Бесплатная электронная книга

УЧУСЬ AngularJS

Free unaffiliated eBook created from
Stack Overflow contributors.

#angularjs

1
1: AngularJS 2
2
2
Examples 9
9
11
13
Hello 14
- 14
15
15
AngularJS 16
2: $ http request 20
Examples 20
$ http 20
$ http 21
$ http 22
3: AngularJS gotchas 24
Examples 24
24
24
html5Mode 25
7 AngularJS 27
4: angularjs , . . 32
32
Examples 32
Angularjs , 32
5: Grunt tasks 34
Examples 34
34

6: HTTP- 38
38
Examples 38
38
HTTPInterceptor 38
Flash- , http- 39
39
40
40
7: SignalR AngularJs 42
42
Examples 42
SignalR And AngularJs [ChatProject] 42
8: UI- 46
46
Examples 46
46
47
49
/ 50
9: 52
52
52
Examples 52
52
53
$ inject 53
AngularJS JavaScript 53
10: 55
Examples 55
angular.equals 55
angular.isString 55

angular.isArray 56
angular.merge 56
angular.isDefined angular.isUndefined 56
angular.isDate 57
angular.isNumber 57
angular.isFunction 58
angular.toJson 58
angular.fromJson 59
angular.noop 59
angular.isObject 60
angular.isElement 60
angular.copy 61
angular.identity 61
angular.forEach 62
11: 63
Examples 63
- 63
ngRepeat 63
ngShow ngHide 67
ngOptions 68
ngModel 71
ngClass 71
ngIf 72
JavaScript 72
..... 72
DOM currentUser 73
DOM currentUser 73
..... 73
ngMouseenter ngMouseleave 73
ngDisabled 74
ngDblclick 74
Cheat Sheet 75

ngClick 77
ngRequired 78
-- 78
ngCloak 79
ngInclude 79
ngSrc 80
ngPattern 80
ngValue 81
ngCopy 81
81
ngPaste 81
ngHref 82
ngList 82
12: 84
84
Examples 84
84
$ digest $ watch 84
$ scope 85
13: 88
88
88
Examples 88
, 88
89
90
14: ng-class 92
Examples 92
ng- 92
1. 92
2. 92

3. 93
15: , ngModelController 94
Examples 94
: 94
: 96
16: 100
..... 100
Examples 100
..... 100
(1,5+) 101
..... 102
..... 102
..... 103
17: AngularJS TypeScript 105
..... 105
Examples 105
..... 105
ControllerAs 106
Bundling / Minification 107
ControllerAs ? 108
..... 108
? 108
$ scope? 109
18: 110
Examples 110
/ HTML 110
19: 112
..... 112
Examples 112
..... 112
20: 115
.....

115

116

Examples 116

116

? 116

: 117

117

«require» 118

JS 118

21: 120

120

Examples 120

120

120

22: 123

123

Examples 123

123

125

, 125

126

JS 126

127

128

23: ES6 130

Examples 130

130

24: 131

131

Examples 131

131

131

132

UI-: 132

ngRoute: 132

132

133

25: ngRoute 134

134

Examples 134

134

135

137

26: 139

Examples 139

139

139

27: - 141

141

141

141

141

Examples 141

142

142

ng-repeat-start + ng-repeat-end 142

28: - 144

144

Examples 144

- 144

144

29: - 146

146

146

Examples....146
ng-....146
30: 147
Examples....147
....147
chg ng-inspect....148
....151
31: 153
Examples....153
Factory VS Service 153
32: AngularJS (`=`, `@`, `&` . .)....155
....155
Examples....155
@ , ...155
= 155
& , ...156
....156
....157
33: 2+....158
....158
Examples....158
AngularJS - 158
....158
?....160
?....160
....161
Webpack ES6....161
34: - Grunt....162
Examples....162
....162
....163

35: 166
..... 166
..... 166
Examples 168
..... 168
..... 170
..... 171
..... 172
..... 174
..... 175
..... 176
..... 177
36: 179
Examples 179
..... 179
example.js 179
example.html 179
..... 179
, ... 179
..... 180
37: ES6 181
Examples 181
FileSize ES6 181
38: 183
..... 183
..... 183
Examples 183
..... 183
..... 184
..... 184
..... 185
..... 185

39: 187
Examples 187
..... 187
..... 188
CSS 188
ngMessages 189
..... 189
..... 189
..... 190
..... 191
..... 191
40: 193
Examples 193
7 193
1) ng-repeat 193
2) 193
3) 194
4) 195
5) ng-if / ng-show 196
6) 196
7) , ... 196
Bind Once 197
..... 198
..... 199
, ? 199
ng-if vs ng-show 201
- 201
- 201
..... 201
..... 201
..... 202

, , 202
41: 204
Examples 204
204
42: 207
207
Examples 207
207
43: 209
209
Examples 209
209
44: 211
Examples 211
211
211
212
$ sce - 212
« » 213
213
214
45: 218
218
Examples 218
218
$. $ 218
$. $ 218
: 219
AngularJS 219
220
$ rootScope. $ $ destory event 222

46: 223
..... 223
Examples 223
ngStorage 223
..... 224
47: MVC 226
..... 226
Examples 226
..... 226
mvc 226
..... 226
..... 226
48: - 227
Examples 227
..... 227
() 228
() 228
49: $ 230
..... 230
Examples 230
$ scope 230
..... 230
, ... 231
$ scope 232
$ scope 233
? 234
50: $ q 236
Examples 236
$ q.all 236
$ q 237
$ q.defer 238
$ q 238

239
240
, $ q.when () 241
$ q.when $ q.resolve 241
$ q - 242
Anti-Pattern 242
51: 243
Examples 243
243
Javascript 244
HTML 244
244
245
245
246
ng-repeat 247
52: 248
Examples 248
angularjs 248
: 248
: 248
250

Около

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

глава 1: Начало работы с AngularJS

замечания

AngularJS - это платформа веб-приложений, предназначенная для упрощения разработки приложений на стороне клиента. Эта документация предназначена для углового 1.x , предшественника более современного углового 2 или см. Документацию по переполнению стека для углового 2 .

Версии

Версия	Дата выхода
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-PK - 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

Версия	Дата выхода
1.5.7	2016-06-15
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-PK - 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

Версия	Дата выхода
1.3.18	2015-08-19
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-PK - 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	2015-02-09
1.3.13	2015-02-09
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

Версия	Дата выхода
1.3.8	2014-12-19
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-PK - 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

Версия	Дата выхода
1.3.0-beta.17	2014-07-25
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

Версия	Дата выхода
1.2.12	2014-02-08
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-PK - 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

Версия	Дата выхода
1.1.1	2012-11-27
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

Версия	Дата выхода
0.10.1	2011-09-09
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

Начиная

Создайте новый HTML-файл и вставьте следующий контент:

```
<!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>
```

Демо-версия

Когда вы открываете файл в браузере, вы увидите поле ввода, за которым следует текст `Hello, World!` , Редактирование значения на входе будет обновлять текст в режиме реального времени, без необходимости обновлять всю страницу.

Объяснение:

1. Загрузите угловую структуру из сети доставки контента.

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

2. Определить HTML-документ в виде углового приложения с помощью директивы `ng-app`

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

3. Инициализировать переменную `name` с помощью `ng-init`

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

   Обратите внимание, что ng-init следует использовать только для демонстрационных и тестовых целей. При создании реального приложения контроллеры должны инициализировать данные.

4. Свяжите данные с модели с представлением на элементах управления HTML. Привязать `<input>` к свойству `name` с `ng-model`

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

5. Отображать содержимое из модели с помощью двойных фигурных скобок `{{ }}`

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

6. Другим способом привязки свойства `name` является использование `ng-bind` вместо

handlebars `"{{ }}"`

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

Последние три шага устанавливают *двустороннюю привязку данных* . Изменения, внесенные на вход, обновляют *модель* , которая отражается в *представлении* .

Существует разница между использованием рулей и `ng-bind` . Если вы используете рули, вы можете увидеть фактический `Hello, {{name}}` поскольку страница загружается до того, как выражение будет разрешено (перед загрузкой данных), тогда как если вы используете `ng-bind` , он будет показывать только данные, когда имя разрешается. В качестве альтернативы директиву `ng-cloak` можно использовать для предотвращения отображения рулей перед его компиляцией.

Отображение всех общих угловых конструкций

В следующем примере показаны общие конструкции AngularJS в одном файле:

```
<!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>
```

Ниже описана каждая строка файла:

Демо-версия

1. `ng-app="myDemoApp"` , директива ngApp, которая загружает приложение и сообщает угловым, что элемент DOM управляется определенным `angular.module` именем `"myDemoApp"` ;
2. `<script src="angular.min.js">` - первый шаг в начальной загрузке библиотеки AngularJS ;

`MyDataService` три функции (`MyDataService` , `DemoController` и `startup`), которые используются (и объясняются) ниже.

3. `angular.module(...)` используемый с массивом, поскольку второй аргумент создает новый модуль. Этот массив используется для предоставления списка зависимостей модулей. В этом примере мы вызываем вызовы на результат функции `module(...)` ;

4. `.service(...)` создает `.service(...)` службу и возвращает модуль для цепочки;

5. `.controller(...)` создает угловой контроллер и возвращает модуль для цепочки;

6. `.config(...)` Используйте этот метод для регистрации работы, которая должна выполняться при загрузке модуля.

7. `.run(...)` гарантирует, что код запускается во время запуска и принимает в качестве параметра массив элементов. Используйте этот метод для регистрации работы, которая должна выполняться при выполнении инжектора, загружая все модули.

    - первый элемент позволяет Angular знать, что для функции `startup` требуется, чтобы встроенная служба `$rootScope` была введена в качестве аргумента;
    - второй элемент позволяет Angular знать, что функция `startup` требует, чтобы встроенная служба `$window` была введена в качестве аргумента;
    - *последний* элемент в массиве, `startup` , является фактической функцией для запуска при запуске;

8. `ng-class` - это директива ngClass для установки динамического `class` , и в этом примере используется `hasStarted` на `$rootScope` динамически

9. `ng-cloak` - это директива по предотвращению ненанесенного шаблона Angular html (например, « `{{ msg }}` »), который должен быть кратко показан до того, как Angular полностью загрузит приложение.

10. `ng-controller` - это директива, которая запрашивает угловое создание нового контроллера определенного имени для организации этой части DOM;

11. `ng-repeat` - это директива, чтобы сделать угловую итерацию по коллекции и клонировать шаблон DOM для каждого элемента;

12. `{{ msg }}` демонстрирует интерполяцию : на месте рендеринга части области или контроллера;

Важность сферы

Поскольку Angular использует HTML для расширения веб-страницы и простого Javascript для добавления логики, упрощает создание веб-страницы с помощью **ng-app** , **ng-controller** и некоторых встроенных директив, таких как **ng-if** , **ng-repeat** и т. Д. . С новым синтаксисом **controllerAs** новички для пользователей Angular могут присоединить функции и данные к своему контроллеру вместо использования `$scope` .

Тем не менее, рано или поздно, это важно , чтобы понять , что именно это `$scope` вещь. Он будет отображаться в примерах, поэтому важно иметь некоторое понимание.

Хорошей новостью является то, что это простая, но мощная концепция.

Когда вы создаете следующее:

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

Где живет **имя** ?

Ответ заключается в том, что Angular создает объект `$rootScope` . Это просто обычный объект Javascript, поэтому **имя** является свойством объекта `$rootScope` :

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

И так же, как с глобальным охватом в Javascript, обычно не так хорошо добавлять элементы в глобальную область или `$rootScope` .

Конечно, большую часть времени мы создаем контроллер и помещаем необходимые функции в этот контроллер. Но когда мы создаем контроллер, Angular делает его магическим и создает объект `$scope` для этого контроллера. Это иногда называют **локальной областью** .

Итак, создав следующий контроллер:

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

позволит доступ к локальной области с помощью параметра `$scope` .

```
angular.module("myApp", [])
  .controller("MyController", function($scope) {
    $scope.name = "Mr Local!";
  });
```

По какой-то причине контроллер без параметра `$scope` может просто не понадобиться. Но важно понимать, что **даже с синтаксисом controllerAs** существует локальная область.

Поскольку `$scope` является объектом JavaScript, Angular волшебным образом устанавливает его для прототипного наследования с `$rootScope` . И, как вы можете себе представить, может быть цепочка областей. Например, вы можете создать модель в родительском контроллере и приложить к ней область действия родительского контроллера как `$scope.model` .

Затем через цепочку прототипов дочерний контроллер может получить доступ к этой же модели локально с помощью `$scope.model` .

Ничего из этого изначально не видно, так как это просто Угловая, делающая свою магию на заднем плане. Но понимание `$scope` - важный шаг в том, чтобы узнать, как работает Angular.

Самый простой возможный угловой мир Hello.

Угловой 1 - это компилятор DOM. Мы можем передать его HTML, либо как шаблон, либо как обычную веб-страницу, а затем скомпилировать приложение.

Мы можем сказать, что Angular обрабатывает область страницы как *выражение,* используя синтаксис стиля `{{ }}` . Все, что между фигурными фигурными скобками будет скомпилировано, вот так:

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

Это приведет к выводу:

```
HelloWorld
```

нг-приложение

Мы указываем Angular, какую часть нашей DOM обрабатываем как главный шаблон,

используя *директиву* `ng-app` . Директива - это настраиваемый атрибут или элемент, которые компилятор Angular template знает, как справиться. Теперь добавим директиву ng-app:

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

Я теперь сказал, что элемент body является корневым шаблоном. Все, что в нем будет скомпилировано.

Директивы

Директивы - это директивы компилятора. Они расширяют возможности компилятора Angular DOM. Вот почему **Misko** , создатель Angular, описывает Angular как:

> «Каким бы веб-браузером он был создан для веб-приложений.

Мы в буквальном смысле создаем новые атрибуты и элементы HTML и имеем Angular для их компиляции в приложение. `ng-app` - это директива, которая просто включает компилятор. Другие директивы включают:

- `ng-click` , который добавляет обработчик кликов,
- `ng-hide` , который условно скрывает элемент и
- `<form>` , который добавляет дополнительное поведение в стандартный элемент формы HTML.

Угловое устройство содержит около 100 встроенных директив, которые позволяют выполнять большинство обычных задач. Мы также можем написать свои собственные, и они будут рассматриваться так же, как встроенные директивы.

Мы создаем Угловое приложение из ряда директив, связанных с HTML.

Минимизация в угловом

Что такое Minification?

Это процесс удаления всех ненужных символов из исходного кода без изменения его функциональности.

Нормальный синтаксис

Если мы используем нормальный угловой синтаксис для написания контроллера, то после миниатюр наших файлов он нарушит нашу функциональность.

Контроллер (до сведения):

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

После использования инструмента минимизации он будет уменьшен, как показано ниже.

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

Здесь minification удалили ненужные пробелы и переменную $ scope из кода. Поэтому, когда мы используем этот минитипированный код, он не собирается печатать что-либо на виду. Поскольку $ scope является важной частью между контроллером и представлением, который теперь заменяется небольшой переменной «e». Поэтому, когда вы запускаете приложение, оно будет выдавать ошибку зависимостей Unknown Provider 'e'.

Существует два способа аннотирования кода с информацией о названии службы, которая безопасна для минимизации:

Встроенный синтаксис аннотации

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

$ inject Синтаксис объявления объявления

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

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

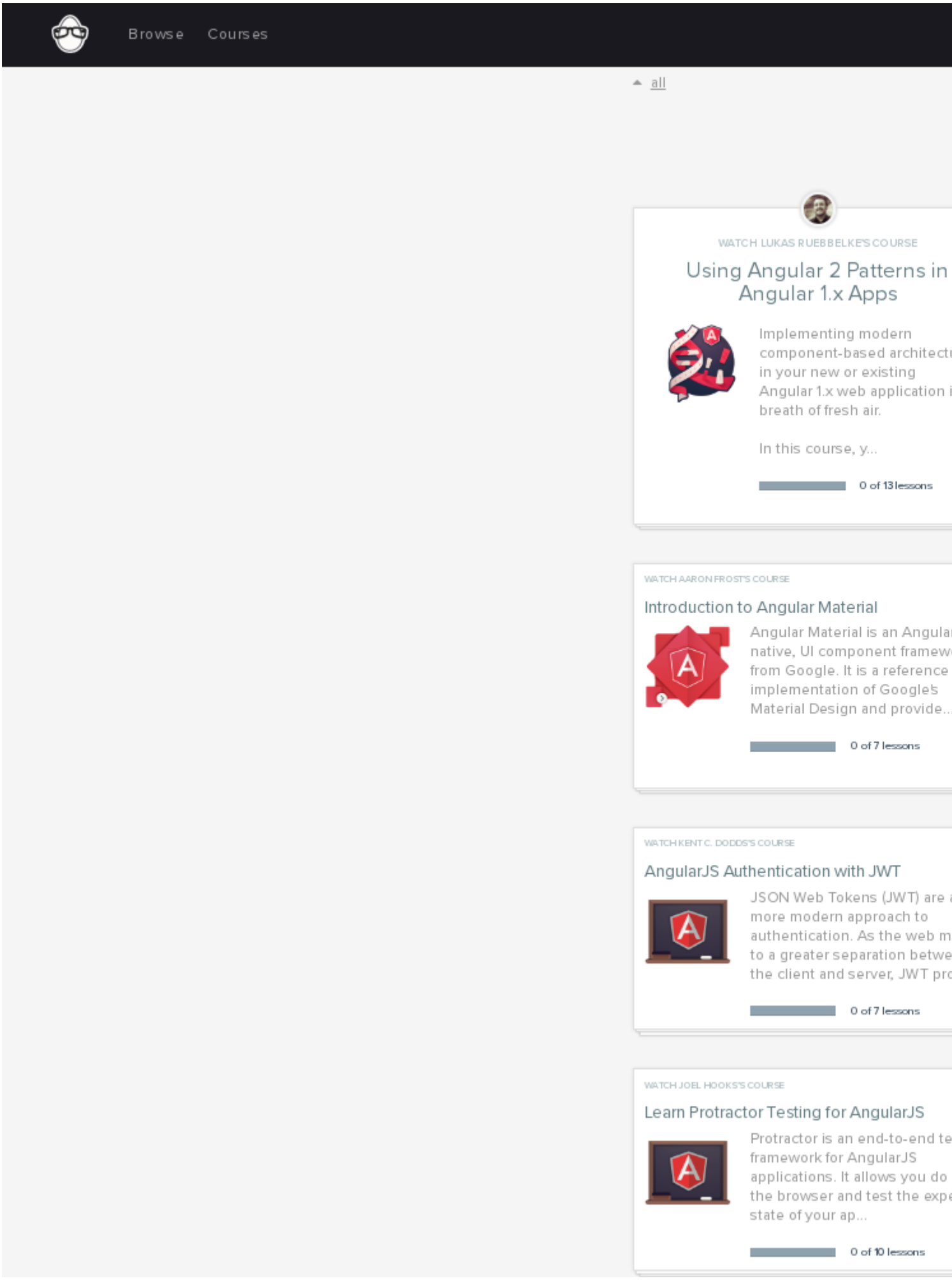
После переоценки этот код будет

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

Здесь угловое будет рассматривать переменную 'a', которая будет рассматриваться как $ scope, и будет отображать вывод как «Hello World!».

AngularJS Начало работы Видеоуроки

Есть много хороших видеоуроков для рамки AngularJS на 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-an-angular-1-x-ionic-application
- https://egghead.io/courses/angular-and-webpack-for-modular-applications
- 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

Прочитайте Начало работы с AngularJS онлайн: https://riptutorial.com/ru/angularjs/topic/295/начало-работы-с-angularjs

глава 2: $ http request

Examples

Использование $ http внутри контроллера

Служба `$http` - это функция, которая генерирует HTTP-запрос и возвращает обещание.

Общее использование

```
// 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.
  });
```

Использование внутри контроллера

```
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.
    });
}])
```

Методы быстрого доступа

`$http` службы `$http` также есть методы быстрого доступа. Читайте о http-методах здесь

Синтаксис

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

Методы быстрого доступа

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

Использование запроса $ http в службе

HTTP-запросы широко используются в каждом веб-приложении, поэтому целесообразно написать метод для каждого общего запроса, а затем использовать его в нескольких местах в приложении.

Создайте `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...
        }
    }
})
```

Служба, указанная выше, выполнит запрос на получение внутри службы. Это будет доступно любому контроллеру, где была введена услуга.

Пример использования

```
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
            })
    }])
```

Используя этот подход, мы теперь можем использовать **httpRequestsService.js в** любое время и в любом контроллере.

Сроки запроса $ http

Запросы $ http требуют времени, которое зависит от сервера, некоторые могут занять несколько миллисекунд, а некоторые могут занять до нескольких секунд. Часто требуется время, необходимое для извлечения данных из запроса. Предполагая, что значение ответа представляет собой массив имен, рассмотрим следующий пример:

некорректный

```
$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]);
```

Доступ к `$scope.names[0]` прямо под запросом $ http часто `$scope.names[0]` ошибку - эта строка кода выполняется до получения ответа от сервера.

Правильный

```
$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);
        });
```

Используя службу $ watch, мы получаем доступ к массиву `$scope.names` только при получении ответа. Во время инициализации функция вызывается даже при инициализации `$scope.names` , поэтому проверка того, `newVal.length` ли `newVal.length` , отличная от 0. Имейте в `$scope.names` - любые изменения, внесенные в `$scope.names` функцию часов.

Прочитайте $ http request онлайн: https://riptutorial.com/ru/angularjs/topic/3620/--http-request

глава 3: AngularJS gotchas и ловушки

Examples

Двухсторонняя привязка данных перестает работать

Следует иметь в виду, что:

1. Угловое связывание данных основывается на прототипном наследовании JavaScript, поэтому оно подвержено переменному затенению .
2. Дочерняя область, обычно прототипически наследуемая от своей родительской области. Единственное исключение из этого правила - это директива, которая имеет изолированную область действия, поскольку она не прототипически наследуется.
3. Существуют некоторые директивы, которые создают новую дочернюю область: `ng-repeat` , `ng-switch` , `ng-view` , `ng-if` , `ng-controller` , `ng-include` и т. Д.

Это означает, что когда вы пытаетесь выполнить двустороннюю привязку некоторых данных к примитиву, находящемуся внутри детской области (или наоборот), все может работать не так, как ожидалось. Вот пример того, как легко *«сломать»* AngularJS.

Эту проблему можно легко избежать, выполнив следующие шаги:

1. Имейте "." внутри вашего HTML-шаблона всякий раз, когда вы связываете некоторые данные
2. Используйте синтаксис `controllerAs` поскольку он способствует использованию привязки к «пунктируемому» объекту
3. $ parent может использоваться для доступа к переменным родительской `scope` а не к области содержимого. как внутри `ng-if` мы можем использовать `ng-model="$parent.foo"` ..

Альтернативой для этого является привязка `ngModel` к функции getter / setter, которая будет обновлять кэшированную версию модели при вызове с аргументами или возвращать ее при вызове без аргументов. Чтобы использовать функцию getter / setter, вам нужно добавить `ng-model-options="{ getterSetter: true }"` к элементу с атрибутом `ngModal` и вызвать функцию getter, если вы хотите отобразить ее значение в выражении (Рабочий пример).

пример

Посмотреть:

```
<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>
```

контроллер:

```
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;
    };
}]);
```

Лучшая практика . Лучше всего держать геттеры быстрыми, потому что Угловые, скорее всего, назовут их чаще, чем другие части вашего кода (ссылка).

Что делать при использовании html5Mode

При использовании `html5Mode([mode])` необходимо:

1. Вы указываете базовый URL-адрес приложения с `<base href="">` в `index.html` вашего `index.html` .
2. Важно, чтобы `base` тег появился перед любыми тегами с запросами url. В противном случае это может привести к этой ошибке - `"Resource interpreted as stylesheet but transferred with MIME type text/html"` . Например:

   ```
   <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. Если вы не хотите указывать `base` тег, настройте `$locationProvider` чтобы не требовать `base` тег, передав объект определения с `requireBase:false` в `$locationProvider.html5Mode()` следующим образом:

   ```
   $locationProvider.html5Mode({
       enabled: true,
   ```

```
    requireBase: false
});
```

4. Чтобы поддерживать прямую загрузку URL-адресов HTML5, вам необходимо активировать переписывание URL-адресов на стороне сервера. From AngularJS / Руководство разработчика / Использование $ location

   > Использование этого режима требует перезаписи URL-адресов на стороне сервера, в основном вы должны переписать все свои ссылки на точку входа вашего приложения (например, `index.html`). Требование `<base>` также важно для этого случая, так как позволяет Angular различать часть URL-адреса, которая является базой приложения, и путь, который должен обрабатывать приложение.

Отличный ресурс для запросов перезаписи запросов для различных реализаций HTTP-сервера можно найти в FAQ по ui-router - Как настроить сервер для работы с html5Mode . Например, сходница

```
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;
    }
}
```

экспресс

```
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 Смертельные грехи AngularJS

Ниже приведен список ошибок, которые разработчики часто делают во время использования функций AngularJS, некоторые извлеченные уроки и решения для них.

1. Манипуляция DOM через контроллер

Это законно, но его следует избегать. Контроллеры - это места, где вы определяете свои зависимости, связываете свои данные с представлением и делаете дальнейшую бизнес-логику. Вы можете технически манипулировать DOM в контроллере, но всякий раз, когда вам нужна такая же или аналогичная манипуляция в другой части вашего приложения, вам понадобится другой контроллер. Поэтому лучшей практикой такого подхода является создание директивы, включающей все манипуляции и использование директивы во всем приложении. Следовательно, контроллер оставляет представление неповрежденным и делает его работу. В директиве функция связывания - лучшее место для манипулирования DOM. Он имеет полный доступ к области и элементу, поэтому, используя директиву, вы также можете воспользоваться преимуществами повторного использования.

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

Вы можете получить доступ к элементам DOM при связывании функции несколькими способами, такими как параметр `element` , `angular.element()` или чистый Javascript.

2. Связывание данных при переходе

AngularJS славится своей двусторонней привязкой данных. Однако иногда вы можете столкнуться с тем, что ваши данные только односторонне связаны внутри директив. Остановитесь там, AngularJS не ошибается, но, вероятно, вы. Директивы - это немного опасные места, так как задействованы дочерние области и изолированные области. Предположим, что у вас есть следующая директива с одним заключением

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

И внутри моего переключения у вас есть некоторые элементы, привязанные к данным во внешней области.

```
<my-dir>
```

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

Вышеприведенный код не будет работать правильно. Здесь переключение создает дочернюю область, и вы можете получить переменную имени, но все изменения, внесенные вами в эту переменную, останутся там. Таким образом, вы можете действительно использовать эту переменную как **$ parent.name** . Однако это использование может быть не лучшей практикой. Лучшим подходом было бы обертывание переменных внутри объекта. Например, в контроллере вы можете создать:

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

Затем в выводе вы можете получить доступ к этой переменной через объект «data» и увидеть, что двусторонняя привязка работает отлично!

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

Не только в переходах, но и во всем приложении, рекомендуется использовать пунктирную нотацию.

3. Несколько директив вместе

Фактически законно использовать две директивы вместе внутри одного и того же элемента, если вы подчиняетесь правилу: две изолированные области не могут существовать на одном элементе. Вообще говоря, при создании новой настраиваемой директивы вы выделяете изолированную область для простой передачи параметров. Предполагая, что в директивах myDirA и myDirB есть выделенные области и myDirC нет, следующий элемент будет действителен:

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

тогда как следующий элемент вызовет ошибку консоли:

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

Поэтому директивы должны использоваться разумно, принимая во внимание области.

4. Неправильное использование $ emit

$ emit, $ broadcast и $ on, они работают в принципе отправителя-получателя. Другими словами, они являются средством связи между контроллерами. Например, следующая строка испускает «someEvent» из контроллера A, который должен быть захвачен

соответствующим контроллером B.

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

И следующая строка ловит 'someEvent'

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

Пока все кажется идеальным. Но помните, что если контроллер B еще не вызывается, событие не будет выловлено, что означает, что для его работы должны быть задействованы как контроллеры-эмиттеры, так и приемники. Итак, опять же, если вы не уверены, что вам определенно нужно использовать $ emit, создание службы кажется лучшим способом.

5. Неправильное использование $ scope. $ Watch

$ scope. $ watch используется для просмотра изменения переменной. Всякий раз, когда переменная изменена, этот метод вызывается. Однако одна распространенная ошибка заключается в изменении переменной внутри $ scope. $ Watch. Это вызовет непоследовательность и бесконечный цикл $ digest в какой-то момент.

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

Поэтому в приведенной выше функции убедитесь, что у вас нет операций с myVariable и newVal.

6. Методы привязки к представлениям

Это один из самых тяжких грехов. У AngularJS есть двусторонняя привязка, и всякий раз, когда что-то меняется, представления обновляются много раз. Таким образом, если вы привязываете метод к атрибуту представления, этот метод потенциально может называться сто раз, что также приводит вас в замешательство во время отладки. Однако для привязки метода существуют только некоторые атрибуты, такие как ng-click, ng-blur, ng-on-change и т. Д., Которые ожидают, что методы будут использоваться как пареметр. Например, предположим, что в вашей разметке есть следующее представление:

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

Здесь вы проверяете отключенное состояние представления с помощью метода isDisabled. В контроллере myCtrl у вас есть:

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

```
    else
        return false;
}
```

Теоретически это может показаться правильным, но технически это вызовет перегрузку, поскольку метод будет работать бесчисленное количество раз. Чтобы решить эту проблему, вы должны привязать переменную. В вашем контроллере должна существовать следующая переменная:

```
vm.isDisabled
```

Вы можете снова инициировать эту переменную при активации контроллера

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

Если условие нестабильно, вы можете связать это с другим событием. Затем вы должны привязать эту переменную к виду:

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

Теперь все атрибуты представления имеют то, что они ожидают, и методы будут выполняться только тогда, когда это необходимо.

7. Не использовать функции Углового

AngularJS обеспечивает большое удобство с некоторыми его функциями, не только упрощает ваш код, но и делает его более эффективным. Некоторые из этих функций перечислены ниже:

1. **angular.forEach** для петель (осторожно, вы не можете «сломать», это может предотвратить только попадание в тело, поэтому рассмотрите производительность здесь.)
2. **угловой.элемент** для DOM-селекторов
3. **angular.copy** : используйте это, когда вы не должны изменять основной объект
4. **Валидации формы** уже потрясающие. Используйте грязную, девственную, тронутую, действительную, необходимую и так далее.
5. Помимо отладчика Chrome, используйте **удаленную отладку** для разработки мобильных устройств.
6. И убедитесь, что вы используете **Batarang** . Это бесплатное расширение Chrome, где вы можете легко просматривать области
   ,

Прочитайте AngularJS gotchas и ловушки онлайн:

https://riptutorial.com/ru/angularjs/topic/3208/angularjs-gotchas-и-ловушки

глава 4: angularjs с фильтром данных, разбиением на страницы и т. д.

Вступление

Пример провайдера и запрос о отображении данных с фильтром, разбиением на страницы и т. Д. В Angularjs.

Examples

Angularjs отображает данные с фильтром, разбиение на страницы

```
<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>
```

Прочитайте angularjs с фильтром данных, разбиением на страницы и т. д. онлайн:

https://riptutorial.com/ru/angularjs/topic/10821/angularjs-с-фильтром-данных--разбиением-на-страницы-и-т--д-

глава 5: Grunt tasks

Examples

Запустить приложение локально

> В следующем примере необходимо установить node.js и доступно npm .
> Полный рабочий код можно разветвить у GitHub @
> https://github.com/mikkoviitala/angular-grunt-run-local

Обычно одна из первых вещей, которые вы хотите сделать при разработке нового веб-приложения, - заставить его работать локально.

Ниже вы найдете полный пример достижения этого, используя grunt (javascript task runner), npm (менеджер пакетов узлов) и bower (еще один менеджер пакетов).

Помимо ваших реальных файлов приложений вам необходимо установить несколько зависимостей сторонних разработчиков, используя перечисленные выше средства. В вашем каталоге проекта, **предпочтительно root** , вам понадобятся три (3) файла.

- package.json (зависимости, управляемые npm)
- bower.json (зависимости, управляемые беседкой)
- gruntfile.js (grunt tasks)

Таким образом, ваш каталог проекта выглядит так:

---

package.json

Мы будем устанавливать **grunt** самостоятельно, **matchdep,** чтобы упростить нашу жизнь, чтобы мы могли фильтровать зависимости по имени, **grunt-express,** используемые для запуска экспресс-веб-сервера через grunt и **grunt-open** для открытия URL-адресов / файлов из задачи grunt.

Таким образом, эти пакеты касаются «инфраструктуры» и помощников, на которые мы будем строить наше приложение.

```
{
  "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

Бауэр (или, по крайней мере, должен быть) должен иметь все об интерфейсе, и мы будем использовать его для установки **углового** .

```
{
  "name": "app",
  "version": "1.0.0",
  "dependencies": {
    "angular": "~1.3.x"
  },
  "devDependencies": {}
}
```

gruntfile.js

Внутри gruntfile.js у нас будет реальная «работающее приложение локально», которое открывает наше приложение в новом окне браузера, запущенном на http: // localhost: 9000 /

```
'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'
  ]);
};
```

использование

Чтобы получить приложение и запустить с нуля, сохраните файлы выше в корневой каталог вашего проекта (любая пустая папка будет делать). Затем запустите консоль / командную строку и введите следующую команду, чтобы установить все необходимые зависимости.

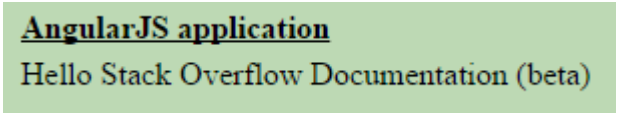
```
npm install -g grunt-cli bower
npm install
```

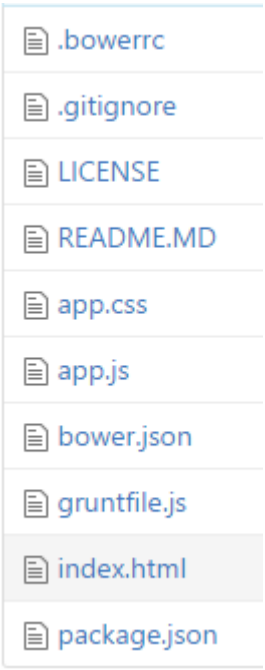
Затем запустите приложение, используя

```
grunt app
```

Обратите внимание, что да, вам также понадобятся ваши фактические файлы приложений.
Для почти-минимального примера просмотрите репозиторий GitHub, упомянутый в начале этого примера.

Там структура не такая уж иная. В `app.js` есть только шаблон `index.html` , угловой код в `app.js` и несколько стилей в `app.css` . Другие файлы предназначены для настройки Git и редактора и некоторых общих вещей. Попробуй!





Прочитайте Grunt tasks онлайн: https://riptutorial.com/ru/angularjs/topic/6077/grunt-tasks

глава 6: HTTP-перехватчик

Вступление

Служба $ http AngularJS позволяет нам связываться с бэкэнд и делать HTTP-запросы. Есть случаи, когда мы хотим захватить каждый запрос и манипулировать им, прежде чем отправлять его на сервер. В других случаях мы хотели бы зафиксировать ответ и обработать его до завершения вызова. Глобальная обработка ошибок HTTP также может служить хорошим примером такой необходимости. Для таких случаев создаются перехватчики.

Examples

Начиная

Встроенная служба `$http` Angular позволяет отправлять HTTP-запросы. Часто возникает необходимость делать что-либо до или после запроса, например, добавляя к каждому запросу токен аутентификации или создавая общую логику обработки ошибок.

Общий HTTPInterceptor шаг за шагом

Создайте HTML-файл со следующим содержимым:

```
<!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>
```

Добавьте файл JavaScript под названием «app.js»:

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

interceptorApp.config(function($httpProvider) {
        $httpProvider.interceptors.push('genericInterceptor');
});
```

Добавьте еще один «appController.js»:

```
(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]);
})();
```

И, наконец, файл, содержащий сам перехватчик '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);
})();
```

«GenericInterceptor» охватывает возможные функции, которые мы можем переопределить добавлением дополнительного поведения в наше приложение.

Flash-сообщение с ответом, использующее http-перехватчик

В файле вида

В базовом html (index.html), где мы обычно включаем угловые скрипты или html, которые

совместно используются в приложении, оставьте пустой элемент div, флэш-сообщения появятся внутри этого элемента div

```
<div class="flashmessage" ng-if="isVisible">
    {{flashMessage}}
</div>
```

Файл сценария

В конфигурационном методе углового модуля введите httpProvider, httpProvider имеет свойство массива перехватчика, нажмите пользовательский перехватчик. В текущем примере пользовательский перехватчик перехватывает только ответ и вызывает метод, прикрепленный к rootScope.

```
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;
                                    }
                    };
        }]);

    }])
```

Поскольку в конфигурационный метод углового модуля (который является httpProvider, а не корнеплодом) могут быть введены только провайдеры, объявите метод, прикрепленный к корневому методу внутри метода запуска углового модуля.

Также отобразите сообщение внутри $ timeout, чтобы сообщение получило свойство flash, которое исчезает после порогового времени. В нашем примере это 3000 мс.

```
interceptorTest.run(["$rootScope","$timeout",function($rootScope,$timeout){
     $rootScope.showFeedBack = function(status,message){

        $rootScope.isVisible = true;
        $rootScope.flashMessage = message;
        $timeout(function(){$rootScope.isVisible = false },3000)
    }
}]);
```

Обычные подводные камни

Пытаясь ввести **$ rootScope или любые другие сервисы** внутри метода **конфигурации** углового модуля, жизненный цикл углового приложения не позволяет выбросить эту и неизвестную ошибку поставщика. В **конфигурационный** метод углового модуля могут быть введены только **поставщики**

Прочитайте HTTP-перехватчик онлайн: https://riptutorial.com/ru/angularjs/topic/6484/http-перехватчик

глава 7: SignalR с AngularJs

Вступление

В этой статье мы сосредоточимся на «Как создать простой проект с использованием AngularJs And SignalR», в этом обучении вам нужно знать о том, как создать приложение с помощью angularjs, «как создать / использовать сервис по угловому» и базовые знания о SignalR «для этого мы рекомендуем https://www.codeproject.com/Tips/590660/Introduction-to-SignalR) .

Examples

SignalR And AngularJs [ChatProject]

Шаг 1. Создание проекта

```
- Application
    - app.js
    - Controllers
        - appController.js
    - Factories
        - SignalR-factory.js
- index.html
- Scripts
    - angular.js
    - jquery.js
    - jquery.signalR.min.js
- Hubs
```

Использование версии сигнала SignalR: signalR-2.2.1

Шаг 2: Startup.cs и ChatHub.cs

Перейдите в каталог *«/ Hubs»* и добавьте 2 файла *[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);
        }
    }
}
```

Шаг 3: создание углового приложения

Перейдите в каталог *«/ Приложение»* и *добавьте* файл *[app.js]*

app.js

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

Шаг 4: создать SignalR Factory

Перейдите в каталог *«/ Приложение / Фабрики»* и добавьте *файл [SignalR-factory.js]*

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;
});
```

Шаг 5: обновите app.js

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

app.run(function(signalR) {
    signalR.url("http://localhost:21991/signalr");
});
```

localhost: 21991 / signalr | **это ваш SignalR Hubs Urls**

Шаг 6: добавьте контроллер

Перейдите в каталог *«/ Приложение / Контроллеры»* и *добавьте* файл *[appController.js]*

```
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] (открытый класс)>** *ChatHub.cs*

Примечание: не вставляйте *HubName* с верхним *регистром* , **первая буква** ниже.

signalR.client () | этот метод пытается подключиться к вашим концентраторам и получить все функции в концентраторах, в этом примере у нас есть «chatHub», чтобы получить функцию «broadcastMessage ()»;

Шаг 7: добавьте index.html в маршрут каталога

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
```

Результат с изображением

Пользователь 1 (отправлять и получать)

Пользователь 2 (отправлять и получать)

Прочитайте SignalR с AngularJs онлайн: https://riptutorial.com/ru/angularjs/topic/9964/signalr-c-angularjs

глава 8: UI-маршрутизатор

замечания

Что такое `ui-router` ?

> Угловой UI-Router - это платформа маршрутизации односторонней страницы на стороне клиента для AngularJS.
>
> Структуры маршрутизации для SPA-центров обновляют URL-адрес браузера, когда пользователь переходит через приложение. И наоборот, это позволяет изменять URL-адрес браузера для навигации по приложению, что позволяет пользователю создавать закладку в месте, расположенном внутри SPA.
>
> Приложения UI-Router моделируются как иерархическое дерево состояний. UI-Router предоставляет конечный автомат для управления переходами между этими состояниями приложения транзакционным образом.

Полезные ссылки

Вы можете найти официальный API документации здесь . По вопросам `ui-router` VS `ng-router` , вы можете найти достаточно подробный ответ здесь . Имейте в виду, что `ng-router` уже выпустил новое обновление ng-router под названием `ngNewRouter` (совместимое с Angular 1.5 + / 2.0), которое поддерживает состояния, подобные ui-router. Вы можете прочитать больше о `ngNewRouter` здесь .

Examples

Основной пример

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>
```

Несколько просмотров

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>
```

Использование функций разрешения для загрузки данных

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>
```

Просмотреть запись WI-Router Wiki здесь разрешает .

Устранение функций должно быть разрешено до того, как будет `$stateChangeSuccess` событие `$stateChangeSuccess` , что означает, что пользовательский интерфейс не будет загружаться, пока *все* функции разрешения в состоянии не закончатся. Это отличный способ гарантировать, что данные будут доступны для вашего контроллера и пользовательского интерфейса. Однако вы можете видеть, что функция разрешения должна быть быстрой, чтобы не повредить пользовательский интерфейс.

Вложенные представления / состояния

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>
```

Прочитайте UI-маршрутизатор онлайн: https://riptutorial.com/ru/angularjs/topic/2545/ui-маршрутизатор

глава 9: Внедрение зависимости

Синтаксис

- myApp.controller ('MyController', function ($ scope) {...}); // недопустимый код
- myApp.controller ('MyController', ['$ scope', function ($ scope) {...}]); // поддержка минимизации
- функция MyController () {}

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

  myApp.controller ('MyController', MyController); // $ вставка аннотации
- $ Injector.get ('инъекционные'); // динамическая / временная инъекция

замечания

Провайдеры не могут быть введены в блоки `run` .

Услуги или значения не могут быть введены в `config` блоки.

Удостоверьтесь, что аннотируете свои инъекции, чтобы ваш код не прерывался при минировании.

Examples

вливания

Простейший пример инъекции в угловом приложении - инъекция `$scope` в угловой `Controller` :

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

Вышеупомянутое иллюстрирует инъекцию `$scope` в `controller` , но все равно, вводите ли вы какой-либо модуль в любой другой. Процесс тот же.

Система Angular отвечает за разрешение зависимостей для вас. Если вы создаете сервис, например, вы можете перечислить его, как в приведенном выше примере, и он будет доступен вам.

Вы можете использовать DI - Dependency Injection - везде, где вы определяете компонент.

Обратите внимание, что в приведенном выше примере мы используем так называемую « Inline Array Annotation». Смысл, мы явно записываем в качестве строк имена наших зависимостей. Мы делаем это, чтобы предотвратить нарушение приложения при кодировании кода для Production. Мидификация кода изменяет имена переменных (но не строк), что прерывает инъекцию. Используя строки, Angular знает, какие зависимости мы хотим.

Очень важно - порядок имен строк должен быть таким же, как и параметры в функции.

Есть инструменты, которые автоматизируют этот процесс и заботятся об этом для вас.

Динамические инъекции

Существует также возможность динамически запрашивать компоненты. Вы можете сделать это, используя службу `$injector` :

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

Примечание. Хотя этот метод можно использовать для предотвращения проблемы с циклической зависимостью, которая может нарушить ваше приложение, не рекомендуется использовать эту проблему, чтобы обойти проблему. Циркулярная зависимость обычно указывает на наличие недостатков в архитектуре вашего приложения, и вам следует обратиться к этому вопросу.

$ inject Объявление свойства

Эквивалентно, мы можем использовать аннотацию свойства `$inject` для достижения того же, что и выше:

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

Динамически загружать службу AngularJS в ванильном JavaScript

Вы можете загружать службы AngularJS в ванильном JavaScript с использованием метода AngularJS `injector()` . Каждый элемент jqLite, полученный из вызывающего `angular.element()` имеет метод `injector()` который может использоваться для извлечения инжектора.

```
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);
}
```

В приведенном выше примере мы пытаемся получить элемент jqLite, содержащий корень приложения AngularJS (`ngAppElement`). Для этого мы используем `angular.element()` , ищем элемент DOM, содержащий атрибут `ng-app` или `data-ng-app` или, если он не существует, мы возвращаемся к элементу `document` . Мы используем `ngAppElement` для извлечения экземпляра инжектора (с помощью `ngAppElement.injector()`). Экземпляр инжектора используется, чтобы проверить, существует ли служба для инъекции (с `injector.has()`), а затем для загрузки службы (с помощью `injector.get()`) внутри `service` переменной.

Прочитайте Внедрение зависимости онлайн: https://riptutorial.com/ru/angularjs/topic/1582/внедрение-зависимости

глава 10: Встроенные вспомогательные функции

Examples

angular.equals

Функция `angular.equals` сравнивает и определяет, равны ли 2 объекта или значения, `angular.equals` выполняет глубокое сравнение и возвращает true тогда и только тогда, `angular.equals` выполняется `angular.equals` одно из следующих условий.

> угловые.равнения (значение1, значение2)

1. Если объекты или значения проходят сравнение `===`
2. Если оба объекта или значения имеют один и тот же тип, и все их свойства также равны с использованием `angular.equals`
3. Оба значения равны `NaN`
4. Оба значения представляют собой результат одного и того же регулярного выражения.

Эта функция полезна, когда вам нужно глубоко сравнивать объекты или массивы по их значениям или результатам, а не только по ссылкам.

Примеры

```
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
```

angular.isString

Функция `angular.isString` возвращает true, если объект или значение, присвоенное ему, имеют `string` типа

> angular.isString (значение1)

Примеры

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

Это эквивалентно выполнению

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

angular.isArray

Функция `angular.isArray` возвращает true тогда и только тогда, когда объект или значение передано ему типа `Array` .

angular.isArray (значение)

Примеры

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

Это эквивалент

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

angular.merge

Функция angular.merge принимает все перечислимые свойства из исходного объекта, чтобы глубоко расширить объект назначения.

Функция возвращает ссылку на теперь расширенный объект назначения

angular.merge (пункт назначения, источник)

Примеры

```
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}}}
```

angular.isDefined и angular.isUndefined

Функция `angular.isDefined` проверяет значение, если оно определено

angular.isDefined (SomeValue)

Это эквивалентно выполнению

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

Примеры

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

Функция `angular.isUndefined` проверяет, не определено ли значение (оно фактически противоположно `angular.isDefined`)

    angular.isUndefined (SomeValue)

Это эквивалентно выполнению

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

Или просто

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

Примеры

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

angular.isDate

Функция `angular.isDate` возвращает true тогда и только тогда, когда объект, переданный ему, имеет тип Date.

    angular.isDate (значение)

Примеры

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

angular.isNumber

Функция `angular.isNumber` возвращает true тогда и только тогда, когда объект или значение, переданное ему, имеет тип Number, это включает в себя + Infinity, -Infinity и NaN

    angular.isNumber (значение)

Эта функция не приведет к принуждению типа, например

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

Примеры

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

Эта функция не приведет к принуждению типа, например

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

angular.isFunction

Функция `angular.isFunction` определяет и возвращает true тогда и только тогда, когда переданное значение является ссылкой на функцию.

Функция возвращает ссылку на теперь расширенный объект назначения

angular.isFunction (п)

Примеры

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

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

angular.toJson

Функция `angular.toJson` возьмет объект и сериализует его в форматированную строку JSON.

В отличие от нативной функции `JSON.stringify` , эта функция удалит все свойства, начинающиеся с `$$` (как угловые обычно префиксы внутренних свойств с `$$`)

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

Поскольку данные должны быть сериализованы до прохождения через сеть, эта функция полезна для превращения любых данных, которые вы хотите передать в JSON.

Эта функция также полезна для отладки, поскольку она работает аналогично методу `.toString` .

Примеры:

```
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
```

angular.fromJson

Функция `angular.fromJson` десериализует действительную строку JSON и возвращает объект или массив.

angular.fromJson (строка | объект)

Обратите внимание, что эта функция не ограничивается только строками, она выводит представление любого переданного ей объекта.

Примеры:

```
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"
```

angular.noop

Функция `angular.noop` - это функция, которая не выполняет никаких операций, вы передаете `angular.noop` когда вам нужно предоставить аргумент функции, который ничего не сделает.

angular.noop ()

Обычным использованием для `angular.noop` может быть предоставление пустой обратного вызова функции, которая иначе выдает ошибку, когда ей передается что-то еще, кроме функции.

Пример:

```
$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
```

Дополнительные примеры:

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

angular.isObject

Функция `angular.isObject` возвращает true тогда и только тогда, когда переданный ей аргумент является объектом, эта функция также вернет true для массива и вернет значение false для `null` даже если `typeof null` является `object` .

angular.isObject (значение)

Эта функция полезна для проверки типов, когда вам нужен определенный объект для обработки.

Примеры:

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

angular.isElement

`angular.isElement` возвращает true, если переданный ему аргумент представляет собой элемент DOM или элемент, обернутый jQuery.

angular.isElement (эль)

Эта функция полезна, чтобы набрать проверку, если переданный аргумент является элементом перед его обработкой как таковым.

Примеры:

```
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
```

angular.copy

Функция `angular.copy` принимает объект, массив или значение и создает его глубокую копию.

> angular.copy ()

Пример:

Объекты:

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

Массивы:

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

В приведенном выше примере `angular.equals(w, q)` будет оцениваться как истинное, потому что `.equals` проверяет равенство по значению. однако `w === q` будет оцениваться как false, потому что строгое сравнение между объектами и массивами выполняется по ссылке.

angular.identity

Функция `angular.identity` возвращает первый переданный ей аргумент.

> angular.identity (аргумент)

Эта функция полезна для функционального программирования, вы можете предоставить эту функцию по умолчанию в случае, если ожидаемая функция не была передана.

Примеры:

```
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
```

angular.forEach

`angular.forEach` принимает объект и функцию итератора. Затем он запускает функцию итератора над каждым перечислимым свойством / значением объекта. Эта функция также работает с массивами.

Подобно JS-версии `Array.prototype.forEach` Функция не выполняет итерацию по унаследованным свойствам (свойства прототипа), однако функция не будет пытаться обрабатывать значение `null` или `undefined` значение и просто вернет его.

> angular.forEach (объект, функция (значение, ключ) {// функция});

Примеры:

```
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
```

Прочитайте Встроенные вспомогательные функции онлайн:
https://riptutorial.com/ru/angularjs/topic/3032/встроенные-вспомогательные-функции

глава 11: Встроенные директивы

Examples

Угловые выражения - текст и номер

В этом примере показано, как вычисляются угловые выражения при использовании `type="text"` и `type="number"` для элемента ввода. Рассмотрим следующий контроллер и представление:

контроллер

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

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

Посмотреть

```
<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>
```

- При использовании `+` в выражении, привязанном к *текстовому* вводу, оператор **объединяет** строки (первый пример), отображая 55 на экране `*` .
- При использовании `+` в выражении, связанном с *номером* ввода, оператор возвращает **сумму** чисел (второй пример), отображая 10 на экране `*` .

`*` - То есть, пока пользователь не изменит значение в поле ввода, после этого дисплей изменится соответствующим образом.

Рабочий пример

ngRepeat

`ng-repeat` - встроенная директива в Angular, которая позволяет вам перебирать массив или объект и дает вам возможность повторять элемент один раз для каждого элемента в коллекции.

ng-повторить массив

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

Куда:
item = отдельный элемент в коллекции
itemCollection = массив, который вы повторяете

ng-повторить объект

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

Куда:
key = имя свойства
value = значение свойства
myObject = объект, который вы повторяете

отфильтруйте свой ng-repeat по пользовательскому вводу

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

Куда:
searchText = текст, который пользователь хочет отфильтровать список
stringArray = массив строк, например `['string', 'array']`

Вы также можете отображать или ссылаться на отфильтрованные элементы в другом месте, назначая выход фильтра псевдонимам с `as aliasName` , например:

```
<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 и ng-repeat-end

Чтобы повторить несколько элементов DOM, определив начало и конечную точку, вы можете использовать директивы `ng-repeat-start` и `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>
```

Выход:

- 1
- 2
- 3
- 4

Важно всегда закрывать `ng-repeat-start` с `ng-repeat-end` .

переменные

`ng-repeat` также предоставляет эти переменные внутри выражения

переменная	Тип	подробности
`$index`	Число	Равна индексу текущей итерации ($ index === 0 будет оценивать значение true на первом итерированном элементе, `$first` увидеть `$first`)
`$first`	логический	Вычисляет значение true при первом повторном элементе
`$last`	логический	Вычисляет значение true при последнем повторном элементе
`$middle`	логический	Вычисляет значение true, если элемент находится между `$first` и `$last`
`$even`	логический	Вычисляет значение true при четной итерации (эквивалентно `$index%2===0`)
`$odd`	логический	Вычисляет значение true при нечетной итерации

переменная	Тип	подробности
		(эквивалентно `$index%2===1`)

Требования к производительности

Рендеринг `ngRepeat` может стать медленным, особенно при использовании больших коллекций.

Если объекты в коллекции имеют свойство идентификатора, вы всегда должны `track by` идентификатор, а не весь объект, который является функциональностью по умолчанию. Если идентификатор отсутствует, вы всегда можете использовать встроенный `$index` .

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

Область действия ngRepeat

`ngRepeat` всегда будет создавать изолированную область для `ngRepeat` , поэтому следует соблюдать осторожность, если к нему нужно получить доступ к родительской области внутри повтора.

Вот простой пример, показывающий, как вы можете установить значение в своей родительской области из события click внутри `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'
}];
```

Если в `ng-click` был только `val = item.value` он не будет обновлять `val` в родительской области из-за изолированной области. Вот почему доступ к родительской области осуществляется с помощью `$parent` reference или с синтаксисом `controllerAs` (например, `ng-controller="mainController as ctrl"`).

Вложенные ng-repeat

Вы также можете использовать вложенный ng-repeat.

```
<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']
  ];
});
```

Чтобы получить доступ к индексу родительского ng-repeat внутри дочернего ng-repeat, вы можете использовать `$parent.$index` .

ngShow и ngHide

Директива `ng-show` показывает или скрывает элемент HTML на основе того, передано ли ему выражение true или false. Если значение выражения ложно, то оно будет скрыто. Если это правда, то это покажет.

Директива `ng-hide` аналогична. Однако, если значение ложно, оно отобразит элемент HTML. Когда выражение правдиво, оно скроет его.

Рабочий пример JSBin

Контроллер :

```
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';

}
```

Посмотреть

```
<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` - это директива, упрощающая создание раскрывающегося окна html для выбора элемента из массива, который будет храниться в модели. Атрибут ngOptions используется для динамического создания списка элементов `<option>` для элемента `<select>` используя массив или объект, полученный путем оценки выражения понимания ngOptions.

С `ng-options` разметка может быть уменьшена до всего лишь тега select, и директива создаст тот же самый выбор:

```
<select ng-model="selectedFruitNgOptions"
        ng-options="curFruit as curFruit.label for curFruit in fruit">
</select>
```

Существует еще один способ создания опций `select` с помощью `ng-repeat` , но не рекомендуется использовать `ng-repeat` поскольку он в основном используется для общего использования, например, `forEach` только для цикла. В то время как `ng-options` специально предназначено для создания опций `select` tag.

Вышеприведенный пример с использованием `ng-repeat` будет

```
<select ng-model="selectedFruit">
    <option ng-repeat="curFruit in fruit" value="{{curFruit}}">
```

```
        {{curFruit.label}}
    </option>
</select>
```

ПОЛНЫЙ ПРИМЕР

Давайте рассмотрим приведенный выше пример подробно и с некоторыми изменениями в нем.

Модель данных для примера:

```
$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>
```

Тег опции, сгенерированный при выборе:

```
 <option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Последствия:

`f.label` будет меткой `<option>` и значение будет содержать весь объект.

ПОЛНЫЙ ПРИМЕР

```
<!-- select as label for value in array -->
<select ng-options="f.value as f.label for f in fruit" ng-model="selectedFruit"></select>
```

Тег опции, сгенерированный при выборе:

```
 <option value="4"> Apples </option>
```

Последствия:

`f.value` (4) будет значением в этом случае, в то время как метка остается прежней.

ПОЛНЫЙ ПРИМЕР

```
<!-- 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>
```

Тег опции, сгенерированный при выборе:

```
<option value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Последствия:

Параметры будут сгруппированы в зависимости от `value` . Варианты с одинаковым `value` будут подпадать под одну категорию

ПОЛНЫЙ ПРИМЕР

```
<!-- 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>
```

Тег опции, сгенерированный при выборе:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Последствия:

«Яблоки» и «Лаймы» будут отключены (невозможно выбрать) из-за того, что условие `disable when f.value==4` . Все параметры со `value=4` должны быть отключены

ПОЛНЫЙ ПРИМЕР

```
<!-- 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>
```

Тег опции, сгенерированный при выборе:

```
<option value="4"> Apples </option>
```

Последствия:

При использовании `trackBy` нет визуальных изменений, но Angular обнаруживает изменения по `id` вместо ссылки, которая всегда является лучшим решением.

ПОЛНЫЙ ПРИМЕР

```
<!-- 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>
```

Тег опции, сгенерированный при выборе:

```
<option disabled="" value="{ label: "Apples", value: 4, id: 2 }"> Apples </option>
```

Последствия:

`orderBy` - это стандартный фильтр AngularJS, который упорядочивает параметры в порядке возрастания (по умолчанию), поэтому «Апельсины» в этом появятся 1-м, так как его `id` = 1.

ПОЛНЫЙ ПРИМЕР

Все `<select>` с `ng-options` должны иметь прикрепленную `ng-model` .

ngModel

С ng-моделью вы можете привязать переменную к любому типу поля ввода. Вы можете отобразить переменную, используя двойные фигурные скобки, например `{{myAge}}` .

```
<input type="text" ng-model="myName">
<p>{{myName}}</p>
```

Когда вы вводите поле ввода или меняете его каким-либо образом, вы сразу увидите значение в обновлении абзаца.

В этом случае переменная ng-model будет доступна в вашем контроллере как `$scope.myName` . Если вы используете синтаксис `controllerAs` :

```
<div ng-controller="myCtrl as mc">
    <input type="text" ng-model="mc.myName">
    <p>{{mc.myName}}</p>
</div>
```

Вам нужно будет обратиться к области контроллера, предварительно отложив псевдоним контроллера, определенный в атрибуте ng-controller, на переменную ng-model. Таким образом вам не нужно вводить `$scope` в ваш контроллер, чтобы ссылаться на вашу переменную ng-model, переменная будет доступна как `this.myName` внутри функции вашего контроллера.

ngClass

Предположим, что вам нужно показать статус пользователя, и у вас есть несколько возможных классов CSS, которые можно использовать. Угловое позволяет легко выбирать из списка нескольких возможных классов, которые позволяют указать список объектов, которые включают условные обозначения. Угловой способ может использовать правильный класс, основанный на правдивости условностей.

Ваш объект должен содержать пары ключ / значение. Ключ - это имя класса, которое будет применяться, когда значение (условное) будет равно true.

```
<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>
```

Угловой будет проверять объект `$scope.user` чтобы увидеть `active` статус и номер `level` . В зависимости от значений этих переменных, Angular будет применять стиль соответствия к `<span>` .

ngIf

`ng-if` - директива, подобная **`ng-show`** но вставляет или удаляет элемент из DOM вместо простого его скрытия. Угловая 1.1.5 введена директива ng-If. Вы можете использовать директиву ng-if выше версий 1.1.5. Это полезно, потому что Angular не будет обрабатывать дайджесты для элементов внутри удаленной `ng-if` уменьшить рабочую нагрузку Angular, особенно для сложных привязок данных.

В отличие от `ng-show` , директива `ng-if` создает дочернюю область, которая использует прототипное наследование. Это означает, что установка примитивного значения в области дочерних объектов не будет применяться к родительскому элементу. Чтобы установить примитив в родительской области, необходимо использовать свойство `$parent` в области дочернего объекта.

JavaScript

```
angular.module('MyApp', []);

angular.module('MyApp').controller('myController', ['$scope', '$window', function
myController($scope, $window) {
    $scope.currentUser= $window.localStorage.getItem('userName');
}]);
```

Посмотреть

```
<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 Если `currentUser` является неопределенным

```
<div ng-controller="myController">
    <div ng-if="currentUser">
        Hello, {{currentUser}}
    </div>
    <!-- ng-if: !currentUser -->
</div>
```

DOM Если `currentUser` не определено

```
<div ng-controller="myController">
    <!-- ng-if: currentUser -->
    <div ng-if="!currentUser">
        <a href="/login">Log In</a>
        <a href="/register">Register</a>
    </div>
</div>
```

Рабочий пример

Функция обещания

Директива ngIf также принимает функции, которые логически требуют возврата true или false.

```
<div ng-if="myFunction()">
    <span>Span text</span>
</div>
```

Текст диапазона появится, только если функция вернет значение true.

```
$scope.myFunction = function() {
    var result = false;
    // Code to determine the boolean value of result
    return result;
};
```

Как любое угловое выражение, функция принимает любые переменные.

ngMouseenter и ngMouseleave

`ng-mouseenter` и `ng-mouseleave` полезны для запуска событий и применения стилизации CSS, когда вы наводите на себя или вне своих элементов DOM.

В директиве `ng-mouseenter` выполняется выражение, которое является одним из событий ввода мыши (когда пользователь вводит указатель мыши над элементом DOM, в котором находится эта директива)

HTML

```
<div ng-mouseenter="applyStyle = true" ng-class="{'active': applyStyle}">
```

В приведенном выше примере, когда пользователь наводит указатель мыши на `div` , `applyStyle` обращается к `true` , что, в свою очередь, применяет класс `.active` CSS в `ng-class` .

Директива `ng-mouseleave` выполняет выражение одно событие выхода мыши (когда пользователь отводит указатель мыши от элемента DOM, в котором находится эта директива)

HTML

```
<div ng-mouseenter="applyStyle = true" ng-mouseleaver="applyStyle = false" ng-
class="{'active': applyStyle}">
```

Повторяя первый пример, теперь, когда пользователь убирает указатель мыши из div, класс `.active` удаляется.

ngDisabled

Эта директива полезна для ограничения входных событий на основе определенных существующих условий.

Директива `ng-disabled` принимает и выражение, которое должно оценивать как правдивые, так и ложные значения.

`ng-disabled` используется для условного применения `disabled` атрибута на `input` элементе.

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` оценивается как true, если длина `input` равна 0, что отключает кнопку, `vm.name.length===0` событие click `ng-click`

ngDblclick

Директива `ng-dblclick` полезна, когда вы хотите привязать событие двойного щелчка к вашим элементам DOM.

Эта директива принимает выражение

HTML

```
<input type="number" ng-model="num = num + 1" ng-init="num=0">

<button ng-dblclick="num++">Double click me</button>
```

В приведенном выше примере значение, удерживаемое на `input` будет увеличиваться при двойном нажатии кнопки.

Встроенные директивы Cheat Sheet

`ng-app` Устанавливает раздел AngularJS.

`ng-init` Устанавливает значение переменной по умолчанию.

`ng-bind` Альтернатива шаблону {{}}.

`ng-bind-template` Привязывает к выражению несколько выражений.

`ng-non-bindable` утверждает, что данные не являются связываемыми.

`ng-bind-html` Связывает внутреннее свойство HTML элемента HTML.

`ng-change` Вычисляет указанное выражение, когда пользователь меняет ввод.

`ng-checked` Устанавливает флажок.

`ng-class` Устанавливает класс css динамически.

`ng-cloak` Предотвращает показ содержимого до тех пор, пока AngularJS не примет управление.

`ng-click` Выполняет метод или выражение при нажатии элемента.

`ng-controller` Присоединяет класс контроллера к представлению.

`ng-disabled` Управляет отключенным свойством элемента формы

`ng-form` Устанавливает форму

`ng-href` Динамически связывать переменные AngularJS с атрибутом href.

`ng-include` Используется для извлечения, компиляции и включения внешнего HTML-фрагмента на вашу страницу.

`ng-if` Удалить или воссоздать элемент в DOM в зависимости от выражения

`ng-switch` Условное управление переключением на основе соответствия выражения.

`ng-model` Привязывает элементы ввода, select, textarea и т. д. с свойством model.

`ng-readonly` Используется для установки атрибута readonly для элемента.

`ng-repeat` Используется для прокрутки каждого элемента в коллекции для создания нового шаблона.

`ng-selected` Используется для установки выбранной опции в элементе.

`ng-show/ng-hide` Показать / скрыть элементы на основе выражения.

`ng-src` Динамически связывать переменные AngularJS с атрибутом src.

`ng-submit` Привязать угловые выражения к событиям onsubmit.

`ng-value` Привяжите угловые выражения к значению.

`ng-required` Привязать угловые выражения к событиям onsubmit.

`ng-style` Устанавливает `ng-style` CSS в HTML-элементе.

`ng-pattern` Добавляет шаблонный шаблон для ngModel.

`ng-maxlength` Добавляет модуль проверки maxlength в ngModel.

`ng-minlength` Добавляет средство проверки длины minlength в ngModel.

`ng-classeven` Работает совместно с ngRepeat и вступает в силу только с нечетными (четными) строками.

`ng-classodd` Работает совместно с ngRepeat и вступает в силу только для нечетных (четных) строк.

`ng-cut` Используется для указания пользовательского поведения на событии cut.

`ng-copy` Используется для указания пользовательского поведения в событии копирования.

`ng-paste` Используется для указания пользовательского поведения при вставке.

`ng-options` Используется для динамического создания списка элементов для элемента.

`ng-list` Используется для преобразования строки в список на основе указанного разделителя.

`ng-open` Используется для установки атрибута open для элемента, если выражение внутри

ngOpen является правдивым.

Источник (отредактирован немного)

ngClick

Директива `ng-click` прикрепляет событие click к элементу DOM.

Директива `ng-click` позволяет указать настраиваемое поведение при нажатии элемента DOM.

Это полезно, когда вы хотите прикреплять события кликов на кнопках и обрабатывать их на контроллере.

Эта директива принимает выражение с объектом событий, доступным как `$event`

HTML

```
<input ng-click="onClick($event)">Click me</input>
```

контроллер

```
.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>
```

контроллер

```
...

$scope.count = function(){
```

```
    $scope.count = $scope.count + 1;
}
...
```

Когда нажимается кнопка, вызов функции `onClick` будет печатать «Hello click event», за которым следует объект события.

ngRequired

`ng-required` добавляют или удаляют `required` атрибут проверки на элемент, который , в свою очередь , будет включать и выключать `require` проверки ключа для `input` .

Он используется для опционального определения того, требуется ли `input` элементу иметь непустое значение. Директива полезна при разработке валидации на сложных HTML-формах.

HTML

```
<input type="checkbox" ng-model="someBooleanValue">
<input type="text" ng-model="username" ng-required="someBooleanValue">
```

нг-модель-опция

`ng-model-options` позволяют изменять поведение `ng-model` по умолчанию, эта директива позволяет регистрировать события, которые будут срабатывать при обновлении ng-модели и приложить эффект отладки.

Эта директива принимает выражение, которое будет оценивать объект определения или ссылку на значение области.

Пример:

```
<input type="text" ng-model="myValue" ng-model-options="{'debounce': 500}">
```

В приведенном выше примере будет добавлен эффект `myValue` 500 миллисекунд на `myValue` , который заставит модель обновить 500 мс после того, как пользователь завершит ввод текста над `input` (то есть, когда `myValue` закончит обновление).

Доступные свойства объекта

1. `updateOn` : указывает, какое событие должно быть привязано к вводу

   ```
   ng-model-options="{ updateOn: 'blur'}" // will update on blur
   ```

2. `debounce` : указывает задержку в несколько миллисекунд к обновлению модели

```
ng-model-options="{'debounce': 500}" // will update the model after 1/2 second
```

3. `allowInvalid` : логический флаг, допускающий недопустимое значение модели, обходя проверку по умолчанию по умолчанию, по умолчанию эти значения будут считаться `undefined` .

4. `getterSetter` : логический флаг, указывающий, следует ли рассматривать `ng-model` как функцию getter / setter вместо значения простой модели. Затем функция запустится и вернет значение модели.

   Пример:

   ```
   <input type="text" ng-model="myFunc" ng-model-options="{'getterSetter': true}">

   $scope.myFunc = function() {return "value";}
   ```

5. `timezone` : определяет часовой пояс для модели, если ввод имеет `date` или `time` . типы

ngCloak

Директива `ngCloak` используется для предотвращения кратковременного отображения браузера в виде необработанной (не скомпилированной) формы, когда приложение загружается. - Просмотр источника

HTML

```
<div ng-cloak>
  <h1>Hello {{ name }}</h1>
</div>
```

`ngCloak` можно применить к элементу body, но предпочтительным применением является применение нескольких директив ngCloak к небольшим частям страницы, чтобы обеспечить прогрессивную визуализацию просмотра браузера.

Директива `ngCloak` не имеет параметров.

См. Также: Предотвращение мерцания

ngInclude

ng-include позволяет делегировать управление одной частью страницы конкретному контроллеру. Вы можете сделать это, потому что сложность этого компонента становится такой, что вы хотите инкапсулировать всю логику в выделенный контроллер.

Пример:

```
  <div ng-include
```

```
      src="'/gridview'"
      ng-controller='gridController as gc'>
  </div>
```

Обратите внимание, что `/gridview` должен обслуживаться веб-сервером как отдельный и законный URL-адрес.

Также обратите внимание, что `src` Attribute принимает угловое выражение. Это может быть вызов переменной или функции, например, или, как в этом примере, строковая константа. В этом случае вам нужно обязательно **обернуть исходный URL в одинарные кавычки** , поэтому он будет оцениваться как строковая константа. Это общий источник путаницы.

Внутри `/gridview` html вы можете ссылаться на `gridController` как если бы он был обернут вокруг страницы, например:

```
<div class="row">
  <button type="button" class="btn btn-default" ng-click="gc.doSomething()"></button>
</div>
```

ngSrc

Использование Угловой разметки, такой как `{{hash}}` в атрибуте src, не работает правильно. Браузер будет извлекать из URL с буквальным текстом `{{hash}}` пока Angular не заменит выражение внутри `{{hash}}` . Директива `ng-src` переопределяет исходный атрибут `src` для элемента тега изображения и решает проблему

```
<div ng-init="pic = 'pic_angular.jpg'">
    <h1>Angular</h1>
    <img ng-src="{{pic}}">
</div>
```

ngPattern

Директива `ng-pattern` принимает выражение, которое оценивает шаблон регулярного выражения и использует этот шаблон для проверки текстового ввода.

Пример:

Допустим, мы хотим, чтобы элемент `<input>` стал действительным, когда его значение (ng-model) является допустимым IP-адресом.

Шаблон:

```
<input type="text" ng-model="ipAddr" ng-pattern="ipRegex" name="ip" required>
```

контроллер:

```
$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

В основном используется в `ng-repeat` ngValue полезно при динамическом создании списков переключателей с помощью ngRepeat

```
<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>
```

Рабочий плнкр

ngCopy

Директива `ngCopy` указывает поведение, которое должно выполняться в событии копирования.

Предотвращение копирования пользователем данных

```
<p ng-copy="blockCopy($event)">This paragraph cannot be copied</p>
```

В контроллере

```
$scope.blockCopy = function(event) {
    event.preventDefault();
    console.log("Copying won't work");
}
```

ngPaste

Директива `ngPaste` определяет пользовательское поведение для запуска, когда пользователь вставляет содержимое

```
<input ng-paste="paste=true" ng-init="paste=false" placeholder='paste here'>
pasted: {{paste}}
```

ngHref

ngHref используется вместо атрибута href, если у нас есть угловые выражения внутри значения href. Директива ngHref переопределяет исходный атрибут href тега html с использованием атрибута href, такого как тег, тег и т. Д.

Директива ngHref гарантирует, что ссылка не будет нарушена, даже если пользователь нажимает ссылку до того, как AngularJS проверит код.

Пример 1

```
<div ng-init="linkValue = 'http://stackoverflow.com'">
    <p>Go to <a ng-href="{{linkValue}}">{{linkValue}}</a>!</p>
</div>
```

Пример 2 Этот пример динамически получает значение href из поля ввода и загружает его как значение href.

```
<input ng-model="value" />
<a id="link" ng-href="{{value}}">link</a>
```

Пример 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

Директива `ng-list` используется для преобразования строки с разделителями из текстового ввода в массив строк или наоборот.

Директива `ng-list` использует разделитель по умолчанию `", "` (запятая).

Вы можете установить разделитель вручную, назначив `ng-list` делителю, подобному этому

`ng-list="; "` .

В этом случае разделитель устанавливается в полуточку с последующим пробелом.

По умолчанию `ng-list` имеет атрибут `ng-trim` которого установлено значение true. `ng-trim` когда false, будет соблюдать пробелы в вашем разделителе. По умолчанию `ng-list` не учитывает пробелы, если вы не установите `ng-trim="false"` .

Пример:

```
angular.module('test', [])
  .controller('ngListExample', ['$scope', function($scope) {
    $scope.list = ['angular', 'is', 'cool!'];
}]);
```

Пользовательский разделитель установлен `;` , И модель окна ввода устанавливается в массив, который был создан в области.

```
  <body ng-app="test" ng-controller="ngListExample">
    <input ng-model="list" ng-list="; " ng-trim="false">
  </body>
```

Поле ввода будет отображаться с содержимым: `angular; is; cool!`

Прочитайте Встроенные директивы онлайн: https://riptutorial.com/ru/angularjs/topic/706/встроенные-директивы

глава 12: дайджест петли прохождение

Синтаксис

- $ scope. $ watch (watchExpression, callback, [deep compare])
- $ Объем. $ Дайджеста ()
- $ Сфера. $ Применять ([exp])

Examples

двусторонняя привязка данных

Угловая магия под капотом. он позволяет связывать DOM с реальными переменными js.

Угловой использует цикл, называемый « *цикл дайджеста* », который вызывается после любого изменения вызывающих переменную вызовов, которые обновляют DOM.

Например, `ng-model` директивы придает `keyup` EventListener к этому входу:

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

Каждый раз, когда срабатывает событие `keyup` , начинается *цикл дайджест* .

В какой-то момент *цикл дайджеста* выполняет повторный вызов, который обновляет содержимое этого диапазона:

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

Основной жизненный цикл этого примера обобщает (очень схематично), как угловые работы ::

1. Угловое сканирование html
    - Директива `ng-model` создает прослушиватель `keyup` на входе
    - `expression` внутри диапазона добавляет обратный вызов *цикла digest*
2. Пользователь взаимодействует со входом
    - прослушиватель `keyup` запускает *цикл дайджеста*
    - *цикл дайджеста* вызывает обратный вызов
    - Содержимое диапазона обратного вызова

$ digest и $ watch

Реализация двухсторонней привязки данных для достижения результата из предыдущего примера может быть выполнена с помощью двух основных функций:

- **$ digest** вызывается после взаимодействия с пользователем (привязка переменной DOM =>)
- **$ watch** устанавливает обратный вызов для вызова после изменения переменной (binding variable => DOM)

Примечание: это пример демонстрации, а не фактический угловой код

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

Нам нужны две функции:

```
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() })
}
```

Теперь мы теперь можем использовать эти функции, чтобы подключить переменную к DOM (угловой поставляется со встроенными директивами, которые сделают это для вас):

```
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;
});
```

Вне курса, реальные реализации сложнее и поддерживают такие параметры, как **какой элемент** привязать и **какую переменную** использовать

Пример работы можно найти здесь: https://jsfiddle.net/azofxd4j/

дерево $ scope

Предыдущий пример достаточно хорош, если нам нужно связать один элемент html с одной переменной.

В действительности - нам нужно связать многие элементы со многими переменными:

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

Этот `ng-repeat` связывает 5 элементов с 5 переменными, называемыми `number` , с другим значением для каждого из них!

Способ углового достижения такого поведения заключается в использовании отдельного контекста для каждого элемента, который нуждается в отдельных переменных. Этот контекст называется областью.

Каждая область содержит свойства, которые являются переменными, связанными с DOM, а функции `$digest` и `$watch` реализованы как методы области.

DOM - это дерево, а переменные должны использоваться на разных уровнях дерева:

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

Но, как мы видели, контекст (или область) переменных внутри `ng-repeat` отличается от контекста над ним. Чтобы решить эту проблему - угловые орудия применения как дерево.

Каждая область имеет массив дочерних элементов, а вызов метода `$digest` будет запускать весь метод `$digest` своего `$digest` метода.

Таким образом - после изменения ввода - `$digest` вызывается для области div, которая затем запускает `$digest` для своих 5 детей - который обновит его содержимое.

Простая реализация для области видимости может выглядеть так:

```
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() })
}
```

Примечание: это пример демонстрации, а не фактический угловой код

Прочитайте дайджест петли прохождение онлайн:
https://riptutorial.com/ru/angularjs/topic/3156/дайджест-петли-прохождение

глава 13: Декораторы

Синтаксис

- декоратор (имя, декоратор);

замечания

> **Decorator** - это функция, которая позволяет модифицировать сервис , фабрику , директиву или фильтр до его использования. Decorator используется для переопределения или изменения поведения сервиса. Возвращаемое значение функции декоратора может быть исходной услугой или новой службой, которая заменяет или обертывает и делегирует исходную услугу.

Любое украшение ***должно быть*** выполнено в фазе `config` углового приложения, введя `$provide` и используя функцию `$provide.decorator` .

> Функция декоратора имеет объект `$delegate` вводится для обеспечения доступа к службе, которая соответствует селектору в декораторе. Этот `$delegate` будет сервисом, который вы украшаете. Возвращаемое значение функции, предоставляемой декоратору, будет иметь место при оформлении услуги, директивы или фильтра.

Следует рассмотреть использование декоратора только в том случае, если какой-либо другой подход не подходит или окажется слишком утомительным. Если большое приложение использует тот же сервис, а одна часть изменяет поведение службы, легко создать путаницу и / или ошибки в этом процессе.

Типичным вариантом использования будет тот факт, что у вас есть зависимая от третьей стороны, которую вы не можете обновить, но вам нужно немного поработать или расширить ее.

Examples

Украсить сервис, завод

Ниже приведен пример декоратора сервисов, переопределяющий `null` дату, возвращаемую службой.

```
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

Указать директиву

Директивы могут быть оформлены так же, как и сервисы, и мы можем модифицировать или заменять любые его функциональные возможности. Обратите внимание, что сама директива доступна в позиции 0 в $ delegate array, а параметр name в декораторе должен содержать суффикс `Directive` (с учетом регистра).

Итак, если директива называется `myDate` , ее можно получить с помощью `myDateDirective` используя `$delegate[0]` .

Ниже приведен простой пример, когда директива показывает текущее время. Мы украсим его, чтобы обновить текущее время за одну секунду. Без украшения он всегда будет показывать одно и то же время.

```
<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

Украсить фильтр

При декорировании фильтров параметр имени должен включать в себя `Filter` суффикс (с учетом регистра). Если фильтр называется `repeat` , параметр декоратора - `repeatFilter` . Ниже мы будем декорировать пользовательский фильтр, который повторяет любую заданную строку *n* раз, чтобы результат был обратным. Вы также можете украшать встроенные фильтры углов одинаково, хотя и не рекомендуется, так как это может повлиять на функциональность фреймворка.

```
<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

Прочитайте Декораторы онлайн: https://riptutorial.com/ru/angularjs/topic/5255/декораторы

глава 14: директива ng-class

Examples

Три типа выражений ng-класса

Угловой поддерживает три типа выражений в директиве `ng-class` .

1. Строка

```
<span ng-class="MyClass">Sample Text</span>
```

Указание выражения, которое оценивается в строке, указывает, что Angular обрабатывает его как переменную $ scope. Угловое будет проверять масштаб $ и искать переменную под названием «MyClass». Любой текст, содержащийся в «MyClass», станет фактическим именем класса, которое применяется к этому `<span>` . Вы можете указать несколько классов, разделив каждый класс на пробел.

В контроллере у вас может быть определение, которое выглядит так:

```
$scope.MyClass = "bold-red deleted error";
```

Угловой будет оценивать выражение `MyClass` и находить определение $ scope. Он применит три класса «полужирный», «удаленный» и «ошибка» к элементу `<span>` .

Указание классов таким образом позволяет легко изменить определения классов в вашем контроллере. Например, вам может потребоваться изменить класс на основе других пользовательских взаимодействий или новых данных, загружаемых с сервера. Кроме того, если у вас есть много выражений для оценки, вы можете сделать это в функции, которая определяет окончательный список классов в переменной `$scope` . Это может быть проще, чем пытаться сжать многие оценки в атрибут `ng-class` в вашем HTML-шаблоне.

---

2. Объект

Это наиболее часто используемый способ определения классов с использованием `ng-class` поскольку он легко позволяет вам определять оценки, определяющие, какой класс использовать.

Укажите объект, содержащий пары ключ-значение. Ключ - это имя класса, которое будет применяться, если значение (условное) оценивается как истинное.

```
<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. Массив

Выражение, которое оценивает массив, позволяет использовать комбинацию **строк** (см. № 1 выше) и **условные объекты** (# 2 выше).

```
<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>
```

Это создает поле ввода текста, привязанное к переменной области видимости `UserStyle` которая позволяет пользователю вводить имена (имена) любого класса. Они будут динамически применяться к элементу `<p>` типу пользователя. Кроме того, пользователь может щелкнуть по флажку, привязанным к данным, к переменной области `warning` . Это также будет динамически применяться к элементу `<p>` .

Прочитайте директива ng-class онлайн: https://riptutorial.com/ru/angularjs/topic/2395/директива-ng-class

глава 15: Директивы, использующие ngModelController

Examples

Простой контроль: рейтинг

Давайте построим простой элемент управления, виджет рейтинга, предназначенный для использования как:

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

На данный момент нет модного CSS; это будет выглядеть так:

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

Нажатие на номер выбирает этот рейтинг; и нажатие «x» устанавливает рейтинг равным нулю.

```
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>'
    };
});
```

Ключевые моменты:

1. Реализовать `ngModel.$render` для передачи *значения просмотра* модели на ваш взгляд.
2. Вызовите `ngModel.$setViewValue()` когда вы чувствуете, что значение представления

должно быть обновлено.
3. Разумеется, управление может быть параметризовано; используйте привязки `'<'` для параметров, если в Angular> = 1.5 четко указать ввод - одностороннее связывание. Если вам нужно предпринимать действия всякий раз, когда изменяется параметр, вы можете использовать свойство JavaScript (см. `Object.defineProperty()`), чтобы сохранить несколько часов.

Примечание 1: Чтобы не усложнять реализацию, значения рейтинга вставляются в массив - `ctrl.options` . Это не нужно; более эффективная, но также более сложная реализация может использовать DOM-манипуляцию для вставки / удаления рейтингов при изменении `min` / `max` .

Примечание 2: За исключением привязок `'<'` scope, этот пример можно использовать в Angular <1.5. Если вы находитесь на Angular> = 1.5, было бы неплохо преобразовать это в компонент и использовать крюк жизненного цикла `$onInit()` для инициализации `min` и `max` вместо этого в конструкторе контроллера.

И необходимая скрипка: https://jsfiddle.net/h81mgxma/

Несколько сложных элементов управления: отредактируйте полный объект

Пользовательский элемент управления не должен ограничиваться тривиальными вещами, такими как примитивы; он может редактировать более интересные вещи. Здесь мы представляем два типа пользовательских элементов управления: один для редактирования лиц и один для редактирования адресов. Управление адресом используется для редактирования адреса человека. Примером использования может быть:

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

Модель для этого примера преднамеренно упрощена:

```
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;
}
```

Редактор адресов:

```
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>'
    };
});
```

Ключевые моменты:

1. Мы редактируем объект; мы не хотим напрямую изменять объект, данный нам от нашего родителя (мы хотим, чтобы наша модель была совместима с принципом непреложности). Поэтому мы создаем мелкие часы на редактируемом объекте и обновляем модель с помощью `$setViewValue()` всякий раз, когда изменяется свойство. Мы передаем *копию* нашему родителю.
2. Всякий раз, когда модель изменяется извне, мы копируем ее и сохраняем копию в нашей области. Принципы неизменности снова, хотя внутренняя копия не является неизменной, внешний может быть очень хорош. Кроме того, мы восстанавливаем часы (`this._unwatch();this._makeWatch();`), чтобы не запускать наблюдателя за изменениями, внесенными нами моделью. (Мы хотим, чтобы часы запускались для изменений, внесенных в пользовательский интерфейс.)
3. Другое, что выше, мы реализуем `ngModel.$render()` и вызываем `ngModel.$setViewValue()` как и для простого `ngModel.$setViewValue()` управления (см. Пример оценки).

Код для персонализированного элемента управления практически идентичен. Шаблон использует `<input-address>` . В более продвинутой реализации мы могли бы извлечь контроллеры в модуль многократного использования.

```
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>'
    };
});
```

Примечание. Здесь набираются объекты, т. Е. У них есть соответствующие конструкторы. Это не обязательно; модель может быть простой JSON-объектами. В этом случае просто используйте `angular.copy()` вместо конструкторов. Дополнительным преимуществом является то, что контроллер становится идентичным для двух элементов управления и может быть легко извлечен в какой-то общий модуль.

Скрипка: https://jsfiddle.net/3tzyqfko/2/

Две версии скрипта извлекли общий код контроллеров: https://jsfiddle.net/agj4cp0e/ и https://jsfiddle.net/ugb6Lw8b/

Прочитайте Директивы, использующие ngModelController онлайн: https://riptutorial.com/ru/angularjs/topic/2438/директивы--использующие-ngmodelcontroller

глава 16: Единичные тесты

замечания

В этом разделе приводятся примеры модульного тестирования различных конструкций в AngularJS. Модульные тесты часто пишутся с использованием Jasmine , популярной платформы тестирования, основанной на поведении. При модульном тестировании угловых конструкций вам нужно будет включить ngMock в качестве зависимости при выполнении модульных тестов.

Examples

Единичный тест фильтра

Фильтр:

```
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;
  }
});
```

Тест:

```
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();
  });
});
```

Бежать!

Примечание. В вызове `inject` в тесте ваш фильтр должен быть указан по имени + *Фильтр* . Причиной этого является то, что всякий раз, когда вы регистрируете фильтр для своего модуля, Angular зарегистрируйте его с помощью `Filter` прикрепленного к его имени.

Единичное тестирование компонента (1,5+)

Код компонента:

```
angular.module('myModule', []).component('myComponent', {
  bindings: {
    myValue: '<'
  },
  controller: function(MyService) {
    this.service = MyService;
    this.componentMethod = function() {
      return 2;
    };
  }
});
```

Тест:

```
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);
  });
});
```

Бежать!

Единичный тест контроллера

Код контроллера:

```
angular.module('myModule', [])
  .controller('myController', function($scope) {
    $scope.num = 2;
    $scope.doSomething = function() {
      $scope.num += 2;
    }
  });
```

Тест:

```
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);
  });
});
```

Бежать!

Единичное тестирование услуги

Код услуги

```
angular.module('myModule', [])
  .service('myService', function() {
    this.doSomething = function(someNumber) {
      return someNumber + 2;
    }
  });
```

Тест

```
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);
  });
});
```

Бежать!

Единичное тестирование директивы

Код директивы

```
angular.module('myModule', [])
  .directive('myDirective', function() {
    return {
      template: '<div>{{greeting}} {{name}}!</div>',
      scope: {
        name: '=',
        greeting: '@'
      }
    };
  });
```

Тест

```
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");
  });
});
```

Бежать!

Прочитайте Единичные тесты онлайн: https://riptutorial.com/ru/angularjs/topic/1689/единичные-тесты

глава 17: Использование AngularJS с TypeScript

Синтаксис

- $ scope: ng.IScope - это путь в машинописном файле для определения типа для конкретной переменной.

Examples

Угловые контроллеры в машинописном машиностроении

Как определено в документации AngularJS

> Когда контроллер подключается к DOM через директиву ng-controller, Angular будет создавать экземпляр нового объекта Controller, используя указанную конструкторскую функцию контроллера. Новая дочерняя область будет создана и доступна как параметр для инъекции функции-конструктора контроллера как $ scope.

Контроллеры могут быть легко сделаны с использованием классов типов.

```
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);
        }
    }
}
```

Результат Javascript

```
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
```

После создания класса контроллера пусть угловой модуль js о контроллере можно сделать простым, используя класс

```
app
 .module('app')
 .controller('exampleController', App.Controller.SampleController)
```

Использование контроллера с синтаксисом ControllerAs

Контроллер, который мы создали, может быть создан и использован с использованием `controller as` синтаксиса. Это потому, что мы поместили переменную непосредственно в класс контроллера, а не в `$scope` .

Использование `controller as someName` - отделить контроллер от самой `$scope` Таким образом, нет необходимости вводить $ scope в качестве зависимости в контроллере.

Традиционный способ:

```
// we are using $scope object.
app.controller('MyCtrl', function ($scope) {
  $scope.name = 'John';
});

<div ng-controller="MyCtrl">
  {{name}}
</div>
```

Теперь с `controller as` синтаксиса :

```
// 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>
```

Если вы создаете экземпляр класса в JavaScript, вы можете сделать это:

```
var jsClass = function () {
  this.name = 'John';
}
var jsObj = new jsClass();
```

Итак, теперь мы можем использовать экземпляр `jsObj` для доступа к любому методу или свойству `jsClass` .

В угловом мы делаем тот же тип вещи. Мы используем контроллер как синтаксис для создания экземпляра.

Использование Bundling / Minification

Способ ввода $ scope в функции конструктора контроллера - это способ продемонстрировать и использовать базовый вариант инъекции угловой зависимости, но не готов к производству, поскольку он не может быть минимизирован. Это потому, что система минимизации изменяет имена переменных и инъекции зависимостей anguar использует имена параметров, чтобы знать, что нужно вводить. Так, например, функция конструктора ExampleController минимизируется следующим кодом.

```
function n(n){this.setUpWatches(n)
```

и `$scope` изменен на `n` !
Чтобы преодолеть это, мы можем добавить массив $ inject (`string[]`). Таким образом, DI интеллектуального сигнала знает, что нужно вводить, в каком положении находится функция конструктора контроллеров.
Таким образом, приведенные выше машинописные изменения

```
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);
        }
    }
}
```

Почему ControllerAs синтаксис?

Функция контроллера

Функция контроллера - это всего лишь функция конструктора JavaScript. Следовательно, когда представление загружает `function context` (**`this`**), устанавливается объект контроллера.

Случай 1 :

```
this.constFunction = function() { ... }
```

Он создается в `controller object` , а не в `$scope` . представления не могут получить доступ к функциям, определенным на объекте контроллера.

Пример :

```
<a href="#123" ng-click="constFunction()"></a> // It will not work
```

Случай 2:

```
$scope.scopeFunction = function() { ... }
```

Он создается в `$scope object` , а не на `controller object` . представления могут обращаться только к функциям, определенным на объекте `$scope` .

Пример :

```
<a href="#123" ng-click="scopeFunction()"></a> // It will work
```

Почему контроллеры?

- Синтаксис **`ControllerAs`** делает его более ясным, когда манипулируют `oneCtrl.name` `anotherCtrl.name` `oneCtrl.name` и `anotherCtrl.name` значительно упрощают идентификацию того, что у вас есть `name` назначенное несколькими разными контроллерами для разных целей, но если оба используются одинаковым `$scope.name` и имеют два разных HTML-элемента на странице, которые оба связаны с **`{{name}}`** тогда трудно определить, какой из них является контроллером.

- Скрытие области `$scope` и отображение элементов с контроллера на представление через `intermediary object` . Установив `this.*` , Мы можем разоблачить только то, что хотим отобразить с контроллера на представление.

```
<div ng-controller="FirstCtrl">
    {{ name }}
    <div ng-controller="SecondCtrl">
        {{ name }}
        <div ng-controller="ThirdCtrl">
            {{ name }}
        </div>
    </div>
</div>
```

Здесь, в приведенном выше случае `{{ name }}` будет очень запутанным для использования, и мы также не знаем, какой из них связан с каким контроллером.

```
<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>
```

Почему $ scope?

- Используйте `$scope` когда вам нужно получить доступ к одному или нескольким методам $ scope, таким как `$watch` , `$digest` , `$emit` , `$http` и т. Д.
- ограничивать, какие свойства и / или методы подвергаются переменной `$scope` , а затем явно передавать их в `$scope` мере необходимости.

Прочитайте Использование AngularJS с TypeScript онлайн:
https://riptutorial.com/ru/angularjs/topic/3477/использование-angularjs-c-typescript

глава 18: Использование встроенных директив

Examples

Скрыть / Показать элементы HTML

В этом примере скрывают элементы html show.

```
<!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>
```

Демо-версия

Пошаговое объяснение:

1. `ng-app="myDemoApp"` , директива ngApp сообщает угловой, что элемент DOM управляется определенным угловым модулем с именем myDemoApp.
2. `<script src="[//angular include]">` включают угловые js.
3. Функция `HideShowController` определена с другой функцией с именем `toggle` которая помогает скрыть отображение элемента.
4. `angular.module(...)` создает новый модуль.
5. `.controller(...)` Angular Controller и возвращает модуль для цепочки;
6. Директива `ng-controller` является ключевым аспектом того, как угловые поддерживают принципы, лежащие в основе шаблона проектирования Model-View-

Controller.

7. Директива `ng-show` показывает данный HTML-элемент, если выражение указано верно.
8. Директива `ng-hide` скрывает данный HTML-элемент, если выражение указано верно.
9. Директива `ng-click` запускает функцию переключения внутри контроллера

Прочитайте Использование встроенных директив онлайн:
https://riptutorial.com/ru/angularjs/topic/7644/использование-встроенных-директив

глава 19: Как работает привязка данных

замечания

Таким образом, хотя эта концепция привязки данных в целом проста в разработке, она довольно тяжелая для браузера, поскольку Angular слушает каждое событие и запускает цикл Digest. Из-за этого, всякий раз, когда мы присоединяем некоторую модель к представлению, убедитесь, что Scope максимально оптимизирован

Examples

Пример привязки данных

```
<p ng-bind="message"></p>
```

Это «сообщение» должно быть привязано к области контроллера текущего элемента.

```
$scope.message = "Hello World";
```

В более поздний момент, даже если модель сообщения обновлена, это обновленное значение отражается в элементе HTML. Когда угловые компиляции, шаблон «Hello World» будет прикреплен к внутреннемуHTML текущего мира. Угловая поддерживает механизм наблюдения всех директив, направленных на представление. Он имеет механизм Digest Cycle, в котором он выполняет итерацию через массив Watchers, он обновит элемент DOM, если произойдет изменение предыдущего значения модели.

Периодически не проверяется, есть ли какие-либо изменения в Объектах, прикрепленных к нему. Наблюдаются не все объекты, привязанные к области. Сфера действия прототипически поддерживает **$$ WatchersArray** . Scope выполняет только итерацию через этот WatchersArray, когда вызывается $ digest.

Угловая добавляет наблюдателя к WatchersArray для каждого из этих

1. {{expression}} - В ваших шаблонах (и где-нибудь еще, где есть выражение) или когда мы определяем ng-model.
2. $ scope. $ watch ('expression / function'). В вашем JavaScript мы можем просто прикрепить объект области для углового просмотра.

Функция **$ watch** принимает три параметра:

1. Первая - это функция наблюдателя, которая просто возвращает объект, или мы можем просто добавить выражение.

2. Вторая - это функция прослушивателя, которая будет вызываться при изменении объекта. В этой функции будут реализованы все такие вещи, как изменения DOM.

3. Третий - необязательный параметр, который принимает логическое значение. Если его истинная, угловая глубокая наблюдает за объектом, и если его ложный угловой просто делает ссылку на объект. Грубая реализация $ watch выглядит так:

```
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
};
```

В Angular называется Digest Cycle. Цикл $ digest начинается в результате вызова $ scope. $ Digest (). Предположим, что вы изменили модель $ scope в функции обработчика с помощью директивы ng-click. В этом случае AngularJS автоматически запускает цикл $ digest, вызывая $ digest (). В дополнение к ng-click есть несколько других встроенных директив / служб, которые позволяют вам изменять модели (например, ng-model, $ timeout и т. Д.), и автоматически запускает цикл $ digest. Грубая реализация $ digest выглядит так.

```
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;
 };
```

Если мы используем функцию **setTimeout ()** JavaScript для обновления модели области, у Angular нет возможности узнать, что вы можете изменить. В этом случае мы несете ответственность за вызов $ apply () вручную, который запускает цикл $ digest. Аналогично,

если у вас есть директива, которая устанавливает прослушиватель событий DOM и меняет некоторые модели внутри функции обработчика, вам необходимо вызвать $ apply (), чтобы гарантировать, что изменения вступят в силу. Большая идея $ apply заключается в том, что мы можем выполнить некоторый код, который не знает Angular, этот код все еще может изменить ситуацию в области. Если мы применим этот код в $ apply, он позаботится о вызове $ digest (). Грубая реализация $ apply ().

```
Scope.prototype.$apply = function(expr) {
       try {
         return this.$eval(expr); //Evaluating code in the context of Scope
       } finally {
         this.$digest();
       }
};
```

Прочитайте Как работает привязка данных онлайн:
https://riptutorial.com/ru/angularjs/topic/2342/как-работает-привязка-данных

глава 20: Компоненты

параметры

параметр	подробности
знак равно	Для использования двусторонней привязки данных. Это означает, что если вы обновите эту переменную в области вашего компонента, это изменение будет отражено в родительской области.
<	Односторонние привязки, когда мы просто хотим прочитать значение из родительской области и не обновлять его.
@	Строковые параметры.
&	Для обратных вызовов в случае, если ваш компонент должен вывести что-то в свою родительскую область.
-	-
Крючки LifeCycle	**Подробности** (требует угловой.версии> = 1.5.3)
$ OnInit ()	Вызывается на каждом контроллере после того, как все контроллеры на элементе были созданы и инициализированы их привязки. Это хорошее место для ввода кода инициализации для вашего контроллера.
$ onChanges (changesObj)	Вызывается каждый раз, когда обновляются односторонние привязки. `changesObj` - хэш, ключи которого являются именами связанных свойств, которые изменились, а значения являются объектом формы `{ currentValue, previousValue, isFirstChange() }`.
$ OnDestroy ()	Вызывается контроллером, когда его сдерживающая область уничтожается. Используйте этот крючок для освобождения внешних ресурсов, часов и обработчиков событий.
$ postLink ()	Вызывается после того, как элемент этого контроллера и его дочерние элементы связаны. Этот крюк можно считать аналогичным крючкам ngAfterViewInit и ngAfterContentInit в Angular 2.
$ doCheck ()	Вызывается каждый ход цикла дайджеста. Предоставляет возможность обнаруживать изменения и вносить изменения в них. Любые действия, которые вы хотите предпринять в ответ на

параметр	подробности
	обнаруженные вами изменения, должны быть вызваны из этого крючка; реализация этого не влияет на то, когда вызывается $ onChanges.

замечания

Компонент - это особый вид директивы, который использует более простую конфигурацию, которая подходит для структуры приложения на основе компонентов. Компоненты были введены в Angular 1.5, примеры в этом разделе **не будут работать** со старыми версиями AngularJS.

Полное руководство разработчика о компонентах доступно на https://docs.angularjs.org/guide/component

Examples

Основные компоненты и крючки жизненного цикла

Что такое компонент?

- Компонент в основном представляет собой директиву, которая использует более простую конфигурацию и подходит для архитектуры на основе компонентов, что и есть Угловая 2. Подумайте о компоненте в виде виджета: фрагмент кода HTML, который можно повторно использовать в нескольких разных местах вашего веб-приложения.

Составная часть

```
angular.module('myApp', [])
    .component('helloWorld', {
        template: '<span>Hello World!</span>'
    });
```

наценка

```
<div ng-app="myApp">
  <hello-world> </hello-world>
</div>
```

Демо-версия

Использование внешних данных в компоненте:

Мы могли бы добавить параметр для передачи имени нашему компоненту, который будет использоваться следующим образом:

```
angular.module("myApp", [])
  .component("helloWorld",{
    template: '<span>Hello {{$ctrl.name}}!</span>',
    bindings: { name: '@' }
  });
```

наценка

```
<div ng-app="myApp">
  <hello-world name="'John'" > </hello-world>
</div>
```

Демо-версия

Использование контроллеров в компонентах

Давайте посмотрим, как добавить к нему контроллер.

```
angular.module("myApp", [])
  .component("helloWorld",{
      template: "Hello {{$ctrl.name}}, I'm {{$ctrl.myName}}!",
      bindings: { name: '@' },
      controller: function(){
        this.myName = 'Alain';
      }
  });
```

наценка

```
<div ng-app="myApp">
  <hello-world name="John"> </hello-world>
</div>
```

Демо-версия CodePen

Параметры, переданные компоненту, доступны в области контроллера непосредственно перед тем, как его функция `$onInit` Angular. Рассмотрим этот пример:

```
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;
        }
      }
  });
```

В шаблоне сверху это отобразит «Привет, Джон, я MacJohn!».

Обратите внимание, что `$ctrl` - значение Angular по умолчанию для `controllerAs` если оно не указано.

Демо-версия

Использование «require» в качестве объекта

В некоторых случаях вам может потребоваться доступ к данным из родительского компонента внутри вашего компонента.

Этого можно добиться, указав, что наш компонент требует этого родительского компонента. Требование даст нам ссылку на требуемый компонентный контроллер, который затем может быть использован в нашем контроллере, как показано в следующем примере:

> Обратите внимание, что требуемые контроллеры гарантированно будут готовы только после крючка $ onInit.

```
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();
        }

      }
  });
```

Однако имейте в виду, что это создает плотную связь между ребенком и родителем.

Компоненты в угловом JS

Компоненты в angularJS могут быть визуализированы как настраиваемая директива (<html>

this в директиве HTML, и что-то вроде этого будет настраиваемой директивой < ANYTHING>). Компонент содержит представление и контроллер. Контроллер содержит бизнес-логику, привязанную к представлению, которое видит пользователь. Компонент отличается от угловой директивы, поскольку он содержит меньшую конфигурацию. Угловой компонент можно определить следующим образом.

```
angular.module("myApp",[]).component("customer", {})
```

Компоненты определены на угловых модулях. Они содержат два аргумента: один - это имя компонента, а второй - это объект, который содержит пару значений ключа, которая определяет, какой вид и какой контроллер он будет использовать следующим образом.

```
angular.module("myApp",[]).component("customer", {
    templateUrl : "customer.html", // your view here
    controller: customerController, //your controller here
    controllerAs: "cust"        //alternate name for your controller
})
```

«myApp» - это имя приложения, которое мы создаем, а клиент - это имя нашего компонента. Теперь для вызова его в основном html-файле мы просто поместим его вот так:

```
<customer></customer>
```

Теперь эта директива будет заменена указанным вами представлением и бизнес-логикой, которую вы написали в своем контроллере.

ПРИМЕЧАНИЕ. Помните, что компонент принимает объект как второй аргумент, а директива принимает фабричную функцию в качестве аргумента.

Прочитайте Компоненты онлайн: https://riptutorial.com/ru/angularjs/topic/892/компоненты

глава 21: Константы

замечания

ВЕРСИЯ Ваша постоянная : постоянная записи в капитале - это общая передовая практика, используемая на многих языках. Также полезно четко определить характер вводимых элементов:

Когда вы видите `.controller('MyController', function($scope, Profile, EVENT))` , вы сразу же знаете, что:

- `$scope` - угловой элемент
- `Profile` - это пользовательский сервис или завод
- `EVENT` - угловая постоянная

Examples

Создайте свою первую константу

```
angular
  .module('MyApp', [])
  .constant('VERSION', 1.0);
```

Теперь ваша константа объявлена и может быть введена в контроллер, службу, фабрику, поставщика и даже в режиме конфигурации:

```
angular
  .module('MyApp')
  .controller('FooterController', function(VERSION) {
    this.version = VERSION;
  });
```

```
<footer ng-controller="FooterController as Footer">{{ Footer.version }}</footer>
```

Случаи применения

Здесь нет революции, но угловая константа может быть полезна специально, когда ваше приложение и / или команда начнет расти ... или если вы просто любите писать красивый код!

- **Код рефакторинга.** Пример с именами событий. Если вы используете много событий в своем приложении, вы каждый раз называете имена событий. А, когда новый разработчик присоединяется к вашей команде, он называет свои события другим

синтаксисом ... Вы можете легко предотвратить это, объединив имена вашего события в константе:

```
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() {
      ...
    });
  })
```

... и теперь имена ваших событий могут воспользоваться преимуществами автозаполнения!

- **Определите конфигурацию.** Найдите всю конфигурацию в одном месте:

```
angular
  .module('MyApp')
  .constant('CONFIG', {
    BASE_URL: {
      APP: 'http://localhost:3000',
      API: 'http://localhost:3001'
    },
    STORAGE: 'S3',
    ...
  });
```

- **Изолировать детали.** Иногда есть некоторые вещи, которые вы не очень гордитесь ... например, как жестко заданное значение. Вместо того, чтобы позволить им в вашем основном коде, вы можете создать угловую константу

```
angular
  .module('MyApp')
  .constant('HARDCODED', {
    KEY: 'KEY',
    RELATION: 'has_many',
    VAT: 19.6
  });
```

... и реорганизовать что-то вроде

```
$scope.settings = {
  username: Profile.username,
```

```
  relation: 'has_many',
  vat: 19.6
}
```

В

```
$scope.settings = {
  username: Profile.username,
  relation: HARDCODED.RELATION,
  vat: HARDCODED.VAT
}
```

Прочитайте Константы онлайн: https://riptutorial.com/ru/angularjs/topic/3967/константы

глава 22: Контроллеры

Синтаксис

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

Examples

Ваш первый контроллер

Контроллер - это базовая структура, используемая в Angular для сохранения области и обработки определенных действий внутри страницы. Каждый контроллер соединен с HTML-представлением.

Ниже приведен базовый шаблон для приложения «Угловое»:

```
<!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>
```

Здесь есть несколько вещей:

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

Настройка имени приложения с помощью `ng-app` позволяет вам получить доступ к приложению во внешнем файле Javascript, который будет рассмотрен ниже.

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

Нам понадобится файл Javascript, в котором вы определяете свои контроллеры и их действия / данные.

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

Атрибут `ng-controller` устанавливает контроллер для этого элемента DOM и всех элементов, которые являются дочерними (рекурсивно) под ним.

У вас может быть несколько одинаковых контроллеров (в данном случае `MyController`), говоря `... as mc` , мы предоставляем этому экземпляру контроллера псевдоним.

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

Обозначение `{{ ... }}` - это угловое выражение. В этом случае этот внутренний текст этого элемента `<h1>` будет установлен независимо от значения `mc.title` .

Примечание. Угловое использование двухсторонней привязки данных, что означает, что независимо от того, как вы обновляете значение `mc.title` , оно будет отображаться как на контроллере, так и на странице.

Также обратите внимание, что угловые выражения *не* должны ссылаться на контроллер. Угловое выражение может быть таким же простым, как `{{ 1 + 2 }}` или `{{ "Hello " + "World" }}` .

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

`ng-click` является Угловой директивой, в этом случае связывания события щелчка на кнопку , чтобы вызвать `clicked()` функцию `MyController` экземпляра.

Имея это в виду, давайте напишем реализацию контроллера `MyController` . В приведенном выше примере вы должны написать этот код в `js/controller.js` .

Во-первых, вам нужно создать экземпляр приложения Angular в вашем Javascript.

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

Обратите внимание, что имя, которое мы передаем здесь, совпадает с именем, указанным вами в вашем HTML, с директивой `ng-app` .

Теперь, когда у нас есть объект приложения, мы можем использовать его для создания контроллеров.

```
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()");
    };
});
```

Примечание. Для всего, что мы хотим быть частью экземпляра контроллера, мы используем `this` ключевое слово.

Это все, что требуется для создания простого контроллера.

Создание контроллеров

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

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

    /* Your code below */
}
```

Примечание. В результате `.$inject` убедитесь, что ваши зависимости не будут скремблированы после минимизации. Кроме того, убедитесь, что это соответствует названной функции.

Создание контроллеров, защита от угроз

Существует несколько способов защитить создание контроллера от минимизации.

Первая называется аннотацией встроенного массива. Это выглядит так:

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

Второй параметр метода контроллера может принимать массив зависимостей. Как вы можете видеть , я определил `$scope` и `$http` , которые должны соответствовать параметрам функции контроллера , в котором будет являться `a` `$scope` , и `b` будет `$http` . Обратите внимание, что последний элемент массива должен быть вашей функцией контроллера.

Второй вариант использует свойство `$inject` . Это выглядит так:

```
var app = angular.module('app');
app.controller('sampleController', sampleController);
sampleController.$inject = ['$scope', '$http'];
```

```
function sampleController(a, b) {
    //logic here
}
```

Это делает то же самое, что и встроенное аннотирование массива, но предоставляет другой стиль для тех, кто предпочитает один вариант над другим.

Порядок вложенных зависимостей важен

При вводе зависимостей с использованием формы массива убедитесь, что список зависимостей соответствует соответствующему списку аргументов, переданных функции контроллера.

Обратите внимание, что в следующем примере `$scope` и `$http` отменены. Это вызовет проблему в коде.

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

Использование контроллеров в Угловой JS

В Angular `$scope` - это клей между контроллером и представлением, который помогает со всеми нашими требованиями к привязке данных. Контроллер. Как другой способ привязки контроллера и просмотра и в основном рекомендуется использовать. В основном это две конструкции контроллера в Angular (т.е. $ scope и Controller As).

Различные способы использования контроллера Как есть -

controllerAs View Синтаксис

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

controllerAs Синтаксис контроллера

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

controllerAs с vm

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

```
    vm.name = {};
    vm.sendMessage = function() { };
}
```

`controllerAs` является синтаксическим сахаром над `$scope` . Вы все еще можете связываться с видом и по- прежнему получить доступ к `$scope` методов. Использование `controllerAs` - одна из лучших практик, предложенная группой углового ядра. Есть много причин для этого, немногие из них -

- `$scope` представляет элементы из контроллера в представление через посреднический объект. Установив `this.*` , Мы можем разоблачить только то, что хотим отобразить с контроллера на представление. Он также соответствует стандартным способам использования JavaScript в JavaScript.
- используя синтаксис `controllerAs` , у нас есть более читаемый код, а родительское свойство можно получить, используя псевдоним родительского контроллера вместо использования синтаксиса `$parent` .
- Это способствует использованию привязки к «пунктируемому» объекту в представлении (например, customer.name вместо имени), который является более контекстуальным, более простым для чтения и позволяет избежать каких-либо проблем с исходными данными, которые могут возникнуть без «точечного».
- Помогает избежать использования `$parent` вызовов в Views с вложенными контроллерами.
- Используйте переменную захвата для этого при использовании синтаксиса `controllerAs` . Выберите имя постоянной переменной, например `vm` , что означает ViewModel. Поскольку `this` ключевое слово является контекстуальным и при использовании внутри функции внутри контроллера может изменить свой контекст. Захват контекста этого позволяет избежать этой проблемы.

ПРИМЕЧАНИЕ. Использование синтаксиса `controllerAs` добавляет ссылку на текущую область действия на текущий контроллер, поэтому она доступна как поле

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

`vm` доступен как `$scope.vm` .

Создание интеллектуальных угловых контроллеров

Чтобы создать интеллектуальные угловые контроллеры, вы измените параметры функции `controller` .

Второй аргумент функции `module.controller` должен быть передан **массивом** , где **последним параметром** является **функция контроллера** , и каждый параметр перед этим

является **именем** каждого введенного значения.

Это отличается от обычной парадигмы; который выполняет **функцию контроллера** с введенными аргументами.

Дано:

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

Контроллер должен выглядеть так:

```
app.controller('ctrlInject',
    [
        /* Injected Parameters */
        '$Injectable1',
        '$Injectable2',
        /* Controller Function */
        function($injectable1Instance, $injectable2Instance) {
            /* Controller Content */
        }
    ]
);
```

Примечание . *Имена введенных параметров не обязательно должны совпадать, но они будут привязаны по порядку.*

Это будет означать что-то похожее на это:

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

Процесс минимизации заменяет каждый экземпляр `app` `a` каждый экземпляр `$Injectable1Instance` с `b` и каждый экземпляр `$Injectable2Instance` с `c` .

Вложенные контроллеры

Контроллеры вложенности также объединяют `$scope` . Изменение переменной `$scope` во вложенном контроллере изменяет одну и ту же переменную `$scope` в родительском контроллере.

```
.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";
    };
```

```
});
```

Теперь давайте попробуем обработать их обоих, вложенных.

```
<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>
```

Контроллеры гнездования могут иметь свои преимущества, но при этом нужно иметь в виду одну вещь. Вызов директивы `ngController` создает новый экземпляр контроллера, который может часто создавать путаницу и неожиданные результаты.

Прочитайте Контроллеры онлайн: https://riptutorial.com/ru/angularjs/topic/601/контроллеры

глава 23: Контроллеры с ES6

Examples

контроллер

очень легко написать угловой контроллер JS с ES6, если вы знакомы с **объектно-ориентированным программированием** :

```
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;
```

Прочитайте Контроллеры с ES6 онлайн: https://riptutorial.com/ru/angularjs/topic/9419/контроллеры-с-es6

глава 24: Ленивая загрузка

замечания

1. Если ваши ленивые загруженные зависимости требуют других ленивых загруженных зависимостей, убедитесь, что вы загрузили их в правильном порядке!

```
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

Подготовка вашего проекта к ленивой загрузке

После включения `oclazyload.js` в ваш индексный файл объявите `ocLazyLoad` как зависимость в `app.js`

```
//Make sure you put the correct dependency! it is spelled different than the service!
angular.module('app', [
 'oc.lazyLoad',
 'ui-router'
])
```

использование

Чтобы лениво загружать файлы, `$ocLazyLoad` службу `$ocLazyLoad` в контроллер или другую службу

```
.controller('someCtrl', function($ocLazyLoad) {
 $ocLazyLoad.load('path/to/file.js').then(...);
});
```

Угловые модули будут автоматически загружаться в угловые.

Другие варианты:

```
$ocLazyLoad.load([
  'bower_components/bootstrap/dist/js/bootstrap.js',
  'bower_components/bootstrap/dist/css/bootstrap.css',
  'partials/template1.html'
]);
```

Полный список вариантов см. В официальной документации

Использование с маршрутизатором

UI-маршрутизатор:

```
.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'
       ]);
      }
     }
    });
```

Использование инъекции зависимостей

Следующий синтаксис позволяет вам указывать зависимости в вашем `module.js` вместо явной спецификации при использовании службы

```
//lazy_module.js
angular.module('lazy', [
 'alreadyLoadedDependency1',
 'alreadyLoadedDependency2',
 ...
 [
  'path/to/lazily/loaded/dependency.js',
  'path/to/lazily/loaded/dependency.css'
```

```
 ]
]);
```

Примечание : этот синтаксис будет работать только для лениво загружаемых модулей!

Использование директивы

```
<div oc-lazy-load="['path/to/lazy/loaded/directive.js',
'path/to/lazy/loaded/directive.html']">

<!-- myDirective available here -->
<my-directive></my-directive>

</div>
```

Прочитайте Ленивая загрузка онлайн: https://riptutorial.com/ru/angularjs/topic/6400/ленивая-загрузка

глава 25: Маршрутизация с использованием ngRoute

замечания

`ngRoute` - это встроенный модуль, предоставляющий услуги маршрутизации и девелопмента и директивы для угловых приложений.

Полная документация о `ngRoute` доступна на https://docs.angularjs.org/api/ngRoute

Examples

Основной пример

В этом примере показано, как настроить небольшое приложение с тремя маршрутами, каждый со своим собственным представлением и контроллером, используя синтаксис `controllerAs` .

Мы настраиваем наш маршрутизатор с помощью угловой функции `.config`

1. Мы `$routeProvider` в `.config`
2. Мы определяем имена маршрутов в методе `.when` с объектом определения маршрута.
3. Мы поставляем метод `.when` с объектом, определяющим наш `template` или `templateUrl` , `controller` и `controllerAs`

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'
    });
  });
```

Затем в нашем HTML мы определяем нашу навигацию с использованием `<a>` элементов с `href` , для имени маршрута `helloRoute` мы будем маршрутизировать как `<a href="#/helloRoute">My route</a>`

Мы также предоставляем наше представление контейнеру и директиву `ng-view` для ввода наших маршрутов.

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>
```

Пример параметров маршрута

Этот пример расширяет базовый пример, передавая параметры на маршруте, чтобы использовать их в контроллере

Для этого нам необходимо:

1. Настроить положение и имя параметра в названии маршрута
2. `$routeParams` услугу `$routeParams` в нашем контроллере

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'
    });
  });
```

Затем, с внесением любых изменений в наши шаблоны, добавив новую ссылку с настраиваемым сообщением, мы можем увидеть новое настраиваемое сообщение на наш взгляд.

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>
```

Определение пользовательского поведения для отдельных маршрутов

Простейший способ определения индивидуального поведения для отдельных маршрутов будет довольно простым.

В этом примере мы используем его для аутентификации пользователя:

1) `routes.js` : создать новое свойство (например, `requireAuth`) для любого желаемого маршрута

```
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) В контроллере верхнего уровня, не связанный с элементом внутри `ng-view` (чтобы избежать конфликта с угловым `$routeProvider`), проверьте , если `newUrl` имеет `requireAuth` свойства и действовать соответствующим образом

```
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");
            }

        });
    }
]);
```

Прочитайте Маршрутизация с использованием ngRoute онлайн:
https://riptutorial.com/ru/angularjs/topic/2391/маршрутизация-с-использованием-ngroute

глава 26: Модули

Examples

Модули

Модуль служит в качестве контейнера различных частей вашего приложения, таких как контроллеры, службы, фильтры, директивы и т. Д. Модули могут ссылаться на другие модули через механизм впрыска зависимостей Angular.

Создание модуля:

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

Массив `[]` в приведенном выше примере, - это *список модулей* `app` зависит `app` , если нет зависимостей, мы передаем пустой массив, т. Е. `[]` .

Ввод модуля в зависимость от другого модуля:

```
angular.module('app', [
    'app.auth',
    'app.dashboard'
]);
```

Ссылка на модуль:

```
angular
    .module('app');
```

Модули

Модуль представляет собой контейнер для различных частей ваших приложений - контроллер, службы, фильтры, директивы и т. Д.

Зачем использовать модули
Большинство приложений имеют основной метод, который создает и объединяет разные части приложения.
Угловые приложения не имеют основного метода.
Но в AngularJs декларативный процесс легко понять, и можно упаковать код в виде повторно используемых модулей.
Модули могут быть загружены в любом порядке, потому что модули замедляют выполнение.

объявить модуль

```
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
});
```

Загрузка модулей и их зависимость

1. Блоки конфигурации: - выполняются во время фазы поставщика и конфигурации.

   ```
   angular.module('myModule', []).
   config(function(injectables) {
     // here you can only inject providers in to config blocks.
   });
   ```

2. Run Blocks: - запускается после создания инжектора и используется для запуска приложения.

   ```
   angular.module('myModule', []).
   run(function(injectables) {
     // here you can only inject instances in to config blocks.
   });
   ```

Прочитайте Модули онлайн: https://riptutorial.com/ru/angularjs/topic/844/модули

глава 27: нг-повтор

Вступление

Директива `ngRepeat` создает экземпляр шаблона один раз за элемент из коллекции. Коллекция должна быть массивом или объектом. Каждый экземпляр шаблона получает свою собственную область, где заданная переменная цикла устанавливается на текущий элемент коллекции, а `$index` - на индекс элемента или ключ.

Синтаксис

- `<element ng-repeat="expression"></element>`
- `<div ng-repeat="(key, value) in myObj">...</div>`
- `<div ng-repeat="variable in expression">...</div>`

параметры

переменная	подробности
`$index`	*число* итераторное смещение повторяющегося элемента (0..length-1)
`$first`	*boolean* true, если повторный элемент является первым в итераторе.
`$middle`	*boolean* true, если повторяющийся элемент находится между первым и последним в итераторе.
`$last`	*boolean* true, если повторный элемент является последним в итераторе.
`$even`	*boolean* true, если позиция итератора `$index` является четной (в противном случае ложной).
`$odd`	*boolean* true, если позиция итератора `$index` нечетна (в противном случае ложна).

замечания

AngularJS предоставляет эти параметры в виде специальных переменных, которые доступны в выражении ng-repeat и в любом месте HTML-тега, на котором живет ng-repeat.

Examples

Итерирование свойств объекта

```
<div ng-repeat="(key, value) in myObj"> ... </div>
```

Например

```
<div ng-repeat="n in [42, 42, 43, 43]">
  {{n}}
</div>
```

Отслеживание и дублирование

`ngRepeat` использует $ watchCollection для обнаружения изменений в коллекции. Когда происходит изменение, `ngRepeat` затем вносит соответствующие изменения в DOM:

- Когда элемент добавляется, в DOM добавляется новый экземпляр шаблона.
- Когда элемент удаляется, его экземпляр шаблона удаляется из DOM.
- Когда элементы переупорядочиваются, их соответствующие шаблоны переупорядочиваются в DOM.

Дубликаты

- `track by` для любого списка, который может включать повторяющиеся значения.
- `track by` также значительно ускоряет внесение изменений в список.
- Если вы не используете `track by` в этом случае, вы получите сообщение об ошибке: `[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` обрабатывает несколько элементов с помощью `ng-repeat-start` и `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>
```

Выход:

Предметы
Пункт 1: Строка 1
Пункт 1: Строка 2
Пункт 2: Строка 1
Пункт 2: Строка 2

Прочитайте нг-повтор онлайн: https://riptutorial.com/ru/angularjs/topic/8118/нг-повтор

глава 28: нг-просмотр

Вступление

ng-view является одной из директив in-build, которая использует углы в качестве контейнера для переключения между представлениями. {info} ngRoute больше не является частью базового файла angular.js, поэтому вам нужно будет включить файл углового маршрута.js после вашего базового углового файла javascript. Мы можем настроить маршрут, используя функцию «когда» $ routeProvider. Нам нужно сначала указать маршрут, затем во втором параметре предоставить объект с свойством templateUrl и свойством контроллера.

Examples

нг-просмотр

`ng-view` - это директива, используемая с `$route` для частичного просмотра в макете главной страницы. Здесь в этом примере Index.html является нашим основным файлом, и когда пользователь приземляется на маршрут «/», templateURL home.html будет отображаться в Index.html, где упоминается `ng-view` .

```
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>
```

Регистрация навигации

1. Мы вводим модуль в приложение

```
  var Registration=angular.module("myApp",["ngRoute"]);
```

2. теперь мы используем $ routeProvider от "ngRoute"

```
    Registration.config(function($routeProvider) {

});
```

3. наконец, мы интегрируем маршрут, мы определяем маршрутизацию «/ add» в приложение, если приложение получает «/ add», оно переадресовывается в regi.htm

```
 Registration.config(function($routeProvider) {
    $routeProvider
    .when("/add", {
        templateUrl : "regi.htm"
    })
});
```

Прочитайте нг-просмотр онлайн: https://riptutorial.com/ru/angularjs/topic/8833/нг-просмотр

глава 29: нг-стиль

Вступление

Директива 'ngStyle' позволяет вам установить стиль CSS на элемент HTML условно. Подобно тому, как мы можем использовать атрибут *стиля* для HTML-элемента в проектах, отличных от AngularJS, мы можем использовать `ng-style` в angularjs, применяем стили на основе некоторого логического условия.

Синтаксис

- `<ANY ng-style="expression"></ANY >`
- `<ANY class="ng-style: expression;"> ... </ANY>`

Examples

Использование ng-стиля

Ниже пример изменяет непрозрачность изображения на основе параметра «статус».

```
<img class="img-responsive"  ng-src="{{imagesrc}}"
    ng-style="{'opacity' : (status == 2) ? 1 : 0.5}">
```

Прочитайте нг-стиль онлайн: https://riptutorial.com/ru/angularjs/topic/8773/нг-стиль

глава 30: отладка

Examples

Базовая отладка в разметке

Тестирование области и выпуск модели

```
<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'
    }];
}
```

Иногда это может помочь увидеть, есть ли новая область для исправления проблем, связанных с определением области охвата. `$scope.$id` можно использовать в выражении везде в вашей разметке, чтобы увидеть, есть ли новая область $.

В этом примере вы можете видеть, что за пределами ul-тега используется та же область ($ id = 2), а внутри `ng-repeat` для каждой итерации есть новые дочерние области.

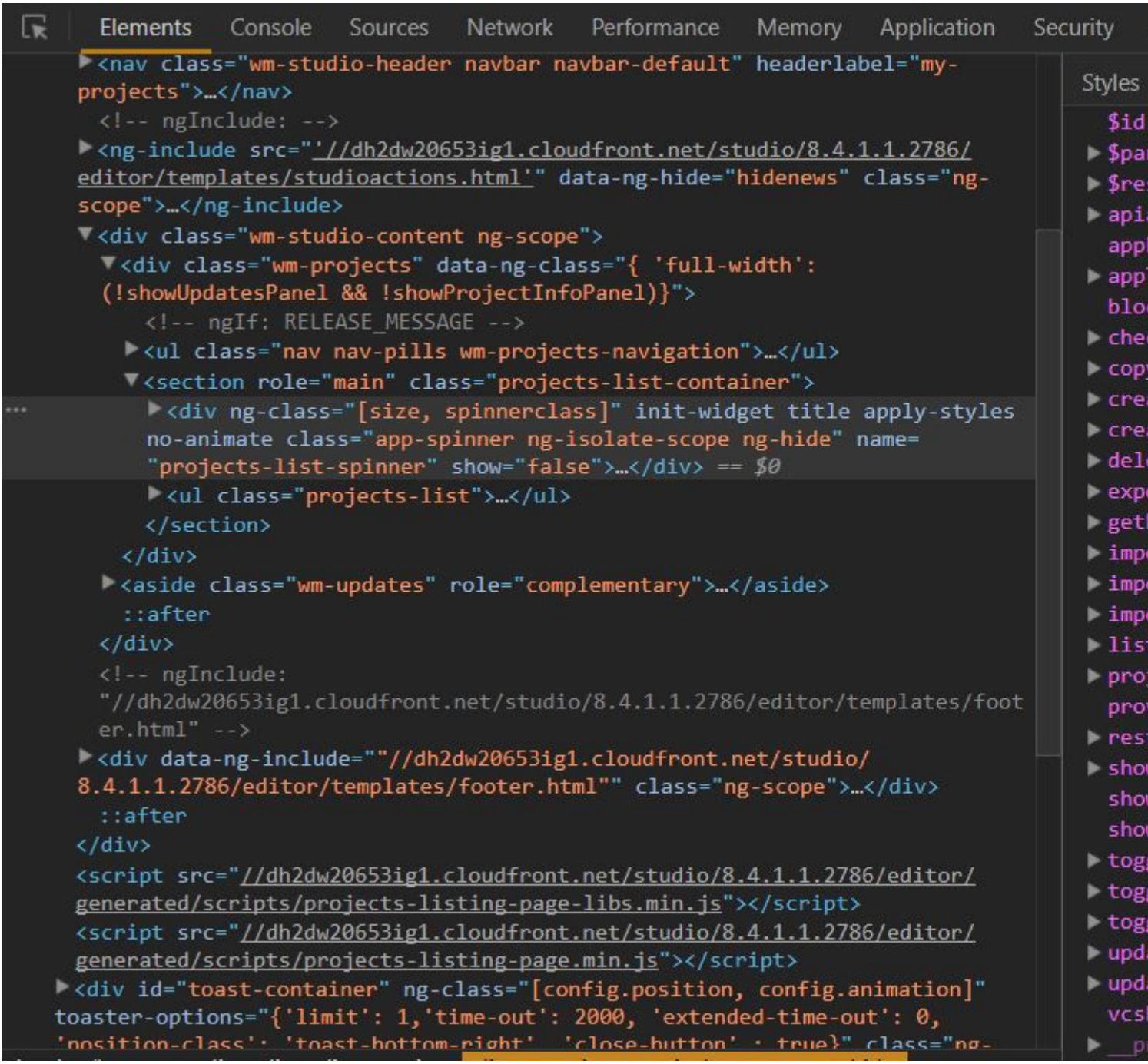
Вывод модели в предтеге полезен для просмотра текущих данных вашей модели. Фильтр `json` создает неплохо выглядящий отформатированный вывод. Предтег используется, потому что внутри этого тега будет отображаться правильный символ новой строки `\n` .

демонстрация

Использование расширения chg ng-inspect

ng-inspect - это небольшое расширение Chrome для отладки приложений AngularJS.

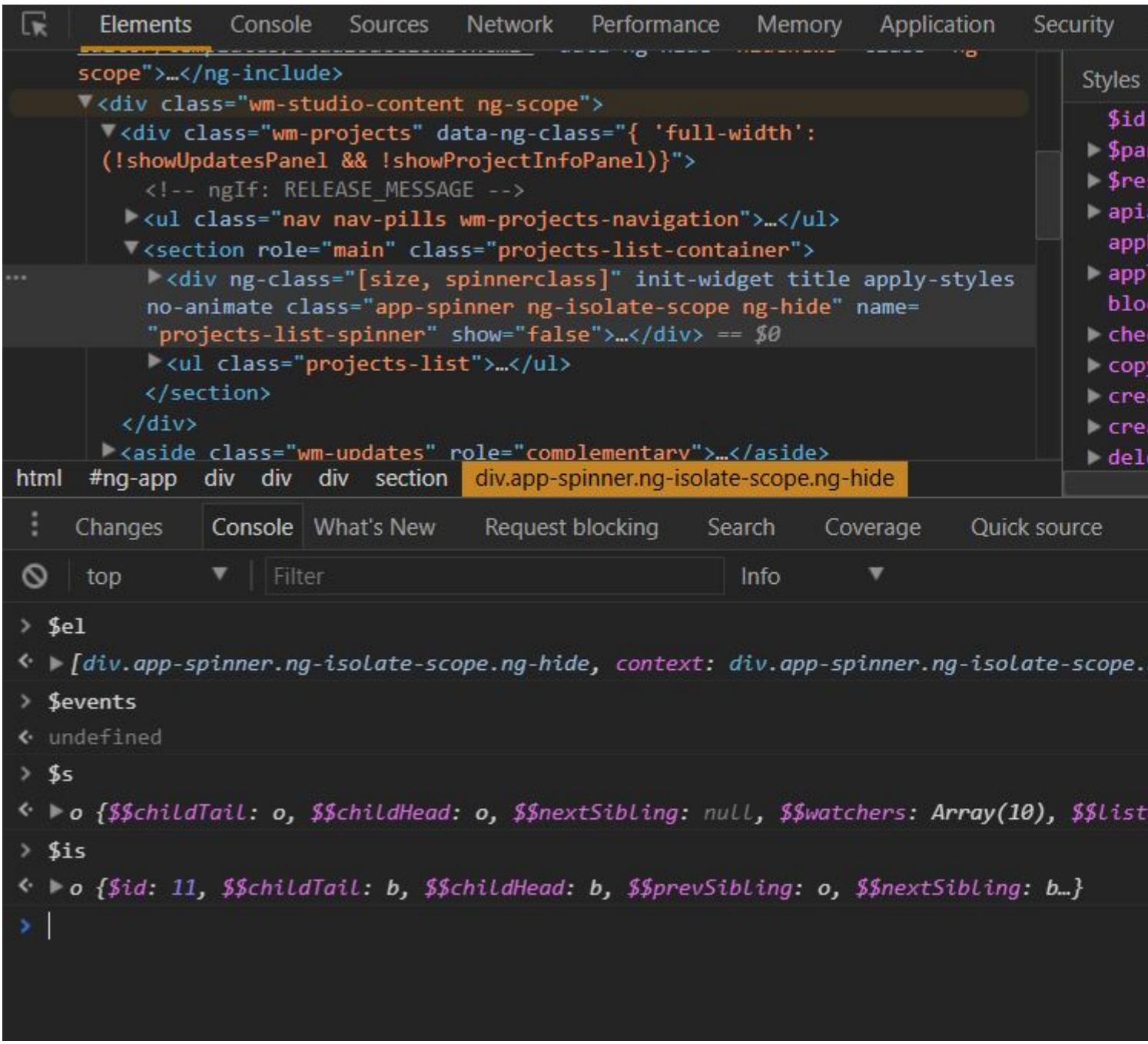
Когда узел выбран на панели элементов, информация об области видимости отображается в панели проверки ng.



Предоставляет несколько глобальных переменных для быстрого доступа к `scope/isolateScope` .

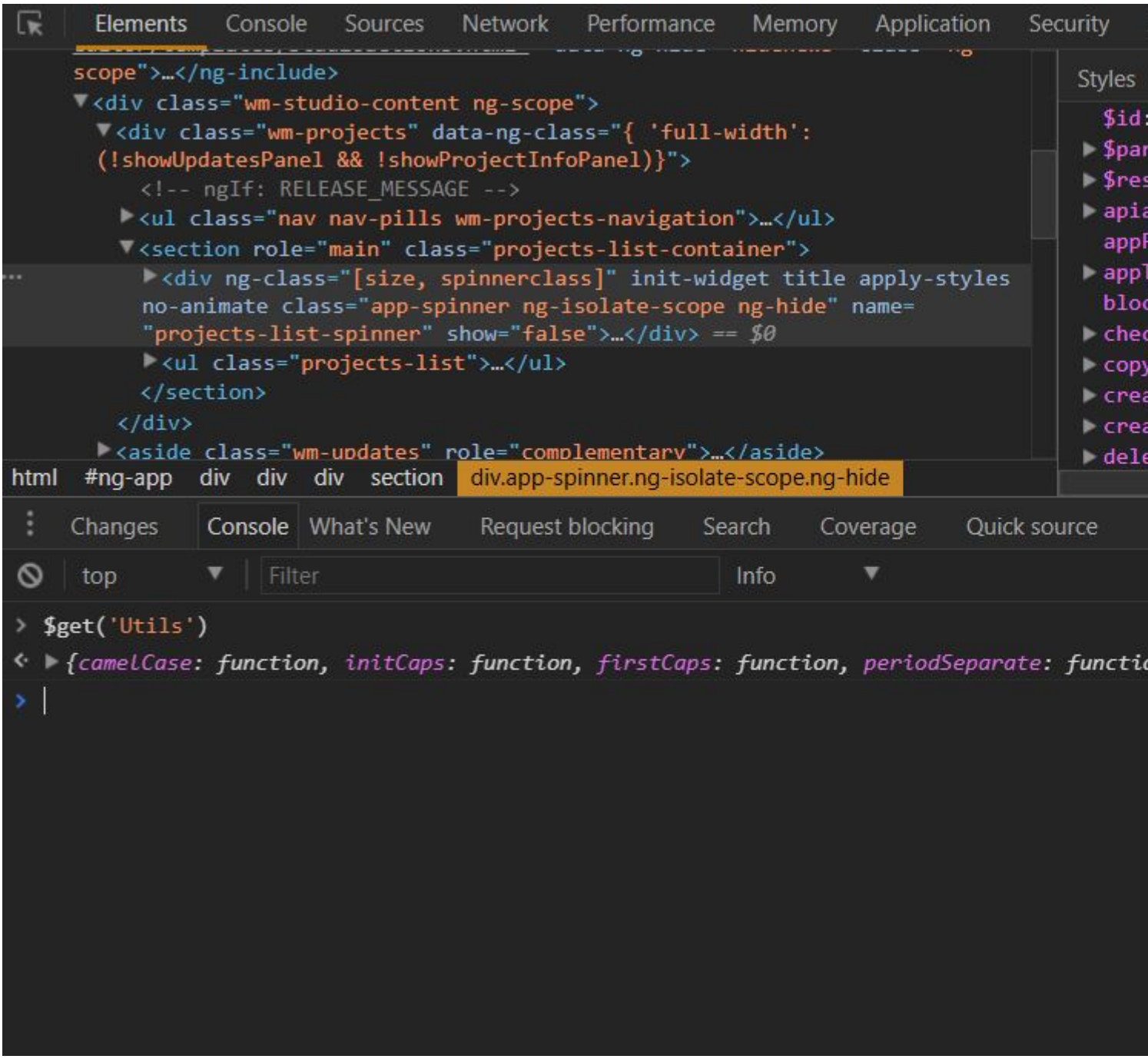
```
$s      -- scope of the selected node
```

```
$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)
```



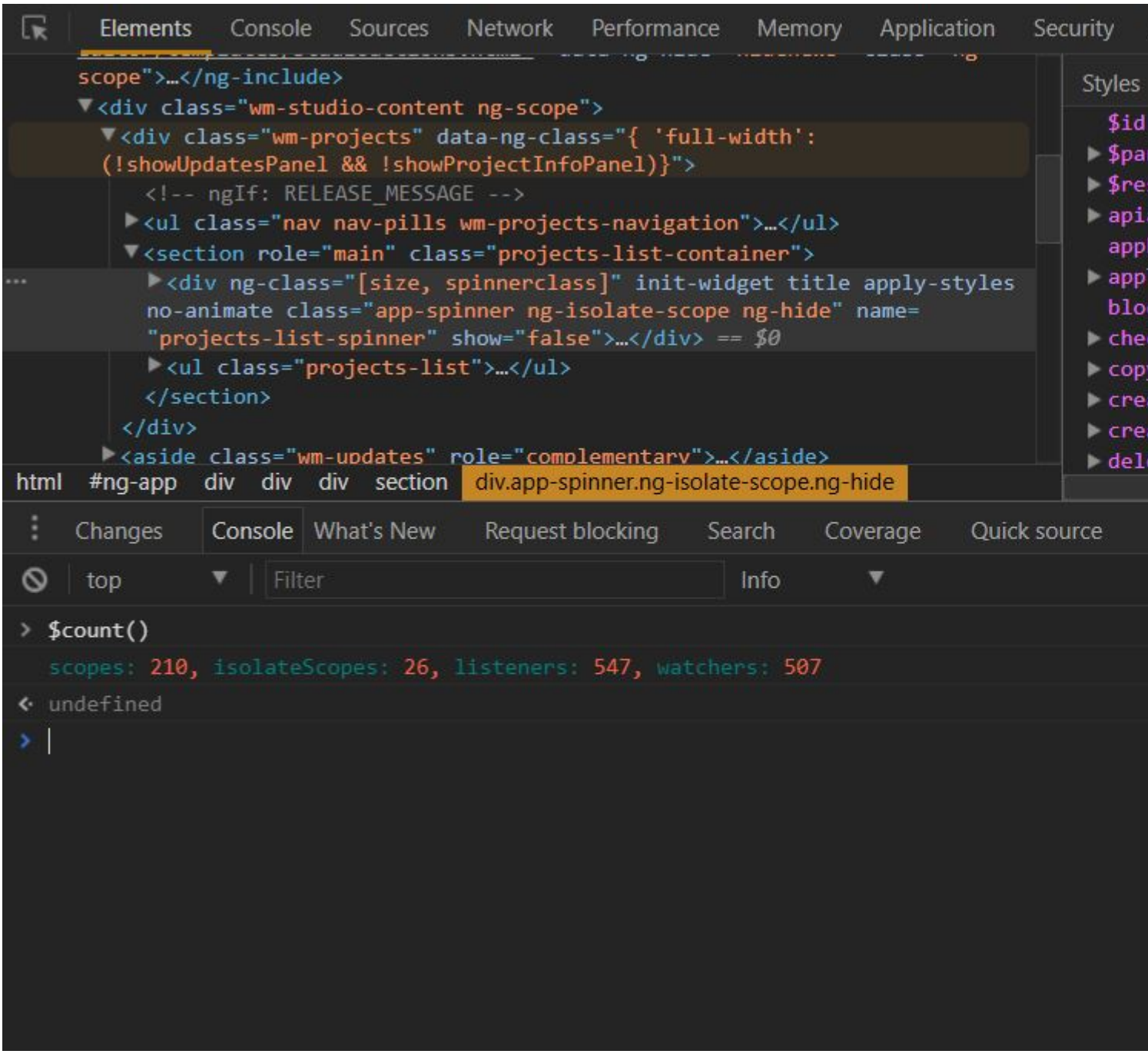
Обеспечивает легкий доступ к Сервисам / Фабрикам.

Используйте `$get()` для извлечения экземпляра службы / фабрики по имени.



Производительность приложения может контролироваться путем подсчета количества областей, изолятов, наблюдателей и слушателей в приложении.

Используйте `$count()` чтобы получить количество областей, isolateScopes, наблюдателей и слушателей.



Примечание. Это расширение будет работать только при включенном debugInfo.

Скачайте ng-inspect здесь

Получение области элемента

В угловом приложении все идет по объему, если мы можем получить область видимости, тогда легко отлаживать угловое приложение. Как получить доступ к объему элемента:

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

Получение объема контроллера: -

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

Еще один простой способ получить доступ к элементу DOM из консоли (как указано в jm) - это щелкнуть по нему на вкладке «Элементы», и он автоматически будет сохранен как $ 0.

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

Прочитайте отладка онлайн: https://riptutorial.com/ru/angularjs/topic/4761/отладка

глава 31: Отличительная служба против фабрики

Examples

Factory VS Service один раз и навсегда

По определению:

Услуги - это в основном функции конструктора. Они используют ключевое слово this.

Заводы - простые функции, поэтому возвращают объект.

Под капотом:

Фабрики внутренне вызывают функцию поставщика.

Служба внутренне вызывает функцию Factory.

Обсуждение:

Заводы могут запускать код до того, как мы вернем наш литерал объекта.

Но в то же время Сервисы также могут быть записаны для возврата литерала объекта и запуска кода перед возвратом. Хотя это непротиворечиво, поскольку службы предназначены для работы в качестве функции конструктора.

Фактически, функции-конструкторы в JavaScript могут возвращать все, что они хотят.

Так какой из них лучше?

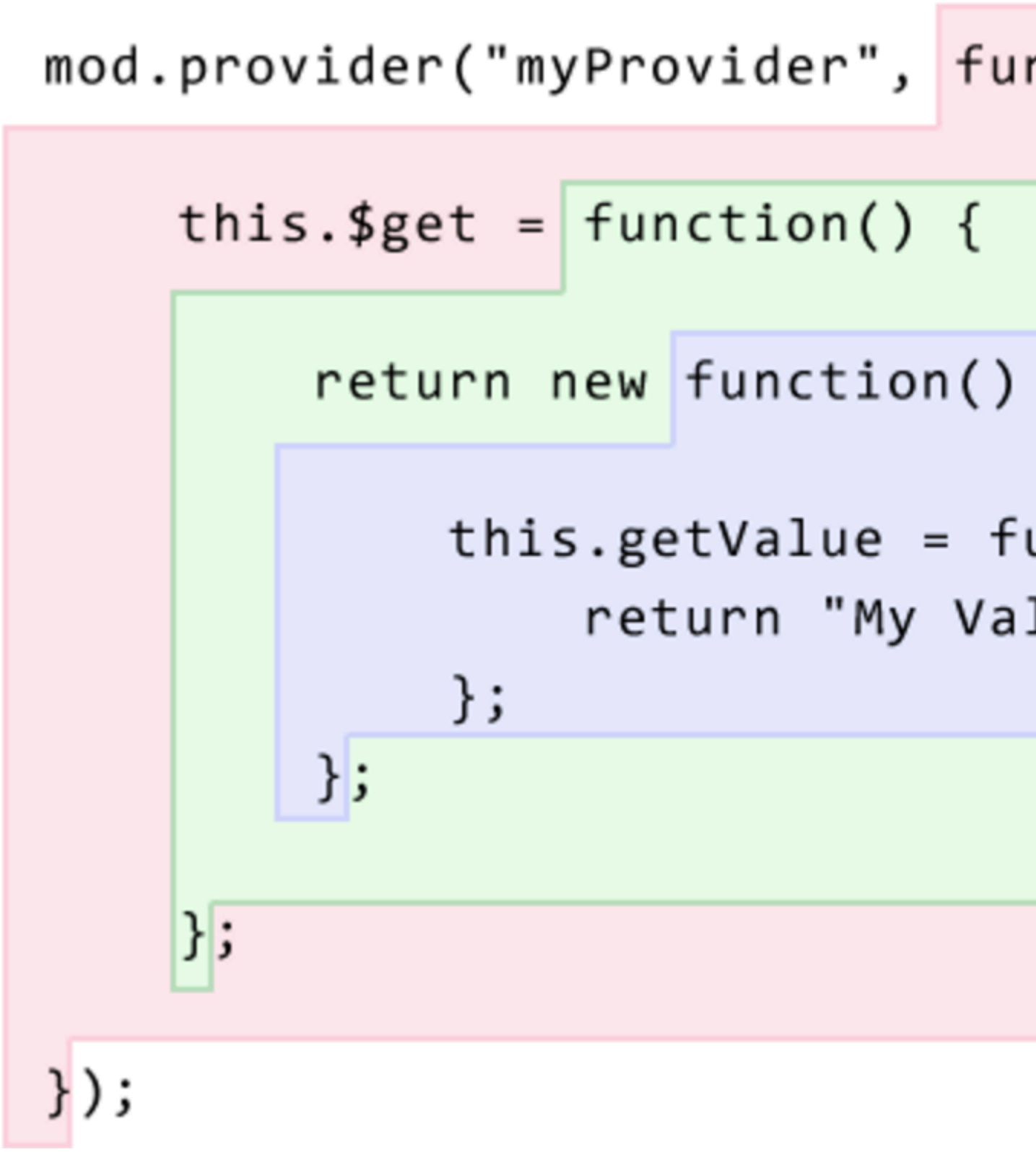
Синтаксис конструктора служб ближе к синтаксису класса ES6. Так что миграция будет легкой.

Резюме

Таким образом, в целом, провайдеры, фабрики и службы являются поставщиками.

Завод является частным случаем поставщика, когда все, что вам нужно в вашем провайдере, - это функция $ get (). Это позволяет вам писать с меньшим количеством кода.

Служба является частным случаем фабрики, когда вы хотите вернуть экземпляр нового объекта, с тем же преимуществом при написании меньшего количества кода.



Прочитайте Отличительная служба против фабрики онлайн:
https://riptutorial.com/ru/angularjs/topic/7099/отличительная-служба-против-фабрики

глава 32: Параметры привязки AngularJS (`=`, `@`, `&` и т. Д.)

замечания

Используйте этот плункер, чтобы играть с примерами.

Examples

@ односторонняя привязка, привязка атрибута.

Перейдите в буквальное значение (а не объект), например строку или номер.

Область Child получает свое значение, если оно обновляет значение, родительская область имеет свое собственное старое значение (дочерняя область не может изменять значение поля parens). Когда значение родительской области видимости изменяется, значение дочернего объекта также будет изменено. Все интерполяции появляются каждый раз при вызове дайджеста, а не только при создании директивы.

```
<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>
```

= двусторонняя привязка.

Передавая значение по ссылке, вы хотите разделить значение между обеими областями и управлять ими из обеих областей. Вы не должны использовать {{...}} для интерполяции.

```
<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>
```

Передача функции по ссылке - это плохая идея: позволить области изменять определение функции, а два ненужных наблюдателя будут созданы, вам нужно минимизировать количество наблюдателей.

& привязка функции, привязка выражения.

Передайте метод в директиву. Он обеспечивает способ выполнения выражения в контексте родительской области. Метод будет выполняться в рамках родителя, вы можете передать некоторые параметры из области дочернего объекта. Вы не должны использовать {{...}} для интерполяции. Когда вы используете & в директиве, она генерирует функцию, которая возвращает значение выражения, вычисленного против родительской области (не то же самое, что = где вы просто передаете ссылку).

```
<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>
```

Все параметры будут включены в функции.

Доступное связывание через простой образец

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

Здесь мы имеем все связывающие элементы.

@ указывает, что нам нужно очень **простое привязку** , от родительской области до области с детьми, без какого-либо наблюдателя. Каждое обновление в родительской области останется в родительской области, и любое обновление дочерней области не будет передано родительской области.

< обозначает **одностороннюю привязку** . Обновления в родительской области будут

распространяться на область «Дети», но любое обновление в области «Дети» не будет применяться к родительской области.

= уже известен как двусторонняя привязка. Каждое обновление родительской области будет применяться к дочерним, и каждое дочернее обновление будет применено к родительской области.

& теперь используется для привязки вывода. Согласно документации по компоненту, ее следует использовать для ссылки на метод родительской области. Вместо того, чтобы манипулировать областью children, просто вызовите родительский метод с обновленными данными!

Связать необязательный атрибут

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

Необязательные атрибуты должны быть отмечены знаком вопроса: `=?` или `=?bar` . Это защита исключения `($compile:nonassign)` .

Прочитайте Параметры привязки AngularJS (`=`, `@`, `&` и т. Д.) онлайн: https://riptutorial.com/ru/angularjs/topic/6149/параметры-привязки-angularjs----------amp---и-т--д--

глава 33: Переход к угловому 2+

Вступление

AngularJS полностью переписан с использованием языка TypeScript и переименован в Angular.

В приложении AngularJS многое можно сделать, чтобы облегчить процесс миграции. Как говорится в официальном руководстве по обновлению , для реорганизации вашего приложения можно выполнить несколько «шагов подготовки», что делает его лучше и ближе к новому угловому стилю.

Examples

Преобразование вашего приложения AngularJS в компонентно-ориентированную структуру

В новой структуре углов **Компоненты** являются основными строительными блоками, которые составляют пользовательский интерфейс. Таким образом, один из первых шагов, который помогает приложению AngularJS переноситься в новый Angular, состоит в том, чтобы реорганизовать его в более ориентированную на компонент структуру.

Компоненты были также представлены в старой версии AngularJS, начиная с версии **1.5+** . Использование компонентов в приложении AngularJS не только приблизит его структуру к новому угловому 2+, но и сделает его более модульным и более простым в обслуживании.

Прежде чем идти дальше, я рекомендую посмотреть официальную страницу документации AngularJS о компонентах , где их преимущества и использование хорошо объясняются.

Я бы хотел упомянуть некоторые советы о том, как преобразовать старый ориентированный на `ng-controller` код в новый `component` ориентированный стиль.

Начните разбивать свое приложение на компоненты

Все приложения, ориентированные на компоненты, обычно имеют один или несколько компонентов, которые включают в себя другие подкомпоненты. Так почему бы не создать первый компонент, который просто будет содержать ваше приложение (или большую часть).

Предположим, что у нас есть кусок кода, назначенный контроллеру с именем `UserListController` , и мы хотим создать его компонент, который мы `UserListComponent` .

текущий HTML:

```
<div ng-controller="UserListController as listctrl">
    <ul>
        <li ng-repeat="user in myUserList">
            {{ user }}
        </li>
    </ul>
</div>
```

текущий JavaScript:

```
app.controller("UserListController", function($scope, SomeService) {

    $scope.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

новый HTML:

```
<user-list></user-list>
```

новый JavaScript:

```
app.component("UserList", {
    templateUrl: 'user-list.html',
    controller: UserListController
});

function UserListController(SomeService) {

    this.myUserList = ['Shin', 'Helias', 'Kalhac'];

    this.someFunction = function() {
        // ...
    }

    // ...
}
```

Обратите внимание, что мы больше не вводим `$scope` в функцию контроллера, и теперь мы объявляем `this.myUserList` вместо `$scope.myUserList` ;

новый файл шаблона `user-list.component.html` :

```
<ul>
    <li ng-repeat="user in $ctrl.myUserList">
        {{ user }}
    </li>
</ul>
```

Обратите внимание, что теперь мы `myUserList` на переменную `myUserList` , которая принадлежит контроллеру, используя `$ctrl.myUserList` из html вместо `$scope.myUserList` .

Это потому, что, как вы, вероятно, выяснили после прочтения документации, `$ctrl` в шаблоне теперь относится к функции контроллера.

Как насчет контроллеров и маршрутов?

В случае, если ваш контроллер привязан к шаблону с использованием системы маршрутизации вместо `ng-controller` , поэтому, если у вас есть что-то вроде этого:

```
$stateProvider
  .state('users', {
    url: '/users',
    templateUrl: 'user-list.html',
    controller: 'UserListController'
  })
  // ..
```

вы можете просто изменить свою декларацию состояния:

```
$stateProvider
    .state('users', {
      url: '/',
      template: '<user-list></user-list>'
    })
    // ..
```

Что дальше?

Теперь, когда у вас есть компонент, содержащий ваше приложение (независимо от того, содержит ли оно все приложение или его часть, например представление), теперь вы должны начать разбивать свой компонент на несколько вложенных компонентов, обертывая его части на новые подкомпоненты , и так далее.

Вы должны начать использовать компоненты Component, такие как

- **Входы** и **выходы**
- **крючки жизненного цикла,** такие как `$onInit()` , `$onChanges()` и т. д. ...

После прочтения документации по компонентам вы уже должны знать, как использовать все эти компоненты, но если вам нужен конкретный пример реального простого приложения, вы можете это проверить.

Кроме того, если внутри контроллера вашего компонента есть некоторые функции, которые содержат много логического кода, хорошая идея может быть рассмотрена, чтобы переместить эту логику в сервисы .

Заключение

Принятие основанного на компонентах подхода подталкивает ваш AngularJS на один шаг ближе, чтобы перенести его в новую угловую структуру, но также делает ее лучше и намного более модульной.

Конечно, есть много других шагов, которые вы можете сделать, чтобы перейти в новое направление Angular 2+, которое я приведу в следующих примерах.

Представляем модули Webpack и ES6

Используя ***загрузчик модуля,*** такой как Webpack, мы можем воспользоваться встроенной системой модулей, доступной в **ES6** (а также в **TypeScript**). Затем мы можем использовать функции импорта и экспорта, которые позволяют нам указать, какие фрагменты кода мы можем разделить между различными частями приложения.

Когда мы принимаем наши приложения в производство, модульные погрузчики также облегчают их упаковку в производственные комплекты с включенными батареями.

Прочитайте Переход к угловому 2+ онлайн: https://riptutorial.com/ru/angularjs/topic/9942/переход-к-угловому-2plus

глава 34: Подготовьтесь к производству - Grunt

Examples

Просмотр предварительной загрузки

Когда запрашивается первый запрос времени, обычно Angular делает запрос `XHR` для получения этого представления. Для проектов среднего размера количество просмотров может быть значительным, и это может замедлить отзывчивость приложения.

Хорошей практикой является предварительная загрузка всех представлений сразу для небольших и средних проектов. Для более крупных проектов хорошо их агрегировать и в некоторых значимых объемах, но некоторые другие методы могут быть полезны для разделения нагрузки. Для автоматизации этой задачи удобно использовать задачи Grunt или Gulp.

Чтобы предварительно загрузить представления, мы можем использовать объект `$templateCache` . Это объект, в котором углы хранят каждое полученное представление с сервера.

Можно использовать модуль `html2js` , который преобразует все наши представления в один модуль - файл js. Тогда нам нужно будет вставить этот модуль в наше приложение, и все.

Чтобы создать конкатенированный файл всех представлений, мы можем использовать эту задачу

```
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']);
};
```

Чтобы использовать этот способ конкатенации, вам нужно сделать 2 изменения: в вашем файле `index.html` вам необходимо ссылаться на файл конкатенированного представления

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

В файле, где вы объявляете свое приложение, вам нужно ввести зависимость

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

Если вы используете популярные роутеры, такие как `ui-router` , никаких изменений в способе, как вы ссылаетесь на шаблоны

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

    })
```

Оптимизация скриптов

Хорошая практика - объединить файлы JS и их минимизировать. Для более крупного проекта могут быть сотни JS-файлов, и он добавляет лишнюю задержку для загрузки каждого файла отдельно от сервера.

Для угловой минимизации требуется, чтобы все функции были аннотированы. Это необходимо для правильной минимальной инъекции угловой зависимости. (Во время минификсации имена функций и переменные будут переименованы, и он будет разорвать инъекцию зависимостей, если никаких дополнительных действий не будет предпринято.)

Во время minificaiton `$scope` и `myService` переменные будут заменены некоторыми другими значениями. Угловые инъекции зависимостей основаны на имени, в результате эти имена не должны меняться

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

Угловой будет понимать обозначение массива, потому что минификация не заменит строковые литералы.

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

- Во-первых, мы будем конкатенировать все файлы до конца.
- Во-вторых, мы будем использовать модуль `ng-annotate` , который подготовит код для минимизации
- Наконец, мы применим модуль `uglify` .

module.exports = function (grunt) {// настраиваем расположение ваших скриптов для повторного использования в скрипте 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']);
};
```

Прочитайте Подготовьтесь к производству - Grunt онлайн:
https://riptutorial.com/ru/angularjs/topic/4434/подготовьтесь-к-производству---grunt

глава 35: Пользовательские директивы

Вступление

Здесь вы узнаете о функции Directives от AngularJS. Ниже вы найдете информацию о том, какие директивы, а также основные и расширенные примеры того, как их использовать.

параметры

параметр	подробности
объем	Свойство, чтобы установить область действия директивы. Он может быть установлен как false, true или как область выделения: {@, =, <, &}.
область действия: ложь	Директива использует родительскую область. Никакой области, созданной для директивы.
scope: true	Директива наследует родительскую область прототипно как новый охват ребенка. Если существует несколько директив для одного элемента с запросом новой области, то они будут разделять одну новую область.
объем: { @ }	Односторонняя привязка свойства области действия директивы к значению атрибута DOM. Поскольку значение атрибута, связанное с родителем, оно изменится в области директивы.
scope: {=}	Двунаправленная привязка атрибута, которая изменяет атрибут родителя, если атрибут директивы изменяется и наоборот.
scope: {<}	Односторонняя привязка свойства области действия директивы и выражения атрибута DOM. Выражение оценивается в родительском. Это отслеживает идентификатор родительского значения, поэтому изменения в объекте в родительском объекте не будут отражены в директиве. Изменения свойства объекта в директиве будут отражены в родительском, поскольку оба ссылаются на один и тот же объект
объем: { & }	Позволяет директиве передать данные в выражение, которое будет оцениваться в родительском.
компиляция: функция	Эта функция используется для выполнения преобразования DOM в шаблоне директивы перед запуском функции связи. Он

параметр	подробности
	принимает `tElement` (шаблон директивы) и `tAttr` (список атрибутов, объявленных в директиве). Он не имеет доступа к области. Он может возвращать функцию, которая будет зарегистрирована в качестве функции `post-link` или может вернуть объект с параметрами `pre` и `post` будет зарегистрирована как функция `pre-link` и `post-link` .
ссылка: функция / объект	Свойство link можно настроить как функцию или объект. Он может принимать следующие аргументы: scope (область директивы), iElement (элемент DOM, где применяется директива), iAttrs (набор атрибутов элемента DOM), контроллер (массив контроллеров, требуемый директивой), transcludeFn. Он в основном используется для настройки DOM-прослушивателей, просмотра свойств модели для изменений и обновления DOM. Он выполняется после клонирования шаблона. Он настроен независимо, если функция компиляции отсутствует.
функция предварительной ссылки	Функция связи, которая выполняется до того, как будет создана какая-либо дочерняя ссылка. По умолчанию функции дочерних директивных ссылок выполняются перед функциями родительской директивной ссылки, а функция предварительной привязки позволяет родительскому соединению сначала. Один случай использования - если ребенок требует данных от родителя.
функция пост-ссылки	Функция ссылок, которую руководители после дочерних элементов связаны с родителем. Он обычно используется для прикрепления обработчиков событий и доступа к дочерним директивам, но данные, требуемые дочерней директивой, не должны устанавливаться здесь, потому что дочерняя директива уже была связана.
ограничивать: строка	Определяет, как вызвать директиву из DOM. Возможные значения (Предположим, что наше директивное имя - `demoDirective`): `E` - Имя элемента (`<demo-directive></demo-directive>`), `A` - Атрибут (`<div demo-directive></div>`), `C` - Соответствующий класс (`<div class="demo-directive"></div>`), `M` - по комментарию (`<!-- directive: demo-directive -->`). Свойство `restrict` может также поддерживать несколько параметров, например - `restrict: "AC"` ограничивает директивой *атрибут* OR *Class* . Если опущено, значением по **умолчанию** является `"EA"` (элемент или атрибут).
требуют: '	Найдите контроллер demoDirective на текущем элементе и

параметр	подробности
demoDirective'	введите его контроллер в качестве четвертого аргумента функции связывания. Выбросьте ошибку, если она не найдена.
require: '? demoDirective'	Попытайтесь найти контроллер demoDirective или передать null в ссылку fn, если он не найден.
require: '^ demoDirective'	Найдите контроллер demoDirective, выполнив поиск элемента и его родителей. Выбросьте ошибку, если она не найдена.
require: '^^ demoDirective'	Найдите контроллер demoDirective, выполнив поиск родителей элемента. Выбросьте ошибку, если она не найдена.
require: '? ^ demoDirective'	Попытайтесь найти контроллер demoDirective, выполнив поиск элемента и его родителей или передав null в ссылку fn, если не найден.
require: '? ^^ demoDirective'	Попытайтесь найти контроллер demoDirective, выполнив поиск родительского элемента или передайте null в ссылку fn, если не найден.

Examples

Создание и использование пользовательских директив

Директивы - одна из самых мощных функций angularjs. Пользовательские директивы angularjs используются для расширения функциональности html путем создания новых элементов html или пользовательских атрибутов для обеспечения определенного поведения тега html.

directive.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;
    }
  };
});
```

Эта директива затем может быть использована в приложении как:

```
<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>
```

Шаблон объекта определения директивы

```
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`** - установить значение true, и любые узлы DOM между началом и концом имени директивы будут собраны и сгруппированы вместе в качестве директивных элементов
2. **`priority`** - позволяет указать порядок применения директив, когда несколько директив определены на одном элементе DOM. Сначала устанавливаются директивы с более высокими номерами.
3. **`terminal`** - установить значение true, а текущий приоритет будет последним набором директив для выполнения
4. **`scope`** - задает область действия директивы
5. **`bind to controller`** - связывает свойства области напрямую с директивным контроллером
6. функция конструктора **`controller`** - контроллера
7. **`require`** - требуется другая директива и ввести свой контроллер в качестве четвертого аргумента функции связывания
8. **`controllerAs`** - имя ссылается на контроллер в области директивы, чтобы позволить на контроллер ссылаться из шаблона директивы.
9. **`restrict`** - ограничить директиву элементом, атрибутом, классом или комментарием
10. **`templateNameSpace`** - устанавливает тип документа, используемый шаблоном директивы: html, svg или math. html - значение по умолчанию
11. **`template`** - html markup, который по умолчанию заменяет содержимое элемента директивы или обертывает содержимое элемента директивы, если transclude is true
12. **`templateUrl`** - url, предоставляемый асинхронно для шаблона
13. **`transclude`** - Извлечь содержимое элемента, в котором появляется директива, и

сделать его доступным для директивы. Содержимое скомпилировано и предоставлено директиве в качестве функции перехода.

14. **`compile`** - функция преобразования шаблона DOM
15. **`link`** - используется только в том случае, если свойство компиляции не определено. Функция связи отвечает за регистрацию DOM-прослушивателей, а также за обновление DOM. Он выполняется после клонирования шаблона.

Пример базовой директивы

супермен-directive.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!')
        });
      }
    }
  });
```

сверхчеловека 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>
```

Вы можете проверить больше о директивах `restrict` и `link` функциях на официальной

документации AngularJS по Директивам

Как создать восстанавливаемый компонент с помощью директивы

Директивы AngularJS - это то, что контролирует рендеринг HTML внутри приложения AngularJS. Они могут быть элементом Html, атрибутом, классом или комментарием. Директивы используются для управления DOM, добавления нового поведения к элементам HTML, привязки данных и многих других. Некоторыми примерами директив, которые являются угловыми, являются ng-model, ng-hide, ng-if.

Аналогичным образом можно создать свою собственную настраиваемую директиву и сделать ее доступной. Для создания Справочника по пользовательским директивам. Смысл создания многоразовых директив заключается в том, чтобы создать набор директив / компонентов, написанных вами, так же, как angularjs предоставляет нам использование angular.js. Эти многоразовые директивы могут быть особенно полезными, когда у вас есть набор приложений / приложений, которые требуют последовательного поведения, внешнего вида и восприятия. Примером такого многоразового компонента может быть простая панель инструментов, которую вы можете использовать в своем приложении или в разных приложениях, но вы хотите, чтобы они вели себя одинаково или выглядели одинаково.

Во-первых, создайте папку с именем resuableComponents в папке приложения и сделайте 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>
```

Директива по умолчанию используется для повторного использования компонентов. Когда

вы делаете директивы в отдельном угловом модуле, он фактически делает его экспортируемым и многоразовым в разных приложениях с угловыми функциями. Новые директивы можно просто добавить внутри reusableModuleApp.js, a reusableModuleApp может иметь собственный контроллер, службы, объект DDO внутри директивы для определения поведения.

Основная директива с шаблоном и изолированной областью

Создание настраиваемой директивы с *изолированной областью* отделяет область **внутри** директивы от **внешней** области, чтобы наша директива не случайно меняла данные в родительской области и ограничивала ее чтением частных данных из родительской области.

Чтобы создать изолированную область действия и позволить нашей настраиваемой директиве связываться с внешней областью, мы можем использовать опцию `scope` которая описывает, как **сопоставить** привязки внутренней области директивы с внешней областью.

Фактические привязки выполняются с дополнительными **атрибутами,** прикрепленными к директиве. Связующие параметры определяются с `scope` опции и объект с пар ключ-значение:

- **Ключ** , который соответствует изолированному свойству свойства нашей директивы
- **Значение** , которое сообщает Angular, как привязать внутреннюю область директивы к **атрибуту** соответствия

Простой пример директивы с изолированной областью:

```
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);
```

Пример использования этой директивы и привязка данных из области действия контроллера к внутренней области директивы:

контроллер:

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

Посмотреть:

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

Построение многоразового компонента

Директивы могут использоваться для создания повторно используемых компонентов. Ниже приведен пример компонента «ящик пользователя»:

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>
```

пользователя box.html

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

Результатом будет:

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

Декоратор директивы

Иногда вам могут понадобиться дополнительные функции из директивы. Вместо того, чтобы переписывать (копировать) директиву, вы можете изменить поведение директивы.

Декоратор будет выполнен во время фазы инъекции.

Чтобы сделать это, подайте код .config в свой модуль. Директива называется myDirective, поэтому вам нужно настроить myDirectiveDirective. (это в угловом соглашении [читайте о провайдерах]).

Этот пример изменит templateUrl директивы:

```
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;
    })
});
```

Этот пример добавляет событие onClick к элементу директивы при щелчке, это происходит во время фазы компиляции.

```
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;
        });

    });
```

Подобный подход может использоваться как для Провайдеров, так и для Услуг.

Наследование и совместимость директив

Угловые js-директивы могут быть вложенными или быть совместимыми.

В этом примере директива Adir предоставляет директиву Bdir - это контроллер $ scope, так как Bdir требует Adir.

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

Удостоверьтесь, что необходимо установить: '^ Adir' (посмотрите на угловую документацию, некоторые версии не требуют символа ^).

```
.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.
                });
```

```
            }
        }
    }]);
```

Вы можете вложить свою директиву таким образом:

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

Не требуется, чтобы директивы были вложены в ваш HTML.

Прочитайте Пользовательские директивы онлайн:
https://riptutorial.com/ru/angularjs/topic/965/пользовательские-директивы

глава 36: Пользовательские фильтры

Examples

Пример простого фильтра

Фильтры форматируют значение выражения для отображения пользователю. Они могут использоваться в виде шаблонов, контроллеров или служб. В этом примере создается фильтр (`addZ`), который затем использует его в представлении. Весь этот фильтр делает добавление капитала «Z» в конец строки.

example.js

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

example.html

Внутри представления фильтр применяется со следующим синтаксисом: `{ variable | filter}` . В этом случае переменная, которую мы определили в контроллере, `sample` , фильтруется фильтром, который мы создали, `addZ` .

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

Ожидаемый результат

```
helloZ
```

Используйте фильтр в контроллере, службе или фильтре

Вам нужно будет ввести `$filter` :

```
angular
  .module('filters', [])
  .filter('percentage', function($filter) {
```

```
    return function (input) {
      return $filter('number')(input * 100) + ' %';
    };
  });
```

Создайте фильтр с параметрами

По умолчанию фильтр имеет единственный параметр: переменную, на которую он применяется. Но вы можете передать больше параметра функции:

```
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) + ' %';
    };
  });
```

Теперь вы можете точно определить `percentage` фильтр:

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

... но другие параметры являются необязательными, вы все равно можете использовать фильтр по умолчанию:

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

Прочитайте Пользовательские фильтры онлайн: https://riptutorial.com/ru/angularjs/topic/2552/пользовательские-фильтры

глава 37: Пользовательские фильтры с ES6

Examples

Фильтр FileSize с использованием ES6

У нас есть файл Размер фильтра, чтобы описать, как добавить фильтр costum в существующий модуль:

```
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;
```

Вызов фильтра в модуль:

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

Код html, где мы называем фильтр:

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

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

Прочитайте Пользовательские фильтры с ES6 онлайн:
https://riptutorial.com/ru/angularjs/topic/9421/пользовательские-фильтры-с-es6

глава 38: Провайдеры

Синтаксис

- константа (имя, значение);
- значение (имя, значение);
- завод (имя, $ getFn);
- сервис (имя, конструктор);
- поставщик (имя, поставщик);

замечания

Провайдеры представляют собой одноэлементные объекты, которые могут быть введены, например, в другие службы, контроллеры и директивы. Все поставщики зарегистрированы с использованием разных «рецептов», где `Provider` является наиболее гибким. Все возможные рецепты:

- постоянная
- Значение
- завод
- обслуживание
- поставщик

Службы, фабрики и поставщики все ленивы инициализированы, компонент инициализируется, только если приложение зависит от него.

Декораторы тесно связаны с Провайдерами. Декораторы используются для перехвата обслуживания или создания фабрики, чтобы изменить его поведение или переопределить (его части).

Examples

постоянная

`Constant` доступна как в конфигурации, так и в фазах запуска.

```
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>
```

---

endpoint = http: //some.rest.endpoint

Значение

`Value` доступно как на этапах конфигурации, так и на этапе запуска.

```
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>
```

---

endpoint = http: //some.rest.endpoint

завод

`Factory` доступна в фазе запуска.

> Рецепт Factory создает новую службу, используя функцию с нулевым или большим аргументом (это зависимости от других сервисов). Возвращаемое значение этой функции является экземпляром службы, созданным этим рецептом.
>
> Factory может создавать сервис любого типа, будь то примитивный, объектный литерал, функция или даже экземпляр пользовательского типа.

```
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>
```

---

endpoint = http: //some.rest.endpoint

обслуживание

`Service` доступна на этапе запуска.

> Рецепт службы создает сервис, аналогичный рецептам Value или Factory, но он делает это, *вызывая конструктор с новым оператором* . Конструктор может принимать ноль или более аргументов, которые представляют зависимости, необходимые экземпляру этого типа.

```
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>
```

---

endpoint = http: //some.rest.endpoint

поставщик

`Provider` доступен как в конфигурации, так и в фазах запуска.

> Рецепт поставщика синтаксически определяется как настраиваемый тип, реализующий метод `$get` .
>
> Вы должны использовать рецепт поставщика только тогда, когда вы хотите открыть API для всей конфигурации приложения, которая должна быть

выполнена до запуска приложения. Обычно это интересно только для многоразовых сервисов, поведение которых может немного меняться между приложениями.

```
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>
```

---

endpoint = http: //some.rest.endpoint

Без `config` фаза результат был бы

endpoint = n / a

Прочитайте Провайдеры онлайн: https://riptutorial.com/ru/angularjs/topic/5169/провайдеры

глава 39: Проверка формы

Examples

Базовая проверка формы

Одной из сильных сторон Angular является проверка на стороне клиента.

Работа с традиционными входными формами и необходимость использования вопросительной обработки в стиле jQuery может быть трудоемкой и сложной. Угловой позволяет вам легко создавать профессиональные *интерактивные* формы.

Директива **ng-model** обеспечивает двустороннюю привязку с полями ввода, и обычно атрибут **novalidate** также помещается в элемент формы, чтобы браузер не выполнял встроенную проверку.

Таким образом, простая форма будет выглядеть так:

```
<form name="form" novalidate>
  <label name="email"> Your email </label>
  <input type="email" name="email" ng-model="email" />
</form>
```

Для Угловой проверки входов используйте тот же синтаксис, что и обычный элемент *ввода* , за исключением добавления атрибута **ng-model,** чтобы указать, какую переменную нужно привязать к области. Электронная почта отображается в предыдущем примере. Чтобы проверить число, синтаксисом будет:

```
<input type="number" name="postalcode" ng-model="zipcode" />
```

Окончательные шаги по базовой проверке формы - это подключение к функции представления формы на контроллере с помощью **ng-submit** , вместо того, чтобы разрешать отправку формы по умолчанию. Это необязательно, но обычно используется, поскольку входные переменные уже доступны в области видимости и поэтому доступны для вашей функции отправки. Как правило, хорошей практикой является предоставление формы имени. Эти изменения приведут к следующему синтаксису:

```
<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>
```

Этот выше код является функциональным, но есть и другие функции, которые предоставляет Angular.

Следующим шагом будет понять, что Angular присоединяет атрибуты класса с использованием **ng-pristine** , **ng-dirty** , **ng-valid** и **ng-invalid** для обработки формы. Использование этих классов в вашем css позволит вам создать **допустимые** / **недействительные** и **нетронутые** / **грязные** поля ввода и таким образом изменить презентацию, когда пользователь вводит данные в форму.

Формы и исходные состояния

Угловые формы и входы имеют различные состояния, которые полезны при проверке содержимого

Входные государства

государственный	Описание
`$touched`	Поле было тронуто
`$untouched`	Поле не тронуто
`$pristine`	Поле не было изменено
`$dirty`	Поле изменено
`$valid`	Полевое содержимое действительно
`$invalid`	Недопустимое содержимое поля.

Все вышеперечисленные состояния являются логическими свойствами и могут быть либо истинными, либо ложными.

С их помощью очень легко отображать сообщения пользователю.

```
<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>
```

Здесь мы используем директиву `ng-show` для отображения сообщения пользователю, если они изменили форму, но это недействительно.

Классы CSS

Угловой также предоставляет некоторые классы CSS для форм и входных данных в зависимости от их состояния

Учебный класс	Описание
`ng-touched`	Поле было тронуто
`ng-untouched`	Поле не тронуто
`ng-pristine`	Поле не было изменено
`ng-dirty`	Поле изменено
`ng-valid`	Поле действительное
`ng-invalid`	Поле недействительно.

Вы можете использовать эти классы для добавления стилей в свои формы

```
input.ng-invalid {
  background-color: crimson;
}
input.ng-valid {
  background-color: green;
}
```

ngMessages

`ngMessages` используется для улучшения стиля отображения сообщений проверки в представлении.

Традиционный подход

Прежде чем `ngMessages` , мы обычно показываем сообщения проверки с использованием предустановленных директив `ng-class` Angular. Этот подход был мусором и `repetitive` задачей.

Теперь, используя `ngMessages` мы можем создавать собственные собственные сообщения.

пример

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";
});
```

Проверка пользовательской формы

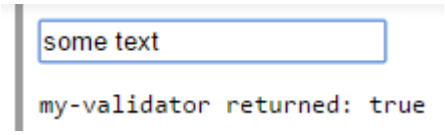
В некоторых случаях базовой проверки недостаточно. Угловая поддержка пользовательской проверки, добавляющая функции `$validators` объекту `$validators` `ngModelController` на `ngModelController` :

```
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;
        };
      }
    };
```

Валидатор определяется как директива, которая требует `ngModel` , поэтому для применения валидатора просто добавьте настраиваемую директиву в элемент управления формы ввода.

```
<form name="form">
  <input type="text"
         ng-model="model"
         name="model"
         my-validator>
  <pre ng-bind="'my-validator returned: ' + form.model.$valid"></pre>
</form>
```

И `my-validator` не обязательно должен применяться для управления собственной формой. Это могут быть любые элементы, если они являются `ng-model` в своих атрибутах. Это полезно, если у вас есть пользовательский компонент сборки ui.



Вложенные формы

Иногда желательно встраивать формы, чтобы логически группировать элементы управления и логические элементы на странице. Однако формы HTML5 не должны быть вложенными. Вместо этого угловые принадлежности представляют собой `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>
```

Каждая часть формы вносит вклад в состояние общей формы. Поэтому, если один из входов `myInput1` был отредактирован и является `$dirty` , его содержащая форма также будет `$dirty` . Это каскадирует каждую содержащую форму, поэтому оба `myNestedForm` и `myForm` будут `$dirty` .

Асинхронные валидаторы

Асинхронные валидаторы позволяют проверять информацию о форме на вашем бэкэнд (используя $ http).

Эти валидаторы необходимы, когда вам нужно получить доступ к хранимой на сервере информации, которую вы не можете иметь на своем клиенте по разным причинам, например, к таблице пользователей и другой информации о базе данных.

Чтобы использовать асинхронные валидаторы, вы `$asyncValidators` к `ng-model` вашего `input` и определяете функции обратного вызова для свойства `$asyncValidators` .

Пример:

В следующем примере проверяется, существует ли предоставленное имя, бэкэнд будет возвращать статус, который отклонит обещание, если имя уже существует или если оно не

было предоставлено. Если имя не существует, оно вернет разрешенное обещание.

```
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
     }
};
```

Теперь каждый раз, когда `ng-model` ввода изменяется, эта функция запускается и возвращает обещание с результатом.

Прочитайте Проверка формы онлайн: https://riptutorial.com/ru/angularjs/topic/3979/проверка-формы

глава 40: Профилирование и производительность

Examples

7 Простые улучшения производительности

1) Используйте ng-repeat экономно

Использование `ng-repeat` во взглядах обычно приводит к низкой производительности, особенно при наличии вложенных `ng-repeat` .

Это супер медленно!

```
<div ng-repeat="user in userCollection">
  <div ng-repeat="details in user">
    {{details}}
  </div>
</div>
```

Старайтесь избегать вложенных повторов как можно больше. Одним из способов улучшить производительность `ng-repeat` является использование `track by $index` (или другому полю id). По умолчанию `ng-repeat` отслеживает весь объект. С помощью `track by` Angular наблюдает за объектом только по `$index` или object.

```
<div ng-repeat="user in userCollection track by $index">
  {{user.data}}
</div>
```

Используйте другие подходы, такие как разбиение на страницы , виртуальные свитки , бесконечные свитки или limitTo: начинайте, когда это возможно, чтобы избежать повторения больших коллекций.

---

2) Привязать один раз

Угловое имеет двунаправленную привязку данных. Это связано с тем, что он слишком медленный, если использовать слишком много.

Более низкая производительность

```
<!-- Default data binding has a performance cost -->
<div>{{ my.data }}</div>
```

Более высокая производительность (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>
```

Используя нотацию «привязать один раз», Angular ожидает, что значение будет стабилизироваться после первой серии циклов дайджеста. Угловой будет использовать это значение в DOM, а затем удалить всех наблюдателей, чтобы он стал статическим значением и больше не привязан к модели.

`{{}}` Намного медленнее.

Это `ng-bind` является директивой и помещает наблюдателя в переданную переменную. Таким образом, `ng-bind` будет применяться только тогда, когда переданное значение действительно изменится.

Скобки, с другой стороны, будут грязно проверены и обновлены в каждом `$digest` , даже если это не обязательно.

---

3) Функции области и фильтры требуют времени

У AngularJS есть цикл дайджеста. Все ваши функции находятся в режиме просмотра, и фильтры выполняются каждый раз, когда цикл дайджест выполняется. Цикл дайджеста будет выполняться всякий раз, когда модель обновляется, и это может замедлить ваше приложение (фильтр может быть удален несколько раз до загрузки страницы).

Избегайте этого:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Лучший подход

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Где контроллер может быть:

```
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) Наблюдатели

Наблюдатели сильно понижают производительность. С большим количеством наблюдателей цикл digest займет больше времени, и пользовательский интерфейс будет замедляться. Если наблюдатель обнаруживает изменения, он начнет цикл дайджеста и повторно отобразит представление.

Существует три способа ручного просмотра переменных изменений в Angular.

`$watch()` - часы для изменения стоимости

`$watchCollection()` - часы для изменений в коллекции (часы больше, чем обычные `$watch`)

`$watch(..., true)` - **Избегайте этого** как можно больше, он будет выполнять «глубокие часы» и `watchCollection` производительность (часы больше, чем `watchCollection`)

Обратите внимание: если вы привязываете переменные в представлении, вы создаете новые часы - используйте `{{::variable}}` чтобы предотвратить создание часов, особенно в циклах.

В результате вам нужно отслеживать, сколько наблюдателей вы используете. Вы можете подсчитать наблюдателей с помощью этого сценария (кредит для @Words Like Jared Количество наблюдателей)

```
(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

Эти функции очень похожи в поведении. **`ng-if`** удаляет элементы из DOM, в то время как **`ng-show`** только скрывает элементы, но сохраняет все обработчики. Если у вас есть части кода, который вы не хотите показывать, используйте `ng-if` .

Это зависит от типа использования, но часто один из них более подходит, чем другой.

- Если элемент не нужен, используйте `ng-if`
- Чтобы быстро включить / выключить, используйте `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>
```

Если есть сомнения - используйте `ng-if` и test!

---

6) Отключить отладку

По умолчанию директивы bind и области действия оставляют дополнительные классы и разметку в коде, чтобы помочь с различными инструментами отладки. Отключение этой опции означает, что вы больше не выполняете эти различные элементы во время цикла дайджеста.

```
angular.module('exampleApp', []).config(['$compileProvider', function ($compileProvider) {
    $compileProvider.debugInfoEnabled(false);
}]);
```

---

7) Используйте инъекцию зависимостей, чтобы

разоблачить ваши ресурсы

Dependency Injection - это шаблон разработки программного обеспечения, в котором объекту присваиваются его зависимости, а не сам объект, создающий их. Речь идет об удалении жестко закодированных зависимостей и их изменении при необходимости.

Вы можете задаться вопросом о стоимости выполнения, связанной с таким синтаксическим разбором всех инъекционных функций. Угловые заботятся об этом путем кэширования $ inject-свойства после первого раза. Так что это не происходит каждый раз, когда нужно вызвать функцию.

PRO TIP: Если вы ищете подход с наилучшей производительностью, перейдите к подстроке аннотаций $ injection. Этот подход полностью исключает синтаксический анализ определения функции, поскольку эта логика завершается в следующей проверке функции аннотации: if (! ($ Inject = fn. $ Inject)). Если $ inject уже доступен, синтаксический анализ не требуется!

```
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 TIP 2: вы можете добавить директиву `ng-strict-di` в том же элементе, что и `ng-app` чтобы перейти в строгий режим DI, который будет вызывать ошибку всякий раз, когда служба пытается использовать неявные аннотации. Пример:

```
<html ng-app="DemoApp" ng-strict-di>
```

Или если вы используете ручную загрузку:

```
angular.bootstrap(document, ['DemoApp'], {
    strictDi: true
});
```

Bind Once

Угловая имеет репутацию за то, что у нее потрясающая двунаправленная привязка данных. По умолчанию «Угловая» непрерывно синхронизирует значения, связанные между моделью и компонентами представления, при любых изменениях данных времени как в модели, так и в представлении.

Это связано со стоимостью немного медленной, если ее использовать слишком много. Это приведет к большему результату:

Плохая производительность: `{{my.data}}`

Добавьте два двоеточия `::` перед именем переменной используйте одноразовую привязку. В этом случае значение обновляется только после определения my.data. Вы явно указываете, что не следите за изменениями данных. Угловой не будет выполнять никаких проверок значений, в результате чего меньшее количество выражений оценивается в каждом цикле дайджеста.

Хорошие примеры производительности с использованием одноразовой привязки

```
{{::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>
```

Примечание. Однако это устраняет двунаправленную привязку данных для `my.data` , поэтому всякий раз, когда это поле изменяется в вашем приложении, оно не будет автоматически отражено в представлении. Поэтому **используйте его только для значений, которые не будут меняться на протяжении всего срока действия вашего приложения** .

Функции и фильтры области

У AngularJS есть цикл дайджеста, и все ваши функции в представлении и фильтры выполняются каждый раз, когда выполняется цикл дайджест. Цикл дайджеста будет выполняться всякий раз, когда модель обновляется, и это может замедлить ваше приложение (фильтр может быть удален несколько раз, прежде чем страница будет загружена).

Вам следует избегать этого:

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.calculateMe()}}</p>
  <p>{{calc.data | heavyFilter}}</p>
</div>
```

Лучший подход

```
<div ng-controller="bigCalulations as calc">
  <p>{{calc.preCalculatedValue}}</p>
  <p>{{calc.data | lightFilter}}</p>
</div>
```

Где контрольный образец:

```
.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
});
```

Зрителей

Наблюдателям необходимо следить за некоторой стоимостью и обнаруживать, что это значение изменено.

После вызова `$watch()` или `$watchCollection` новый наблюдатель добавит во внутреннюю коллекцию наблюдателей в текущей области.

Итак, что такое наблюдатель?

Watcher - простая функция, которая вызывается для каждого цикла дайджеста и возвращает некоторое значение. Угловой проверяет возвращаемое значение, если оно не совпадает с предыдущим вызовом - будет выполнен обратный вызов, который был передан во втором параметре для функции `$watch()` или `$watchCollection` .

```
(function() {
  angular.module("app", []).controller("ctrl", function($scope) {
    $scope.value = 10;
    $scope.$watch(
      function() { return $scope.value; },
      function() { console.log("value changed"); }
    );
  }
})();
```

Наблюдатели - убийцы производительности. Чем больше наблюдателей у вас есть, тем дольше они берут, чтобы сделать цикл дайджеста, более медленный пользовательский интерфейс. Если наблюдатель обнаруживает изменения, он начнет цикл дайджеста (пересчет на весь экран)

Существует три способа ручного просмотра для переменных изменений в Угловом.

`$watch()` - просто наблюдает за изменениями стоимости

`$watchCollection()` - часы для изменений в коллекции (часы больше, чем обычные $ watch)

`$watch(..., true)`

- **Избегайте этого** как можно больше, он будет выполнять «глубокие часы» и убьет производительность (часы больше, чем watchCollection)

Обратите внимание: если вы привязываете переменные в представлении, вы создаете новых наблюдателей - используйте `{{::variable}}` чтобы не создавать наблюдателей, особенно в циклах

В результате вам нужно отслеживать, сколько наблюдателей вы используете. Вы можете подсчитать наблюдателей с помощью этого сценария (кредит на @Words Like Jared). Как подсчитать общее количество часов на странице?

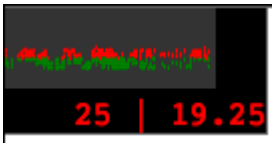
```
(function() {
  var root = angular.element(document.getElementsByTagName("body")),
      watchers = [];

  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);

})();
```

Если вы не хотите создавать свой собственный скрипт, есть утилита с открытым исходным кодом, называемая ng-stats, которая использует диаграмму реального времени, встроенную в страницу, чтобы дать вам представление о количестве часов, которыми управляет Angular, а также частоты и продолжительности циклов дайджеста с течением времени. Утилита предоставляет глобальную функцию с именем `showAngularStats` которую вы можете вызвать, чтобы настроить, как вы хотите, чтобы график работал.

```
showAngularStats({
  "position": "topleft",
```

```
  "digestTimeThreshold": 16,
  "autoload": true,
  "logDigest": true,
  "logWatches": true
});
```

В приведенном выше примере приведена следующая диаграмма на странице (интерактивная демонстрация).



ng-if vs ng-show

Эти функции очень похожи в поведении. Разница в том, что `ng-if` удаляет элементы из DOM. Если есть большие части кода, которые не будут показаны, то `ng-if` - путь. `ng-show` только скроет элементы, но сохранит все обработчики.

нг-если

Директива ngIf удаляет или воссоздает часть дерева DOM на основе выражения. Если выражение, присвоенное ngIf, оценивается как ложное значение, тогда элемент удаляется из DOM, иначе клон элемента снова вводится в DOM.

нг-шоу

Директива ngShow показывает или скрывает данный HTML-элемент на основе выражения, предоставляемого атрибуту ngShow. Элемент отображается или скрывается удалением или добавлением класса CSS ng-hide в элемент.

пример

```
<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>
```

Заключение

Это зависит от типа использования, но часто один из них более подходит, чем другой (например, если в 95% случаев элемент не нужен, используйте `ng-if` , если вам нужно переключить видимость элемента DOM, используйте `ng-show`).

Если вы сомневаетесь, используйте `ng-if` и test!

Примечание : `ng-if` создает новую изолированную область, в то время как `ng-show` и `ng-hide` - нет. Используйте `$parent.property` если свойство родительской области недоступно в нем.

Откажитесь от своей модели

```
<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>
```

В приведенном выше примере мы устанавливаем значение дебюта 1000 миллисекунд, которое составляет 1 секунду. Это значительная задержка, но предотвратит неоднократное измельчение ввода `ng-model` со многими циклами `$digest` .

Используя debounce в ваших полях ввода и где-либо еще, где мгновенное обновление не требуется, вы можете существенно увеличить производительность ваших приложений с угловым выражением. Вы не только можете задерживаться по времени, но также можете задерживать время срабатывания. Если вы не хотите обновлять свою ng-модель при каждом нажатии клавиши, вы также можете обновить ее.

Всегда отменять регистрацию слушателей, зарегистрированных в других областях, отличных от текущей области

Вы всегда должны отменить регистрацию областей, отличных от текущей, как показано ниже:

```
//always deregister these
$rootScope.$on(...);
$scope.$parent.$on(...);
```

Вам не нужно отменять регистрацию списков в текущей области, поскольку угловые позаботятся об этом:

```
//no need to deregister this
$scope.$on(...);
```

`$rootScope.$on` слушателях останется в памяти, если вы перейдете к другому контроллеру. Это приведет к утечке памяти, если контроллер выпадет из области видимости.

не

```
angular.module('app').controller('badExampleController', badExample);
badExample.$inject = ['$scope', '$rootScope'];

function badExample($scope, $rootScope) {
    $rootScope.$on('post:created', function postCreated(event, data) {});
}
```

Делать

```
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();
    });
}
```

Прочитайте Профилирование и производительность онлайн:
https://riptutorial.com/ru/angularjs/topic/1921/профилирование-и-производительность

глава 41: Профилирование производительности

Examples

Все о профилировании

Что такое профилирование?

По определению Профилирование представляет собой форму динамического анализа программ, который измеряет, например, пространство (память) или временную сложность программы, использование конкретных инструкций или частоту и продолжительность вызовов функций.

Почему это необходимо?

Профилирование важно, потому что вы не можете эффективно оптимизировать, пока не знаете, что тратит ваша программа большую часть времени. Без измерения времени выполнения программы (профилирования) вы не узнаете, действительно ли вы ее улучшили.

Инструменты и методы:

1. Встроенные инструменты разработчика Chrome

   Это включает в себя полный набор инструментов для профилирования. Вы можете углубиться в поиск узких мест в вашем файле javascript, css-файлах, анимациях, потреблении процессора, утечке памяти, сети, безопасности и т. Д.

   Сделайте запись временной шкалы и посмотрите подозрительно долгое событие «Оценить сценарий». Если вы их найдете, вы можете включить JS Profiler и повторно записать свою запись, чтобы получить более подробную информацию о том, какие функции JS были вызваны и сколько времени они занимали. Прочитайте больше...

2. FireBug (использование с Firefox)

3. Dynatrace (использование с IE)

4. Batarang (использование с Chrome)

   Это устаревшее дополнение для браузера Chrome, хотя оно стабильно и может использоваться для мониторинга моделей, производительности, зависимостей для углового приложения. Он отлично подходит для применения в малых масштабах и может дать вам представление о том, что переменная области поддерживается на

разных уровнях. В нем рассказывается о активных наблюдателях, часах выражений, коллекциях часов в приложении.

5. Наблюдатель (используйте с Chrome)

   Хороший и упрощенный пользовательский интерфейс, чтобы подсчитать количество наблюдателей в приложении «Угловое».

6. Используйте следующий код, чтобы вручную узнать количество наблюдателей в вашем угловом приложении (кредит на @Words Like Jared Количество наблюдателей)

```
(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. Существует несколько онлайн-инструментов / веб-сайтов, которые облегчают широкий спектр функций для создания профиля вашего приложения.

   Один из таких сайтов: https://www.webpagetest.org/

   С помощью этого вы можете запустить бесплатный тест скорости сайта из нескольких мест по всему миру с использованием реальных браузеров (IE и Chrome) и на реальных скоростях подключения к потребителю. Вы можете запускать простые тесты или выполнять расширенное тестирование, включая многошаговые транзакции, захват видео, блокирование контента и многое другое.

Следующие шаги:

Выполнено с профилированием. Это только приведет вас на полпути вниз по дороге. Следующей задачей является фактическое превращение ваших данных в элементы действия для оптимизации вашего приложения. Посмотрите эту документацию о том, как вы можете улучшить производительность своего углового приложения с помощью простых трюков.

Счастливое кодирование :)

Прочитайте Профилирование производительности онлайн:
https://riptutorial.com/ru/angularjs/topic/7033/профилирование-производительности

глава 42: Распечатать

замечания

Создайте класс ng-hide в файле css. ng-show / hide не будет работать без класса.

Подробнее

Examples

Служба печати

Обслуживание:

```
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
        }
    }
]);
```

Контроллер:

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

Прочитайте Распечатать онлайн: https://riptutorial.com/ru/angularjs/topic/6750/распечатать

глава 43: Самостоятельная или эта переменная в контроллере

Вступление

Это объяснение общей картины и, как правило, считается лучшей практикой, которую вы можете увидеть в коде AngularJS.

Examples

Понимание цели самопеременной

При использовании «контроллера в качестве синтаксиса» вы должны дать вашему контроллеру псевдоним в html при использовании директивы ng-controller.

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

Затем вы можете получить доступ к свойствам и методам из ***основной*** переменной, представляющей наш экземпляр контроллера. Например, давайте посмотрим на свойство ***приветствия*** нашего контроллера и отобразим его на экране:

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

Теперь в нашем контроллере нам нужно установить значение свойства приветствия нашего экземпляра контроллера (в отличие от $ scope или чего-то еще):

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

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

Для того , чтобы правильно иметь дисплей HTML нам нужно установить свойство приветственного на ***этом*** внутри нашего тела контроллера. Я создаю промежуточную переменную с именем ***self,*** которая содержит ссылку на это. Зачем? Рассмотрим этот код:

```
angular
.module('ngNjOrg')
.controller('ForgotPasswordController',function ($log) {
```

```
  var self = this;

  self.greeting = "Hello World";

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

})
```

В этом вышеприведенном коде вы можете ожидать, что текст на экране будет обновляться при *вызове* метода *itsLate* , но на самом деле это не так. JavaScript использует правила определения уровня функции, поэтому «this» внутри itsLate относится к чему-то другому, что «это» вне тела метода. Однако мы можем получить желаемый результат, если мы используем переменную ***self*** :

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

  self.greeting = "Hello World";

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

})
```

Это красота использования «собственной» переменной в контроллерах - вы можете получить доступ к ней в любом месте своего контроллера и всегда можете быть уверены, что она ссылается на ваш экземпляр контроллера.

Прочитайте Самостоятельная или эта переменная в контроллере онлайн: https://riptutorial.com/ru/angularjs/topic/8867/самостоятельная-или-эта-переменная-в-контроллере

глава 44: Сервисы

Examples

Как создать службу

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

    var service = {
        number: 0
    };

    return service;
   });
```

Как пользоваться услугой

```
    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;
        })
```

В шаблоне с использованием этого контроллера вы должны написать:

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

или же

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

Конечно, в реальном коде вы будете взаимодействовать с сервисом, используя методы на контроллере, которые, в свою очередь, делегируют услугу. Приведенный выше пример просто увеличивает значение счетчика каждый раз, когда контроллер используется в шаблоне.

Услуги в Angularjs - это одиночные игры:

Службы - это одноэлементные объекты, которые создаются только один раз на приложение (с помощью инжектора) и ленивы загружаются (создаются только при

необходимости).

> Singleton - это класс, который позволяет только создать один экземпляр для себя - и дает простой и легкий доступ к указанному экземпляру. Как указано здесь

Создание сервиса с использованием угловой.фабрики

Сначала определите сервис (в этом случае он использует заводскую модель):

```
.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;
})
```

Теперь вы можете использовать службу для обмена данными между контроллерами:

```
.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 - санировать и отображать контент и ресурсы в шаблонах

> $ sce («Строгое контекстное экранирование») - это встроенная угловая служба, которая автоматически дезинфицирует контент и внутренние источники в шаблонах.

для ввода **внешних** источников и **необработанного HTML** в шаблон требуется ручная упаковка `$sce` .

В этом примере мы создадим простой фильтр сателлитов $ sce: `.

демонстрация

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

Использование в шаблоне

```
<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>
```

Как создать службу с зависимостями с помощью «синтаксиса массива»

```
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;
   }]);
```

Регистрация службы

Наиболее распространенный и гибкий способ создания сервиса использует фабрику API углового.module:

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

```
});
```

Функция фабрики услуг может быть либо функцией, либо массивом, точно так же, как мы создаем контроллеры:

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

Чтобы разоблачить метод нашей службы, мы можем поместить его как атрибут в объект службы.

```
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');
    }
};
```

Разница между сервисом и фабрикой

1) Услуги

Служба - это функция- `constructor` которая вызывается один раз во время выполнения с `new` , точно так же, как то, что мы будем делать с простым javascript, с той лишь разницей, что `AngularJs` вызывает `new` за кулисами.

В случае обслуживания есть одно правило большого пальца

1. Услуги - это конструкторы, которые называются `new`

Давайте посмотрим простой пример, в котором мы зарегистрировали бы службу, которая использует службу `$http` для получения сведений о студенте и использования ее в контроллере

```
function StudentDetailsService($http) {
```

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

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

Мы просто вводим эту услугу в контроллер

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

Когда использовать?

Используйте `.service()` везде, где вы хотите использовать конструктор. Обычно он используется для создания общедоступного API, как и для `getStudentDetails()` . Но если вы не хотите использовать конструктор и хотите использовать простой шаблон API, то в `.service()` нет большой гибкости.

2) Фабрика

Несмотря на то, что мы можем достичь всего, что использует `.factory()` которое мы использовали бы, используя `.services()` , он не делает `.factory()` «таким же, как« `.service()` . Он намного более мощный и гибкий, чем `.service()`

А `.factory()` - шаблон проектирования, который используется для возврата значения.

В случае фабрик есть два правила большого пальца

1. Возвращаемые значения фабрик
2. Заводы (могут) создавать объекты (Любой объект)

Давайте посмотрим некоторые примеры того, что мы можем сделать, используя `.factory()`

Литералы возвращающихся объектов

Давайте посмотрим пример, когда фабрика используется для возврата объекта с использованием шаблона базового модуля раскрытия

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

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

Использование внутри контроллера

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

Возвратные закрытия

Что такое закрытие?

Закрытие - это функции, которые относятся к переменным, которые используются локально, но определены в охватывающей области.

Ниже приведен пример закрытия

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

«Замечательная» часть состоит в том, что она может получить доступ к `name` которое находится в родительской области.

Давайте воспользуемся приведенным выше примером закрытия внутри `.factory()`

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

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

Использование внутри контроллера

```
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);
```

Создание конструкторов / экземпляров

`.service()` создает конструкторы с вызовом `new` как показано выше. `.factory()` также может создавать конструкторы с призывом к `new`

Давайте посмотрим пример того, как достичь этого.

```
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);
```

Использование внутри контроллера

```
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);
```

Прочитайте Сервисы онлайн: https://riptutorial.com/ru/angularjs/topic/1486/сервисы

глава 45: События

параметры

параметры	Типы значений
событие	Object {name: "eventName", targetScope: Scope, defaultPrevented: false, currentScope: ChildScope}
арг	данные, которые были переданы вместе с выполнением события

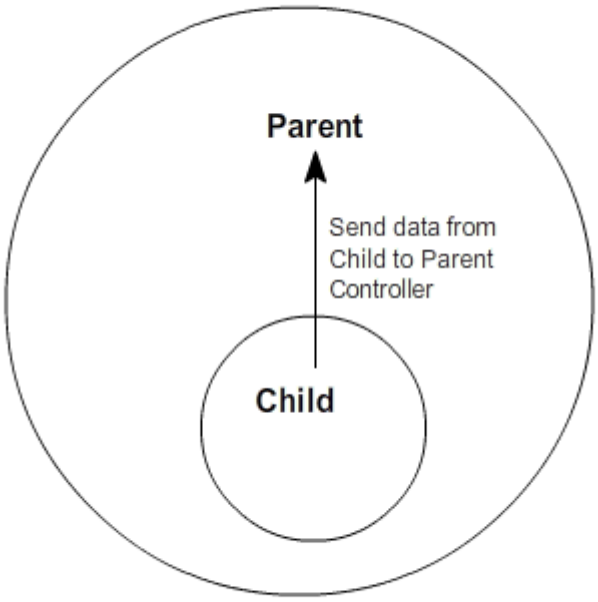
Examples

Использование системы угловых событий

$ Объем. $ Испускают

Используя `$scope.$emit` загонит имя события вверх по иерархии областей и уведомит об этом в области `$scope` . Жизненный цикл события начинается с области, в которой был вызван `$emit` .

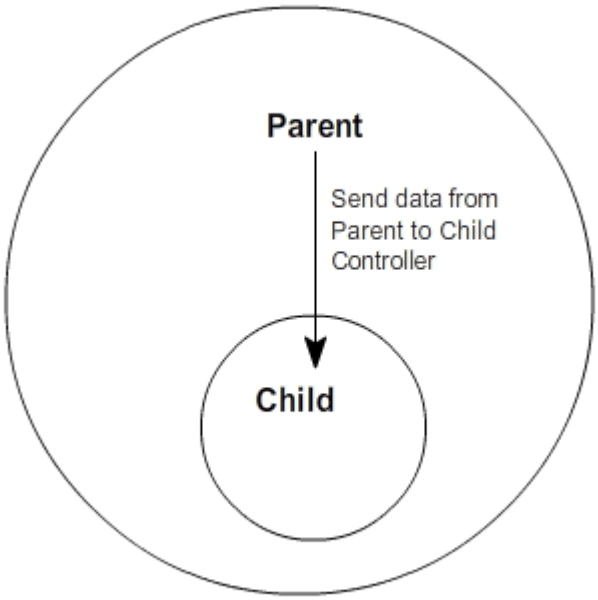
Рабочий каркас:



$ Объем. $ Широковещательный

Использование `$scope.$broadcast` приведет к сбою события в `$scope` . Мы можем слушать эти события, используя `$scope.$on`

Рабочий каркас:



Синтаксис:

```
// 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'
});
```

Вместо `$scope` вы можете использовать `$rootScope` , в этом случае ваше событие будет доступно во всех контроллерах вне зависимости от области контроля

Чистое зарегистрированное событие в

Причина очистки зарегистрированных событий, потому что даже контроллер был уничтожен, обработка зарегистрированного события все еще жива. Таким образом, код будет работать как неожиданно.

```
// 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();
});
```

Использование и значение

Эти события могут использоваться для связи между двумя или более контроллерами.

`$emit` отправляет событие вверх по иерархии областей, а `$broadcast` отправляет событие вниз всем дочерним областям. Это было прекрасно объяснено здесь .

Взаимодействие между контроллерами может быть в основном двух типов:

---

1. Когда контроллеры имеют отношения «Родитель-ребенок». (мы можем в основном использовать `$scope` в таких сценариях)

---

2. Когда контроллеры не независимы друг от друга и необходимы для информирования о деятельности друг друга. (мы можем использовать `$rootScope` в таких сценариях)

например: для любого веб-сайта электронной коммерции предположим, что у нас есть `ProductListController` (который контролирует страницу с листингом продукта при нажатии на любую марку) и `CartController` (для управления элементами корзины). Теперь, когда мы нажимаем кнопку « **Добавить в корзину**» , она также должна быть `CartController` в `CartController` , так что она может отражать количество / количество элементов корзины в навигационной панели сайта. Этого можно добиться с помощью `$rootScope` .

С `$scope.$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>
```

C `$scope.$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>
```

Всегда отменить регистрацию $ rootScope. $ На слушателях в области $ destory event

$ rootScope. $ на слушателях останется в памяти, если вы перейдете к другому контроллеру. Это приведет к утечке памяти, если контроллер выпадет из области видимости.

не

```
angular.module('app').controller('badExampleController', badExample);

badExample.$inject = ['$scope', '$rootScope'];
function badExample($scope, $rootScope) {

    $rootScope.$on('post:created', function postCreated(event, data) {});

}
```

Делать

```
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();
    });

}
```

Прочитайте События онлайн: https://riptutorial.com/ru/angularjs/topic/1922/события

глава 46: Совместное использование данных

замечания

Очень распространенный вопрос при работе с Angular - это обмен данными между контроллерами. Использование службы является наиболее частым ответом, и это простой пример, демонстрирующий фабричный шаблон для обмена любым типом данных между двумя или несколькими контроллерами. Поскольку это ссылка на общий объект, обновление в одном контроллере будет немедленно доступно во всех других контроллерах, использующих службу. Обратите внимание, что как сервис, так и завод и оба поставщика .

Examples

Использование ngStorage для обмена данными

Во-первых, **включите** источник **ngStorage** в ваш index.html.

Пример ввода `ngStorage` src:

```
<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` дает вам 2 хранилища, а именно: `$localStorage` и `$sessionStorage` . Вам нужно будет потребовать ngStorage и Inject services.

Предположим, что если `ng-app="myApp"` , то вы будете вводить `ngStorage` следующим образом:

```
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`

и `$sessionStorage` доступны глобально через любые контроллеры, если вы вводите эти службы в контроллеры.

Вы также можете использовать `localStorage` и `sessionStorage` для `HTML5` . Однако, используя `HTML5 localStorage` , вам потребуется сериализовать и десериализовать объекты перед их использованием или сохранением.

Например:

```
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"));
```

Обмен данными с одного контроллера на другой с помощью службы

Мы можем создать `service` для `set` и `get` данных между `controllers` а затем ввести эту службу в функцию контроллера, где мы хотим ее использовать.

Обслуживание :

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

Контроллеры:

```
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 !!

}]);
```

Здесь мы видим, что `myCtrl1` используется для `setting` данных, а `myCtrl2` используется для

`getting` данных. Таким образом, мы можем обмениваться данными с одного контроллера на другой contrller, как это.

Прочитайте Совместное использование данных онлайн:
https://riptutorial.com/ru/angularjs/topic/1923/совместное-использование-данных

глава 47: Угловой MVC

Вступление

В **AngularJS** шаблон **MVC** реализован в JavaScript и HTML. Представление определено в HTML, а модель и контроллер реализованы в JavaScript. Существует несколько способов объединения этих компонентов в AngularJS, но простейшая форма начинается с представления.

Examples

Статический просмотр с контроллером

демонстрация mvc

Привет, мир

Определение функции контроллера

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

Добавление информации в модель

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

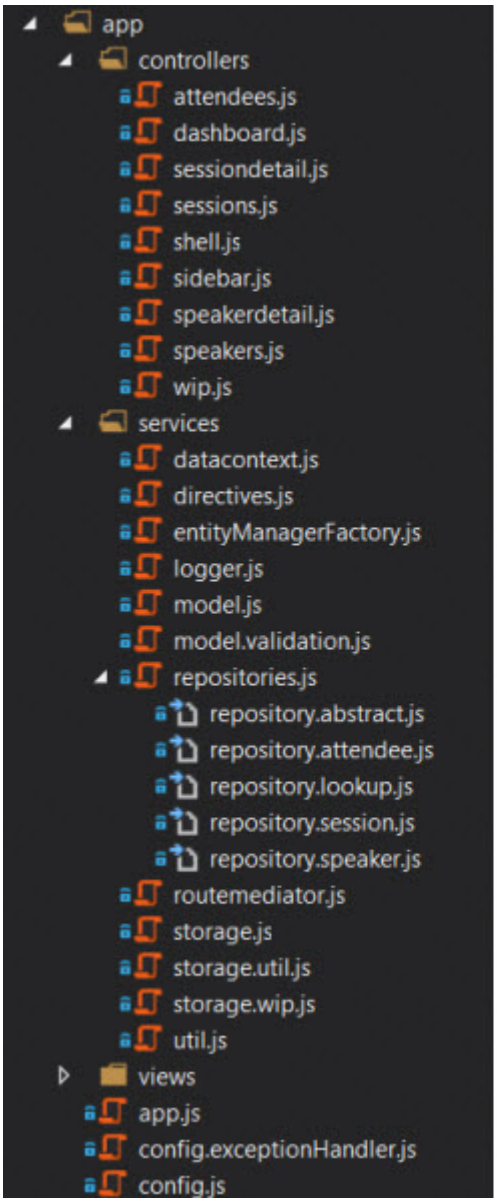
Прочитайте Угловой MVC онлайн: https://riptutorial.com/ru/angularjs/topic/8667/угловой-mvc

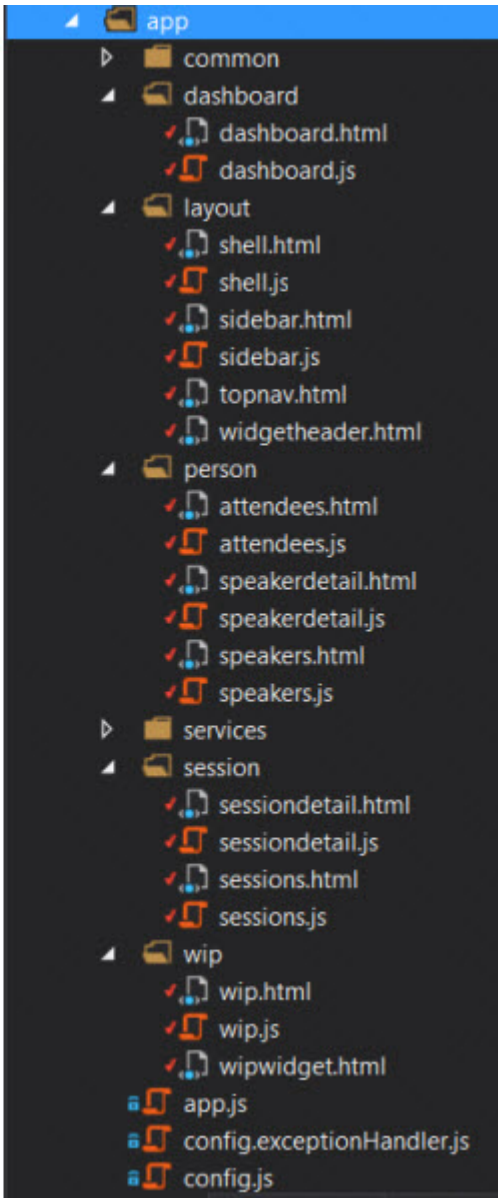
глава 48: Угловой проект - Структура каталогов

Examples

Структура каталога

Обычный вопрос среди новых программистов на уголках - «Какова должна быть структура проекта?». Хорошая структура помогает в развитии масштабируемого приложения. Когда мы начинаем проект, у нас есть два варианта: **Sort By Type** (слева) и **Sort By Feature** (справа). Во-вторых, лучше, особенно в крупных приложениях, проект становится намного проще управлять.





Сортировать по типу (слева)

Приложение организовано типом файлов.

- **Преимущество** - хорошо для небольших приложений, поскольку программисты только начинают использовать Angular и легко конвертируются во второй метод.
- **Недостаток.** Даже для небольших приложений начинает усложняться поиск определенного файла. Например, представление и его контроллер находятся в двух отдельных папках.

Сортировать по функции (справа)

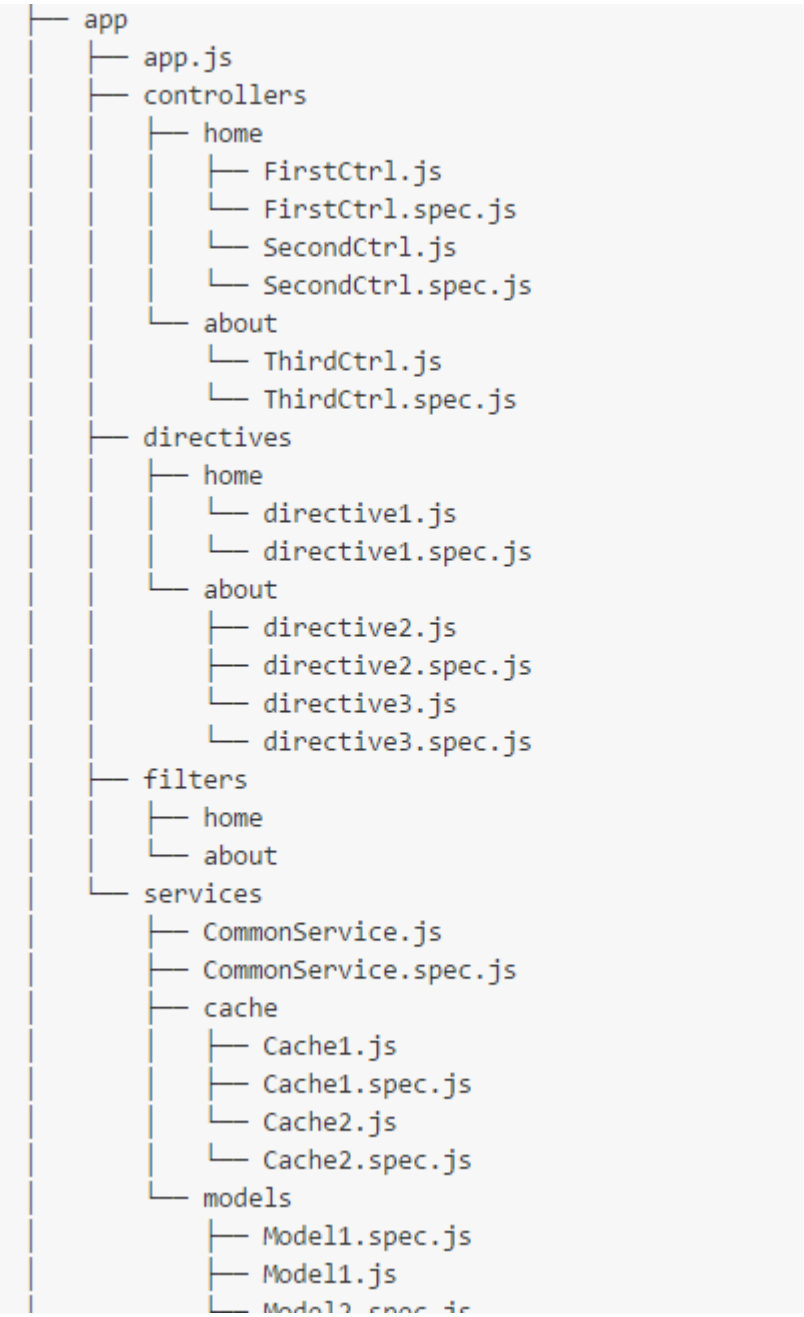
Предлагаемый метод организации, в котором поданная информация сортируется по типу признаков.

Все представления макетов и контроллеры входят в папку макета, содержимое администратора идет в папке администратора и так далее.

- **Преимущество.** При поиске раздела кода, определяющего определенную функцию, все находится в одной папке.
- **Недостаток** - услуги немного отличаются, поскольку они «обслуживают» многие функции.

Вы можете больше узнать об этом на Угловой структуре: Рефакторинг для роста

Предложенная файловая структура, объединяющая оба вышеупомянутых метода:



Кредит: Руководство *по* угловому стилю

Прочитайте Угловой проект - Структура каталогов онлайн:
https://riptutorial.com/ru/angularjs/topic/6148/угловой-проект---структура-каталогов

глава 49: Угловые $ области

замечания

Угловое использует **дерево** областей для привязки логики (от контроллеров, директив и т. Д.) К виду и является основным механизмом обнаружения изменений в AngularJS. Более подробную ссылку для областей можно найти в docs.angularjs.org

Корень дерева доступен так же, как через инъекционную службу **$ rootScope** . Все дочерние $ scopes наследуют методы и свойства их родительской области $ scope, что позволяет детям получить доступ к методам без использования Angular Services.

Examples

Основной пример наследования $ 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>
```

В этом примере директива ng-repeat создает новую область для каждого из вновь созданных детей.

Эти созданные области являются дочерними элементами их родительской области (в данном случае областью, созданной myController), и, следовательно, они наследуют все свои доли, например, человека.

Избегайте наследования примитивных значений

В javascript, присваивая не примитивное значение (например, Object, Array, Function и многие другие), сохраняет ссылку (адрес в памяти) на заданное значение.

Присвоение примитивному значению (String, Number, Boolean или Symbol) двум переменным и изменению одного из них не изменится:

```
var x = 5;
var y = x;
y = 6;
```

```
console.log(y === x, x, y); //false, 5, 6
```

Но с не примитивное значение, поскольку обе переменные просто сохраняя ссылки на тот же объект, изменение одной переменной изменит другой:

```
var x = { name : 'John Doe' };
var y = x;
y.name = 'Jhon';
console.log(x.name === y.name, x.name, y.name); //true, John, John
```

В угловом режиме, когда создается область, ему присваиваются все свойства родителя. Однако изменение свойств впоследствии влияет только на родительскую область, если это не примитивное значение:

```
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>
```

Помните: в Angular scope могут быть созданы разными способами (например, встроенные или настраиваемые директивы или функция `$scope.$new()` Функция `$scope.$new()`), и отслеживание дерева области видимости, вероятно, невозможно.

Использование только не-примитивных значений в качестве свойств области будет держать вас в безопасности (если вам не требуется свойство, чтобы не наследовать, или другие случаи, когда вам известно о наследовании области).

Функция, доступная во всем приложении

Будьте осторожны, этот подход может считаться плохим дизайном для угловых приложений, поскольку он требует от программистов запоминания того, где функции помещаются в дерево областей видимости, и знать о наследовании области. Во многих случаях было бы предпочтительнее вводить услугу (Угловая практика - с использованием наследования сферы по сравнению с инъекцией) .

В этом примере показано, как можно использовать наследование областей для наших нужд и как вы могли бы воспользоваться им, а не лучшие методы проектирования всего приложения.

В некоторых случаях мы могли бы воспользоваться наложением наследования и установить функцию как свойство rootScope. Таким образом, все области приложения (за исключением изолированных областей) наследуют эту функцию и могут быть вызваны из любого места приложения.

```
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>
```

Создание пользовательских событий $ scope

Подобно обычным элементам HTML, для $ scopes возможно иметь собственные события. События $ scope могут быть подписаны следующим образом:

```
 $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
});
```

Если вам нужно отменить регистрацию прослушивателя событий, функция **$ on** вернет функцию открепления. Чтобы продолжить описанный выше пример:

```
var unregisterMyEvent = $scope.$on('my-event', function(event, args) {
    console.log(args); // { custom: 'data' }
    unregisterMyEvent();
});
```

Существует два способа запуска вашего собственного события **$** scope event **$ broadcast** и **$ emit** . Чтобы уведомить родителя (ов) о масштабе конкретного события, используйте **$ emit**

```
$scope.$emit('my-event', { custom: 'data' });
```

Вышеприведенный пример вызовет прослушиватели событий для `my-event` в родительской области и продолжит **доведение** дерева до **$ rootScope** до **тех** пор, пока слушатель не

`stopPropagation` на событие. Только события, вызванные с помощью **$** `stopPropagation` могут вызвать `stopPropagation`

Реверс **$ emit** - это **$ broadcast** , который инициирует прослушиватели событий во всех дочерних областях в дереве областей, которые являются дочерними элементами области, которая называется **$ broadcast** .

```
$scope.$broadcast('my-event', { custom: 'data' });
```

События, инициированные с помощью **$ broadcast,** не могут быть отменены.

Использование функций $ scope

Хотя объявление функции в корневом каталоге $ имеет свои преимущества, мы также можем объявить функцию $ scope любой частью кода, который вводится службой $ scope. Контроллер, например.

контроллер

```
myApp.controller('myController', ['$scope', function($scope){
    $scope.myFunction = function () {
        alert("You are in myFunction!");
    };
}]);
```

Теперь вы можете вызвать свою функцию с контроллера, используя:

```
$scope.myfunction();
```

Или через HTML, который находится под этим конкретным контроллером:

```
<div ng-controller="myController">
    <button ng-click="myFunction()"> Click me! </button>
</div>
```

директива

Угловая директива - это еще одно место, которое вы можете использовать в своей области:

```
myApp.directive('triggerFunction', function() {
    return {
        scope: {
            triggerFunction: '&'
        },
        link: function(scope, element) {
            element.bind('mouseover', function() {
                scope.triggerFunction();
            });
```

```
        }
    };
});
```

И в вашем HTML-коде под одним и тем же контроллером:

```
<div ng-controller="myController">
    <button trigger-function="myFunction()"> Hover over me! </button>
</div>
```

Конечно, вы можете использовать ngMouseover для одного и того же, но что особенно важно в директивах, так это то, что вы можете настроить их так, как вы хотите. И теперь вы знаете, как использовать свои функции $ scope внутри них, быть творческими!

Как вы можете ограничить область действия директивы и почему вы это сделаете?

Область применения используется как «клей», который мы используем для связи между родительским контроллером, директивой и шаблоном директивы. Всякий раз, когда приложение AngularJS загружается, создается объект rootScope. Каждая область, созданная контроллерами, директивами и службами, прототипически унаследована от rootScope.

Да, мы можем ограничить область действия директивы. Мы можем сделать это, создав изолированную область действия для директивы.

> Существует три типа директивных областей:
>
> 1. Область действия: False (Директива использует свою родительскую область)
> 2. Область действия: True (директива получает новую область)
> 3. Область действия: {} (Директива получает новую изолированную область)

Директивы с новой изолированной областью: когда мы создаем новую изолированную область, она не будет унаследована от родительской области. Эта новая область называется изолированной областью, потому что она полностью отделена от родительской области. Зачем? мы должны использовать изолированную область действия: мы должны использовать изолированную область, когда хотим создать настраиваемую директиву, потому что она будет обеспечивать, чтобы наша директива была общей и размещалась в любом месте приложения. Родительский охват не будет мешать сфере действия.

Пример изолированного объема:

```
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'/>"
    };
});
```

Существует 3 типа префиксов, которые AngularJS предоставляет для выделенной области:

1. "@" (Привязка текста / односторонняя привязка)
2. "=" (Привязка прямой модели / двусторонняя привязка)
3. "&" (Привязка к действию / привязка метода)

Все эти префиксы получают данные из атрибутов элемента директивы, например:

```
<div my-directive
  class="directive"
  name="{{name}}"
  reverse="reverseName()"
  color="color" >
</div>
```

Прочитайте Угловые $ области онлайн: https://riptutorial.com/ru/angularjs/topic/3157/угловые--области

глава 50: Угловые обещания с услугой $ q

Examples

Использование $ q.all для обработки нескольких обещаний

Вы можете использовать функцию `$q.all` для вызова метода `.then` после успешного решения массива обещаний и получения данных, с которыми они разрешили.

Пример:

JS:

```
$scope.data = []

$q.all([
   $http.get("data.json"),
   $http.get("more-data.json"),
   ]).then(function(responses) {
     $scope.data = responses.map((resp) => resp.data);
   });
```

Вышеприведенный код запускает `$http.get` 2 раза для данных в локальных json-файлах, когда оба метода `get` завершают, они разрешают связанные с ними обещания, когда все обещания в массиве разрешаются, метод `.then` начинается с обеим обеим данным в `responses` аргумент массива.

Затем данные отображаются таким образом, чтобы их можно было отобразить на шаблоне, затем мы можем показать

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"
}]
```

Использование конструктора $ q для создания обещаний

Функция конструктора `$q` используется для создания обещаний от асинхронных API, которые используют обратные вызовы для возврата результатов.

    $ q (функция (разрешение, отклонение) {...})

Функция-конструктор получает функцию, которая вызывается с двумя аргументами, `resolve` и `reject` которые являются функциями, которые используются для разрешения или отклонения обещания.

Пример 1:

```
function $timeout(fn, delay) {
    return = $q(function(resolve, reject) {
         setTimeout(function() {
              try {
                 let r = fn();
                 resolve(r);
              }
              catch (e) {
                 reject(e);
              }
         }, delay);
     };
}
```

В приведенном выше примере создается обещание API WindowTimers.setTimeout . Рамка AngularJS обеспечивает более сложную версию этой функции. Для использования см. Справочную информацию об API-интерфейсе AngularJS $.

Пример 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);
      }
    });
}
```

Вышеприведенный код, демонстрирующий функцию обобщенного деления, возвращает обещание с результатом или отклонение по причине, если расчет невозможно.

Затем вы можете позвонить и использовать. `.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.
})
```

Операции отсрочки с использованием $ q.defer

Мы можем использовать `$q` для отсрочки операций в будущем, имея в настоящее время ожидающий объект обещания, используя `$q.defer` мы создаем обещание, которое будет либо разрешаться, либо отклоняться в будущем.

Этот метод не эквивалентен использованию конструктора `$q` , так как мы используем `$q.defer` чтобы обещать существующую процедуру, которая может или не может возвращать (или когда-либо возвращал) обещание вообще.

Пример:

```
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. Убедитесь, что вы всегда возвращаете объект `deferred.promise` Или можете `.then` при вызове. `.then`
2. Убедитесь , что вы всегда разрешить или отклонить отложенный объект или `.then` не может работать , и вы рискуете утечку памяти

Использование угловых обещаний с помощью услуги $ q

`$q` - это встроенный сервис, который помогает выполнять асинхронные функции и использовать их возвращаемые значения (или исключение), когда они закончены с обработкой.

`$q` интегрируется с механизмом наблюдения за моделью `$rootScope.Scope` , что означает

более быстрое распространение разрешения или отклонения в ваших моделях и исключение ненужных переходов браузера, что приведет к мерцанию пользовательского интерфейса.

В нашем примере мы называем наш завод `getMyData` , который возвращает объект обещания. Если объект `resolved` , он возвращает случайное число. Если он `rejected` , он возвращает отказ с сообщением об ошибке через 2 секунды.

В Угловом заводе

```
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;
  }
}
```

Использование обещаний по вызову

```
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())
    })
})
```

Чтобы использовать обещания, введите `$q` качестве зависимости. Здесь мы вводили `$q` в фабрику `getMyData` .

```
var defer = $q.defer();
```

Новый экземпляр отложенных построен путем вызова `$q.defer()`

Отложенный объект - это просто объект, который предоставляет обещание, а также связанные с ним методы для решения этого обещания. Он построен с использованием функции `$q.deferred()` и предоставляет три основных метода: `resolve()` , `reject()` и `notify()` .

- `resolve(value)` - разрешает производное обещание со значением.
- `reject(reason)` - отклоняет полученное обещание по причине.
- `notify(value)` - предоставляет обновления статуса выполнения обещания. Это можно назвать несколько раз до того, как обещание будет разрешено или отклонено.

свойства

Связанный объект обещания доступен через свойство prom. `promise` - {Обещание} - объект обещания, связанный с этим отсроченным.

Новый экземпляр обещания создается, когда созданный отложенный экземпляр создается и может быть получен путем вызова `deferred.promise` .

Цель объекта `promise` - предоставить заинтересованным сторонам возможность получить доступ к результатам отложенной задачи, когда она будет завершена.

Методы обещания -

- `then(successCallback, [errorCallback], [notifyCallback])` Независимо от того, когда обещание было или будет разрешено или отклонено, он вызывает асинхронный вызов одного из успешных или ошибочных обратных вызовов, как только результат будет доступен. Обратные вызовы вызываются с помощью одного аргумента: причина или отклонение. Кроме того, обратный вызов уведомления может быть вызван ноль или более раз, чтобы обеспечить индикацию выполнения, прежде чем обещание будет разрешено или отклонено.

- `catch(errorCallback)` - сокращенное `catch(errorCallback)` для обещания. then (null, errorCallback)

- `finally(callback, notifyCallback)` - позволяет вам наблюдать за выполнением или отказом от обещания, но делать это без изменения окончательного значения.

Одной из самых мощных возможностей обещаний является способность объединить их вместе. Это позволяет данным проходить через цепочку и манипулировать и мутировать на каждом шаге. Это продемонстрировано в следующем примере:

Пример 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);
  });
```

Оберните простое значение в обещание, используя $ q.when ()

Если все, что вам нужно, это превратить ценность в обещание, вам не нужно использовать длинный синтаксис, как здесь:

```
//OVERLY VERBOSE
var defer;
defer = $q.defer();
defer.resolve(['one', 'two']);
return defer.promise;
```

В этом случае вы можете просто написать:

```
//BETTER
return $q.when(['one', 'two']);
```

$ q.when и его псевдоним $ q.resolve

Обертывает объект, который может быть значением или (сторонним), а затем - обещанием в обещание в $ q. Это полезно, когда вы имеете дело с объектом, который может или не может быть обещанием, или если обещание исходит от

источника, которому нельзя доверять.

- AngularJS $ q Service API Reference - $ q.when

С выпуском AngularJS v1.4.1

Вы можете также использовать ES6-последовательный псевдоним `resolve`

```
//ABSOLUTELY THE SAME AS when
return $q.resolve(['one', 'two'])
```

Избегайте $ q отложенного анти-шаблона

Избегайте этого Anti-Pattern

```
var myDeferred = $q.defer();

$http(config).then(function(res) {
   myDeferred.resolve(res);
}, function(error) {
   myDeferred.reject(error);
});

return myDeferred.promise;
```

Нет необходимости `$q.defer` обещание с помощью `$q.defer` поскольку служба $ http уже возвращает обещание.

```
//INSTEAD
return $http(config);
```

Просто верните обещание, созданное службой $ http.

Прочитайте Угловые обещания с услугой $ q онлайн:
https://riptutorial.com/ru/angularjs/topic/4379/угловые-обещания-с-услугой---q

глава 51: фильтры

Examples

Ваш первый фильтр

Фильтры - это особый тип функции, который может изменять, как что-то распечатывается на странице, или может использоваться для фильтрации массива или действия `ng-repeat` . Вы можете создать фильтр, вызвав метод `app.filter()` , передав ему имя и функцию. Подробнее о синтаксисе см. Приведенные ниже примеры.

Например, давайте создадим фильтр, который изменит строку на все прописные (по существу, обертку функции `.toUpperCase()` javascript):

```
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();
    };
});
```

Давайте подробнее рассмотрим, что происходит выше.

Во-первых, мы создаем фильтр под названием «toUppercase», который похож на контроллер; `app.filter(...)` . Затем функция фильтра возвращает фактическую функцию фильтра. Эта функция принимает один объект, который является объектом для фильтрации, и должен возвращать отфильтрованную версию объекта.

Примечание. *В этой ситуации мы предполагаем, что объект, передаваемый в фильтр, является строкой и поэтому знает, что он всегда использует фильтр только для строк. При этом может быть сделано дальнейшее улучшение фильтра, которое будет проходить через объект (если это массив), а затем делает каждый элемент, который является строкой в верхнем регистре.*

Теперь давайте использовать наш новый фильтр в действии. Наш фильтр можно использовать двумя способами: либо в угловом шаблоне, либо в виде функции javascript (в качестве введенной угловой ссылки).

Javascript

Просто добавьте угловой объект `$filter` к вашему контроллеру, а затем используйте его для извлечения функции фильтра, используя свое имя.

```
app.controller("MyController", function($scope, $filter){
    this.rawString = "Foo";
    this.capsString = $filter("toUppercase")(this.rawString);
});
```

HTML

Для угловой директивы используйте символ трубы (`|`), а затем имя фильтра в директиве после фактической строки. Например, допустим, у нас есть контроллер под названием `MyController` который имеет строку с именем `rawString` как элемент этого.

```
<div ng-controller="MyController as ctrl">
    <span>Capital rawString: {{ ctrl.rawString | toUppercase }}</span>
</div>
```

Примечание редактора: У *углового есть ряд встроенных фильтров, включая «верхний регистр», а фильтр «toUppercase» предназначен только для демонстрации, чтобы легко показать, как работают фильтры, но вам* не *нужно* создавать *собственную функцию верхнего регистра.*

Пользовательский фильтр для удаления значений

Типичным примером использования фильтра является удаление значений из массива. В этом примере мы передаем массив и удалим все найденные в нем нули, возвращая массив.

```
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;
    };
}
```

Это будет использоваться в HTML как

```
{{listOfItems | removeNulls}}
```

или в контроллере, таком как

```
listOfItems = removeNullsFilter(listOfItems);
```

Пользовательский фильтр для форматирования значений

Другим вариантом использования фильтров является форматирование одного значения. В этом примере мы передаем значение, и нам возвращается соответствующее истинное булевское значение.

```
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;
    };
}
```

Что в HTML будет использоваться следующим образом:

```
{{isAvailable | convertToBooleanValue}}
```

Или в контроллере, например:

```
var available = convertToBooleanValueFilter(isAvailable);
```

Выполнение фильтра в дочернем массиве

Этот пример был выполнен, чтобы продемонстрировать, как вы можете выполнять глубокий фильтр в *дочернем* массиве без необходимости настраиваемого фильтра.

контроллер:

```
(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"
            }
         ]
      }
   ];
  }
})();
```

Посмотреть:

```
<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>
```

Проверьте все DEMO .

Использование фильтров в контроллере или услуге

Вводя `$filter` , любой определенный фильтр в вашем Угловом модуле может использоваться в контроллерах, службах, директивах или даже в других фильтрах.

```
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
}
```

Доступ к фильтруемому списку извне ng-repeat

Иногда вам нужно получить доступ к результатам ваших фильтров из-за пределов `ng-repeat` , возможно, для указания количества элементов, которые были отфильтрованы. Вы можете сделать это, используя синтаксис `as [variablename]` в `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>
```

Прочитайте фильтры онлайн: https://riptutorial.com/ru/angularjs/topic/1401/фильтры

глава 52: Хранение сессии

Examples

Обработка хранилища сеансов через службу с помощью angularjs

Служба хранения сеансов:

Общее заводское обслуживание, которое будет сохранять и возвращать сохраненные данные сеанса на основе ключа.

```
 '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);
        }
    };
}]);
```

В контроллере:

Внедрите зависимость storageService в контроллер, чтобы установить и получить данные из хранилища сеансов.

```
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');

});
```

Прочитайте Хранение сессии онлайн: https://riptutorial.com/ru/angularjs/topic/8201/хранение-

сессии

кредиты

S. No	Главы	Contributors
1	Начало работы с 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 request	CENT1PEDE, jaredsk, Liron Ilayev
3	AngularJS gotchas и ловушки	Alon Eitan, Cosmin Ababei, doctorsherlock, Faruk Yazıcı, ngLover, Phil
4	angularjs с фильтром данных, разбиением на страницы и т. д.	Paresh Maghodiya
5	Grunt tasks	Mikko Viitala
6	HTTP-перехватчик	G Akshay, Istvan Reiter, MeanMan, Mistalis, mnoronha
7	SignalR с AngularJs	Maher
8	UI-маршрутизатор	George Kagan, H.T, Michael P. Bazos, Ryan Hamley, sgarcia.dev
9	Внедрение зависимости	Andrea, badzilla, Gavishiddappa Gadagi, George Kagan, MoLow, Omri Aharon
10	Встроенные вспомогательные функции	MoLow, Pranav C Balan, svarog
11	Встроенные директивы	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
12	дайджест петли прохождение	Alon Eitan, chris, MoLow, prit4fun
13	Декораторы	Mikko Viitala
14	директива ng-class	Dr. Cool
15	Директивы, использующие ngModelController	Nikos Paraskevopoulos
16	Единичные тесты	daniellmb, elliot-j, fracz, Gabriel Pires, Nico, ronapelbaum
17	Использование AngularJS с TypeScript	Parv Sharma, Rohit Jindal
18	Использование встроенных директив	Gourav Garg
19	Как работает привязка данных	Lucas L, Sasank Sunkavalli, theblindprophet
20	Компоненты	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
21	Константы	Sylvain
22	Контроллеры	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
23	Контроллеры с ES6	Bouraoui KACEM
24	Ленивая загрузка	Muli Yulzary

25	Маршрутизация с использованием ngRoute	Alon Eitan, Alvaro Vazquez, camabeh, DotBot, sgarcia.dev, svarog
26	Модули	Alon Eitan, Ankit, badzilla, Bon Macalindong, Matthew Green, Nad Flores, ojus kulkarni, sgarcia.dev, thegreenpizza
27	нг-повтор	Divya Jain, Jim, Sender, zucker
28	нг-просмотр	Aayushi Jain, Manikandan Velayutham, Umesh Shende
29	нг-стиль	Divya Jain, Jim
30	отладка	Aman, AWolf, Vinay K
31	Отличительная служба против фабрики	Deepak Bansal
32	Параметры привязки AngularJS (`=`, `@`, `&` и т. Д.)	Alon Eitan, Lucas L, Makarov Sergey, Nico, zucker
33	Переход к угловому 2+	ShinDarth
34	Подготовьтесь к производству - Grunt	JanisP
35	Пользовательские директивы	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
36	Пользовательские фильтры	doodhwala, Pat, Sylvain
37	Пользовательские фильтры с ES6	Bouraoui KACEM
38	Провайдеры	Mikko Viitala
39	Проверка формы	Alon Eitan, fantarama, garyx, Mikko Viitala, Richard Hamilton, Rohit Jindal, shane, svarog, timbo
40	Профилирование и производительность	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
41	Профилирование производительности	Deepak Bansal
42	Распечатать	ziaulain
43	Самостоятельная или эта переменная в контроллере	It-Z, Jim
44	Сервисы	Abdellah Alaoui, Alvaro Vazquez, AnonDCX, DotBot, elliot-j, Flash, Gavishiddappa Gadagi, Hubert Grzeskowiak, Lex, Nishant123
45	События	CodeWarrior, Nguyen Tran, Rohit Jindal, RyanDawkins, sgarcia.dev, shaN, Shashank Vivek
46	Совместное использование данных	elliot-j, Grundy, Lex, Mikko Viitala, Mistalis, Nix, prit4fun, Rohit Jindal, sgarcia.dev, Sunil Lama
47	Угловой MVC	Ashok choudhary, Gavishiddappa Gadagi, Jim
48	Угловой проект - Структура каталогов	jitender, Liron Ilayev
49	Угловые $ области	Abhishek Maurya, elliot-j, Eugene, jaredsk, Liron Ilayev, MoLow , Prateek Gupta, RamenChef, ryansstack, Tony
50	Угловые обещания с услугой $ q	Alon Eitan, caiocpricci2, ganqqwerty, georgeawg, John F., Muli Yulzary, Praveen Poonia, Richard Hamilton, Rohit Jindal, svarog
51	фильтры	Aeolingamenfel, developer033, Ed Hinchliffe, fracz, gustavohenke, Matthew Green, Nico
52	Хранение сессии	Rohit Jindal