

    .state('home', {
        url: '/home',
        templateUrl: 'home.html',
        controller: function($scope){
            $scope.text = 'This is the Home'
        }
    })

    .state('home.nested1',{
        url: '/nested1',
        templateUrl:'nested1.html',
        controller: function($scope){
            $scope.text1 = 'This is the nested view 1'
        }
    })

    .state('home.nested2',{
        url: '/nested2',
        templateUrl:'nested2.html',
        controller: function($scope){
            $scope.fruits = ['apple','mango','oranges'];
        }
    });

    $urlRouterProvider.otherwise('/home');

});
```

index.html

```
  <div ui-view></div>
   <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.8/angular.min.js"></script>
   <script src="angular-ui-router.min.js"></script>
   <script src="app.js"></script>
```

home.html

```
<div>
<h1> {{text}} </h1>
<br>
    <a ui-sref="home.nested1">Show nested1</a>
    <br>
    <a ui-sref="home.nested2">Show nested2</a>
    <br>
```

```
    <div ui-view></div>
</div>
```

nested1.html

```
<div>
<h1> {{text1}} </h1>
</div>
```

nested2.html

```
<div>
    <ul>
      <li ng-repeat="fruit in fruits">{{ fruit }}</li>
    </ul>
</div>
```

ui-router online lesen: https://riptutorial.com/de/angularjs/topic/2545/ui-router

Kapitel 42: Unit-Tests

Bemerkungen

Dieses Thema enthält Beispiele für das Testen von Einheiten der verschiedenen Konstrukte in AngularJS. Unit-Tests werden oft mit Jasmine geschrieben , einem beliebten verhaltensorientierten Test-Framework. Beim Komponententest für Winkelkonstrukte müssen Sie ngMock als Abhängigkeit beim Ausführen der Komponententests angeben .

Examples

Gerätetest einen Filter

Filtercode:

```
angular.module('myModule', []).filter('multiplier', function() {
  return function(number, multiplier) {
    if (!angular.isNumber(number)) {
      throw new Error(number + " is not a number!");
    }
    if (!multiplier) {
      multiplier = 2;
    }
    return number * multiplier;
  }
});
```

Der Test:

```
describe('multiplierFilter', function() {
  var filter;

  beforeEach(function() {
    module('myModule');
    inject(function(multiplierFilter) {
      filter = multiplierFilter;
    });
  });

  it('multiply by 2 by default', function() {
    expect(filter(2)).toBe(4);
    expect(filter(3)).toBe(6);
  });

  it('allow to specify custom multiplier', function() {
    expect(filter(2, 4)).toBe(8);
  });

  it('throws error on invalid input', function() {
    expect(function() {
      filter(null);
    }).toThrow();
```

```
  });
});
```

Lauf!

Anmerkung: Bei dem `inject` Aufruf im Test muss Ihr Filter durch seinen Namen + *Filter angegeben werden* . Der Grund dafür ist, dass Angular jedes Mal, wenn Sie einen Filter für Ihr Modul registrieren, diesen mit einem an seinen Namen angehängten `Filter` registriert.

Komponententest einer Komponente (1.5+)

Komponentencode:

```
angular.module('myModule', []).component('myComponent', {
  bindings: {
    myValue: '<'
  },
  controller: function(MyService) {
    this.service = MyService;
    this.componentMethod = function() {
      return 2;
    };
  }
});
```

Der Test:

```
describe('myComponent', function() {
  var component;

  var MyServiceFake = jasmine.createSpyObj(['serviceMethod']);

  beforeEach(function() {
    module('myModule');
    inject(function($componentController) {
      // 1st - component name, 2nd - controller injections, 3rd - bindings
      component = $componentController('myComponent', {
        MyService: MyServiceFake
      }, {
        myValue: 3
      });
    });
  });

  /** Here you test the injector. Useless. */

  it('injects the binding', function() {
    expect(component.myValue).toBe(3);
  });

  it('has some cool behavior', function() {
    expect(component.componentMethod()).toBe(2);
  });
});
```

Lauf!

Gerätetest einer Steuerung

Controller-Code:

```
angular.module('myModule', [])
  .controller('myController', function($scope) {
    $scope.num = 2;
    $scope.doSomething = function() {
      $scope.num += 2;
    }
  });
```

Der Test:

```
describe('myController', function() {
  var $scope;
  beforeEach(function() {
    module('myModule');
    inject(function($controller, $rootScope) {
      $scope = $rootScope.$new();
      $controller('myController', {
        '$scope': $scope
      })
    });
  });
  it('should increment `num` by 2', function() {
    expect($scope.num).toEqual(2);
    $scope.doSomething();
    expect($scope.num).toEqual(4);
  });
});
```

Lauf!

Testen Sie einen Dienst

Service code

```
angular.module('myModule', [])
  .service('myService', function() {
    this.doSomething = function(someNumber) {
      return someNumber + 2;
    }
  });
```

Der Test

```
describe('myService', function() {
  var myService;
  beforeEach(function() {
    module('myModule');
    inject(function(_myService_) {
      myService = _myService_;
    });
  });
```

```
  it('should increment `num` by 2', function() {
    var result = myService.doSomething(4);
    expect(result).toEqual(6);
  });
});
```

Lauf!

Unit-Test eine Anweisung

Richtliniencode

```
angular.module('myModule', [])
  .directive('myDirective', function() {
    return {
      template: '<div>{{greeting}} {{name}}!</div>',
      scope: {
        name: '=',
        greeting: '@'
      }
    };
  });
```

Der Test

```
describe('myDirective', function() {
  var element, scope;
  beforeEach(function() {
    module('myModule');
    inject(function($compile, $rootScope) {
      scope = $rootScope.$new();
      element = angular.element("<my-directive name='name' greeting='Hello'></my-directive>");
      $compile(element)(scope);
      /* PLEASE NEVER USE scope.$digest(). scope.$apply use a protection to avoid to run a
digest loop when there is already one, so, use scope.$apply() instead. */
      scope.$apply();
    })
  });

  it('has the text attribute injected', function() {
    expect(element.html()).toContain('Hello');
  });

  it('should have proper message after scope change', function() {
    scope.name = 'John';
    scope.$apply();
    expect(element.html()).toContain("John");
    scope.name = 'Alice';
    expect(element.html()).toContain("John");
    scope.$apply();
    expect(element.html()).toContain("Alice");
  });
});
```

Lauf!

Unit-Tests online lesen: https://riptutorial.com/de/angularjs/topic/1689/unit-tests

Kapitel 43: Unterscheidungsdienst vs Fabrik

Examples

Werks-VS-Service ein für alle Mal

Per Definition:

Dienste sind im Grunde Konstruktorfunktionen. Sie verwenden dieses Schlüsselwort.

Fabriken sind einfache Funktionen, die ein Objekt zurückgeben.

Unter der Haube:

Fabriken rufen intern den Provider an.

Dienste rufen intern die Werksfunktion auf.

Debatte:

Fabriken können Code ausführen, bevor wir unser Objektliteral zurückgeben.

Services können aber auch so geschrieben werden, dass sie ein Objektliteral zurückgeben und Code ausführen, bevor sie zurückgegeben werden. Das ist kontraproduktiv, da Services als Konstruktorfunktion fungieren sollen.

Tatsächlich können Konstruktorfunktionen in JavaScript das zurückgeben, was sie wollen.

Also welches ist besser?

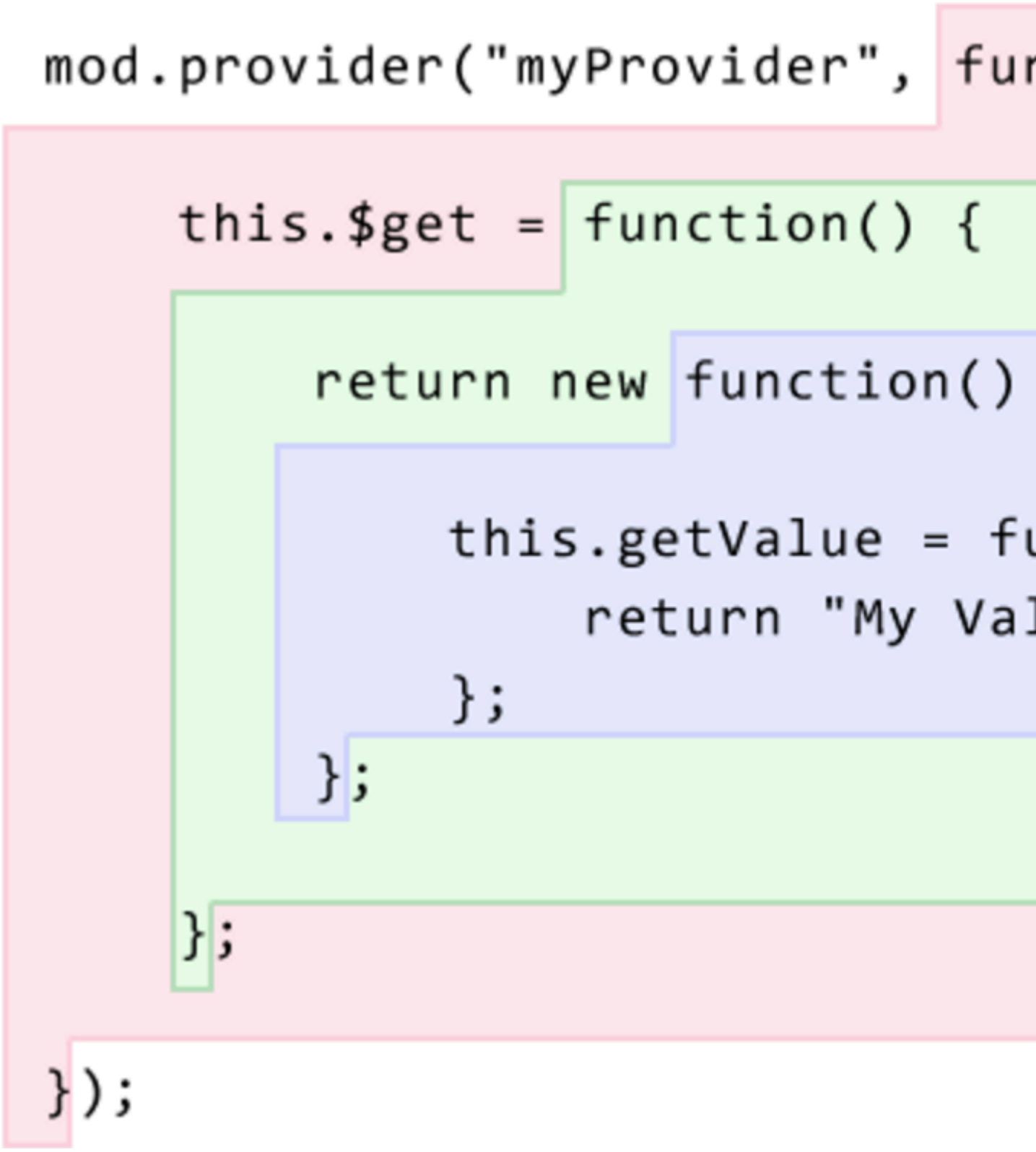
Die Konstruktorsyntax von Services entspricht eher der Klassensyntax von ES6. Die Migration wird also einfach sein.

Zusammenfassung

Zusammenfassend sind also Provider, Werk und Service Anbieter.

Eine Factory ist ein Spezialfall eines Providers, wenn Sie in Ihrem Provider nur eine $ get () - Funktion benötigen. Es erlaubt Ihnen, es mit weniger Code zu schreiben.

Ein Service ist ein Sonderfall einer Factory, in der Sie eine Instanz eines neuen Objekts mit dem gleichen Vorteil zurückgeben möchten, dass Sie weniger Code schreiben.



Unterscheidungsdienst vs Fabrik online lesen:
https://riptutorial.com/de/angularjs/topic/7099/unterscheidungsdienst-vs-fabrik

Kapitel 44: Veranstaltungen

Parameter

Parameter	Werttypen
Veranstaltung	Objekt {name: "eventName", targetScope: Scope, defaultPrevented: false, currentScope: ChildScope}
args	Daten, die zusammen mit der Ereignisausführung übergeben wurden

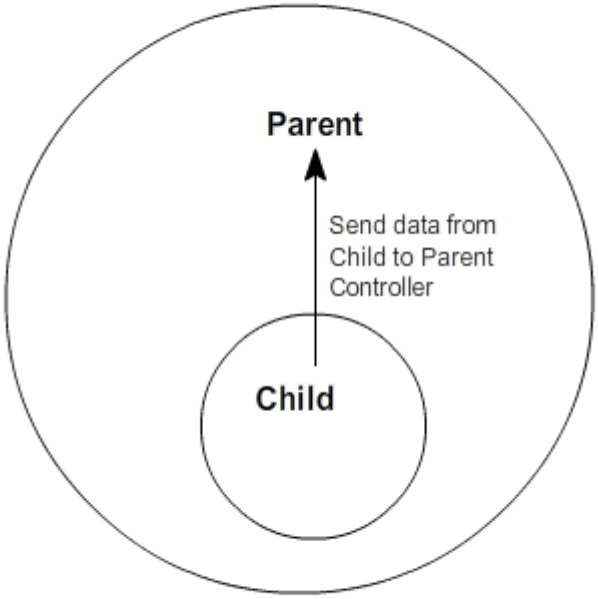
Examples

Verwenden eines Winkelereignissystems

$ scope. $ emit

Die Verwendung von `$scope.$emit` löst einen Ereignisnamen durch die Gültigkeitsbereichshierarchie nach oben aus und informiert den `$scope` Der Ereignislebenszyklus beginnt in dem Bereich, in dem `$emit` aufgerufen wurde.

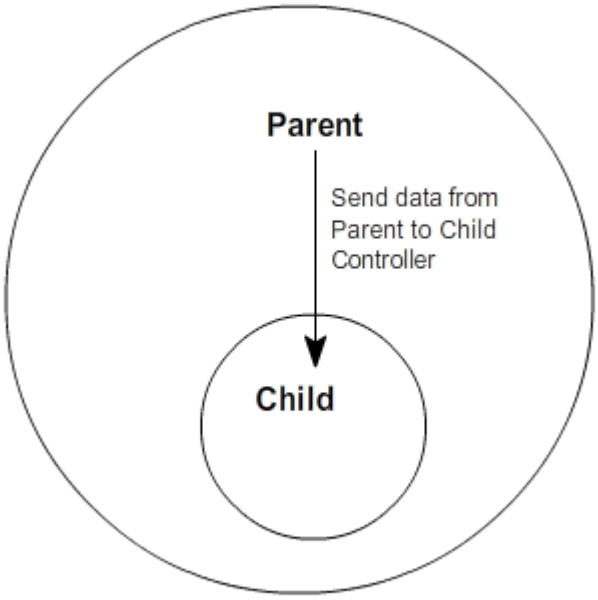
Wireframe arbeiten:



$ scope. $ broadcast

Mit `$scope.$broadcast` wird ein Ereignis im `$scope` . Wir können diese Ereignisse mit `$scope.$on`

Wireframe arbeiten:



Syntax :

```
// firing an event upwards
$scope.$emit('myCustomEvent', 'Data to send');

// firing an event downwards
$scope.$broadcast('myCustomEvent', {
  someProp: 'some value'
});

// listen for the event in the relevant $scope
$scope.$on('myCustomEvent', function (event, data) {
  console.log(data); // 'Data from the event'
});
```

Anstelle von `$scope` Sie `$rootScope` . In diesem Fall ist Ihr Ereignis in allen Controllern verfügbar, unabhängig von diesem Controller-Bereich

Saubere registrierte Veranstaltung in AngularJS

Der Grund für das Bereinigen der registrierten Ereignisse, da selbst der Controller die Verarbeitung registrierter Ereignisse zerstört hat, ist immer noch am Leben. Der Code wird also als unerwartet ausgeführt.

```
// firing an event upwards
$rootScope.$emit('myEvent', 'Data to send');

// listening an event
var listenerEventHandler = $rootScope.$on('myEvent', function(){
    //handle code
});

$scope.$on('$destroy', function() {
    listenerEventHandler();
});
```

Verwendung und Bedeutung

Diese Ereignisse können verwendet werden, um zwischen zwei oder mehr Controllern zu kommunizieren.

`$emit` ein Ereignis Versendungen nach oben durch den Umfang Hierarchie, während `$broadcast` ein Ereignis nach unten auf alle Kind entsendet scopes.This schön erklärt wurde hier .

Bei der Kommunikation zwischen Steuerungen können grundsätzlich zwei Arten von Szenarien verwendet werden:

---

1. Wenn Controller eine Eltern-Kind-Beziehung haben. (In solchen Szenarien können wir meistens `$scope` verwenden.)

---

2. Wenn Controller nicht unabhängig voneinander sind und über die Aktivitäten des anderen informiert werden müssen. (In solchen Szenarien können wir `$rootScope` verwenden.)

Beispiel: Für jede E-Commerce-Website haben wir `ProductListController` (die die Produktlistenseite kontrolliert, wenn auf eine Produktmarke geklickt wird) und `CartController` (zum Verwalten von `CartController`). Wenn wir nun auf **die** Schaltfläche "In den **Warenkorb"** klicken, muss dies ebenfalls `CartController` mitgeteilt werden, damit die Anzahl / Details der neuen Artikel in der Navigationsleiste der Website `CartController` können. Dies kann mit `$rootScope` erreicht werden.

Mit `$scope.$emit`

```
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
```

```
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
            });
        });

        app.controller("SecondController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$emit('eventName', {message: msg});
            };
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px ;padding:5px;">
        <h1>Parent Controller</h1>
        <p>Emit Message : {{message}}</p>
        <br />
        <div ng-controller="SecondController" style="border:2px;padding:5px;">
            <h1>Child Controller</h1>
            <input ng-model="msg">
            <button ng-click="handleClick(msg);">Emit</button>
        </div>
    </div>
  </body>
</html>
```

Mit `$scope.$broadcast` :

```
<html>
  <head>
    <title>Broadcasting</title>
    <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.4/angular.js"></script>
    <script>
        var app = angular.module('app', []);

        app.controller("FirstController", function ($scope) {
            $scope.handleClick = function (msg) {
                $scope.$broadcast('eventName', {message: msg});
            };

        });

        app.controller("SecondController", function ($scope) {
            $scope.$on('eventName', function (event, args) {
                $scope.message = args.message;
            });
        });

    </script>
  </head>
  <body ng-app="app">
    <div ng-controller="FirstController" style="border:2px solid ; padding:5px;">
        <h1>Parent Controller</h1>
        <input ng-model="msg">
        <button ng-click="handleClick(msg);">Broadcast</button>
        <br /><br />
        <div ng-controller="SecondController" style="border:2px solid ;padding:5px;">
            <h1>Child Controller</h1>
            <p>Broadcast Message : {{message}}</p>
```

```
        </div>
    </div>
  </body>
</html>
```

Entfernen Sie $ rootScope. $ Immer auf Listener des scope $ destory - Ereignisses

$ rootScope. $ on Listener bleibt im Speicher, wenn Sie zu einem anderen Controller navigieren. Dies führt zu einem Speicherverlust, wenn der Controller außerhalb des gültigen Bereichs liegt.

Nicht

```
angular.module('app').controller('badExampleController', badExample);

badExample.$inject = ['$scope', '$rootScope'];
function badExample($scope, $rootScope) {

    $rootScope.$on('post:created', function postCreated(event, data) {});

}
```

Tun

```
angular.module('app').controller('goodExampleController', goodExample);

goodExample.$inject = ['$scope', '$rootScope'];
function goodExample($scope, $rootScope) {

    var deregister = $rootScope.$on('post:created', function postCreated(event, data) {});

    $scope.$on('$destroy', function destroyScope() {
        deregister();
    });

}
```

Veranstaltungen online lesen: https://riptutorial.com/de/angularjs/topic/1922/veranstaltungen

Kapitel 45: Verwenden von AngularJS mit TypeScript

Syntax

- $ scope: ng.IScope - Dies ist eine Methode in Typescript, um den Typ für eine bestimmte Variable zu definieren.

Examples

Winkelsteuerungen in Typoscript

Wie in der AngularJS- Dokumentation definiert

> Wenn ein Controller über die ng-controller-Direktive an das DOM angeschlossen wird, instanziiert Angular ein neues Controller-Objekt mit der angegebenen Konstruktorfunktion des Controllers. Ein neuer untergeordneter Bereich wird erstellt und der Konstruktorfunktion des Controllers als ein injizierbarer Parameter als $ scope zur Verfügung gestellt.

Controller lassen sich mit den Typoscript-Klassen sehr einfach erstellen.

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
                //n is string and so is o
            });
        };
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

Das resultierende Javascript ist

```
var App;
(function (App) {
```

```
    var Controllers;
    (function (Controllers) {
        var Address = (function () {
            function Address() {
            }
            return Address;
        }());
        var SampleController = (function () {
            function SampleController($scope) {
                this.setUpWatches($scope);
            }
            SampleController.prototype.setUpWatches = function ($scope) {
                var _this = this;
                $scope.$watch(function () { return _this.firstName; }, function (n, o) {
                    //n is string and so is o
                });
            };
            ;
            return SampleController;
        }());
        Controllers.SampleController = SampleController;
    })(Controllers = App.Controllers || (App.Controllers = {}));
})(App || (App = {}));
//# sourceMappingURL=ExampleController.js.map
```

Nachdem Sie die Controller-Klasse erstellt haben, lassen Sie das Winkelmodul js über den Controller mithilfe der Klasse einfach ausführen

```
app
 .module('app')
 .controller('exampleController', App.Controller.SampleController)
```

Verwenden des Controllers mit der ControllerAs-Syntax

Der von uns erstellte Controller kann instanziiert und unter Verwendung des `controller as` Syntax verwendet werden. Das liegt daran, dass wir Variable direkt in die Controller-Klasse und nicht in den `$scope` .

`controller as someName` zu verwenden, bedeutet, den Controller von `$scope` `controller as someName` ist es nicht erforderlich, $ scope als Abhängigkeit in den Controller `controller as someName` .

Traditioneller Weg:

```
// we are using $scope object.
app.controller('MyCtrl', function ($scope) {
  $scope.name = 'John';
});

<div ng-controller="MyCtrl">
  {{name}}
</div>
```

Jetzt mit `controller as` Syntax :

```
// we are using the "this" Object instead of "$scope"
app.controller('MyCtrl', function() {
  this.name = 'John';
});

<div ng-controller="MyCtrl as info">
  {{info.name}}
</div>
```

Wenn Sie eine "Klasse" in JavaScript instanziieren, können Sie Folgendes tun:

```
var jsClass = function () {
  this.name = 'John';
}
var jsObj = new jsClass();
```

Jetzt können wir die **`jsObj`** Instanz verwenden, um auf jede Methode oder Eigenschaft von **`jsClass`** .

In angle verwenden wir die gleiche Art von Dingen. Wir verwenden Controller als Syntax für die Instantiierung.

Bündelung / Minimierung verwenden

Durch die Injektion des $ scope in die Konstruktorfunktionen des Controllers kann die grundlegende Option der Winkelabhängigkeitseinspritzung demonstriert und verwendet werden , sie ist jedoch nicht produktionsbereit, da sie nicht minimiert werden kann. Das liegt daran, dass das Minifizierungssystem die Variablennamen ändert und die Abhängigkeitseingabe von Anguar anhand der Parameternamen weiß, was injiziert werden muss. Für ein Beispiel ist die Konstruktorfunktion von ExampleController auf den folgenden Code reduziert.

```
function n(n){this.setUpWatches(n)
```

und `$scope` wird in `n` geändert!
Um dies zu überwinden, können wir ein $ inject-Array (`string[]`) hinzufügen. Der DI dieses Winkels weiß also, an welcher Position die Konstrukteur der Regler zu injizieren ist.
Das obige Typoskript ändert sich also zu

```
module App.Controllers {
    class Address {
        line1: string;
        line2: string;
        city: string;
        state: string;
    }
    export class SampleController {
        firstName: string;
        lastName: string;
        age: number;
        address: Address;
        setUpWatches($scope: ng.IScope): void {
            $scope.$watch(() => this.firstName, (n, o) => {
```

```
                //n is string and so is o
            });
        };
        static $inject : string[] = ['$scope'];
        constructor($scope: ng.IScope) {
            this.setUpWatches($scope);
        }
    }
}
```

Warum ControllerAs-Syntax?

Controller-Funktion

Die Controller-Funktion ist nichts weiter als eine JavaScript-Konstruktorfunktion. Wenn also eine Ansicht geladen wird, wird der `function context` (**`this`**) auf das Controller-Objekt gesetzt.

Fall 1 :

```
this.constFunction = function() { ... }
```

Sie wird im `controller object` , nicht in `$scope` . Ansichten können nicht auf die auf dem Controller-Objekt definierten Funktionen zugreifen.

Beispiel:

```
<a href="#123" ng-click="constFunction()"></a> // It will not work
```

Fall 2:

```
$scope.scopeFunction = function() { ... }
```

Sie wird im `$scope object` , nicht im `controller object` . Ansichten können nur auf die für das `$scope` Objekt definierten Funktionen zugreifen.

Beispiel:

```
<a href="#123" ng-click="scopeFunction()"></a> // It will work
```

Warum ControllerAs?

- **`ControllerAs`** Syntax macht es viel klarer , wo Objekte werden manipulated.Having `oneCtrl.name` und `anotherCtrl.name` macht es viel einfacher zu erkennen , dass Sie einen `name` von mehreren verschiedenen Controllern für unterschiedliche Zwecke zugewiesen , aber wenn beide verwendeten gleicher `$scope.name` und mit Wenn auf einer Seite zwei verschiedene HTML-Elemente vorhanden sind, die beide an **`{{name}}`** gebunden sind, ist es

schwierig zu erkennen, welches Element von welchem Controller stammt.

- Den `$scope` ausblenden und die Mitglieder über ein `intermediary object` aus der Steuerung für die Ansicht `intermediary object` . Mit `this.*` Einstellung `this.*` können wir genau das zeigen, was wir vom Controller für die Ansicht verfügbar machen möchten.

```
<div ng-controller="FirstCtrl">
    {{ name }}
    <div ng-controller="SecondCtrl">
        {{ name }}
        <div ng-controller="ThirdCtrl">
            {{ name }}
        </div>
    </div>
</div>
```

In diesem Fall ist `{{ name }}` sehr verwirrend in der Anwendung und wir wissen auch nicht, welcher Controller sich auf welchen Controller bezieht.

```
<div ng-controller="FirstCtrl as first">
    {{ first.name }}
    <div ng-controller="SecondCtrl as second">
        {{ second.name }}
        <div ng-controller="ThirdCtrl as third">
            {{ third.name }}
        </div>
    </div>
</div>
```

Warum $ Umfang?

- Verwenden Sie `$scope` wenn Sie auf eine oder mehrere Methoden von $ scope wie `$watch` , `$digest` , `$emit` , `$http` usw. zugreifen müssen.
- Beschränken Sie, welche Eigenschaften und / oder Methoden für `$scope` , und übergeben Sie sie `$scope` Bedarf explizit an `$scope` .

Verwenden von AngularJS mit TypeScript online lesen:
https://riptutorial.com/de/angularjs/topic/3477/verwenden-von-angularjs-mit-typescript

Kapitel 46: Verwendung von eingebauten Anweisungen

Examples

HTML-Elemente ausblenden / anzeigen

In diesem Beispiel werden show html-Elemente ausgeblendet.

```
<!DOCTYPE html>
<html ng-app="myDemoApp">
  <head>
    <script src="https://code.angularjs.org/1.5.8/angular.min.js"></script>
    <script>

      function HideShowController() {
        var vm = this;
        vm.show=false;
        vm.toggle= function() {
          vm.show=!vm.show;
        }
      }

      angular.module("myDemoApp", [/* module dependencies go here */])
        .controller("hideShowController", [HideShowController]);
    </script>
  </head>
  <body ng-cloak>
    <div ng-controller="hideShowController as vm">
      <a style="cursor: pointer;" ng-show="vm.show" ng-click="vm.toggle()">Show Me!</a>
      <a style="cursor: pointer;" ng-hide="vm.show" ng-click="vm.toggle()">Hide Me!</a>
    </div>
  </body>
</html>
```

Live Demo

Schritt für Schritt Erklärung:

1. `ng-app="myDemoApp"` , die ngApp- Direktive teilt wink mit, dass ein DOM-Element von einem bestimmten angle.module mit dem Namen "myDemoApp" gesteuert wird.
2. `<script src="[//angular include]">` angle `<script src="[//angular include]">` enthält winklige js.
3. `HideShowController` Funktion enthält eine weitere Funktion namens `toggle` deren Hilfe das Element `HideShowController` kann.
4. `angular.module(...)` erstellt ein neues Modul.
5. `.controller(...)` Angular Controller und `.controller(...)` das Modul zur Verkettung zurück;
6. `ng-controller` Direktive ist ein wesentlicher Aspekt dafür, wie eckig die Prinzipien des Model-View-Controller-Entwurfsmusters unterstützt.
7. `ng-show` Direktive zeigt das angegebene HTML-Element, wenn der angegebene Ausdruck

"true" ist.

8. `ng-hide` Anweisung `ng-hide` verbirgt das angegebene HTML-Element, wenn der angegebene Ausdruck "true" ist.
9. `ng-click` Direktive löst eine Umschaltfunktion im Controller aus

Verwendung von eingebauten Anweisungen online lesen:
https://riptutorial.com/de/angularjs/topic/7644/verwendung-von-eingebauten-anweisungen

Kapitel 47: Wie funktioniert die Datenbindung?

Bemerkungen

Obwohl dieses Datenbindungskonzept insgesamt für den Entwickler einfach ist, ist es für den Browser ziemlich schwer, da Angular jede Ereignisänderung überwacht und den Digest-Zyklus ausführt. Stellen Sie daher immer sicher, dass der Umfang so optimiert ist, wie wir das Modell an die Ansicht anhängen

Examples

Beispiel für die Datenbindung

```
<p ng-bind="message"></p>
```

Diese "Nachricht" muss an den Gültigkeitsbereich des aktuellen Elementcontrollers angehängt werden.

```
$scope.message = "Hello World";
```

Zu einem späteren Zeitpunkt, selbst wenn das Nachrichtenmodell aktualisiert wird, wird dieser aktualisierte Wert im HTML-Element angezeigt. Beim Erstellen von Winkeln wird die Vorlage "Hello World" an die innerHTML der aktuellen Welt angehängt. Angular unterhält einen Beobachtungsmechanismus für alle Anweisungen, die der Ansicht zugeordnet sind. Es verfügt über einen Digest-Cycle-Mechanismus, bei dem es das Watchers-Array durchläuft. Es aktualisiert das DOM-Element, wenn sich der vorherige Wert des Modells ändert.

Es wird keine regelmäßige Überprüfung des Bereichs durchgeführt, ob sich an den daran angebundenen Objekten Änderungen ergeben. Nicht alle Objekte, die an den Bereich angehängt sind, werden überwacht. Scope verwaltet prototypisch ein **$$ WatchersArray** . Scope durchläuft diesen WatchersArray nur dann, wenn $ Digest aufgerufen wird.

Angular fügt dem WatchersArray für jedes dieser Elemente einen Watcher hinzu

1. {{expression}} - In Ihren Vorlagen (und überall dort, wo ein Ausdruck vorhanden ist) oder wenn Sie ng-model definieren.
2. $ scope. $ watch ('Ausdruck / Funktion') - In Ihrem JavaScript können wir nur ein Oszilloskopobjekt für die Winkelung anhängen.

Die $ watch- Funktion umfasst drei Parameter:

1. Die erste ist eine Watcher-Funktion, die das Objekt zurückgibt oder wir können nur einen Ausdruck hinzufügen.

2. Die zweite ist eine Listener-Funktion, die aufgerufen wird, wenn sich das Objekt ändert. Alle Dinge wie DOM-Änderungen werden in dieser Funktion implementiert.

3. Der dritte ist ein optionaler Parameter, der einen Booleschen Wert enthält. Wenn es wahr ist, beobachtet eckig tief das Objekt und wenn es falsch ist, führt Angular lediglich eine Referenz auf das Objekt aus. Die grobe Implementierung von $ watch sieht so aus

```
Scope.prototype.$watch = function(watchFn, listenerFn) {
   var watcher = {
       watchFn: watchFn,
       listenerFn: listenerFn || function() { },
       last: initWatchVal  // initWatchVal is typically undefined
   };
   this.$$watchers.push(watcher); // pushing the Watcher Object to Watchers
};
```

In Angular gibt es eine interessante Sache namens Digest Cycle. Der $ digest-Zyklus startet als Ergebnis eines Aufrufs an $ scope. $ Digest (). Angenommen, Sie ändern ein $ scope-Modell in einer Handlerfunktion über die Direktive ng-click. In diesem Fall löst AngularJS automatisch einen $ digest-Zyklus aus, indem es $ digest () aufruft. Zusätzlich zu ng-click gibt es mehrere integrierte Direktiven / Dienste, mit denen Sie Modelle ändern können (z. B. ng-model, $ timeout usw.). und automatisch einen $ Digest-Zyklus auslösen. Die grobe Implementierung von $ Digest sieht so aus.

```
Scope.prototype.$digest = function() {
      var dirty;
      do {
          dirty = this.$$digestOnce();
      } while (dirty);
}
Scope.prototype.$$digestOnce = function() {
   var self = this;
   var newValue, oldValue, dirty;
   _.forEach(this.$$watchers, function(watcher) {
          newValue = watcher.watchFn(self);
          oldValue = watcher.last;   // It just remembers the last value for dirty checking
          if (newValue !== oldValue) { //Dirty checking of References
   // For Deep checking the object , code of Value
   // based checking of Object should be implemented here
              watcher.last = newValue;
              watcher.listenerFn(newValue,
                    (oldValue === initWatchVal ? newValue : oldValue),
                     self);
          dirty = true;
          }
     });
   return dirty;
 };
```

Wenn Sie die JavaScript-Funktion **setTimeout ()** verwenden, um ein Bereichsmodell zu aktualisieren, kann Angular nicht wissen, was Sie ändern könnten. In diesem Fall ist es unsere Verantwortung, $ apply () manuell aufzurufen, wodurch ein $ Digest-Zyklus ausgelöst wird. Wenn

Sie über eine Direktive verfügen, die einen DOM-Ereignislistener einrichtet und einige Modelle innerhalb der Handlerfunktion ändert, müssen Sie $ apply () aufrufen, um sicherzustellen, dass die Änderungen wirksam werden. Die große Idee von $ apply ist, dass wir einen Code ausführen können, der Angular nicht kennt. Dieser Code kann jedoch auch Auswirkungen auf den Geltungsbereich haben. Wenn wir diesen Code in $ apply einhüllen, wird $ digest () aufgerufen. Grobe Implementierung von $ apply ().

```
Scope.prototype.$apply = function(expr) {
       try {
         return this.$eval(expr); //Evaluating code in the context of Scope
       } finally {
         this.$digest();
       }
};
```

Wie funktioniert die Datenbindung? online lesen:
https://riptutorial.com/de/angularjs/topic/2342/wie-funktioniert-die-datenbindung-

Kapitel 48: Wiederholung

Einführung

Die Direktive `ngRepeat` instanziiert eine Vorlage einmal pro Element aus einer Auflistung. Die Sammlung muss ein Array oder ein Objekt sein. Jede Vorlageninstanz erhält ihren eigenen Gültigkeitsbereich, wobei die angegebene Schleifenvariable auf das aktuelle Collection-Element und `$index` auf den Elementindex oder -schlüssel gesetzt wird.

Syntax

- `<element ng-repeat="expression"></element>`
- `<div ng-repeat="(key, value) in myObj">...</div>`
- `<div ng-repeat="variable in expression">...</div>`

Parameter

Variable	Einzelheiten
`$index`	*Nummern*- Iterator-Offset des wiederholten Elements (0..Länge-1)
`$first`	*boolean* true, wenn sich das wiederholte Element zuerst im Iterator befindet.
`$middle`	*boolean* true, wenn sich das wiederholte Element im ersten Schritt zwischen dem ersten und dem letzten Element befindet.
`$last`	*boolean* true, wenn das wiederholte Element das letzte Element im Iterator ist.
`$even`	*boolean* true, wenn der Iteratorposition `$index` ist (andernfalls false).
`$odd`	*boolean* true, wenn der Iteratorposition `$index` ungerade ist (andernfalls false).

Bemerkungen

AngularJS stellt diese Parameter als spezielle Variablen zur Verfügung, die im Ausdruck ng-repeat und an beliebiger Stelle innerhalb des HTML-Tags verfügbar sind, an dem der ng-repeat lebt.

Examples

Iteration über Objekteigenschaften

```
<div ng-repeat="(key, value) in myObj"> ... </div>
```

Zum Beispiel

```
<div ng-repeat="n in [42, 42, 43, 43]">
  {{n}}
</div>
```

Tracking und Duplikate

`ngRepeat` verwendet $ watchCollection , um Änderungen in der Sammlung zu erkennen. Wenn eine Änderung `ngRepeat` , nimmt `ngRepeat` die entsprechenden Änderungen am DOM vor:

- Wenn ein Element hinzugefügt wird, wird eine neue Instanz der Vorlage zum DOM hinzugefügt.
- Wenn ein Element entfernt wird, wird seine Vorlageninstanz aus dem DOM entfernt.
- Wenn Artikel neu angeordnet werden, werden ihre entsprechenden Vorlagen im DOM neu angeordnet.

Duplikate

- `track by` für jede Liste, die doppelte Werte enthalten kann.
- `track by` beschleunigt auch Listenänderungen erheblich.
- Wenn Sie in diesem Fall `track by` nicht verwenden, wird der Fehler `[ngRepeat:dupes]`

```
$scope.numbers = ['1','1','2','3','4'];

<ul>
  <li ng-repeat="n in numbers track by $index">
    {{n}}
  </li>
</ul>
```

ng-repeat-start + ng-repeat-end

AngularJS 1.2 `ng-repeat` behandelt mehrere Elemente mit `ng-repeat-start` und `ng-repeat-end` :

```
// table items
$scope.tableItems = [
    {
        row1: 'Item 1: Row 1',
        row2: 'Item 1: Row 2'
    },
    {
        row1: 'Item 2: Row 1',
        row2: 'Item 2: Row 2'
    }
];

// template
<table>
    <th>
        <td>Items</td>
    </th>
    <tr ng-repeat-start="item in tableItems">
        <td ng-bind="item.row1"></td>
    </tr>
    <tr ng-repeat-end>
```

```
        <td ng-bind="item.row2"></td>
    </tr>
</table>
```

Ausgabe:

Artikel
Punkt 1: Zeile 1
Punkt 1: Zeile 2
Punkt 2: Zeile 1
Punkt 2: Zeile 2

Wiederholung online lesen: https://riptutorial.com/de/angularjs/topic/8118/wiederholung

Kapitel 49: Winkelige $ -Optionen

Bemerkungen

Angular verwendet einen **Baum** von Gültigkeitsbereichen, um die Logik (von Steuerungen, Anweisungen usw.) an die Ansicht zu binden und ist der Hauptmechanismus für die Änderungserkennung in AngularJS. Eine detailliertere Referenz für Bereiche finden Sie unter docs.angularjs.org

Die Wurzel des Baums ist über den **injektierbaren** Dienst **$ rootScope zugänglich** . Alle untergeordneten $ -Bereiche erben die Methoden und Eigenschaften ihres übergeordneten $ -Bereichs, sodass Kinder ohne Verwendung von Angular Services auf Methoden zugreifen können.

Examples

Grundlegendes Beispiel für die Vererbung von $ scope

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' };
}]);

<div ng-app="app" ng-conroller="myController">
    <input ng-model="person.name" />
    <div ng-repeat="number in [0,1,2,3]">
        {{person.name}} {{number}}
    </div>
</div>
```

In diesem Beispiel erstellt die Direktive ng-repeat einen neuen Bereich für jedes der neu erstellten untergeordneten Elemente.

Diese erstellten Bereiche sind Kinder ihres übergeordneten Bereichs (in diesem Fall der von myController erstellte Bereich) und erben daher alle ihre Proportionen, z. B. person.

Vermeiden Sie das Erben primitiver Werte

In JavaScript einen nicht zuweisen primitiven Wertes (wie Objekt, Array, Funktion und viele mehr), hält eine Referenz (eine Adresse in dem Speicher) an den zugewiesenen Wert.

Durch das Zuweisen eines primitiven Werts (String, Number, Boolean oder Symbol) zu zwei verschiedenen Variablen und Ändern einer Variablen werden beide nicht geändert:

```
var x = 5;
var y = x;
y = 6;
console.log(y === x, x, y); //false, 5, 6
```

Da bei einem nicht primitiven Wert jedoch nur Verweise auf dasselbe Objekt beibehalten **werden,** ändert das Ändern einer Variablen die andere:

```
var x = { name : 'John Doe' };
var y = x;
y.name = 'Jhon';
console.log(x.name === y.name, x.name, y.name); //true, John, John
```

Wenn ein Bereich erstellt wird, werden ihm alle Eigenschaften des übergeordneten Objekts zugewiesen. Die nachträgliche Änderung der Eigenschaften wirkt sich jedoch nur auf den übergeordneten Bereich aus, wenn es sich um einen nicht primitiven Wert handelt:

```
angular.module('app', [])
.controller('myController', ['$scope', function($scope){
    $scope.person = { name: 'John Doe' }; //non-primitive
    $scope.name = 'Jhon Doe'; //primitive
}])
.controller('myController1', ['$scope', function($scope){}]);

<div ng-app="app" ng-controller="myController">
    binding to input works: {{person.name}}<br/>
    binding to input does not work: {{name}}<br/>
    <div ng-controller="myController1">
        <input ng-model="person.name" />
        <input ng-model="name" />
    </div>
</div>
```

Denken Sie daran: In Angular-Bereichen kann auf verschiedene Arten erstellt werden (z. B. eingebaute oder benutzerdefinierte Anweisungen oder die Funktion `$scope.$new()`), und es ist möglicherweise unmöglich, den Gültigkeitsbereich zu verfolgen.

Wenn Sie nur nicht-primitive Werte als Bereichseigenschaften verwenden, bleiben Sie auf der sicheren Seite (es sei denn, Sie benötigen eine Eigenschaft, die nicht erbt, oder in anderen Fällen, in denen Sie die Vererbung des Bereichs kennen.).

Eine Funktion, die in der gesamten App verfügbar ist

Beachten Sie, dass dieser Ansatz möglicherweise als schlechtes Design für eckige Apps betrachtet wird, da Programmierer sich daran erinnern müssen, wo sich Funktionen in der Bereichsstruktur befinden, und dass die Vererbung des Bereichs bekannt ist. In vielen Fällen wäre es vorzuziehen, einen Dienst zu injizieren (Angular-Praxis - Verwendung der Vererbung des Bereichs in Abhängigkeit von der Injektion) .

Dieses Beispiel zeigt nur, wie die Vererbung des Bereichs für unsere Anforderungen verwendet werden kann und wie Sie davon profitieren können, und nicht die bewährten Methoden zum Entwerfen einer gesamten App.

In einigen Fällen könnten wir die Vererbung des Bereichs nutzen und eine Funktion als Eigenschaft des rootScope festlegen. Auf diese Weise erben alle Bereiche in der App (mit Ausnahme von isolierten Bereichen) diese Funktion und können von überall in der App aufgerufen

werden.

```
angular.module('app', [])
.run(['$rootScope', function($rootScope){
    var messages = []
    $rootScope.addMessage = function(msg){
        messages.push(msg);
    }
}]);

<div ng-app="app">
    <a ng-click="addMessage('hello world!')">it could be accsessed from here</a>
    <div ng-include="inner.html"></div>
</div>
```

inner.html:

```
<div>
    <button ng-click="addMessage('page')">and from here to!</button>
</div>
```

Erstellen von benutzerdefinierten $ scope-Ereignissen

Wie bei normalen HTML-Elementen können $ scopes eigene Ereignisse haben. $ scope-Ereignisse können auf folgende Weise abonniert werden:

```
 $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
});
```

Wenn Sie die Registrierung eines Ereignis-Listeners aufheben müssen, gibt die **$ on-** Funktion eine unverbindliche Funktion zurück. Um mit dem obigen Beispiel fortzufahren:

```
var unregisterMyEvent = $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
    unregisterMyEvent();
});
```

Es gibt zwei Möglichkeiten , Ihre eigenen $ scope Ereignis **$ Sendung** auszulösen und **$ emittieren.** Verwenden Sie **$ emit,** um die Eltern über einen Bereich eines bestimmten Ereignisses zu informieren

```
$scope.$emit('my-event', { custom: 'data' });
```

Das obige Beispiel löst alle Ereignis-Listener für `my-event` im übergeordneten Bereich aus und setzt die Gültigkeitsbereichsstruktur auf **$ rootScope hoch,** sofern nicht ein Listener `stopPropagation` für das Ereignis `stopPropagation` . Nur mit **$** `stopPropagation` ausgelöste Ereignisse können `stopPropagation`

Die Umkehrung von **$ emit** ist **$ broadcast** , wodurch alle Ereignis-Listener in allen untergeordneten Bereichen in der Bereichsstruktur ausgelöst werden, die untergeordnete

Bereiche des Bereichs sind, der **$ broadcast** aufgerufen hat.

```
$scope.$broadcast('my-event', { custom: 'data' });
```

Mit **$ broadcast** ausgelöste Ereignisse können nicht abgebrochen werden.

$ Scope-Funktionen verwenden

Während die Deklaration einer Funktion im $ rootscope ihre Vorteile hat, können wir auch eine $ scope-Funktion an jedem Teil des Codes deklarieren, der vom $ scope-Service eingefügt wird. Controller zum Beispiel.

Regler

```
myApp.controller('myController', ['$scope', function($scope){
    $scope.myFunction = function () {
        alert("You are in myFunction!");
    };
}]);
```

Jetzt können Sie Ihre Funktion vom Controller aus aufrufen:

```
$scope.myfunction();
```

Oder über HTML, das sich unter diesem speziellen Controller befindet:

```
<div ng-controller="myController">
    <button ng-click="myFunction()"> Click me! </button>
</div>
```

Richtlinie

Eine Winkelanweisung ist ein weiterer Ort, an dem Sie Ihren Gültigkeitsbereich verwenden können:

```
myApp.directive('triggerFunction', function() {
    return {
        scope: {
            triggerFunction: '&'
        },
        link: function(scope, element) {
            element.bind('mouseover', function() {
                scope.triggerFunction();
            });
        }
    };
});
```

Und in Ihrem HTML-Code unter demselben Controller:

```
<div ng-controller="myController">
```

```
    <button trigger-function="myFunction()"> Hover over me! </button>
</div>
```

Natürlich können Sie ngMouseover für dasselbe verwenden, aber das Besondere an Anweisungen ist, dass Sie sie nach Ihren Wünschen anpassen können. Und jetzt wissen Sie, wie Sie Ihre $ scope-Funktionen in ihnen einsetzen können, seien Sie kreativ!

Wie können Sie den Geltungsbereich einer Richtlinie einschränken und warum sollten Sie dies tun?

Der Bereich wird als "Klebstoff" verwendet, mit dem wir zwischen dem übergeordneten Controller, der Direktive und der Direktive-Vorlage kommunizieren. Bei jedem Bootstrapping der AngularJS-Anwendung wird ein RootScope-Objekt erstellt. Jeder durch Controller, Direktiven und Services erstellte Bereich wird prototypisch von rootScope übernommen.

Ja, wir können den Anwendungsbereich einer Richtlinie einschränken. Wir können dies tun, indem wir einen isolierten Geltungsbereich für die Richtlinie schaffen.

Es gibt drei Arten von Richtlinienbereichen:

1. Geltungsbereich: Falsch (Richtlinie verwendet den übergeordneten Geltungsbereich)
2. Geltungsbereich: Richtig (Richtlinie erhält einen neuen Geltungsbereich)
3. Geltungsbereich: {} (Richtlinie erhält einen neuen isolierten Geltungsbereich)

Direktiven mit dem neuen isolierten Bereich: Wenn wir einen neuen isolierten Bereich erstellen, wird er nicht vom übergeordneten Bereich geerbt. Dieser neue Bereich wird als isolierter Bereich bezeichnet, da er vollständig vom übergeordneten Bereich getrennt ist. Warum? Verwenden Sie den isolierten Bereich: Wir sollten den isolierten Bereich verwenden, wenn Sie eine benutzerdefinierte Direktive erstellen möchten, da dadurch sichergestellt wird, dass unsere Direktive generisch ist und sich innerhalb der Anwendung befindet. Der übergeordnete Bereich wird den Richtlinienbereich nicht beeinträchtigen.

Beispiel für einen isolierten Umfang:

```
var app = angular.module("test",[]);

app.controller("Ctrl1",function($scope){
    $scope.name = "Prateek";
    $scope.reverseName = function(){
        $scope.name = $scope.name.split('').reverse().join('');
    };
});
app.directive("myDirective", function(){
    return {
        restrict: "EA",
        scope: {},
        template: "<div>Your name is : {{name}}</div>"+
        "Change your name : <input type='text' ng-model='name'/>"
    };
});
```

Es gibt drei Arten von Präfixen, die AngularJS für den isolierten Umfang bereitstellt.

1. "@" (Textbindung / Einwegbindung)
2. "=" (Direkte Modellbindung / bidirektionale Bindung)
3. "&" (Verhaltensbindung / Methodenbindung)

Alle diese Präfixe erhalten Daten von den Attributen des Direktive-Elements wie:

```
<div my-directive
  class="directive"
  name="{{name}}"
  reverse="reverseName()"
  color="color" >
</div>
```

Winkelige $ -Optionen online lesen: https://riptutorial.com/de/angularjs/topic/3157/winkelige----optionen

Kapitel 50: Winkeljs mit Datenfilter, Seitenumbruch etc

Einführung

Provider-Beispiel und Abfrage von Anzeigedaten mit Filter, Seitenumbruch usw. in Angularjs.

Examples

Angularjs zeigt Daten mit Filter und Seitenumbruch an

```
<div ng-app="MainApp" ng-controller="SampleController">
    <input ng-model="dishName" id="search" class="form-control" placeholder="Filter text">
    <ul>
        <li dir-paginate="dish in dishes |  filter : dishName | itemsPerPage: pageSize"
pagination-id="flights">{{dish}}</li>
    </ul>
    <dir-pagination-controls boundary-links="true" on-page-
change="changeHandler(newPageNumber)" pagination-id="flights"></dir-pagination-controls>
</div>
<script type="text/javascript" src="angular.min.js"></script>
<script type="text/javascript" src="pagination.js"></script>
<script type="text/javascript">

var MainApp = angular.module('MainApp', ['angularUtils.directives.dirPagination'])
MainApp.controller('SampleController', ['$scope', '$filter', function ($scope, $filter) {

    $scope.pageSize = 5;

    $scope.dishes = [
    'noodles',
    'sausage',
    'beans on toast',
    'cheeseburger',
    'battered mars bar',
    'crisp butty',
    'yorkshire pudding',
    'wiener schnitzel',
    'sauerkraut mit ei',
    'salad',
    'onion soup',
    'bak choi',
    'avacado maki'
    ];

    $scope.changeHandler = function (newPage) { };
}]);
</script>
```

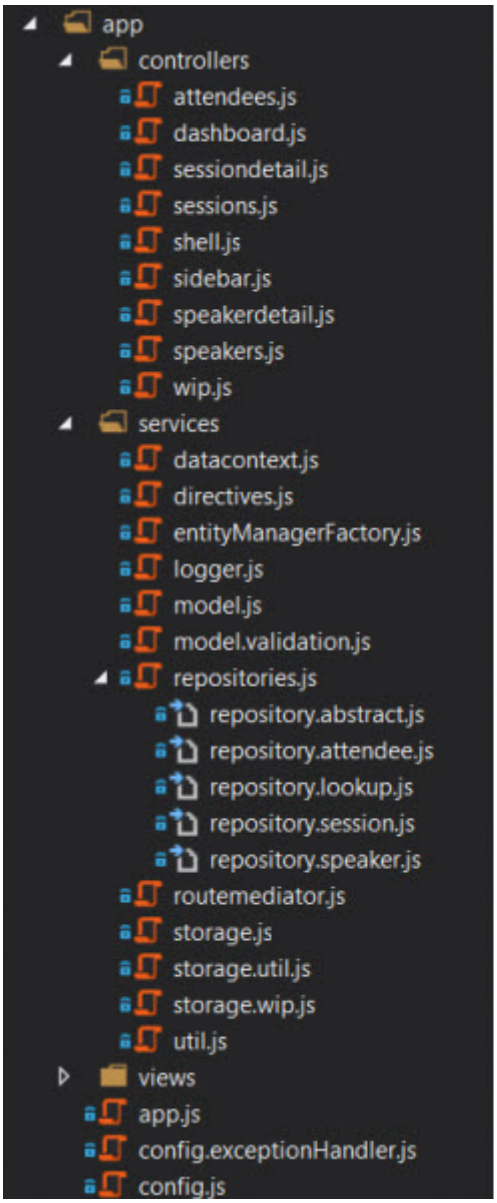
Winkeljs mit Datenfilter, Seitenumbruch etc online lesen:
https://riptutorial.com/de/angularjs/topic/10821/winkeljs-mit-datenfilter--seitenumbruch-etc

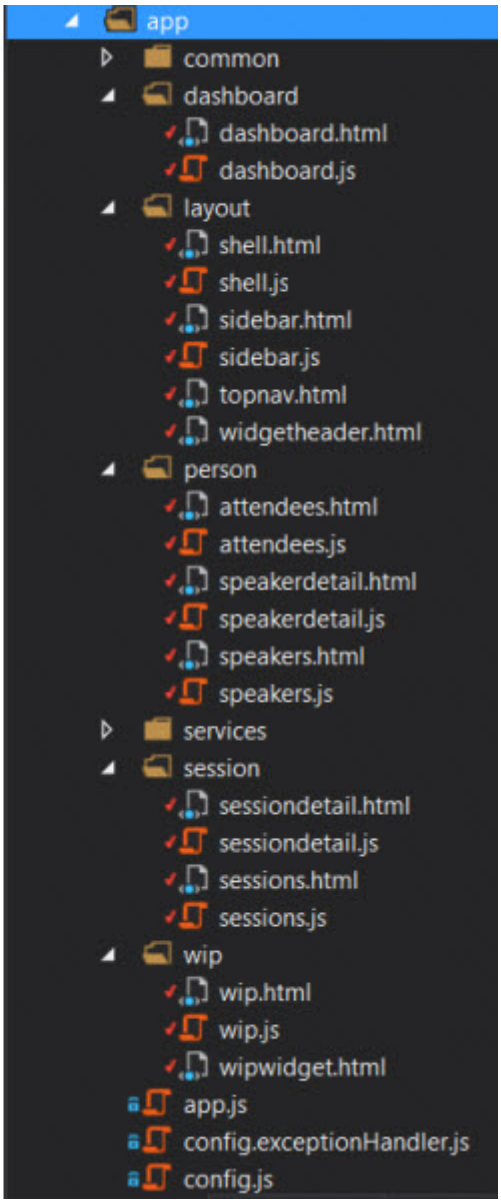
Kapitel 51: Winkelprojekt - Verzeichnisstruktur

Examples

Verzeichnisaufbau

Eine häufig gestellte Frage unter neuen Angular-Programmierern: "Wie sollte die Struktur des Projekts aussehen?". Eine gute Struktur hilft bei der Entwicklung einer skalierbaren Anwendung. Wenn wir ein Projekt starten, haben wir zwei Möglichkeiten: **Sort By Type** (links) und **Sort By Feature** (rechts). Der zweite ist besser, vor allem in großen Anwendungen wird das Projekt viel einfacher zu verwalten.





Nach Typ sortieren (links)

Die Anwendung ist nach Dateityp organisiert.

- **Vorteil** - Geeignet für kleine Apps, nur für Programmierer, die mit Angular beginnen und leicht auf die zweite Methode umwandeln können.
- **Nachteil** - Selbst für kleine Apps wird es schwieriger, eine bestimmte Datei zu finden. Eine Ansicht und ihr Controller befinden sich beispielsweise in zwei separaten Ordnern.

Sortiere nach Funktion (rechts)

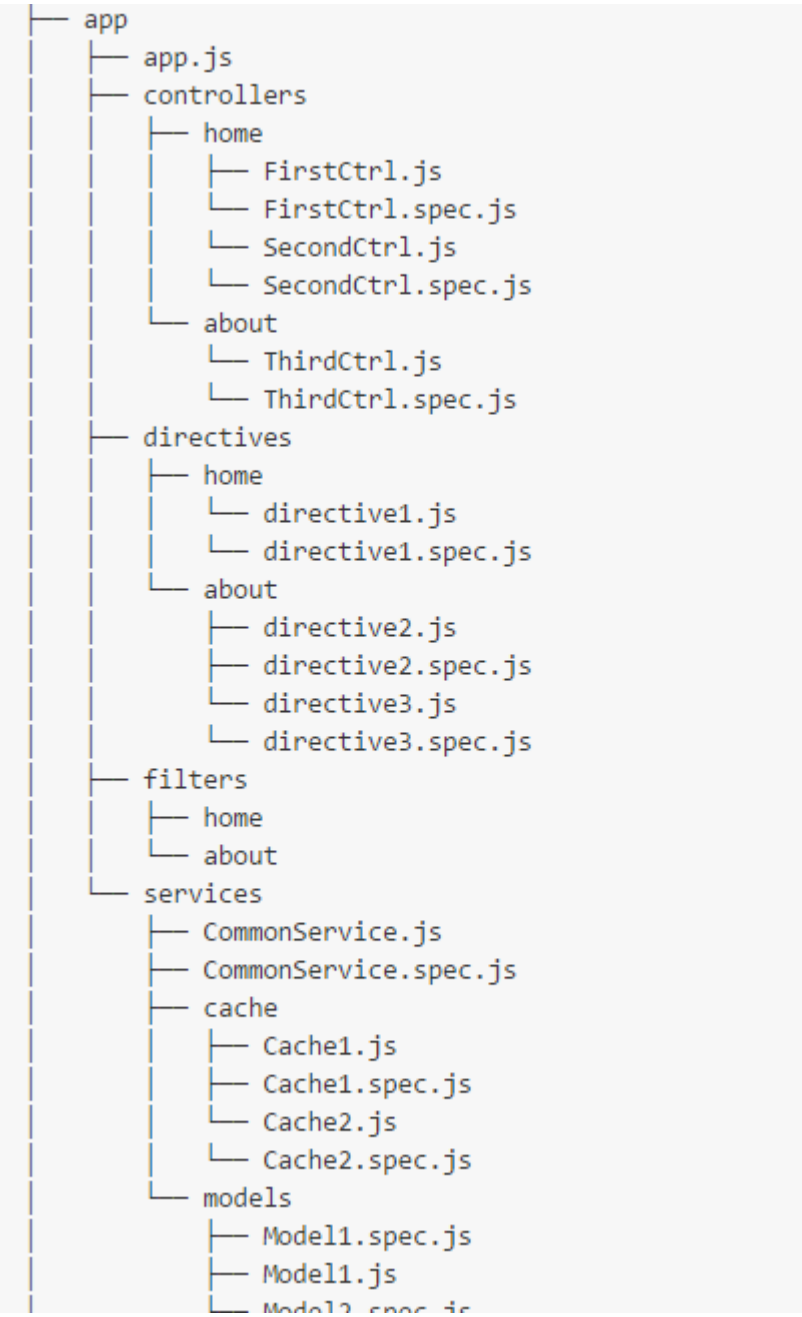
Die empfohlene Organisationsmethode, bei der die Ablage nach Merkmalstyp sortiert wird.

Alle Layoutansichten und Controller werden im Layoutordner gespeichert, der Admin-Inhalt wird im Admin-Ordner gespeichert und so weiter.

- **Vorteil** - Bei der Suche nach einem Codeabschnitt, der eine bestimmte Funktion bestimmt, befindet sich dieser in einem Ordner.
- **Nachteil** - Dienste sind etwas anders, da sie viele Funktionen „bedienen“.

Weitere Informationen dazu finden Sie unter Winkelstruktur: Refactoring für Wachstum

Die vorgeschlagene Dateistruktur, die die beiden zuvor genannten Methoden kombiniert:



Kredit zu: Angular Style Guide

Winkelprojekt - Verzeichnisstruktur online lesen:
https://riptutorial.com/de/angularjs/topic/6148/winkelprojekt---verzeichnisstruktur

Kapitel 52: Winkelversprechen mit $ q-Service

Examples

Verwenden Sie $ q.all, um mehrere Versprechen zu bearbeiten

Sie können die `$q.all` Funktion verwenden, um eine `.then` Methode `.then` , nachdem ein Array von Zusagen erfolgreich aufgelöst wurde, und die Daten `.then` , mit denen sie aufgelöst wurden.

Beispiel:

JS:

```
$scope.data = []

$q.all([
   $http.get("data.json"),
   $http.get("more-data.json"),
   ]).then(function(responses) {
     $scope.data = responses.map((resp) => resp.data);
   });
```

Der obige Code läuft `$http.get` 2mal für Daten in lokalen json - Dateien, wenn beide `get` Methode vollständig sie ihre zugehörigen Versprechungen lösen, wenn alle Versprechungen in der Anordnung gelöst werden, die `.then` beginnt Methode mit beiden Versprechen Daten innerhalb der `responses` Array-Argument.

Die Daten werden dann zugeordnet, sodass sie auf der Vorlage angezeigt werden können, und wir können sie anzeigen

HTML:

```
<ul>
   <li ng-repeat="d in data">
      <ul>
         <li ng-repeat="item in d">{{item.name}}: {{item.occupation}}</li>
      </ul>
   </li>
</ul>
```

JSON:

```
[{
  "name": "alice",
  "occupation": "manager"
}, {
  "name": "bob",
  "occupation": "developer"
```

```
}]
```

Verwenden des $ q-Konstruktors zum Erstellen von Versprechen

Die `$q` Konstruktorfunktion wird verwendet, um Zusagen aus asynchronen APIs zu erstellen, die Rückrufe verwenden, um Ergebnisse zurückzugeben.

    $ q (Funktion (auflösen, ablehnen) {...})

Die Konstruktorfunktion empfängt eine Funktion, die mit zwei Argumenten aufgerufen wird: `resolve` und `reject` Dies sind Funktionen, mit denen das Versprechen entweder aufgelöst oder abgelehnt wird.

Beispiel 1:

```
function $timeout(fn, delay) {
    return = $q(function(resolve, reject) {
         setTimeout(function() {
              try {
                 let r = fn();
                 resolve(r);
              }
              catch (e) {
                 reject(e);
              }
         }, delay);
     };
}
```

Im obigen Beispiel wird ein Versprechen aus der WindowTimers.setTimeout-API erstellt . Das AngularJS-Framework bietet eine ausführlichere Version dieser Funktion. Informationen zur Verwendung finden Sie in der AngularJS $ Timeout Service API-Referenz .

Beispiel 2

```
$scope.divide = function(a, b) {
    return $q(function(resolve, reject) {
      if (b===0) {
        return reject("Cannot devide by 0")
      } else {
        return resolve(a/b);
      }
    });
}
```

Der obige Code, der eine versprochene Divisionsfunktion zeigt, gibt ein Versprechen mit dem Ergebnis zurück oder weist einen Grund zurück, wenn die Berechnung nicht möglich ist.

Sie können dann anrufen und `.then`

```
$scope.divide(7, 2).then(function(result) {
    // will return 3.5
```

```
}, function(err) {
    // will not run
})

$scope.divide(2, 0).then(function(result) {
    // will not run as the calculation will fail on a divide by 0
}, function(err) {
    // will return the error string.
})
```

Verschieben von Operationen mit $ q.defer

Wir können `$q` , um Operationen in die Zukunft zu verschieben, während wir derzeit ein ausstehendes Versprechungsobjekt haben. Mit `$q.defer` erstellen wir ein Versprechen, das entweder gelöst oder in der Zukunft abgelehnt wird.

Diese Methode ist nicht gleichbedeutend mit der Verwendung des `$q` Konstruktors, da mit `$q.defer` eine vorhandene Routine versprochen wird, die möglicherweise ein Versprechen zurückgibt (oder jemals zurückgegeben hat).

Beispiel:

```
var runAnimation = function(animation, duration) {
    var deferred = $q.defer();
    try {
        ...
        // run some animation for a given duration
        deferred.resolve("done");
    } catch (err) {
        // in case of error we would want to run the error hander of .then
        deferred.reject(err);
    }
    return deferred.promise;
}

// and then
runAnimation.then(function(status) {}, function(error) {})
```

1. `.then` Sie sicher, dass Sie immer das Objekt `deferred.promise` , oder riskieren Sie einen Fehler, wenn Sie `.then`

2. `.then` Sie sicher, dass Sie Ihr zurückgestelltes Objekt immer auflösen oder ablehnen, `.then` möglicherweise nicht ausgeführt, und Sie riskieren einen Speicherverlust

Verwenden von winkligen Versprechen mit dem $ q-Service

`$q` ist ein integrierter Dienst, der bei der Ausführung asynchroner Funktionen und der Verwendung ihrer Rückgabewerte (oder Ausnahme) hilft, wenn die Verarbeitung abgeschlossen ist.

`$q` ist in den `$rootScope.Scope` Modellbeobachtungsmechanismus integriert, was eine schnellere Ausbreitung der Auflösung oder Ablehnung in Ihren Modellen bedeutet und unnötige Browser-Neulackierungen vermeidet, die zu einer flimmernden Benutzeroberfläche führen.

In unserem Beispiel rufen wir unsere Factory `getMyData` , die ein Versprechenobjekt `getMyData` . Wenn das Objekt `resolved` , wird eine Zufallszahl zurückgegeben. Wenn es `rejected` , wird nach 2 Sekunden eine Ablehnung mit einer Fehlermeldung zurückgegeben.

In der Winkelfabrik

```
function getMyData($timeout, $q) {
  return function() {
    // simulated async function
    var promise = $timeout(function() {
      if(Math.round(Math.random())) {
        return 'data received!'
      } else {
        return $q.reject('oh no an error! try again')
      }
    }, 2000);
    return promise;
  }
}
```

Versprechen auf Abruf verwenden

```
angular.module('app', [])
.factory('getMyData', getMyData)
.run(function(getData) {
  var promise = getData()
    .then(function(string) {
      console.log(string)
    }, function(error) {
      console.error(error)
    })
    .finally(function() {
      console.log('Finished at:', new Date())
    })
})
```

Um Versprechungen zu verwenden, fügen Sie `$q` als Abhängigkeit hinzu. Hier haben wir `$q` in die `getMyData` Fabrik eingespritzt.

```
var defer = $q.defer();
```

Eine neue verzögerte Instanz wird erstellt, indem `$q.defer()`

Ein aufgeschobenes Objekt ist einfach ein Objekt, das ein Versprechen sowie die zugehörigen Methoden zur Lösung dieses Versprechens aufdeckt. Es ist so konstruiert , mit der `$q.deferred()` Funktion und macht drei Hauptmethoden: `resolve()` , `reject()` und `notify()` .

- `resolve(value)` - löst das abgeleitete Versprechen mit dem Wert auf.
- `reject(reason)` - lehnt das abgeleitete Versprechen mit dem Grund ab.
- `notify(value)` - informiert über den Status der Ausführung des Versprechens. Dies kann mehrmals aufgerufen werden, bevor das Versprechen entweder gelöst oder abgelehnt wird.

Eigenschaften

Auf das zugehörige Versprechungsobjekt wird über die Versprechenseigenschaft zugegriffen. `promise` - {Versprechen} - Versprechungsobjekt, das diesem zurückgestellten zugeordnet ist.

Eine neue Versprechungsinstanz wird erstellt, wenn eine verzögerte Instanz erstellt wird, die durch Aufrufen von `deferred.promise` abgerufen werden kann.

Der Zweck des `promise` besteht darin, interessierten Parteien nach Abschluss des Vorgangs den Zugriff auf das Ergebnis der zurückgestellten Aufgabe zu ermöglichen.

Versprechungsmethoden -

- `then(successCallback, [errorCallback], [notifyCallback])` - Unabhängig davon, wann das Versprechen aufgelöst wurde oder abgelehnt wird, ruft einer der Erfolgs- oder Fehlerrückrufe asynchron auf, sobald das Ergebnis verfügbar ist. Die Callbacks werden mit einem einzigen Argument aufgerufen: dem Ergebnis oder dem Ablehnungsgrund. Zusätzlich kann der Benachrichtigungsrückruf null oder mehrmals aufgerufen werden, um eine Fortschrittsanzeige bereitzustellen, bevor das Versprechen gelöst oder abgelehnt wird.

- `catch(errorCallback)` - Abkürzung für promise.then (null, errorCallback)

- `finally(callback, notifyCallback)` - Ermöglicht Ihnen, die Erfüllung oder Ablehnung eines Versprechens zu beobachten, dies jedoch ohne Änderung des Endwerts.

Eine der wichtigsten Eigenschaften von Versprechen ist die Fähigkeit, sie miteinander zu verketten. Dadurch können die Daten durch die Kette fließen und in jedem Schritt manipuliert und mutiert werden. Dies wird am folgenden Beispiel demonstriert:

Beispiel 1:

```
// Creates a promise that when resolved, returns 4.
function getNumbers() {

  var promise = $timeout(function() {
    return 4;
  }, 1000);

  return promise;
}

// Resolve getNumbers() and chain subsequent then() calls to decrement
// initial number from 4 to 0 and then output a string.
getNumbers()
  .then(function(num) {
      // 4
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 3
      console.log(num);
      return --num;
```

```
  })
   .then(function (num) {
      // 2
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 1
      console.log(num);
      return --num;
  })
  .then(function (num) {
      // 0
      console.log(num);
      return 'And we are done!';
  })
  .then(function (text) {
      // "And we are done!"
      console.log(text);
  });
```

Einfaches Versprechen mit $ q.when () in ein Versprechen

Wenn Sie den Wert nur in ein Versprechen einpacken müssen, müssen Sie nicht die lange Syntax wie hier verwenden:

```
//OVERLY VERBOSE
var defer;
defer = $q.defer();
defer.resolve(['one', 'two']);
return defer.promise;
```

In diesem Fall können Sie einfach schreiben:

```
//BETTER
return $q.when(['one', 'two']);
```

$ q.when und sein Alias $ q.resolve

> Schließt ein Objekt, das ein Wert oder ein (Drittanbieter) dann mögliches Versprechen ist, in ein $ q-Versprechen ein. Dies ist nützlich, wenn Sie es mit einem Objekt zu tun haben, das möglicherweise ein Versprechen ist oder nicht, oder wenn das Versprechen aus einer Quelle stammt, der nicht vertraut werden kann.
>
> - AngularJS $ q Service-API-Referenz - $ q.when

Mit der Veröffentlichung von AngularJS v1.4.1

Sie können auch eine ES6-konsistente Alias- `resolve`

```
//ABSOLUTELY THE SAME AS when
```

```
return $q.resolve(['one', 'two'])
```

Vermeiden Sie das $ q Deferred Anti-Pattern

Vermeiden Sie dieses Anti-Pattern

```
var myDeferred = $q.defer();

$http(config).then(function(res) {
   myDeferred.resolve(res);
}, function(error) {
   myDeferred.reject(error);
});

return myDeferred.promise;
```

Mit `$q.defer` kein Versprechen `$q.defer` da der $ http-Service bereits ein Versprechen zurückgibt.

```
//INSTEAD
return $http(config);
```

Geben Sie einfach das vom $ http-Service erstellte Versprechen zurück.

Winkelversprechen mit $ q-Service online lesen:
https://riptutorial.com/de/angularjs/topic/4379/winkelversprechen-mit---q-service

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit AngularJS	Abhishek Pandey, After Class, Andrés Encarnación, AnonymousNotReally, badzilla, Charlie H, Chirag Bhatia - chirag64, Community, daniellmb, David G., Devid Farinelli, Eugene, fracz, Franck Dernoncourt, Gabriel Pires, Gourav Garg, H. Pauwelyn, Igor Raush, jengeb, Jeroen, John F., Léo Martin, Lotus91, LucyMarieJ, M. Junaid Salaat, Maaz.Musa, Matt, Mikko Viitala, Mistalis, Nemanja Trifunovic, Nhan, Nico, pathe.kiran, Patrick, Pushpendra, Richard Hamilton, Stepan Suvorov, Stephen Leppik, Sunil Lama, superluminary, Syed Priom, timbo, Ven, vincentvanjoe, Yasin Patel, Ze Rubeus, Артем Комаров
2	$ http-Anfrage	CENT1PEDE, jaredsk, Liron Ilayev
3	Abhängigkeitsspritze	Andrea, badzilla, Gavishiddappa Gadagi, George Kagan, MoLow, Omri Aharon
4	Anbieter	Mikko Viitala
5	Angular MVC	Ashok choudhary, Gavishiddappa Gadagi, Jim
6	AngularJS Fallstiche und Fallen	Alon Eitan, Cosmin Ababei, doctorsherlock, Faruk Yazıcı, ngLover, Phil
7	AngularJS-Bindungsoptionen (`=`, @, `&` usw.)	Alon Eitan, Lucas L, Makarov Sergey, Nico, zucker
8	Anweisungen, die ngModelController verwenden	Nikos Paraskevopoulos
9	Benutzerdefinierte Filter	doodhwala, Pat, Sylvain
10	Benutzerdefinierte Filter mit ES6	Bouraoui KACEM
11	Benutzerdefinierte Richtlinien	Alon Eitan, br3w5, casraf, Cody Stott, Daniel, Everettss, Filipe Amaral, Gaara, Gavishiddappa Gadagi, Jinw, jkris, mnoronha, Pushpendra, Rahul Bhooteshwar, Sajal, sgarcia.dev, Stephan, theblindprophet, TheChetan, Yuri Blanc

12	Bereiten Sie sich auf die Produktion vor - Grunt	JanisP
13	Controller	Adam Harrison, Aeolingamenfel, Alon Eitan, badzilla, Bon Macalindong, Braiam, chatuur, DerekMT12, Dr. Cool, Florian, George Kagan, Grundy, Jared Hooper, Liron Ilayev, M. Junaid Salaat, Mark Cidade, Matthew Green, Mike, Nad Flores, Praveen Poonia, RamenChef, Sébastien Deprez, sgarcia.dev, thegreenpizza, timbo, Und3rTow, WMios
14	Controller mit ES6	Bouraoui KACEM
15	Das Selbst oder diese Variable in einem Controller	It-Z, Jim
16	Daten teilen	elliot-j, Grundy, Lex, Mikko Viitala, Mistalis, Nix, prit4fun, Rohit Jindal, sgarcia.dev, Sunil Lama
17	Debuggen	Aman, AWolf, Vinay K
18	Dekorateure	Mikko Viitala
19	Dienstleistungen	Abdellah Alaoui, Alvaro Vazquez, AnonDCX, DotBot, elliot-j, Flash, Gavishiddappa Gadagi, Hubert Grzeskowiak, Lex, Nishant123
20	Digest Loop Komplettlösung	Alon Eitan, chris, MoLow, prit4fun
21	Drucken	ziaulain
22	Eingebaute Hilfsfunktionen	MoLow, Pranav C Balan, svarog
23	Eingebaute Richtlinien	Adam Harrison, Alon Eitan, Aron, AWolf, Ayan, Bon Macalindong, CENT1PEDE, Devid Farinelli, DillonChanis, Divya Jain, Dr. Cool, Eric Siebeneich, George Kagan, Grinn, gustavohenke, IncrediApp, kelvinelove, Krupesh Kotecha, Liron Ilayev, m.e.conroy, Maciej Gurban, Mansouri, Mikko Viitala, Mistalis, Mitul, MoLow, Naga2Raja, ngLover, Nishant123, Piet, redunderthebed, Richard Hamilton, svarog, tanmay, theblindprophet, timbo, Tomislav Stankovic, vincentvanjoe, Vishal Singh
24	Faules Laden	Muli Yulzary
25	Filter	Aeolingamenfel, developer033, Ed Hinchliffe, fracz, gustavohenke, Matthew Green, Nico

26	Formularüberprüfung	Alon Eitan, fantarama, garyx, Mikko Viitala, Richard Hamilton, Rohit Jindal, shane, svarog, timbo
27	Grunzen Sie Aufgaben	Mikko Viitala
28	HTTP-Interceptor	G Akshay, Istvan Reiter, MeanMan, Mistalis, mnoronha
29	Komponenten	Alon Eitan, Artem K., badzilla, BarakD, Hubert Grzeskowiak, John F., Juri, M. Junaid Salaat, Mansouri, Pankaj Parkar, Ravi Singh, sgarcia.dev, Syed Priom, Yogesh Mangaj
30	Konstanten	Sylvain
31	Leistungsprofilierung	Deepak Bansal
32	Migration nach Angular 2+	ShinDarth
33	Module	Alon Eitan, Ankit, badzilla, Bon Macalindong, Matthew Green, Nad Flores, ojus kulkarni, sgarcia.dev, thegreenpizza
34	ng-class Direktive	Dr. Cool
35	ng-style	Divya Jain, Jim
36	ng-view	Aayushi Jain, Manikandan Velayutham, Umesh Shende
37	Profiling und Leistung	Ajeet Lakhani, Alon Eitan, Andrew Piliser, Anfelipe, Anirudha, Ashwin Ramaswami, atul mishra, Braiam, bwoebi, chris, Dania , Daniel Molin, daniellmb, Deepak Bansal, Divya Jain, DotBot, Dr. Cool, Durgpal Singh, fracz, Gabriel Pires, George Kagan, Grundy, JanisP, Jared Hooper, jhampton, John Slegers, jusopi , M22an, Matthew Green, Mistalis, Mudassir Ali, Nhan, Psaniko, Richard Hamilton, RyanDawkins, sgarcia.dev, theblindprophet, user3632710, vincentvanjoe, Yasin Patel, Ze Rubeus
38	Routing mit ngRoute	Alon Eitan, Alvaro Vazquez, camabeh, DotBot, sgarcia.dev, svarog
39	SignalR mit AngularJs	Maher
40	Sitzungsspeicher	Rohit Jindal
41	ui-router	George Kagan, H.T, Michael P. Bazos, Ryan Hamley, sgarcia.dev
42	Unit-Tests	daniellmb, elliot-j, fracz, Gabriel Pires, Nico, ronapelbaum
43	Unterscheidungsdienst	Deepak Bansal

	vs Fabrik	
44	Veranstaltungen	CodeWarrior, Nguyen Tran, Rohit Jindal, RyanDawkins, sgarcia.dev, shaN, Shashank Vivek
45	Verwenden von AngularJS mit TypeScript	Parv Sharma, Rohit Jindal
46	Verwendung von eingebauten Anweisungen	Gourav Garg
47	Wie funktioniert die Datenbindung?	Lucas L, Sasank Sunkavalli, theblindprophet
48	Wiederholung	Divya Jain, Jim, Sender, zucker
49	Winkelige $ -Optionen	Abhishek Maurya, elliot-j, Eugene, jaredsk, Liron Ilayev, MoLow, Prateek Gupta, RamenChef, ryansstack, Tony
50	Winkeljs mit Datenfilter, Seitenumbruch etc	Paresh Maghodiya
51	Winkelprojekt - Verzeichnisstruktur	jitender, Liron Ilayev
52	Winkelversprechen mit $ q-Service	Alon Eitan, caiocpricci2, ganqqwerty, georgeawg, John F., Muli Yulzary, Praveen Poonia, Richard Hamilton, Rohit Jindal, svarog