



Бесплатная электронная книга

УЧУСЬ

Angular 2

Free unaffiliated eBook created from
Stack Overflow contributors.

#angular2

.....	1
1: Angular 2	2
.....	2
.....	2
Examples.....	3
.....	3
:	3
.....	3
.....	4
.....	4
, ,	4
Angular 2	6
1.....	6
2.....	6
3.....	9
5.....	9
6.....	10
7.....	11
8.....	11
?.....	11
Visual Studios NPM NODE.....	11
-.....	12
Angular 2 backend node.js / expressjs (http).....	13
.....	13
.....	13
1.....	13
2.....	14
3.....	14
Angular 4!.....	19
2: Angular2 CanActivate	24

Examples.....	24
Angular2 CanActivate.....	24
3: Angular2 Databinding.....	25
Examples.....	25
@Input ().....	25
:	25
4: Angular2 - Memory.....	27
.....	27
Examples.....	27
.....	27
API	28
5: Angular2	30
.....	30
Examples.....	30
.....	30
6: Angular2 webpack.....	32
Examples.....	32
.....	32
7: CRUD Angular2 API.....	36
.....	36
Examples.....	36
Restful API Angular2.....	36
8: Dropzone Angular2.....	38
Examples.....	38
.....	38
9: Mocking @ ngrx /	40
.....	40
.....	40
.....	40
Examples.....	41
.....	41
.....	41

- 42
- 2 - Mock Observable (+)..... 43
- 46
- 10: ngrx..... 50**
- 50
- Examples..... 50
- : / 50
- 1) IUser..... 50**
- 2) User..... 51**
- 3) UserReducer..... 52**
- 4) UserReducer..... 52**
- : - 53
- 5) UserReducer , Store..... 54**
- 6) Store..... 54**
- 11: Zone.js..... 57**
- Examples..... 57
- NgZone..... 57
- NgZone HTTP- 57
- 12:..... 59**
- Examples..... 59
- 59
- 59
- 13:..... 61**
- 61
- Examples..... 61
- 61
- 14: 2..... 62**
- Examples..... 62
- 62
- 62

.....	63
.....	63
15: Angular2 Input () ()	64
Examples.....	64
().....	64
:	64
.....	65
16: ViewContainerRef.createComponent	66
Examples.....	66
-,	66
().....	67
html Angular2.....	68
17:	72
.....	72
.....	72
Examples.....	72
.....	72
-	72
.....	72
.....	72
* ngFor.....	73
.....	74
.....	75
18: @H	78
Examples.....	78
@HostBinding.....	78
19: : @Input @Output	79
.....	79
Examples.....	79
.....	79
Angular2 @Input @Output	80

Angular2 @Input	81
.....	82
,	82
20:	84
.....	84
.....	84
Examples.....	84
.....	84
21: , jQuery Angular 2	85
.....	85
Examples.....	85
.....	85
NPM.....	85
.....	85
.....	85
jQuery Angular 2.x.....	85
22: - Angular 2	87
.....	87
Examples.....	87
.....	87
-	87
23: ngfor	88
.....	88
Examples.....	88
.....	88
.....	88
.....	88
2	88
* ngFor	89
24: ngif	90
.....	90
.....	90

Examples.....	90
.....	90
.....	91
* ngFor * ngIf.....	91
* ngIf * ngFor.....	91
25: (AOT) Angular 2.....	93
Examples.....	93
1. Angular 2	93
2. `angularCompilerOptions` `tsconfig.json`	93
3. ngc,	93
4. `main.ts` NgFactory	93
, ?.....	94
AoT CLI.....	95
26:	96
.....	96
.....	96
Examples.....	96
- @Input & @Output.....	96
- ViewChild.....	98
-	99
27:	101
.....	101
Examples.....	101
.....	101
28:	109
.....	109
Examples.....	109
.....	109
.....	109
.....	110
.....	110

.....	110
.....	110
.....	110
.....	112
29:	113
Examples.....	113
Md2Select.....	113
Md2Tooltip.....	113
Md2Toast.....	113
Md2Datepicker.....	114
Md2Accordion Md2Collapse.....	114
30:	116
.....	116
.....	116
.....	116
.....	116
Examples.....	116
OnInit.....	116
OnDestroy.....	117
OnChanges.....	117
AfterContentInit.....	117
AfterContentChecked.....	118
AfterViewInit.....	118
AfterViewChecked.....	119
DoCheck.....	119
31:	120
Examples.....	120
.....	120
32:	122
Examples.....	122
.....	122
.....	124

ResolveData.....	125
.....	128
33: (3.0.0+)	131
.....	131
Examples.....	131
.....	131
-.....	131
().....	132
.....	133
.....	134
.....	134
.....	134
.....	135
.....	135
.....	135
.....	136
Guard	136
.....	137
34:	139
.....	139
.....	139
Examples.....	139
CLI.....	139
35:	140
.....	140
Examples.....	140
.....	140
.....	140
36:	142
Examples.....	142
-.....	142
.....

37: ASP.net Core Angular 2 TypeScript.....144
144
 Examples.....144
 Asp.Net Core + Angular2 + Gulp.....144
 [Seed] Asp.Net Core + Angular2 + Gulp Visual Studio 2017.....148
 MVC <-> 2.....148

38:150
 Examples.....150
 ,150

39:151
 Examples.....151
 , : WARN151

40: Sanitizing152
152
152

!.....152
 XSS ()152
 Examples.....152
 Sanitizing ().....153

41:156
156
 Examples.....156
 156
 AsyncPipe.....156
 2,157
 157

42: ChangeDetectionStrategy.....159
 Examples.....159
 vs OnPush.....159

43: Angular2 Visual Studio.....161

Examples.....	161
Launch.json	161
44: Http.....	163
.....	163
Examples.....	163
Http.....	163
Angular's Http.....	164
HttpClient AuthToken Interceptor (Angular 4.3+).....	165
45: ngx-bootstrap datepicker + input.....	166
Examples.....	166
ngx-bootstrap datepicker.....	166
46: , / route / subroute URL-.....	169
Examples.....	169
.....	169
47:	170
.....	170
Examples.....	170
Image Picker	170
.....	171
48: -.....	173
.....	173
Examples.....	173
Angular2 cli.....	173
, ,	173
.....	173
.....	174
scss / sass	174
@ angular / cli.....	175
.....	175
49:	176
.....	176
Examples.....	176

Service Worker	176
50: EventEmitter	179
Examples	179
.....	179
.....	179
.....	179
.....	179
.....	180
51: 2+ NPM	181
.....	181
Examples	181
.....	181
.....	181
.gitignore	181
.npmignore	181
gulpfile.js	182
index.d.ts	182
index.js	182
package.json	182
/tsconfig.json	183
SRC / -- NPM-package.component.ts	184
SRC / -- NPM-package.component.html	184
SRC / -- table.component.css	184
SRC / -- NPM-package.module.ts	184
.....	185
.....	185
52: npm	186
.....	186
Examples	186
.....	186
.....	186
.....	186

186	187
NPM	188
.....	189
53: ngModel	191
.....	191
Examples	191
.....	191
54: Angular 2	193
Examples	193
.....	193
.....	193
.....	193
Gulp, Webpack, Karma Jasmine	193
Http	198
-	200
55:	202
.....	202
Examples	202
.....	202
56:	205
.....	205
.....	205
.....	205
.....	205
Examples	205
.....	205
.....	206
.....	206
Angular2	206
.....	207
-reservation.component.ts	207

-reservation.template.html.....	207
.....	207
JsonPipe.....	208
.....	208
.....	208
.....	208
.....	209
async-.....	209
.....	210
.....	210
.....	211
.....	213
57: - ForLoop.....	215
.....	215
.....	215
Examples.....	215
2.....	215
NgFor -	216
* ngFor	216
* ngFor	217
* ngFor X	217
58: 2.....	218
.....	218
Examples.....	218
- ,	218
59: 2-.....	220
.....	220
Examples.....	220
.....	220
PW-change.template.html.....	220
PW-change.component.ts.....	221
PW-validators.ts.....	221

2:	222
2 - /	223
2: ()	224
-form.component.ts	224
-form.html	225
2 ()	225
app.module.ts	225
app.component.ts	225
app.component.html	226
validators.ts	227
Angular2 - Form Builder	227
60:	229
	229
Examples	229
	229
	229
61:	231
Examples	231
	231
	232
	232
	233
62: 2 -	234
Examples	234
Navbar Protractor	234
2 -	235
63: 2	237
Examples	237
	237
64: 2	239
	239
Examples	240

.....	240
65: RXJS API	242
.....	242
Examples.....	242
.....	242
API.....	243
.....	243
66: 2	245
.....	245
Examples.....	245
:	245
Formbuilder.....	245
get / set FormBuilder	246
67:	247
Examples.....	247
.....	247
Promise.resolve.....	248
.....	249
68: angular-cli@1.0.0-beta.10	252
.....	252
Examples.....	252
jquery angular-cli.....	252
,	254
69:	256
Examples.....	256
.....	256
70:	257
.....	257
Examples.....	257
2	257
.....	259

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [angular-2](#)

It is an unofficial and free Angular 2 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 Angular 2.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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 Angular 2

замечания

В этом разделе приведен обзор того, как устанавливать и настраивать Angular2 + для использования в различных средах, а также с помощью инструментов IDE, таких как сообщество, разработанное [angular-cli](#) .

Предыдущая версия Angular - [AngularJS](#) или также называется Angular 1. См. Здесь [документацию](#) .

Версии

Версия	Дата выхода
4.3.3	2017-08-02
4.3.2	2017-07-26
4.3.1	2017-07-19
4.3.0	2017-07-14
4.2.0	2017-06-08
4.1.0	2017-04-26
4.0.0	2017-03-23
2.3.0	2016-12-08
2.2.0	2016-11-14
2.1.0	2016-10-13
2.0.2	2016-10-05
2.0.1	2016-09-23
2.0.0	2016-09-14
2.0.0-rc.7	2016-09-13
2.0.0-rc.6	2016-08-31
2.0.0-rc.5	2016-08-09

Версия	Дата выхода
2.0.0-rc.4	2016-06-30
2.0.0-rc.3	2016-06-21
2.0.0-ПК-2	2016-06-15
2.0.0-RC.1	2016-05-03
2.0.0-rc.0	2016-05-02

Examples

Установите угловое с угловым кли

Этот пример представляет собой быструю настройку Angular 2 и как создать быстрый пример проекта.

Предпосылки:

- [Node.js v4](#) или выше.
- [npm v3](#) или больше или [пряжа](#) .

Откройте терминал и выполните команды один за другим:

```
npm install -g @angular/cli
```

или же

```
yarn global add @angular/cli
```

в зависимости от вашего выбора менеджера пакетов.

Предыдущая команда устанавливает **@ angular / cli** глобально, добавляя исполняемый файл `ng` к `PATH`.

Чтобы настроить новый проект

Перейдите с помощью терминала в папку, в которой вы хотите настроить новый проект.

Запустите команды:

```
ng new PROJECT_NAME
```

```
cd PROJECT_NAME
ng serve
```

Вот и все, теперь у вас есть простой пример проекта с угловым 2. Теперь вы можете перейти к ссылке, отображаемой в терминале, и посмотреть, что она работает.

Чтобы добавить к существующему проекту

Перейдите в корень вашего текущего проекта.

Выполните команду:

```
ng init
```

Это добавит необходимые строительные леса в ваш проект. Файлы будут созданы в текущем каталоге, поэтому обязательно запустите его в пустой директории.

Выполнение проекта локально

Чтобы видеть и взаимодействовать с вашим приложением во время работы в браузере, вы должны запустить локальный сервер разработки, в котором размещаются файлы для вашего проекта.

```
ng serve
```

Если сервер успешно запущен, он должен отображать адрес, на котором работает сервер. Обычно это:

```
http://localhost:4200
```

Исходя из этого, этот локальный сервер разработки подключается с помощью Hot Module Reloading, поэтому любые изменения в html, машинописном тексте или css приведут к тому, что браузер будет автоматически перезагружен (но может быть отключен по желанию).

Создание компонентов, директив, труб и услуг

Команда `ng generate <scaffold-type> <name>` (или просто `ng g <scaffold-type> <name>`)

позволяет автоматически генерировать угловые компоненты:

```
# The command below will generate a component in the folder you are currently at
ng generate component my-generated-component
# Using the alias (same outcome as above)
ng g component my-generated-component
```

Существует несколько возможных типов лесов, которые могут генерировать угловые cli:

Тип лесов	использование
модуль	<code>ng g module my-new-module</code>
Составная часть	<code>ng g component my-new-component</code>
директива	<code>ng g directive my-new-directive</code>
труба	<code>ng g pipe my-new-pipe</code>
обслуживание	<code>ng g service my-new-service</code>
Учебный класс	<code>ng g class my-new-class</code>
Интерфейс	<code>ng g interface my-new-interface</code>
Enum	<code>ng g enum my-new-enum</code>

Вы также можете заменить имя типа своей первой буквой. Например:

`ng gm my-new-module` для создания нового модуля или `ng gc my-new-component` для создания компонента.

Строительство / Пакетирование

Когда вы все закончите создание своего веб-приложения Angular 2, и вы хотите установить его на веб-сервере, таком как Apache Tomcat, все, что вам нужно сделать, это запустить команду сборки либо с установленным флагом, либо без него. Производство будет минимизировать код и оптимизировать производственные параметры.

```
ng build
```

или же

```
ng build --prod
```

Затем зайдите в корневой каталог проектов для папки `/dist`, которая содержит сборку.

Если вы хотите получить преимущества небольшого пакета продуктов, вы также можете

использовать компиляцию шаблона Ahead-of-Time, которая удаляет компилятор шаблона из окончательной сборки:

```
ng build --prod --aot
```

Тестирование устройства

Угловой 2 обеспечивает встроенное модульное тестирование, и каждый элемент, созданный угловым кли, генерирует базовый модульный тест, который можно расширить. Модульные тесты записываются с использованием жасмина и выполняются через Карму. Чтобы начать тестирование, выполните следующую команду:

```
ng test
```

Эта команда выполнит все тесты в проекте и будет повторно выполнять их каждый раз при изменении исходного файла, будь то тест или код из приложения.

Для получения дополнительной информации также посетите страницу [angular-cli github](#)

Начало работы с Angular 2 без углового кли.

Угловой 2.0.0-rc.4

В этом примере мы создадим «Hello World!». приложение с одним корневым компонентом (`AppComponent`) для простоты.

Предпосылки:

- [Node.js](#) v5 или новее
- `npm` v3 или новее

Примечание. Вы можете проверить версии, запустив `node -v` и `npm -v` в консоли / терминале.

Шаг 1

Создайте и введите новую папку для своего проекта. Назовем это `angular2-example` .

```
mkdir angular2-example
cd angular2-example
```

Шаг 2

Прежде чем мы начнем писать наш код приложения, мы добавим следующие 4 файла:

`package.json` , `tsconfig.json` , `typings.json` и `systemjs.config.js` .

Отказ от ответственности: те же файлы можно найти в [официальном 5-минутном старте](#) .

package.json - Позволяет загружать все зависимости с помощью npm и обеспечивает простое выполнение скриптов, чтобы упростить жизнь для простых проектов. (Вы должны в будущем использовать что-то вроде [Gulp](#) для автоматизации задач).

```
{
  "name": "angular2-example",
  "version": "1.0.0",
  "scripts": {
    "start": "tsc && concurrently \"npm run tsc:w\" \"npm run lite\" ",
    "lite": "lite-server",
    "postinstall": "typings install",
    "tsc": "tsc",
    "tsc:w": "tsc -w",
    "typings": "typings"
  },
  "license": "ISC",
  "dependencies": {
    "@angular/common": "2.0.0-rc.4",
    "@angular/compiler": "2.0.0-rc.4",
    "@angular/core": "2.0.0-rc.4",
    "@angular/forms": "0.2.0",
    "@angular/http": "2.0.0-rc.4",
    "@angular/platform-browser": "2.0.0-rc.4",
    "@angular/platform-browser-dynamic": "2.0.0-rc.4",
    "@angular/router": "3.0.0-beta.1",
    "@angular/router-deprecated": "2.0.0-rc.2",
    "@angular/upgrade": "2.0.0-rc.4",
    "systemjs": "0.19.27",
    "core-js": "^2.4.0",
    "reflect-metadata": "^0.1.3",
    "rxjs": "5.0.0-beta.6",
    "zone.js": "^0.6.12",
    "angular2-in-memory-web-api": "0.0.14",
    "bootstrap": "^3.3.6"
  },
  "devDependencies": {
    "concurrently": "^2.0.0",
    "lite-server": "^2.2.0",
    "typescript": "^1.8.10",
    "typings": "^1.0.4"
  }
}
```

tsconfig.json - настраивает транспилятор TypeScript.

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
```

```
    "noImplicitAny": false
  }
}
```

typings.json - typings.json библиотеки распознавания TypeScript, которые мы используем.

```
{
  "globalDependencies": {
    "core-js": "registry:dt/core-js#0.0.0+20160602141332",
    "jasmine": "registry:dt/jasmine#2.2.0+20160621224255",
    "node": "registry:dt/node#6.0.0+20160621231320"
  }
}
```

systemjs.config.js - Настраивает [SystemJS](#) (вы также можете использовать [webpack](#)).

```
/**
 * System configuration for Angular 2 samples
 * Adjust as necessary for your application's needs.
 */
(function(global) {
  // map tells the System loader where to look for things
  var map = {
    'app': 'app', // 'dist',
    '@angular': 'node_modules/@angular',
    'angular2-in-memory-web-api': 'node_modules/angular2-in-memory-web-api',
    'rxjs': 'node_modules/rxjs'
  };
  // packages tells the System loader how to load when no filename and/or no extension
  var packages = {
    'app': { main: 'main.js', defaultExtension: 'js' },
    'rxjs': { defaultExtension: 'js' },
    'angular2-in-memory-web-api': { main: 'index.js', defaultExtension: 'js' },
  };
  var ngPackageNames = [
    'common',
    'compiler',
    'core',
    'forms',
    'http',
    'platform-browser',
    'platform-browser-dynamic',
    'router',
    'router-deprecated',
    'upgrade',
  ];
  // Individual files (~300 requests):
  function packIndex(pkgName) {
    packages['@angular/' + pkgName] = { main: 'index.js', defaultExtension: 'js' };
  }
  // Bundled (~40 requests):
  function packUmd(pkgName) {
    packages['@angular/' + pkgName] = { main: '/bundles/' + pkgName + '.umd.js',
    defaultExtension: 'js' };
  }
  // Most environments should use UMD; some (Karma) need the individual index files
  var setPackageConfig = System.packageWithIndex ? packIndex : packUmd;
  // Add package entries for angular packages
```



```
ngPackageNames.forEach(setPackageConfig);
var config = {
  map: map,
  packages: packages
};
System.config(config);
})(this);
```

Шаг 3

Давайте установим зависимости, набрав

```
npm install
```

в консоли / терминале.

Шаг 4

Создайте `index.html` внутри папки `angular2-example`.

```
<html>
  <head>
    <title>Angular2 example</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. Load libraries -->
    <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <!-- 2. Configure SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>
  </head>
  <!-- 3. Display the application -->
  <body>
    <my-app></my-app>
  </body>
</html>
```

Ваше приложение будет отображаться между тегами `my-app`.

Тем не менее, Angular все еще не знает, что делать. Чтобы сказать это, мы определим `AppComponent`.

Шаг 5

Создайте подпапку под названием `app` где мы можем определить компоненты и **сервисы**, составляющие наше приложение. (В данном случае, это будет просто содержать

AppComponent КОД И main.ts .)

```
mkdir app
```

Шаг 6

Создайте файл `app/app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `
    <h1>{{title}}</h1>
    <ul>
      <li *ngFor="let message of messages">
        {{message}}
      </li>
    </ul>
  `
})
export class AppComponent {
  title = "Angular2 example";
  messages = [
    "Hello World!",
    "Another string",
    "Another one"
  ];
}
```

Что происходит? Во-первых, мы импортируем декоратор `@Component` который мы используем, чтобы дать Angular HTML-тег и шаблон для этого компонента. Затем мы создаем класс `AppComponent` с переменными `title` и `messages` которые мы можем использовать в шаблоне.

Теперь давайте посмотрим на этот шаблон:

```
<h1>{{title}}</h1>
<ul>
  <li *ngFor="let message of messages">
    {{message}}
  </li>
</ul>
```

Мы отображаем переменную `title` в теге `h1` а затем `*ngFor` список, показывающий каждый элемент массива `messages`, используя директиву `*ngFor`. Для каждого элемента массива `*ngFor` создает переменную `message` которую мы используем в элементе `li`. Результатом будет:

```
<h1>Angular 2 example</h1>
<ul>
  <li>Hello World!</li>
  <li>Another string</li>
```

```
<li>Another one</li>
</ul>
```

Шаг 7

Теперь мы создаем файл `main.ts`, который будет первым файлом, на который смотрит Angular.

Создайте файл `app/main.ts`

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent);
```

Мы импортируем функцию `bootstrap` и класс `AppComponent`, а затем `AppComponent bootstrap` чтобы сообщить Angular, какой компонент использовать в качестве корня.

Шаг 8

Пришло время запустить ваше первое приложение. Тип

```
npm start
```

в вашей консоли / терминале. Это запустит подготовленный скрипт из `package.json` который запустит Lite-сервер, откроет ваше приложение в окне браузера и запустит транспилятор TypeScript в режиме просмотра (файлы `.ts` будут переданы и браузер обновится при сохранении изменений),

Что теперь?

Ознакомьтесь с [официальным руководством Angular 2](#) и другими темами в [документации StackOverflow](#).

Вы также можете редактировать `AppComponent` для использования внешних шаблонов, стилей или добавления / редактирования переменных компонента. Вы должны увидеть свои изменения сразу после сохранения файлов.

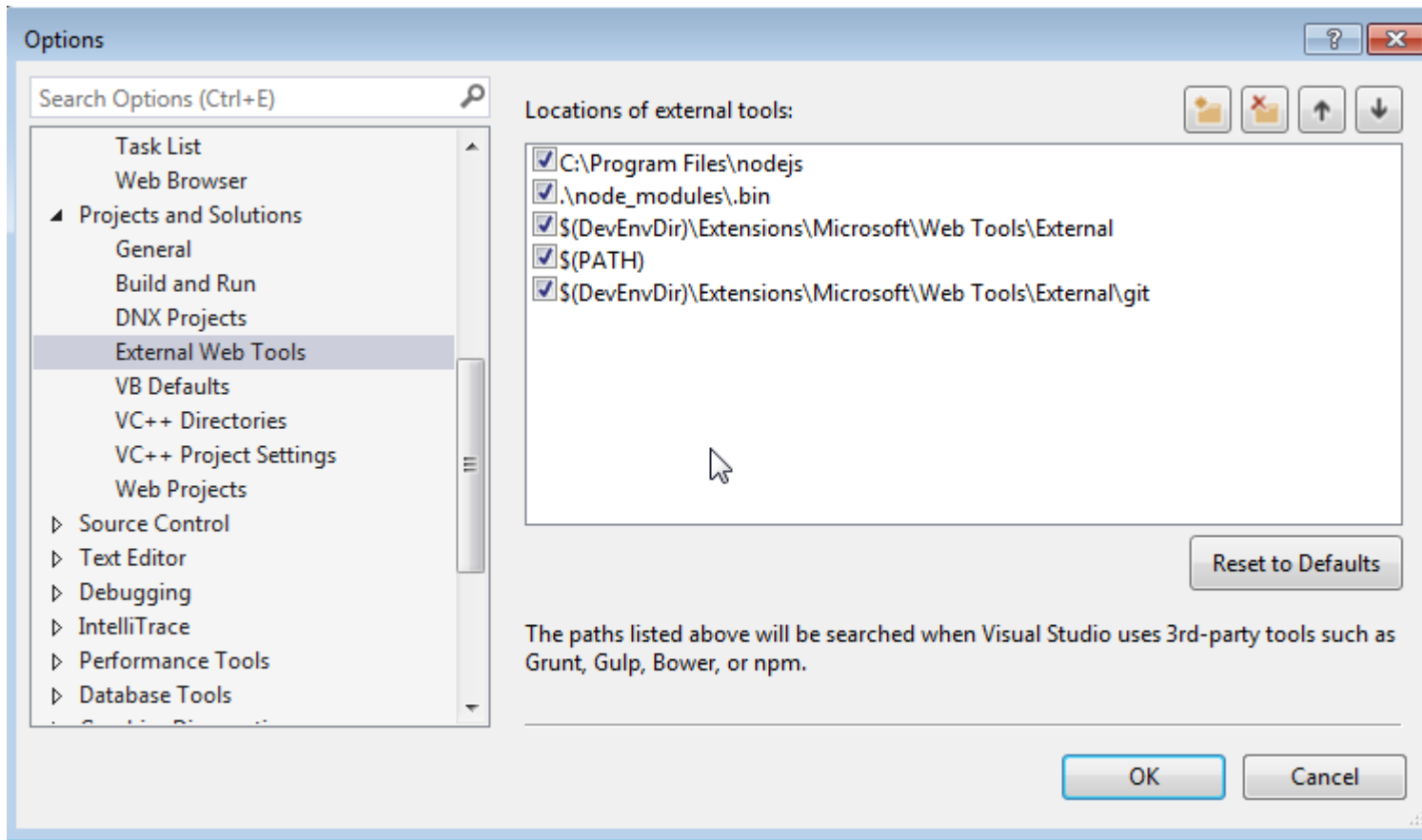
Сохранение Visual Studios в синхронизации с обновлениями NPM и NODE

Шаг 1. Найдите вашу загрузку Node.js, как правило, она устанавливается под файлами `C:/program / nodejs`

Шаг 2. Откройте Visual Studios и перейдите к «Инструменты»> «Параметры».

Шаг 3: В окне параметров перейдите к разделу «Проекты и решения» Внешние веб-инструменты»

Шаг 4: Добавьте новую запись с вашим местоположением файла Node.js (C: / program files / nodejs), ВАЖНО используйте кнопки со стрелками в меню, чтобы переместить ссылку на верхнюю часть списка.



Шаг 5: Перезапустите Visual Studios и запустите npm install, против вашего проекта, из окна командной строки npm

Пройдя через этот досадный прокси-сервер

Если вы пытаетесь получить сайт Angular2 на рабочем компьютере Windows на XYZ MegaCorp, есть вероятность, что у вас возникают проблемы с прокси-сервером компании.

Есть (по крайней мере) два менеджера пакетов, которым необходимо пройти через прокси:

1. NPM
2. типизации

Для NPM вам нужно добавить следующие строки в файл .npmrc :

```
proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
https-proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
```

Для Typings вам нужно добавить следующие строки в файл `.typingsrc` :

```
proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
https-proxy=http://[DOMAIN]%5C[USER]:[PASS]@[PROXY]:[PROXYPORT] /
rejectUnauthorized=false
```

Эти файлы, вероятно, еще не существуют, поэтому вы можете создавать их в виде пустых текстовых файлов. Они могут быть добавлены в корень проекта (то же самое место, что и `package.json` или вы можете поместить их в `%HOMEPATH%` и они будут доступны для всех ваших проектов.

Бит, который не является очевидным, и является основной причиной того, что люди считают, что настройки прокси-сервера не работают, это `%5C` который является URL-кодированием для `\` для разделения имен доменов и пользователей. Спасибо Стиву Робертсу за это: [использование npm за корпоративным прокси .pac](#)

Начало работы с Angular 2 с помощью backend узла node.js / expressjs (включен пример http)

Мы создадим простой «Hello World!». приложение с Angular2 2.4.1 (изменение `@NgModule`) с помощью узла node.js (expressjs).

Предпосылки

- [Node.js](#) v4.xx или выше
- [npm](#) v3.xx или выше или [пряжа](#)

Затем запустите `npm install -g typescript` или `yarn global add typescript` для установки машинописных файлов по всему миру

Дорожная карта

Шаг 1

Создайте новую папку (и корневой каталог нашего back-end) для нашего приложения. Назовем это `Angular2-express` .

командной строки :

```
mkdir Angular2-express
cd Angular2-express
```

Шаг 2

Создание `package.json` (для зависимостей) и `app.js` (для самозагрузки) для нашего `node.js` приложения.

`package.json`:

```
{
  "name": "Angular2-express",
  "version": "1.0.0",
  "description": "",
  "scripts": {
    "start": "node app.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.13.3",
    "express": "^4.13.3"
  }
}
```

`app.js`:

```
var express = require('express');
var app = express();
var server = require('http').Server(app);
var bodyParser = require('body-parser');

server.listen(process.env.PORT || 9999, function(){
  console.log("Server connected. Listening on port: " + (process.env.PORT || 9999));
});

app.use(bodyParser.json());
app.use(bodyParser.urlencoded({extended: true}));

app.use( express.static(__dirname + '/front' ) );

app.get('/test', function(req,res){ //example http request receiver
  return res.send(myTestVar);
});

//send the index.html on every page refresh and let angular handle the routing
app.get('/*', function(req, res, next) {
  console.log("Reloading");
  res.sendFile('index.html', { root: __dirname });
});
```

Затем запустите `npm install` или `yarn` чтобы установить зависимости.

Теперь наша внутренняя структура завершена. Давайте перейдем к интерфейсу.

Шаг 3

Наш интерфейс должен находиться в папке с именем `front` внутри нашей `Angular2-express` .

командная строка:

```
mkdir front
cd front
```

Точно так же, как мы сделали это с нашим back-end, нашему интерфейсу нужны файлы зависимостей. Давайте продолжим и создадим следующие файлы: `package.json` , `systemjs.config.js` , `tsconfig.json`

`package.json` :

```
{
  "name": "Angular2-express",
  "version": "1.0.0",
  "scripts": {
    "tsc": "tsc",
    "tsc:w": "tsc -w"
  },
  "licenses": [
    {
      "type": "MIT",
      "url": "https://github.com/angular/angular.io/blob/master/LICENSE"
    }
  ],
  "dependencies": {
    "@angular/common": "~2.4.1",
    "@angular/compiler": "~2.4.1",
    "@angular/compiler-cli": "^2.4.1",
    "@angular/core": "~2.4.1",
    "@angular/forms": "~2.4.1",
    "@angular/http": "~2.4.1",
    "@angular/platform-browser": "~2.4.1",
    "@angular/platform-browser-dynamic": "~2.4.1",
    "@angular/platform-server": "^2.4.1",
    "@angular/router": "~3.4.0",
    "core-js": "^2.4.1",
    "reflect-metadata": "^0.1.8",
    "rxjs": "^5.0.2",
    "systemjs": "0.19.40",
    "zone.js": "^0.7.4"
  },
  "devDependencies": {
    "@types/core-js": "^0.9.34",
    "@types/node": "^6.0.45",
    "typescript": "2.0.2"
  }
}
```

`systemjs.config.js`:

```
/**
 * System configuration for Angular samples
 * Adjust as necessary for your application needs.
 */
```

```

(function (global) {
  System.config({
    defaultJSExtensions:true,
    paths: {
      // paths serve as alias
      'npm:': 'node_modules/'
    },
    // map tells the System loader where to look for things
    map: {
      // our app is within the app folder
      app: 'app',
      // angular bundles
      '@angular/core': 'npm:@angular/core/bundles/core.umd.js',
      '@angular/common': 'npm:@angular/common/bundles/common.umd.js',
      '@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',
      '@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-
browser.umd.js',
      '@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-
dynamic/bundles/platform-browser-dynamic.umd.js',
      '@angular/http': 'npm:@angular/http/bundles/http.umd.js',
      '@angular/router': 'npm:@angular/router/bundles/router.umd.js',
      '@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',
      // other libraries
      'rxjs': 'npm:rxjs',
      'angular-in-memory-web-api': 'npm:angular-in-memory-web-api',
    },
    // packages tells the System loader how to load when no filename and/or no extension
    packages: {
      app: {
        main: './main.js',
        defaultExtension: 'js'
      },
      rxjs: {
        defaultExtension: 'js'
      }
    }
  });
})(this);

```

tsconfig.json:

```

{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  },
  "compileOnSave": true,
  "exclude": [
    "node_modules/*"
  ]
}

```

Затем запустите `npm install` или `yarn` чтобы установить зависимости.

Теперь, когда наши файлы зависимостей завершены. Перейдем к нашему `index.html` :

index.html:

```
<html>
  <head>
    <base href="/">
    <title>Angular2-express</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. Load libraries -->
    <!-- Polyfill(s) for older browsers -->
    <script src="node_modules/core-js/client/shim.min.js"></script>
    <script src="node_modules/zone.js/dist/zone.js"></script>
    <script src="node_modules/reflect-metadata/Reflect.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <!-- 2. Configure SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>

  </head>
  <!-- 3. Display the application -->
  <body>
    <my-app>Loading...</my-app>
  </body>
</html>
```

Теперь мы готовы создать наш первый компонент. Создайте папку с именем `app` в нашей `front` папке.

командная строка:

```
mkdir app
cd app
```

Давайте `main.ts` следующие файлы с именем `main.ts` , `app.module.ts` , `app.component.ts`

main.ts:

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app.module';

const platform = platformBrowserDynamic();
platform.bootstrapModule(AppModule);
```

app.module.ts:

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from "@angular/http";

import { AppComponent } from './app.component';
```

```

@NgModule({
  imports:      [
    BrowserModule,
    HttpClientModule
  ],
  declarations: [
    AppComponent
  ],
  providers: [ ],
  bootstrap:   [ AppComponent ]
})
export class AppModule {}

```

app.component.ts:

```

import { Component } from '@angular/core';
import { Http } from '@angular/http';

@Component({
  selector: 'my-app',
  template: 'Hello World!',
  providers: []
})
export class AppComponent {
  constructor(private http: Http){
    //http get example
    this.http.get('/test')
      .subscribe((res)=>{
        console.log(res);
      });
  }
}

```

После этого скомпилируйте файлы машинописных файлов в файлы javascript. Перейдите на 2 уровня вверх от текущего каталога (внутри папки Angular2-express) и выполните приведенную ниже команду.

командная строка:

```

cd ..
cd ..
tsc -p front

```

Структура нашей папки должна выглядеть так:

```

Angular2-express
├─ app.js
├─ node_modules
├─ package.json
├─ front
│   ├─ package.json
│   ├─ index.html
│   ├─ node_modules
│   ├─ systemjs.config.js
│   └─ tsconfig.json

```

```
| | | | | app
| | | | |   └─ app.component.ts
| | | | |   └─ app.component.js.map
| | | | |   └─ app.component.js
| | | | |   └─ app.module.ts
| | | | |   └─ app.module.js.map
| | | | |   └─ app.module.js
| | | | |   └─ main.ts
| | | | |   └─ main.js.map
| | | | |   └─ main.js
```

Наконец, внутри папки Angular2-express запустите команду `node app.js` в командной строке. Откройте свой любимый браузер и проверьте `localhost:9999` чтобы увидеть свое приложение.

Давайте погрузиться в Angular 4!

Угловое 4 теперь доступно! На самом деле Angular использует `semver` с угловым 2, что требует увеличения основного числа при введении изменений. Угловая команда отложила функции, которые вызывают нарушения, которые будут выпущены с помощью Angular 4. Угловая 3 была пропущена, чтобы выровнять номера версий основных модулей, потому что у маршрутизатора уже была версия 3.

Согласно команде «Угловая», приложения с угловым 4 будут занимать меньше места и быстрее, чем раньше. Они отделили пакет анимации от пакета `@angular/core`. Если кто-то не использует анимационный пакет, значит, дополнительное пространство кода не закончится в процессе производства. Синтаксис привязки шаблона теперь поддерживает синтаксис `if / else`. Угловая 4 теперь совместима с последней версией TypeScript 2.1 и 2.2. Итак, Angular 4 станет более захватывающим.

Теперь я покажу вам, как настроить Angular 4 в вашем проекте.

Давайте начнем Угловую настройку тремя способами:

Вы можете использовать Angular-CLI (интерфейс командной строки), он будет устанавливать для вас все зависимости.

- Вы можете перейти от углового 2 к угловому 4.
- Вы можете использовать `github` и клонировать Angular4-шаблон. (Это самый простой.)
- Угловая настройка с использованием Angular-CLI (интерфейс командной строки).

Прежде чем вы начнете использовать Angular-CLI, убедитесь, что на вашем компьютере установлен узел. Здесь я использую узел `v7.8.0`. Теперь откройте терминал и введите следующую команду для Angular-CLI.

```
npm install -g @angular/cli
```

или же

```
yarn global add @angular/cli
```

в зависимости от используемого менеджера пакетов.

Давайте установим Angular 4 с помощью Angular-CLI.

```
ng new Angular4-boilerplate
```

cd Angular4-schemeplate Мы все настроены для Angular 4. Его довольно простой и простой метод.

Угловая настройка путем перехода от углового 2 к угловому 4

Теперь посмотрим на второй подход. Я покажу вам, как перенести Angular 2 на Angular 4. Для этого вам нужно клонировать любой проект Angular 2 и обновлять зависимости Angular 2 с помощью функции Angular 4 Dependency в вашем package.json следующим образом:

```
"dependencies": {
  "@angular/animations": "^4.1.0",
  "@angular/common": "4.0.2",
  "@angular/compiler": "4.0.2",
  "@angular/core": "^4.0.1",
  "@angular/forms": "4.0.2",
  "@angular/http": "4.0.2",
  "@angular/material": "^2.0.0-beta.3",
  "@angular/platform-browser": "4.0.2",
  "@angular/platform-browser-dynamic": "4.0.2",
  "@angular/router": "4.0.2",
  "typescript": "2.2.2"
}
```

Это основные зависимости для Angular 4. Теперь вы можете установить npm и затем запустить npm для запуска приложения. Для справки мой package.json.

Угловая настройка из проекта github

Перед началом этого шага убедитесь, что у вас установлена git на вашем компьютере. Откройте ваш терминал и клонируйте угловой4-шаблон, используя команду:

```
git@github.com: CypherTree/angular4-boilerplate.git
```

Затем установите все зависимости и запустите его.

```
npm install
npm start
```

И вы закончили настройку Angular 4. Все шаги очень просты, поэтому вы можете выбрать

любой из них.

Структура каталога углового4-шаблона

```
Angular4-boilerplate
-karma
-node_modules
-src
  -mocks
  -models
    -loginform.ts
    -index.ts
  -modules
    -app
      -app.component.ts
      -app.component.html
      -login
    -login.component.ts
    -login.component.html
    -login.component.css
      -widget
    -widget.component.ts
    -widget.component.html
    -widget.component.css
    .....
-services
  -login.service.ts
  -rest.service.ts
-app.routing.module.ts
-app.module.ts
-bootstrap.ts
-index.html
-vendor.ts
-typings
-webpack
-package.json
-tsconfig.json
-tslint.json
-typings.json
```

Основное понимание структуры каталогов:

Весь код находится в папке src.

папка «mocks» предназначена для макетов данных, которые используются в целях тестирования.

model содержит класс и интерфейс, которые используются в компоненте.

modules содержит список компонентов, таких как приложение, логин, виджет и т. д. Все компоненты содержат файлы машинописных, html и css. index.ts предназначен для экспорта всего класса.

services содержит список служб, используемых в приложении. Я разделял службу отдыха и обслуживание различных компонентов. В службе отдыха есть различные методы http.

Служба входа в систему работает как посредник между компонентом входа и службой отдыха.

Файл `app.routing.ts` описывает все возможные маршруты для приложения.

`app.module.ts` описывает модуль приложения как корневой компонент.

`bootstrap.ts` будет запускать все приложение.

Папка `webpack` содержит файл конфигурации `webpack`.

Файл `package.json` предназначен для всех списков зависимостей.

`karma` содержит конфигурацию кармы для модульного теста.

`node_modules` содержит список пакетов.

Давайте начнем с компонента входа. В `login.component.html`

```
<form>Dreamfactory - Addressbook 2.0
  <label>Email</label> <input id="email" form="" name="email" type="email" />
  <label>Password</label> <input id="password" form="" name="password"
  type="password" />
  <button form="">Login</button>
</form>
```

В `login.component.ts`

```
import { Component } from '@angular/core';
import { Router } from '@angular/router';
import { Form, FormGroup } from '@angular/forms';
import { LoginForm } from '../models';
import { LoginService } from '../services/login.service';

@Component({
  selector: 'login',
  template: require('./login.component.html'),
  styles: [require('./login.component.css')]
})
export class LoginComponent {

  constructor(private loginService: LoginService, private router: Router, form: LoginForm) {
  }

  getLogin(form: LoginForm): void {
    let username = form.email;
    let password = form.password;
    this.loginService.getAuthenticate(form).subscribe(() => {
      this.router.navigate(['/calender']);
    });
  }
}
```

Нам нужно экспортировать этот компонент в `index.ts`.

```
export * from './login/login.component';
```

нам нужно установить маршруты для входа в `app.routes.ts`

```
const appRoutes: Routes = [  
  {  
    path: 'login',  
    component: LoginComponent  
  },  
  .....  
  {  
    path: '',  
    pathMatch: 'full',  
    redirectTo: '/login'  
  }  
];
```

В корневом компоненте, файле `app.module.ts` вам просто нужно импортировать этот компонент.

```
.....  
import { LoginComponent } from './modules';  
.....  
@NgModule({  
  bootstrap: [AppComponent],  
  declarations: [  
    LoginComponent  
    .....  
    .....  
  ]  
  .....  
})  
export class AppModule { }
```

и после этого установить `npm` и запустить `npm`. Ну вот! Вы можете проверить экран входа в свой локальный хост. В случае каких-либо трудностей вы можете обратиться к угловому4-шаблону.

В принципе, я могу чувствовать меньше строительный пакет и более быстрый отклик с помощью приложения Angular 4. И хотя я нашел точно подобный Angular 2 в кодировании.

Прочитайте Начало работы с Angular 2 онлайн: <https://riptutorial.com/ru/angular2/topic/789/начало-работы-с-angular-2>

глава 2: Angular2 CanActivate

Examples

Angular2 CanActivate

Реализовано в маршрутизаторе:

```
export const MainRoutes: Route[] = [{
  path: '',
  children: [ {
    path: 'main',
    component: MainComponent ,
    canActivate : [CanActivateRoute]
  } ]
}];
```

Файл `canActivateRoute` :

```
@Injectable()
export class CanActivateRoute implements CanActivate{
  constructor() {}
  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    return true;
  }
}
```

Прочитайте [Angular2 CanActivate онлайн](https://riptutorial.com/ru/angular2/topic/8899/angular2-canactivate):

<https://riptutorial.com/ru/angular2/topic/8899/angular2-canactivate>

глава 3: Angular2 Databinding

Examples

@Input ()

Родительский компонент: Инициализировать списки пользователей.

```
@Component ({
  selector: 'parent-component',
  template: '<div>
    <child-component [users]="users"></child-component>
  </div>'
})
export class ParentComponent implements OnInit {
  let users : List<User> = null;

  ngOnInit() {
    users.push(new User('A', 'A', 'A@gmail.com'));
    users.push(new User('B', 'B', 'B@gmail.com'));
    users.push(new User('C', 'C', 'C@gmail.com'));
  }
}
```

Детский компонент получает пользователя от родительского компонента с помощью Input ()

```
@Component ({
  selector: 'child-component',
  template: '<div>
    <table *ngIf="users !== null">
      <thead>
        <th>Name</th>
        <th>FName</th>
        <th>Email</th>
      </thead>
      <tbody>
        <tr *ngFor="let user of users">
          <td>{{user.name}}</td>
          <td>{{user.fname}}</td>
          <td>{{user.email}}</td>
        </tr>
      </tbody>
    </table>

  </div>',
})
export class ChildComponent {
  @Input () users : List<User> = null;
}
```

```
export class User {
  name : string;
  fname : string;
  email : string;

  constructor(_name : string, _fname : string, _email : string){
    this.name = _name;
    this.fname = _fname;
    this.email = _email;
  }
}
```

Прочитайте [Angular2 Databinding](https://riptutorial.com/ru/angular2/topic/9036/angular2-databinding) онлайн:

<https://riptutorial.com/ru/angular2/topic/9036/angular2-databinding>

глава 4: Angular2 В интерфейсе веб-интерфейса Memory

замечания

Я в основном просил эту тему, потому что я не мог найти никакой информации о настройке нескольких маршрутов API с помощью Angular2-In-Memory-Web-API. Закончился сам, поняв, что это может быть полезно для других.

Examples

Основная настройка

макетный data.ts

Создайте данные mock api

```
export class MockData {
  createDb() {
    let mock = [
      { id: '1', name: 'Object A' },
      { id: '2', name: 'Object B' },
      { id: '3', name: 'Object C' },
      { id: '4', name: 'Object D' }
    ];

    return {mock};
  }
}
```

main.ts

Имейте инжекторы зависимости, чтобы обеспечить InMemoryBackendService для запросов XHRBackend. Кроме того, укажите класс, который включает

```
createDb()
```

(в данном случае, MockData), определяющий маршрутизированные маршруты API для запросов SEED_DATA.

```
import { XHRBackend, HTTP_PROVIDERS } from '@angular/http';
import { InMemoryBackendService, SEED_DATA } from 'angular2-in-memory-web-api';
import { MockData } from './mock-data';
import { bootstrap } from '@angular/platform-browser-dynamic';

import { AppComponent } from './app.component';
```

```
bootstrap(AppComponent, [
  HTTP_PROVIDERS,
  { provide: XHRBackend, useClass: InMemoryBackendService },
  { provide: SEED_DATA, useClass: MockData }
]);
```

mock.service.ts

Пример вызова запроса получения для созданного маршрута API

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';
import { Mock } from './mock';

@Injectable()
export class MockService {
  // URL to web api
  private mockUrl = 'app/mock';

  constructor (private http: Http) {}

  getData(): Promise<Mock[]> {
    return this.http.get(this.mockUrl)
      .toPromise()
      .then(this.extractData)
      .catch(this.handleError);
  }

  private extractData(res: Response) {
    let body = res.json();
    return body.data || { };
  }

  private handleError (error: any) {
    let errMsg = (error.message) ? error.message :
      error.status ? `${error.status} - ${error.statusText}` : 'Server error';
    console.error(errMsg);
    return Promise.reject(errMsg);
  }
}
```

Настройка маршрутов API с несколькими тестами

макетный data.ts

```
export class MockData {
  createDb() {
    let mock = [
      { id: '1', name: 'Object A' },
      { id: '2', name: 'Object B' },
      { id: '3', name: 'Object C' }
    ];

    let data = [
      { id: '1', name: 'Data A' },
      { id: '2', name: 'Data B' },
      { id: '3', name: 'Data C' }
    ];
  }
}
```

```
];  
  
    return { mock, data };  
  }  
}
```

Теперь вы можете взаимодействовать с обоими

```
app/mock
```

а также

```
app/data
```

для извлечения их соответствующих данных.

Прочитайте [Angular2 В интерфейсе веб-интерфейса Memory онлайн](#):

<https://riptutorial.com/ru/angular2/topic/6576/angular2-в-интерфейсе-веб-интерфейса-memory>

глава 5: Angular2 предоставляет внешние данные для приложения перед загрузкой

Вступление

В этом сообщении я продемонстрирую, как передавать внешние данные в приложение Angular перед загрузкой приложения. Это внешние данные могут быть данными конфигурации, устаревшими данными, обработанными серверами и т. Д.

Examples

Через инъекцию зависимости

Вместо того, чтобы напрямую ссылаться на код начальной загрузки Angular, оберните код начальной загрузки в функцию и экспортируйте функцию. Эта функция также может принимать параметры.

```
import { platformBrowserDynamic } from "@angular/platform-browser-dynamic";
import { AppModule } from "./src/app";
export function runAngular2App(legacyModel: any) {
  platformBrowserDynamic([
    { provide: "legacyModel", useValue: model }
  ]).bootstrapModule(AppModule)
  .then(success => console.log("Ng2 Bootstrap success"))
  .catch(err => console.error(err));
}
```

Затем в любых сервисах или компонентах мы можем вводить «устаревшую модель» и получать к ней доступ.

```
import { Injectable } from "@angular/core";
@Injectable()
export class MyService {
  constructor(@Inject("legacyModel") private legacyModel) {
    console.log("Legacy data - ", legacyModel);
  }
}
```

Требовать приложения, а затем запустить его.

```
require(["myAngular2App"], function(app) {
  app.runAngular2App(legacyModel); // Input to your APP
});
```

Прочитайте [Angular2 предоставляет внешние данные для приложения перед загрузкой онлайн](https://riptutorial.com/ru/angular2/topic/9203/angular2-предоставляет-внешние-данные-онлайн): <https://riptutorial.com/ru/angular2/topic/9203/angular2-предоставляет-внешние-данные-онлайн>

[для-приложения-перед-загрузкой](#)

глава 6: Angular2 с помощью webpack

Examples

Угловая настройка веб-страницы

webpack.config.js

```
const webpack = require("webpack")
const helpers = require('./helpers')
const path = require("path")
const WebpackNotifierPlugin = require('webpack-notifier');

module.exports = {

  // set entry point for your app module
  "entry": {
    "app": helpers.root("app/main.module.ts"),
  },

  // output files to dist folder
  "output": {
    "filename": "[name].js",
    "path": helpers.root("dist"),
    "publicPath": "/",
  },

  "resolve": {
    "extensions": ['.ts', '.js'],
  },

  "module": {
    "rules": [
      {
        "test": /\.ts$/,
        "loaders": [
          {
            "loader": 'awesome-typescript-loader',
            "options": {
              "configFileName": helpers.root("./tsconfig.json")
            }
          },
          "angular2-template-loader"
        ]
      },
    ],
  },

  "plugins": [

    // notify when build is complete
    new WebpackNotifierPlugin({title: "build complete"}),

    // get reference for shims
    new webpack.DllReferencePlugin({
      "context": helpers.root("src/app"),
    })
  ]
}
```



```

    "manifest": helpers.root("config/polyfills-manifest.json")
  }},
  // get reference of vendor DLL
  new webpack.DllReferencePlugin({
    "context": helpers.root("src/app"),
    "manifest": helpers.root("config/vendor-manifest.json")
  }),
  // minify compiled js
  new webpack.optimize.UglifyJsPlugin(),
],
}

```

vendor.config.js

```

const webpack = require("webpack")
const helpers = require('./helpers')
const path = require("path")

module.exports = {
  // specify vendor file where all vendors are imported
  "entry": {
    // optionally add your shims as well
    "polyfills": [helpers.root("src/app/shims.ts")],
    "vendor": [helpers.root("src/app/vendor.ts")],
  },

  // output vendor to dist
  "output": {
    "filename": "[name].js",
    "path": helpers.root("dist"),
    "publicPath": "/",
    "library": "[name]"
  },

  "resolve": {
    "extensions": ['.ts', '.js'],
  },

  "module": {
    "rules": [
      {
        "test": /\.ts$/,
        "loaders": [
          {
            "loader": 'awesome-typescript-loader',
            "options": {
              "configFileName": helpers.root("./tsconfig.json")
            }
          }
        ],
      }
    ],
  },

  "plugins": [

    // create DLL for entries
    new webpack.DllPlugin({

```

```

        "name": "[name]",
        "context": helpers.root("src/app"),
        "path": helpers.root("config/[name]-manifest.json")
    }
  ],
  // minify generated js
  new webpack.optimize.UglifyJsPlugin(),
],
}

```

helpers.js

```

var path = require('path');

var _root = path.resolve(__dirname, '..');

function root(args) {
  args = Array.prototype.slice.call(arguments, 0);
  return path.join.apply(path, [_root].concat(args));
}

exports.root = root;

```

vendor.ts

```

import "@angular/platform-browser"
import "@angular/platform-browser-dynamic"
import "@angular/core"
import "@angular/common"
import "@angular/http"
import "@angular/router"
import "@angular/forms"
import "rxjs"

```

index.html

```

<!DOCTYPE html>
<html>
<head>
  <title>Angular 2 webpack</title>

  <script src="/dist/vendor.js" type="text/javascript"></script>
  <script src="/dist/app.js" type="text/javascript"></script>
</head>
<body>
  <app>loading...</app>
</body>
</html>

```

package.json

```

{
  "name": "webpack example",
  "version": "0.0.0",
  "description": "webpack",
  "scripts": {

```

```
"build:webpack": "webpack --config config/webpack.config.js",
"build:vendor": "webpack --config config/vendor.config.js",
"watch": "webpack --config config/webpack.config.js --watch"
},
"devDependencies": {
  "@angular/common": "2.4.7",
  "@angular/compiler": "2.4.7",
  "@angular/core": "2.4.7",
  "@angular/forms": "2.4.7",
  "@angular/http": "2.4.7",
  "@angular/platform-browser": "2.4.7",
  "@angular/platform-browser-dynamic": "2.4.7",
  "@angular/router": "3.4.7",
  "webpack": "^2.2.1",
  "awesome-typescript-loader": "^3.1.2",
},
"dependencies": {
}
}
```

Прочитайте Angular2 с помощью webpack онлайн:

<https://riptutorial.com/ru/angular2/topic/9554/angular2-с-помощью-webpack>

глава 7: CRUD в Angular2 с остальным API

Синтаксис

- `@Injectable ()` // Указывает, что инжектор зависимостей вводит зависимости при создании экземпляра этой службы.
- `request.subscribe ()` // Это где вы на *самом деле* сделать запрос. Без этого ваш запрос не будет отправлен. Также вы хотите прочитать ответ в функции обратного вызова.
- `constructor (private wikiService: WikipediaService) {}` // Поскольку и наша служба, и ее зависимости внедряются инжектором зависимостей, это хорошая практика, чтобы внедрить службу компоненту для модульного тестирования приложения.

Examples

Читайте из Restful API в Angular2

Чтобы отделить API-интерфейс от компонента, мы создаем клиент API как отдельный класс. Этот примерный класс делает запрос к API Википедии, чтобы получить случайные статьи wiki.

```
import { Http, Response } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/Rx';

@Injectable()
export class WikipediaService{
  constructor(private http: Http) {}

  getRandomArticles(numberOfArticles: number)
  {
    var request =
this.http.get("https://en.wikipedia.org/w/api.php?action=query&list=random&format=json&rnlimit="
+ numberOfArticles);
    return request.map((response: Response) => {
      return response.json();
    }, (error) => {
      console.log(error);
      //your want to implement your own error handling here.
    });
  }
}
```

И иметь компонент, чтобы потреблять наш новый клиент API.

```
import { Component, OnInit } from '@angular/core';
import { WikipediaService } from './wikipedia.Service';
```

```
@Component({
  selector: 'wikipedia',
  templateUrl: 'wikipedia.component.html'
})
export class WikipediaComponent implements OnInit {
  constructor(private wikiService: WikipediaService) { }

  private articles: any[] = null;
  ngOnInit() {
    var request = this.wikiService.getRandomArticles(5);
    request.subscribe((res) => {
      this.articles = res.query.random;
    });
  }
}
```

Прочитайте CRUD в Angular2 с остальным API онлайн:

<https://riptutorial.com/ru/angular2/topic/7343/crud-в-angular2-с-остальным-api>

глава 8: Dropzone в Angular2

Examples

Зона сброса

Угловая библиотека обложек 2 для Dropzone.

```
npm установка угловой2-dropzone-wrapper --save-dev
```

Загрузите модуль для вашего приложения-модуля

```
import { DropzoneModule } from 'angular2-dropzone-wrapper';
import { DropzoneConfigInterface } from 'angular2-dropzone-wrapper';

const DROPZONE_CONFIG: DropzoneConfigInterface = {
  // Change this to your upload POST address:
  server: 'https://example.com/post',
  maxFileSize: 10,
  acceptedFiles: 'image/*'
};

@NgModule({
  ...
  imports: [
    ...
    DropzoneModule.forRoot(DROPZONE_CONFIG)
  ]
})
```

КОМПОНЕНТНОЕ ИСПОЛЬЗОВАНИЕ

Просто замените элемент, который обычно передается Dropzone с компонентом dropzone.

```
<dropzone [config]="config" [message]="Click or drag images here to upload"
(error)="onUploadError($event)" (success)="onUploadSuccess($event)"></dropzone>
```

Создать компонент Dropzone

```
import {Component} from '@angular/core';
@Component({
  selector: 'app-new-media',
  templateUrl: './dropzone.component.html',
  styleUrls: ['./dropzone.component.scss']
})
export class DropZoneComponent {

  onUploadError(args: any) {
    console.log('onUploadError:', args);
  }
}
```

```
}  
  
onUploadSuccess(args: any) {  
    console.log('onUploadSuccess:', args);  
}  
}
```

Прочитайте Dropzone в Angular2 онлайн:

<https://riptutorial.com/ru/angular2/topic/10010/dropzone-в-angular2>

глава 9: Mocking @ ngrx / Магазин

Вступление

@ ngrx / Store становится все более широко использоваться в проектах Angular 2. Таким образом, магазин необходимо вставить в конструктор любого Компонента или Службы, который хочет его использовать. Тестирование модулей не так просто, как тестирование простого сервиса. Как и во многих проблемах, существует множество способов реализации решений. Однако основной рецепт заключается в том, чтобы написать класс mock для интерфейса Observer и написать класс mock для Store. Затем вы можете ввести Store в качестве поставщика в свой TestBed.

параметры

название	описание
значение	следующее значение
ошибка	описание
заблуждаться	ошибка, которую нужно выбросить
супер	описание
действие \$	mock Observer, который ничего не делает, если не определено, чтобы сделать это в классе mock
actionReducer \$	mock Observer, который ничего не делает, если не определено, чтобы сделать это в классе mock
набл \$	макет Наблюдаемый

замечания

Наблюдатель является общим, но должен иметь тип `any` чтобы избежать сложности тестирования модулей. Причина такой сложности заключается в том, что конструктор Store ожидает аргументы Observer с разными родовыми типами. Использование `any` метода позволяет избежать этого осложнения.

В суперконструкторе StoreMock можно передавать значения NULL, но это ограничивает количество утверждений, которые могут быть использованы для тестирования класса дальше по дороге.

Компонент, используемый в этом примере, просто используется в качестве контекста для того, как можно было бы внедрить *Store* в качестве обеспечения в тестовую настройку.

Examples

Наблюдатель Мок

```
class ObserverMock implements Observer<any> {
  closed?: boolean = false; // inherited from Observer
  nextVal: any = ''; // variable I made up

  constructor() {}

  next = (value: any): void => { this.nextVal = value; };
  error = (err: any): void => { console.error(err); };
  complete = (): void => { this.closed = true; }
}

let actionReducer$: ObserverMock = new ObserverMock();
let action$: ObserverMock = new ObserverMock();
let obs$: Observable<any> = new Observable<any>();

class StoreMock extends Store<any> {
  constructor() {
    super(action$, actionReducer$, obs$);
  }
}

describe('Component:Typeahead', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [...],
      declarations: [Typeahead],
      providers: [
        {provide: Store, useClass: StoreMock} // NOTICE useClass instead of useValue
      ]
    }).compileComponents();
  });
});
```

Единый тест для компонента с матовым хранилищем

Это единый тест компонента, который имеет *Store* как зависимость. Здесь мы создаем новый класс *MockStore*, который вводится в наш компонент вместо обычного *Store*.

```
import { Injectable } from '@angular/core';
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { DumbComponentComponent } from './dumb-component/dumb-component.component';
import { SmartComponentComponent } from './smart-component/smart-component.component';
import { mainReducer } from './state-management/reducers/main-reducer';
import { StoreModule } from '@ngrx/store';
import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';
```

```

class MockStore {
  public dispatch(obj) {
    console.log('dispatching from the mock store!')
  }

  public select(obj) {
    console.log('selecting from the mock store!');

    return Observable.of({})
  }
}

describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        SmartComponentComponent,
        DumbComponentComponent,
      ],
      imports: [
        StoreModule.provideStore({mainReducer})
      ],
      providers: [
        {provide: Store, useClass: MockStore}
      ]
    });
  });

  it('should create the app', async(() => {

    let fixture = TestBed.createComponent(AppComponent);
    let app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();

  }));
}

```

Единичный тест для компонентного шпионажа в магазине

Это единичный тест компонента, который имеет *Store* как зависимость. Здесь мы можем использовать хранилище со стандартным «начальным состоянием», предотвращая его фактическое диспетчеризацию при *вызове store.dispatch ()*.

```

import {TestBed, async} from '@angular/core/testing';
import {AppComponent} from './app.component';
import {DumbComponentComponent} from './dumb-component/dumb-component.component';
import {SmartComponentComponent} from './smart-component/smart-component.component';
import {mainReducer} from './state-management/reducers/main-reducer';
import {StoreModule} from '@ngrx/store';
import {Store} from '@ngrx/store';
import {Observable} from 'rxjs';

describe('AppComponent', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,

```

```

        SmartComponentComponent,
        DumbComponentComponent,
    ],
    imports: [
        StoreModule.provideStore({mainReducer})
    ]
});

});

it('should create the app', async(() => {
    let fixture = TestBed.createComponent(AppComponent);
    let app = fixture.debugElement.componentInstance;

    var mockStore = fixture.debugElement.injector.get(Store);
    var storeSpy = spyOn(mockStore, 'dispatch').and.callFake(function () {
        console.log('dispatching from the spy!');
    });

}));

});

```

Угловое 2 - Mock Observable (сервис + компонент)

оказание услуг

- Я создал почтовую службу с методом postRequest.

```

import {Injectable} from '@angular/core';
import {Http, Headers, Response} from "@angular/http";
import {PostModel} from "./PostModel";
import 'rxjs/add/operator/map';
import {Observable} from "rxjs";

@Injectable()
export class PostService {

    constructor(private _http: Http) {
    }

    postRequest(postModel: PostModel) : Observable<Response> {
        let headers = new Headers();
        headers.append('Content-Type', 'application/json');
        return this._http.post("/postUrl", postModel, {headers})
            .map(res => res.json());
    }
}

```

Составная часть

- Я создал компонент с параметром результата и функцией postExample, которые вызывают postService.

- когда успешное повторение сообщения, чем параметр результата, должно быть «Success» еще «Fail»

```
import {Component} from '@angular/core';
import {PostService} from "../PostService";
import {PostModel} from "../PostModel";

@Component({
  selector: 'app-post',
  templateUrl: './post.component.html',
  styleUrls: ['./post.component.scss'],
  providers : [PostService]
})
export class PostComponent{

  constructor(private _postService : PostService) {

    let postModel = new PostModel();
    result : string = null;
    postExample(){
      this._postService.postRequest(this.postModel)
        .subscribe(
          () => {
            this.result = 'Success';
          },
          err => this.result = 'Fail'
        )
    }
  }
}
```

тестовая услуга

- когда вы хотите проверить службу, используя http, вы должны использовать mockBackend. и приложить к нему.
- вам также нужно ввести postService.

```
describe('Test PostService', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpModule],
      providers: [
        PostService,
        MockBackend,
        BaseRequestOptions,
        {
          provide: Http,
          deps: [MockBackend, BaseRequestOptions],
          useFactory: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        }
      ]
    });
  });
});
```

```

it('sendPostRequest function return Observable', inject([PostService, MockBackend],
(service: PostService, mockBackend: MockBackend) => {
  let mockPostModel = PostModel();

  mockBackend.connections.subscribe((connection: MockConnection) => {
    expect(connection.request.method).toEqual(RequestMethod.Post);
    expect(connection.request.url.indexOf('postUrl')).not.toEqual(-1);
    expect(connection.request.headers.get('Content-Type')).toEqual('application/json');
  });

  service
    .postRequest(PostModel)
    .subscribe((response) => {
      expect(response).toBeDefined();
    });
}));
});

```

ТЕСТОВЫЙ КОМПОНЕНТ

```

describe('testing post component', () => {
  let component: PostComponent;
  let fixture: ComponentFixture<postComponent>;

  let mockRouter = {
    navigate: jasmine.createSpy('navigate')
  };

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [PostComponent],
      imports: [RouterTestingModule.withRoutes([], ModalModule.forRoot() )],
      providers: [PostService, MockBackend, BaseRequestOptions,
        {provide: Http, deps: [MockBackend, BaseRequestOptions],
          useFactory: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        },
        {provide: Router, useValue: mockRouter}
      ],
      schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(PostComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('test postRequest success', inject([PostService, MockBackend], (service: PostService,
mockBackend: MockBackend) => {
    fixturePostComponent = TestBed.createComponent(PostComponent);
    componentPostComponent = fixturePostComponent.componentInstance;
    fixturePostComponent.detectChanges();

```

```

component.postExample();
let postModel = new PostModel();
let response = {
  'message' : 'message',
  'ok'      : true
};
mockBackend.connections.subscribe((connection: MockConnection) => {
  postComponent.result = 'Success'
  connection.mockRespond(new Response(
    new ResponseOptions({
      body: response
    })
  ))
});
service.postRequest(postModel)
  .subscribe((data) => {
    expect(component.result).toBeDefined();
    expect(PostComponent.result).toEqual('Success');
    expect(data).toEqual(response);
  });
}));
});

```

Простой магазин

simple.action.ts

```

import { Action } from '@ngrx/store';

export enum simpleActionType {
  add = "simpleAction_Add",
  add_Success = "simpleAction_Add_Success"
}

export class simpleAction {
  type: simpleActionType
  constructor(public payload: number) { }
}

```

simple.effects.ts

```

import { Effect, Actions } from '@ngrx/effects';
import { Injectable } from '@angular/core';
import { Action } from '@ngrx/store';
import { Observable } from 'rxjs';

import { simpleAction, simpleActionType } from './simple.action';

@Injectable()
export class simpleEffects {

  @Effect()
  addAction$: Observable<simpleAction> = this.actions$
    .ofType(simpleActionType.add)
    .switchMap((action: simpleAction) => {

```

```

        console.log(action);

        return Observable.of({ type: simpleActionType.add_Success, payload: action.payload
    })
        // if you have an api use this code
        // return this.http.post(url).catch().map(res=>{ type: simpleAction.add_Success,
payload:res})
        });
    constructor(private actions$: Actions) { }
}

```

simple.reducer.ts

```

import { Action, ActionReducer } from '@ngrx/store';

import { simpleAction, simpleActionType } from './simple.action';

export const simpleReducer: ActionReducer<number> = (state: number = 0, action: simpleAction):
number => {
    switch (action.type) {
        case simpleActionType.add_Success:
            console.log(action);
            return state + action.payload;
        default:
            return state;
    }
}

```

магазин / index.ts

```

import { combineReducers, ActionReducer, Action, StoreModule } from '@ngrx/store';
import { EffectsModule } from '@ngrx/effects';
import { ModuleWithProviders } from '@angular/core';
import { compose } from '@ngrx/core';

import { simpleReducer } from './simple/simple.reducer';
import { simpleEffects } from './simple/simple.effects';

export interface IAppState {
    sum: number;
}

// all new reducers should be define here
const reducers = {
    sum: simpleReducer
};

export const store: ModuleWithProviders = StoreModule.forRoot(reducers);
export const effects: ModuleWithProviders[] = [
    EffectsModule.forRoot([simpleEffects])
];

```

app.module.ts

```

import { BrowserModule } from '@angular/platform-browser'
import { NgModule } from '@angular/core';

```

```

import { effects, store } from './Store/index';
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    // store
    store,
    effects
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

app.component.ts

```

import { Component } from '@angular/core';

import { Store } from '@ngrx/store';
import { Observable } from 'rxjs';

import { IAppState } from './Store/index';
import { simpleActionTpye } from './Store/simple/simple.action';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'app';

  constructor(private store: Store<IAppState>) {
    store.select(s => s.sum).subscribe((res) => {
      console.log(res);
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 1
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 2
    })
    this.store.dispatch({
      type: simpleActionTpye.add,
      payload: 3
    })
  }
}

```

результат 0 1 3 6

Прочитайте [Mocking @ ngrx / Магазин онлайн](#):

<https://riptutorial.com/ru/angular2/topic/8038/mocking---ngrx---магазин>

глава 10: ngrx

Вступление

Ngrx - мощная библиотека, которую вы можете использовать с **Angular2** . Идея состоит в том, чтобы объединить две концепции, которые хорошо сочетаются, чтобы иметь **реактивное приложение** с предсказуемым **контейнером состояния** : - [Redux] [1] - [RxJs] [2] Основные преимущества: - Совместное использование данных в приложении между вашими компонентами будет проще. Тестирование основной логики приложения состоит в проверке чистых функций без какой-либо зависимости от Angular2 (очень просто!) [1]: <http://redux.js.org> [2]: <http://reactivex.io/rxjs>

Examples

Полный пример: Вход / выход пользователя

Предпосылки

Эта тема **не** касается Redux и / или Ngrx:

- Вы должны быть довольны Redux
- По крайней мере, понять основы RxJs и наблюдаемого шаблона

Сначала давайте определим пример с самого начала и поиграем с некоторым кодом:

Как разработчик, я хочу:

1. Имейте интерфейс `IUser` который определяет свойства `User`
2. Объявите действия, которые мы будем использовать позже, чтобы манипулировать `User` в `Store`
3. Определить начальное состояние `UserReducer`
4. Создайте редуктор `UserReducer`
5. Импортируйте наш `UserReducer` в наш основной модуль, чтобы создать `Store`
6. Используйте данные из `Store` для отображения информации в нашем представлении

Предупреждение о спойлере : если вы хотите попробовать демо сразу или прочитать код, прежде чем мы начнем, вот Plunkr ([встроенный просмотр](#) или [просмотр](#)).

1) Определить интерфейс `IUser`

Мне нравится разделить мои интерфейсы на две части:

- Свойства, которые мы получим с сервера
- Свойства, которые мы определяем только для UI (например, при нажатии кнопки)

И вот интерфейс `IUser` мы будем использовать:

user.interface.ts

```
export interface IUser {
  // from server
  username: string;
  email: string;

  // for UI
  isConnected: boolean;
  isConnecting: boolean;
};
```

2) Объявить действия для манипулирования `User`

Теперь нам нужно подумать о том, какие действия должны выполнять наши **редукторы**. Скажем здесь:

user.actions.ts

```
export const UserActions = {
  // when the user clicks on login button, before we launch the HTTP request
  // this will allow us to disable the login button during the request
  USR_IS_CONNECTING: 'USR_IS_CONNECTING',
  // this allows us to save the username and email of the user
  // we assume those data were fetched in the previous request
  USR_IS_CONNECTED: 'USR_IS_CONNECTED',

  // same pattern for disconnecting the user
  USR_IS_DISCONNECTING: 'USR_IS_DISCONNECTING',
  USR_IS_DISCONNECTED: 'USR_IS_DISCONNECTED'
};
```

Но прежде чем мы будем использовать эти действия, позвольте мне объяснить, почему нам понадобится служба для отправки **некоторых** из этих действий для нас:

Скажем, что мы хотим подключить пользователя. Таким образом, мы будем нажимать кнопку входа в систему, и вот что произойдет:

- Нажмите на кнопку
- Компонент поймает событие и вызовет `userService.login`
- Метод `userService.login` dispatch событие для обновления нашего свойства хранилища: `user.isConnected`
- Вызывается HTTP-вызов (мы будем использовать `setTimeout` в демо, чтобы

имитировать поведение `async`)

- Как только `HTTP` вызов будет завершен, мы отправим другое действие, чтобы предупредить наш магазин о том, что пользователь зарегистрирован

user.service.ts

```
@Injectable()
export class UserService {
  constructor(public store$: Store<AppState>) { }

  login(username: string) {
    // first, dispatch an action saying that the user's trying to connect
    // so we can lock the button until the HTTP request finish
    this.store$.dispatch({ type: UserActions.USR_IS_CONNECTING });

    // simulate some delay like we would have with an HTTP request
    // by using a timeout
    setTimeout(() => {
      // some email (or data) that you'd have get as HTTP response
      let email = `${username}@email.com`;

      this.store$.dispatch({ type: UserActions.USR_IS_CONNECTED, payload: { username, email }
    });
  }, 2000);
}

  logout() {
    // first, dispatch an action saying that the user's trying to connect
    // so we can lock the button until the HTTP request finish
    this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTING });

    // simulate some delay like we would have with an HTTP request
    // by using a timeout
    setTimeout(() => {
      this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTED });
    }, 2000);
  }
}
```

3) Определите начальное состояние `UserReducer`

user.state.ts

```
export const UserFactory: IUser = () => {
  return {
    // from server
    username: null,
    email: null,

    // for UI
    isConnected: false,
    isConnecting: false,
    isDisconnecting: false
  };
};
```

4) Создайте редуктор UserReducer

Редуктор принимает 2 аргумента:

- Текущее состояние
- Action типа `Action<{type: string, payload: any}>`

Напоминание: редуктор должен быть инициализирован в какой-то момент

Поскольку мы определили состояние по умолчанию нашего редуктора в части 3), мы сможем использовать его так:

`user.reducer.ts`

```
export const UserReducer: ActionReducer<IUser> = (user: IUser, action: Action) => {
  if (user === null) {
    return userFactory();
  }

  // ...
}
```

Надеюсь, есть более простой способ написать, что, используя нашу `factory` функцию для возврата объекта и внутри редуктора, используйте [значение параметров по умолчанию \(ES6\)](#):

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  // ...
}
```

Затем нам нужно обработать все действия в нашем редукторе: **СОВЕТ**: используйте функцию [ES6 `Object.assign`](#) чтобы сохранить наше состояние неизменным

```
export const UserReducer: ActionReducer<IUser> = (user: IUser = UserFactory(), action: Action)
=> {
  switch (action.type) {
    case UserActions.USER_IS_CONNECTING:
      return Object.assign({}, user, { isConnecting: true });

    case UserActions.USER_IS_CONNECTED:
      return Object.assign({}, user, { isConnecting: false, isConnected: true, username:
action.payload.username });

    case UserActions.USER_IS_DISCONNECTING:
      return Object.assign({}, user, { isDisconnecting: true });

    case UserActions.USER_IS_DISCONNECTED:
```

```
    return Object.assign({}, user, { isDisconnecting: false, isConnected: false });

    default:
      return user;
  }
};
```

5) Импортируйте наш `UserReducer` в наш основной модуль, чтобы создать `Store`

app.module.ts

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    // angular modules
    // ...

    // declare your store by providing your reducers
    // (every reducer should return a default state)
    StoreModule.provideStore({
      user: UserReducer,
      // of course, you can put as many reducers here as you want
      // ...
    }),

    // other modules to import
    // ...
  ]
});
```

6) Используйте данные из `Store` для отображения информации на наш взгляд

Теперь все готово с логической стороны, и нам просто нужно отобразить то, что мы хотим, в двух компонентах:

- `UserComponent` : **[Dumb component]** Мы просто передадим объект пользователя из магазина с использованием свойства `@Input` и `async pipe`. Таким образом, компонент получит пользователя только после его доступности (и `user` будет иметь тип `IUser` а не тип `Observable<IUser>` !)
- `LoginComponent` **[Интеллектуальный компонент]** Мы будем непосредственно вводить `Store` в этот компонент и работать только с `user` в качестве `Observable` .

user.component.ts

```

@Component({
  selector: 'user',
  styles: [
    '.table { max-width: 250px; }',
    '.truthy { color: green; font-weight: bold; }',
    '.falsy { color: red; }'
  ],
  template: `
    <h2>User information :</h2>

    <table class="table">
      <tr>
        <th>Property</th>
        <th>Value</th>
      </tr>

      <tr>
        <td>username</td>
        <td [class.truthy]="user.username" [class.falsy]="!user.username">
          {{ user.username ? user.username : 'null' }}
        </td>
      </tr>

      <tr>
        <td>email</td>
        <td [class.truthy]="user.email" [class.falsy]="!user.email">
          {{ user.email ? user.email : 'null' }}
        </td>
      </tr>

      <tr>
        <td>isConnecting</td>
        <td [class.truthy]="user.isConnecting" [class.falsy]="!user.isConnecting">
          {{ user.isConnecting }}
        </td>
      </tr>

      <tr>
        <td>isConnected</td>
        <td [class.truthy]="user.isConnected" [class.falsy]="!user.isConnected">
          {{ user.isConnected }}
        </td>
      </tr>

      <tr>
        <td>isDisconnecting</td>
        <td [class.truthy]="user.isDisconnecting" [class.falsy]="!user.isDisconnecting">
          {{ user.isDisconnecting }}
        </td>
      </tr>
    </table>
  `
})
export class UserComponent {
  @Input() user;

  constructor() { }
}

```

login.component.ts

```

@Component ({
  selector: 'login',
  template: `
    <form
      *ngIf="!(user | async).isConnected"
      #loginForm="ngForm"
      (ngSubmit)="login(loginForm.value.username) "
    >
      <input
        type="text"
        name="username"
        placeholder="Username"
        [disabled]="(user | async).isConnecting"
        ngModel
      >

      <button
        type="submit"
        [disabled]="(user | async).isConnecting || (user | async).isConnected"
      >Log me in</button>
    </form>

    <button
      *ngIf="(user | async).isConnected"
      (click)="logout () "
      [disabled]="(user | async).isDisconnecting"
    >Log me out</button>
  `
})
export class LoginComponent {
  public user: Observable<IUser>;

  constructor(public store$: Store<AppState>, private userService: UserService) {
    this.user = store$.select('user');
  }

  login(username: string) {
    this.userService.login(username);
  }

  logout () {
    this.userService.logout ();
  }
}

```

Поскольку `Ngrx` является слиянием концепций `Redux` и `RxJs`, `RxJs` может быть трудно понять входы. Но это мощный шаблон, который позволяет вам, как мы видели в этом примере, иметь *реактивное приложение*, и вы можете легко поделиться своими данными. Не забывайте, что есть [Plunkr](https://plunkr.io/), и вы можете разветвить его, чтобы сделать свои собственные тесты!

Надеюсь, это было полезно, даже если тема довольно длинная, ура!

Прочитайте `ngrx` онлайн: <https://riptutorial.com/ru/angular2/topic/8086/ngrx>

глава 11: Zone.js

Examples

Получение ссылки на NgZone

NgZone может быть введена через Injection Dependency (DI).

my.component.ts

```
import { Component, NgOnInit, NgZone } from '@angular/core';

@Component({...})
export class Mycomponent implements NgOnInit {
  constructor(private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      // Do something outside Angular so it won't get noticed
    });
  }
}
```

Использование NgZone для выполнения нескольких HTTP-запросов перед показом данных

runOutsideAngular можно использовать для запуска кода вне углового 2, чтобы он не runOutsideAngular ненужное изменение. Это можно использовать, например, для запуска нескольких HTTP-запросов для получения всех данных перед их рендерингом. Чтобы снова выполнить код внутри Angular 2, можно run метод NgZone .

my.component.ts

```
import { Component, OnInit, NgZone } from '@angular/core';
import { Http } from '@angular/http';

@Component({...})
export class Mycomponent implements OnInit {
  private data: any[];
  constructor(private http: Http, private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      this.http.get('resource1').subscribe((data1:any) => {
        // First response came back, so its data can be used in consecutive request
        this.http.get(`resource2?id=${data1['id']}`).subscribe((data2:any) => {
          this.http.get(`resource3?id1=${data1['id']}&id2=${data2}`).subscribe((data3:any) =>
        {
          this._ngZone.run(() => {
            this.data = [data1, data2, data3];
          });
        });
      });
    });
  }
}
```

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

Прочитайте Zone.js онлайн: <https://riptutorial.com/ru/angular2/topic/4184/zone-js>

глава 12: Анимация

Examples

Переход между нулевыми состояниями

```
@Component ({
  ...
  animations: [
    trigger('appear', [
      transition(':enter', [
        style({
          //style applied at the start of animation
        }),
        animate('300ms ease-in', style({
          //style applied at the end of animation
        }))
      ])
    ])
  ]
})
class AnimComponent {
}
]
```

Анимация между несколькими состояниями

<div> в этом шаблоне растёт до 50px а затем 100px а затем сжимается до 20px при нажатии кнопки.

Каждое state имеет связанный стиль, описанный в метаданных @Component .

Логикой для любого state является активное управление в логике компонентов. В этом случае size переменной компонента содержит строковое значение «маленький», «средний» или «большой».

Элемент <div> отвечает на это значение через trigger указанный в метаданных @Component :
[@size]="size" .

```
@Component ({
  template: '<div [@size]="size">Some Text</div><button
(click)="toggleSize()">TOGGLE</button>',
  animations: [
    trigger('size', [
      state('small', style({
        height: '20px'
      })),
      state('medium', style({
        height: '50px'
      })),
    ])
  ]
})
```

```
state('large', style({
  height: '100px'
})),
transition('small => medium', animate('100ms')),
transition('medium => large', animate('200ms')),
transition('large => small', animate('300ms'))
])
])
})
export class TestComponent {

  size: string;

  constructor(){
    this.size = 'small';
  }
  toggleSize(){
    switch(this.size) {
      case 'small':
        this.size = 'medium';
        break;
      case 'medium':
        this.size = 'large';
        break;
      case 'large':
        this.size = 'small';
    }
  }
}
```

Прочитайте Анимация онлайн: <https://riptutorial.com/ru/angular2/topic/8127/анимация>

глава 13: бочка

Вступление

Баррель - это способ свернуть экспорт из нескольких модулей ES2015 в единый модуль ES2015. Сам ствол является файлом модуля ES2015, который реэкспортирует выбранный экспорт других модулей ES2015.

Examples

Использование ствола

Например, без барреля потребителю потребуется три оператора импорта:

```
import { HeroComponent } from '../heroes/hero.component.ts';
import { Hero }          from '../heroes/hero.model.ts';
import { HeroService }  from '../heroes/hero.service.ts';
```

Мы можем добавить баррель, создав файл в той же папке компонента. В этом случае папка называется «heroes» с именем index.ts (используя соглашения), которая экспортирует все эти элементы:

```
export * from './hero.model.ts'; // re-export all of its exports
export * from './hero.service.ts'; // re-export all of its exports
export { HeroComponent } from './hero.component.ts'; // re-export the named thing
```

Теперь потребитель может импортировать то, что ему нужно из ствола.

```
import { Hero, HeroService } from '../heroes/index';
```

Тем не менее, это может стать очень длинной линией; которые могут быть еще более сокращены.

```
import * as h from '../heroes/index';
```

Это довольно мало! * as h импортирует все модули и псевдонимы как h

Прочитайте бочка онлайн: <https://riptutorial.com/ru/angular2/topic/10717/бочка>

глава 14: Бутстрап Пустой модуль в угловом 2

Examples

Пустой модуль

```
import { NgModule } from '@angular/core';

@NgModule({
  declarations: [], // components your module owns.
  imports: [], // other modules your module needs.
  providers: [], // providers available to your module.
  bootstrap: [] // bootstrap this root component.
})
export class MyModule {}
```

Это пустой модуль, не содержащий деклараций, импорта, поставщиков или компонентов для загрузки. Используйте эту ссылку.

Модуль с сетью в веб-браузере.

```
// app.module.ts

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/http';
import { MyRootComponent } from './app.component';

@NgModule({
  declarations: [MyRootComponent],
  imports: [BrowserModule, HttpClientModule],
  bootstrap: [MyRootComponent]
})
export class MyModule {}
```

MyRootComponent - это корневой компонент, упакованный в MyModule . Это точка входа в ваше приложение Angular 2.

Загрузочный модуль

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { MyModule } from './app.module';

platformBrowserDynamic().bootstrapModule( MyModule );
```

В этом примере MyModule - это модуль, содержащий ваш корневой компонент. MyModule ваше

приложение Angular 2 будет готово к работе.

Корневой модуль приложения

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';

@NgModule({
  imports: [ BrowserModule ],
  declarations: [ AppComponent ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Статическая загрузка с заводскими классами

Мы можем статически загружать приложение, используя простой вывод ES5 Javascript сгенерированных фабричных классов. Затем мы можем использовать этот вывод для загрузки приложения:

```
import { platformBrowser } from '@angular/platform-browser';
import { AppModuleNgFactory } from './main.ngfactory';

// Launch with the app module factory.
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

Это приведет к значительному сокращению набора приложений, потому что вся компиляция шаблона уже была выполнена на этапе сборки, используя либо ngc, либо напрямую вызывающий его внутренние элементы.

Прочитайте [Бутстрап Пустой модуль в угловом 2 онлайн](https://riptutorial.com/ru/angular2/topic/5508/бутстрап-пустой-модуль-в-угловом-2):

<https://riptutorial.com/ru/angular2/topic/5508/бутстрап-пустой-модуль-в-угловом-2>

глава 15: Выход Angular2 Input () ()

Examples

Ввод ()

Родительский компонент: Инициализировать списки пользователей.

```
@Component ({
  selector: 'parent-component',
  template: '<div>
    <child-component [users]="users"></child-component>
  </div>'
})
export class ParentComponent implements OnInit {
  let users : List<User> = null;

  ngOnInit () {
    users.push(new User('A', 'A', 'A@gmail.com'));
    users.push(new User('B', 'B', 'B@gmail.com'));
    users.push(new User('C', 'C', 'C@gmail.com'));
  }
}
```

Детский компонент получает пользователя от родительского компонента с помощью Input ()

```
@Component ({
  selector: 'child-component',
  template: '<div>
    <table *ngIf="users !== null">
      <thead>
        <th>Name</th>
        <th>FName</th>
        <th>Email</th>
      </thead>
      <tbody>
        <tr *ngFor="let user of users">
          <td>{{user.name}}</td>
          <td>{{user.fname}}</td>
          <td>{{user.email}}</td>
        </tr>
      </tbody>
    </table>

  </div>',
})
export class ChildComponent {
  @Input () users : List<User> = null;
}
```



```
export class User {
  name : string;
  fname : string;
  email : string;

  constructor(_name : string, _fname : string, _email : string){
    this.name = _name;
    this.fname = _fname;
    this.email = _email;
  }
}
```

Простой пример свойств ввода

Родительский элемент html

```
<child-component [isSelected]="inputPropValue"></child-component>
```

Родительский элемент ts

```
export class AppComponent {
  inputPropValue: true
}
```

Детский компонент ts:

```
export class ChildComponent {
  @Input() inputPropValue = false;
}
```

Детский компонент html:

```
<div [class.simpleCssClass]="inputPropValue"></div>
```

Этот код отправит `inputPropValue` из родительского компонента в дочерний элемент и он будет иметь значение, которое мы установили в родительском компоненте, когда оно поступит туда, - `false` в нашем случае. Затем мы можем использовать это значение в дочернем компоненте, например, добавить класс к элементу.

Прочитайте Выход Angular2 Input () () онлайн: <https://riptutorial.com/ru/angular2/topic/8943/выход-angular2-input----->

глава 16: Динамически добавлять компоненты с помощью ViewContainerRef.createComponent

Examples

Компонент-оболочка, который декларативно добавляет динамические компоненты

Пользовательский компонент, который принимает тип компонента как входной и создает экземпляр этого типа компонента внутри себя. Когда ввод обновляется, ранее добавленный динамический компонент удаляется, а новый добавляется вместо него.

```
@Component({
  selector: 'dcl-wrapper',
  template: `<div #target></div>`
})
export class DclWrapper {
  @ViewChild('target', {
    read: ViewContainerRef
  }) target;
  @Input() type;
  cmpRef: ComponentRef;
  private isViewInitialized: boolean = false;

  constructor(private resolver: ComponentResolver) {}

  updateComponent() {
    if (!this.isViewInitialized) {
      return;
    }
    if (this.cmpRef) {
      this.cmpRef.destroy();
    }
    this.resolver.resolveComponent(this.type).then((factory: ComponentFactory < any > ) => {
      this.cmpRef = this.target.createComponent(factory)
      // to access the created instance use
      // this.cmpRef.instance.someProperty = 'someValue';
      // this.cmpRef.instance.someOutput.subscribe(val => doSomething());
    });
  }

  ngOnChanges() {
    this.updateComponent();
  }

  ngAfterViewInit() {
    this.isViewInitialized = true;
    this.updateComponent();
  }
}
```

```
ngOnDestroy() {
  if (this.cmpRef) {
    this.cmpRef.destroy();
  }
}
```

Это позволяет создавать динамические компоненты, такие как

```
<dcl-wrapper [type]="someComponentType"></dcl-wrapper>
```

Пример Plunker

Динамически добавлять компонент к определенному событию (щелкните)

Основной файл компонента:

```
//our root app component
import {Component, NgModule, ViewChild, ViewContainerRef, ComponentFactoryResolver,
ComponentRef} from '@angular/core'
import {BrowserModule} from '@angular/platform-browser'
import {ChildComponent} from './childComp.ts'

@Component({
  selector: 'my-app',
  template: `
    <div>
      <h2>Hello {{name}}</h2>
      <input type="button" value="Click me to add element" (click) = addElement()> // call the
function on click of the button
      <div #parent> </div> // Dynamic component will be loaded here
    </div>
  `,
})
export class App {
  name:string;

  @ViewChild('parent', {read: ViewContainerRef}) target: ViewContainerRef;
  private componentRef: ComponentRef<any>;

  constructor(private componentFactoryResolver: ComponentFactoryResolver) {
    this.name = 'Angular2'
  }

  addElement(){
    let childComponent =
this.componentFactoryResolver.resolveComponentFactory(ChildComponent);
    this.componentRef = this.target.createComponent(childComponent);
  }
}
```

childComp.ts:

```
import{Component} from '@angular/core';
```

```

@Component({
  selector: 'child',
  template: `
    <p>This is Child</p>
  `,
})
export class ChildComponent {
  constructor() {

  }
}

```

app.module.ts:

```

@NgModule({
  imports: [ BrowserModule ],
  declarations: [ App, ChildComponent ],
  bootstrap: [ App ],
  entryComponents: [ChildComponent] // define the dynamic component here in module.ts
})
export class AppModule {}

```

Пример Plunker

Отображение динамически созданного массива компонентов на шаблоне html в Angular2

Мы можем создать динамический компонент и получить экземпляры компонента в массив и, наконец, отобразить его на шаблоне.

Например, мы можем рассмотреть два компонента виджета: ChartWidget и PatientWidget, которые расширили класс WidgetComponent, который я хотел добавить в контейнер.

ChartWidget.ts

```

@Component({
  selector: 'chart-widget',
  templateUrl: 'chart-widget.component.html',
  providers: [{provide: WidgetComponent, useExisting: forwardRef(() => ChartWidget) }]
})

export class ChartWidget extends WidgetComponent implements OnInit {
  constructor(ngEl: ElementRef, renderer: Renderer) {
    super(ngEl, renderer);
  }
  ngOnInit() {}
  close() {
    console.log('close');
  }
  refresh() {
    console.log('refresh');
  }
  ...
}

```

chart-widget.component.html (с использованием панели primeng)

```
<p-panel [style]="{'margin-bottom':'20px'}">
  <p-header>
    <div class="ui-helper-clearfix">
      <span class="ui-panel-title" style="font-size:14px;display:inline-block;margin-top:2px">Chart Widget</span>
      <div class="ui-toolbar-group-right">
        <button pButton type="button" icon="fa-window-minimize"
(click)="minimize()" ></button>
        <button pButton type="button" icon="fa-refresh" (click)="refresh()" ></button>
        <button pButton type="button" icon="fa-expand" (click)="expand()" ></button>
        <button pButton type="button" (click)="close()" icon="fa-window-close"></button>
      </div>
    </div>
  </p-header>
  some data
</p-panel>
```

DataWidget.ts

```
@Component({
  selector: 'data-widget',
  templateUrl: 'data-widget.component.html',
  providers: [{provide: WidgetComponent, useExisting: forwardRef(() =>DataWidget) }]
})

export class DataWidget extends WidgetComponent implements OnInit {
  constructor(ngEl: ElementRef, renderer: Renderer) {
    super(ngEl, renderer);
  }
  ngOnInit() {}
  close(){
    console.log('close');
  }
  refresh(){
    console.log('refresh');
  }
  ...
}
```

data-widget.component.html (так же, как виджет диаграммы с использованием панели primeng)

WidgetComponent.ts

```
@Component({
  selector: 'widget',
  template: '<ng-content></ng-content>'
})
export class WidgetComponent{
}
```

мы можем создавать экземпляры динамических компонентов, выбирая уже существующие компоненты. Например,

```

@Component({
  selector: 'dynamic-component',
  template: `<div #container><ng-content></ng-content></div>`
})
export class DynamicComponent {
  @ViewChild('container', {read: ViewContainerRef}) container: ViewContainerRef;

  public addComponent(ngItem: Type<WidgetComponent>): WidgetComponent {
    let factory = this.compFactoryResolver.resolveComponentFactory(ngItem);
    const ref = this.container.createComponent(factory);
    const newItem: WidgetComponent = ref.instance;
    this._elements.push(newItem);
    return newItem;
  }
}

```

Наконец, мы используем его в компоненте приложения. `app.component.ts`

```

@Component({
  selector: 'app-root',
  templateUrl: './app/app.component.html',
  styleUrls: ['./app/app.component.css'],
  entryComponents: [ChartWidget, DataWidget],
})

export class AppComponent {
  private elements: Array<WidgetComponent>=[];
  private WidgetClasses = {
    'ChartWidget': ChartWidget,
    'DataWidget': DataWidget
  }
  @ViewChild(DynamicComponent) dynamicComponent:DynamicComponent;

  addComponent(widget: string ): void{
    let ref= this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
    this.elements.push(ref);
    console.log(this.elements);

    this.dynamicComponent.resetContainer();
  }
}

```

`app.component.html`

```

<button (click)="addComponent('ChartWidget')">Add ChartWidget</button>
<button (click)="addComponent('DataWidget')">Add DataWidget</button>

<dynamic-component [hidden]="true" ></dynamic-component>

<hr>
Dynamic Components
<hr>
<widget *ngFor="let item of elements">
  <div>{{item}}</div>
  <div [innerHTML]="item._ngEl.nativeElement.innerHTML | sanitizeHtml">
  </div>
</widget>

```

<https://plnkr.co/edit/lugU2pPsSBd3XhPHiUP1?p=preview>

Некоторая модификация @yurzui для использования события мыши в виджетах

view.directive.ts

import {ViewRef, Directive, Input, ViewContainerRef} из '@ angular / core';

```
@Directive({
  selector: '[view]'
})
export class ViewDirective {
  constructor(private vcRef: ViewContainerRef) {}

  @Input ()
  set view(view: ViewRef) {
    this.vcRef.clear();
    this.vcRef.insert(view);
  }

  ngOnDestroy() {
    this.vcRef.clear()
  }
}
```

app.component.ts

```
private elements: Array<{ view: ViewRef, component: WidgetComponent}> = [];

...
addComponent(widget: string ): void{
  let component = this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
  let view: ViewRef = this.dynamicComponent.container.detach(0);
  this.elements.push({view, component});

  this.dynamicComponent.resetContainer();
}
```

app.component.html

```
<widget *ngFor="let item of elements">
  <ng-container *view="item.view"></ng-container>
</widget>
```

<https://plnkr.co/edit/JHpIHR43SvJd0OxJVMfV?p=preview>

Прочитайте [Динамически добавлять компоненты с помощью](#)

[ViewContainerRef.createComponent](#) онлайн: <https://riptutorial.com/ru/angular2/topic/831/динамически-добавлять-компоненты-с-помощью-viewcontainerref-createcomponent>

глава 17: Директивы

Синтаксис

- `<input [value]="value">` - Персональное значение атрибут член класса `name` .
- `<div [attr.data-note]="note">` - Персональный атрибут `data-note` к переменной `note` .
- `<p green></p>` - Пользовательская директива

замечания

Основным источником информации о директивах Angular 2 является официальная документация <https://angular.io/docs/ts/latest/guide/attribute-directives.html>

Examples

Директива атрибута

```
<div [class.active]="isActive"></div>

<span [style.color]="red"></span>

<p [attr.data-note]="This is value for data-note attribute">A lot of text here</p>
```

Компонент - это директива с шаблоном

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `
    <h1>Angular 2 App</h1>
    <p>Component is directive with template</p>
  `
})
export class AppComponent {
}
```

Структурные директивы

```
<div *ngFor="let item of items">{{ item.description }}</div>

<span *ngIf="isVisible"></span>
```

Пользовательская директива


```
import {Directive, ElementRef, Renderer} from '@angular/core';

@Directive({
  selector: '[green]',
})

class GreenDirective {
  constructor(private _elementRef: ElementRef,
              private _renderer: Renderer) {
    _renderer.setStyle(_elementRef.nativeElement, 'color', 'green');
  }
}
```

Использование:

```
<p green>A lot of green text here</p>
```

* ngFor

form1.component.ts:

```
import { Component } from '@angular/core';

// Defines example component and associated template
@Component({
  selector: 'example',
  template: `
    <div *ngFor="let f of fruit"> {{f}} </div>
    <select required>
      <option *ngFor="let f of fruit" [value]="f"> {{f}} </option>
    </select>
  `
})

// Create a class for all functions, objects, and variables
export class ExampleComponent {
  // Array of fruit to be iterated by *ngFor
  fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];
}
```

Выход:

```
<div>Apples</div>
<div>Oranges</div>
<div>Bananas</div>
<div>Limes</div>
<div>Lemons</div>
<select required>
  <option value="Apples">Apples</option>
  <option value="Oranges">Oranges</option>
  <option value="Bananas">Bananas</option>
  <option value="Limes">Limes</option>
  <option value="Lemons">Lemons</option>
</select>
```

В самой простой форме `*ngFor` имеет две части: `let variableName of object/array`

В случае `fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];`,

Яблоки, апельсины и так далее являются значением внутри массива `fruit`.

`[value]="f"` будет равно каждому текущему `fruit (f)`, что `*ngFor` имеет итерацию.

В отличие от AngularJS, Angular2 не продолжался с использованием `ng-options` для `<select>` и `ng-repeat` для всех других общих повторений.

`*ngFor` очень похож на `ng-repeat` со слегка измененным синтаксисом.

Рекомендации:

Angular2 | [Отображение данных](#)

Angular2 | [ngFor](#)

Angular2 | [формы](#)

Скопировать в буфер обмена

В этом примере мы собираемся создать директиву для копирования текста в буфер обмена, нажав на элемент

Тетради `text.directive.ts`

```
import {
  Directive,
  Input,
  HostListener
} from "@angular/core";

@Directive({
  selector: '[text-copy]'
})
export class TextCopyDirective {

  // Parse attribute value into a 'text' variable
  @Input('text-copy') text:string;

  constructor() {
  }

  // The HostListener will listen to click events and run the below function, the
  // HostListener supports other standard events such as mouseenter, mouseleave etc.
  @HostListener('click') copyText() {

    // We need to create a dummy textarea with the text to be copied in the DOM
    var textArea = document.createElement("textarea");
```

```

// Hide the textarea from actually showing
textarea.style.position = 'fixed';
textarea.style.top = '-999px';
textarea.style.left = '-999px';
textarea.style.width = '2em';
textarea.style.height = '2em';
textarea.style.padding = '0';
textarea.style.border = 'none';
textarea.style.outline = 'none';
textarea.style.boxShadow = 'none';
textarea.style.background = 'transparent';

// Set the texarea's content to our value defined in our [text-copy] attribute
textarea.value = this.text;
document.body.appendChild(textArea);

// This will select the textarea
textarea.select();

try {
  // Most modern browsers support execCommand('copy'|'cut'|'paste'), if it doesn't
  it should throw an error
  var successful = document.execCommand('copy');
  var msg = successful ? 'successful' : 'unsuccessful';
  // Let the user know the text has been copied, e.g toast, alert etc.
  console.log(msg);
} catch (err) {
  // Tell the user copying is not supported and give alternative, e.g alert window
  with the text to copy
  console.log('unable to copy');
}

// Finally we remove the textarea from the DOM
document.body.removeChild(textArea);
}
}

export const TEXT_COPY_DIRECTIVES = [TextCopyDirective];

```

некоторые-page.component.html

Не забудьте ввести TEXT_COPY_DIRECTIVES в массив директив вашего компонента

```

...
<!-- Insert variable as the attribute's value, let textToBeCopied = 'http://facebook.com/'
-->
<button [text-copy]="textToBeCopied">Copy URL</button>
<button [text-copy]=" 'https://www.google.com/' ">Copy URL</button>
...

```

Тестирование пользовательской директивы

Учитывая директиву, которая выделяет текст на событиях мыши

```

import { Directive, ElementRef, HostListener, Input } from '@angular/core';

@Directive({ selector: '[appHighlight]' })

```

```

export class HighlightDirective {
  @Input('appHighlight') // tslint:disable-line no-input-rename
  highlightColor: string;

  constructor(private el: ElementRef) { }

  @HostListener('mouseenter')
  onMouseEnter() {
    this.highlight(this.highlightColor || 'red');
  }

  @HostListener('mouseleave')
  onMouseLeave() {
    this.highlight(null);
  }

  private highlight(color: string) {
    this.el.nativeElement.style.backgroundColor = color;
  }
}

```

Его можно протестировать следующим образом

```

import { ComponentFixture, ComponentFixtureAutoDetect, TestBed } from '@angular/core/testing';

import { Component } from '@angular/core';
import { HighlightDirective } from './highlight.directive';

@Component({
  selector: 'app-test-container',
  template: `
    <div>
      <span id="red" appHighlight>red text</span>
      <span id="green" [appHighlight]="'green'">green text</span>
      <span id="no">no color</span>
    </div>
  `
})
class ContainerComponent { }

const mouseEvents = {
  get enter() {
    const mouseenter = document.createEvent('MouseEvent');
    mouseenter.initEvent('mouseenter', true, true);
    return mouseenter;
  },
  get leave() {
    const mouseleave = document.createEvent('MouseEvent');
    mouseleave.initEvent('mouseleave', true, true);
    return mouseleave;
  },
};

describe('HighlightDirective', () => {
  let fixture: ComponentFixture<ContainerComponent>;
  let container: ContainerComponent;
  let element: HTMLElement;

  beforeEach(() => {
    TestBed.configureTestingModule({

```

```

    declarations: [ContainerComponent, HighlightDirective],
    providers: [
      { provide: ComponentFixtureAutoDetect, useValue: true },
    ],
  });

  fixture = TestBed.createComponent(ContainerComponent);
  // fixture.detectChanges(); // without the provider
  container = fixture.componentInstance;
  element = fixture.nativeElement;
});

it('should set background-color to empty when mouse leaves with directive without
arguments', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#red');

  targetElement.dispatchEvent(mouseEvents.leave);
  expect(targetElement.style.backgroundColor).toEqual('');
});

it('should set background-color to empty when mouse leaves with directive with arguments',
() => {
  const targetElement = <HTMLSpanElement>element.querySelector('#green');

  targetElement.dispatchEvent(mouseEvents.leave);
  expect(targetElement.style.backgroundColor).toEqual('');
});

it('should set background-color red with no args passed', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#red');

  targetElement.dispatchEvent(mouseEvents.enter);
  expect(targetElement.style.backgroundColor).toEqual('red');
});

it('should set background-color green when passing green parameter', () => {
  const targetElement = <HTMLSpanElement>element.querySelector('#green');

  targetElement.dispatchEvent(mouseEvents.enter);
  expect(targetElement.style.backgroundColor).toEqual('green');
});
});

```

Прочитайте Директивы онлайн: <https://riptutorial.com/ru/angular2/topic/2202/директивы>

глава 18: Директивы атрибутов влияют на значение свойств узла узла с помощью декоратора @HostBinding.

Examples

@HostBinding

Декодер @HostBinding позволяет нам программно установить значение свойства в элементе хоста директивы. Он работает аналогично привязке свойств, определенной в шаблоне, за исключением того, что он специально нацелен на элемент хоста. Связывание проверяется на каждый цикл обнаружения изменений, поэтому при желании он может динамически меняться. Например, предположим, что мы хотим создать директиву для кнопок, которые динамически добавляют класс, когда мы нажимаем на него. Это может выглядеть примерно так:

```
import { Directive, HostBinding, HostListener } from '@angular/core';

@Directive({
  selector: '[appButtonPress]'
})
export class ButtonPressDirective {
  @HostBinding('attr.role') role = 'button';
  @HostBinding('class.pressed') isPressed: boolean;

  @HostListener('mousedown') hasPressed() {
    this.isPressed = true;
  }
  @HostListener('mouseup') hasReleased() {
    this.isPressed = false;
  }
}
```

Обратите внимание, что для обоих случаев использования @HostBinding мы передаем строковое значение, для которого мы хотим повлиять. Если мы не предоставим строку декоратору, вместо этого будет использоваться имя члена класса. В первом @HostBinding мы статически устанавливаем атрибут роли на кнопку. Во втором примере нажатый класс будет применен, когда isPressed is true

Прочитайте [Директивы атрибутов влияют на значение свойств узла узла с помощью декоратора @HostBinding](https://riptutorial.com/ru/angular2/topic/9455/директивы-атрибутов-влияют-на-значение-свойств-узла-узла-с-помощью-декоратора-@HostBinding). онлайн: <https://riptutorial.com/ru/angular2/topic/9455/директивы-атрибутов-влияют-на-значение-свойств-узла-узла-с-помощью-декоратора--hostbinding->

глава 19: Директивы и компоненты: @Input @Output

Синтаксис

1. Один способ привязки родительского компонента к вложенному компоненту: [propertyName]
2. Связывание одного из вложенных компонентов с родительским компонентом: (propertyName)
3. Двусторонняя привязка (ака банановая коробка): [(propertyName)]

Examples

Пример ввода

@input полезно связывать данные между компонентами

Сначала импортируйте его в свой компонент

```
import { Input } from '@angular/core';
```

Затем добавьте ввод как свойство класса компонента

```
@Input() car: any;
```

Предположим, что селектор вашего компонента является «автомобильным компонентом», когда вы вызываете компонент, добавляете атрибут «автомобиль»,

```
<car-component [car]="car"></car-component>
```

Теперь ваш автомобиль доступен как атрибут вашего объекта (this.car)

Полный пример:

1. car.entity.ts

```
export class CarEntity {
  constructor(public brand : string, public color : string) {
  }
}
```

2. car.component.ts

```

import { Component, Input } from '@angular/core';
import { CarEntity } from "../car.entity";

@Component({
  selector: 'car-component',
  template: require('./templates/car.html'),
})

export class CarComponent {
  @Input() car: CarEntity;

  constructor() {
    console.log('gros');
  }
}

```

3. garage.component.ts

```

import { Component } from '@angular/core';
import { CarEntity } from "../car.entity";
import { CarComponent } from "../car.component";

@Component({
  selector: 'garage',
  template: require('./templates/garage.html'),
  directives: [CarComponent]
})

export class GarageComponent {
  public cars : Array<CarEntity>;

  constructor() {
    var carOne : CarEntity = new CarEntity('renault', 'blue');
    var carTwo : CarEntity = new CarEntity('fiat', 'green');
    var carThree : CarEntity = new CarEntity('citroen', 'yellow');
    this.cars = [carOne, carTwo, carThree];
  }
}

```

4. garage.html

```

<div *ngFor="let car of cars">
  <car-component [car]="car"></car-component>
</div>

```

5. car.html

```

<div>
  <span>{{ car.brand }}</span> |
  <span>{{ car.color }}</span>
</div>

```

Angular2 @Input и @Output в вложенном компоненте

Директива Button, которая принимает @Input() чтобы указать ограничение на клик, пока

кнопка не будет отключена. Родительский компонент может прослушивать событие, которое будет выдаваться, когда ограничение на клик будет достигнуто с помощью @Output :

```
import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'limited-button',
  template: `<button (click)="onClick()"
    [disabled]="disabled">
    <ng-content></ng-content>
  </button>`,
  directives: []
})

export class LimitedButton {
  @Input() clickLimit: number;
  @Output() limitReached: EventEmitter<number> = new EventEmitter();

  disabled: boolean = false;

  private clickCount: number = 0;

  onClick() {
    this.clickCount++;
    if (this.clickCount === this.clickLimit) {
      this.disabled = true;
      this.limitReached.emit(this.clickCount);
    }
  }
}
```

Родительский компонент, который использует директиву Button и предупреждает о том, когда достигнут предел клика:

```
import { Component } from '@angular/core';
import { LimitedButton } from './limited-button.component';

@Component({
  selector: 'my-parent-component',
  template: `<limited-button [clickLimit]="2"
    (limitReached)="onLimitReached($event)">
    You can only click me twice
  </limited-button>`,
  directives: [LimitedButton]
})

export class MyParentComponent {
  onLimitReached(clickCount: number) {
    alert('Button disabled after ' + clickCount + ' clicks.');
```

Angular2 @Input с асинхронными данными

Иногда вам нужно получать данные асинхронно, прежде чем передавать их дочернему

компоненту. Если дочерний компонент пытается использовать данные до их получения, он выдаст ошибку. Вы можете использовать `ngOnChanges` для обнаружения изменений в компоненте `@Input` `s` и дождаться, пока они не будут определены, прежде чем действовать на них.

Родительский компонент с асинхронным вызовом конечной точки

```
import { Component, OnChanges, OnInit } from '@angular/core';
import { Http, Response } from '@angular/http';
import { ChildComponent } from './child.component';

@Component ({
  selector : 'parent-component',
  template : `
    <child-component [data]="asyncData"></child-component>
  `
})
export class ParentComponent {

  asyncData : any;

  constructor(
    private _http : Http
  ){}

  ngOnInit () {
    this._http.get('some.url')
      .map(this.extractData)
      .subscribe(this.handleData)
      .catch(this.handleError);
  }

  extractData (res:Response) {
    let body = res.json();
    return body.data || { };
  }

  handleData (data:any) {
    this.asyncData = data;
  }

  handleError (error:any) {
    console.error(error);
  }
}
```

Детский компонент, который имеет асинхронные данные как входные данные

Этот дочерний компонент принимает данные async как входные данные. Поэтому он должен дожидаться, пока данные будут существовать до его использования. Мы используем ngOnChanges, который срабатывает всякий раз, когда изменяется вход компонента, проверьте, существуют ли данные и используют ли они это, если это произойдет. Обратите внимание, что шаблон для дочернего объекта не будет отображаться, если свойство, которое полагается на передаваемые данные, неверно.

```
import { Component, OnChanges, Input } from '@angular/core';

@Component ({
  selector : 'child-component',
  template : `
    <p *ngIf="doesDataExist">Hello child</p>
  `
})
export class ChildComponent {

  doesDataExist: boolean = false;

  @Input('data') data : any;

  // Runs whenever component @Inputs change
  ngOnChanges () {
    // Check if the data exists before using it
    if (this.data) {
      this.useData(data);
    }
  }

  // contrived example to assign data to reliesOnData
  useData (data) {
    this.doesDataExist = true;
  }
}
```

Прочитайте Директивы и компоненты: @Input @Output онлайн:

<https://riptutorial.com/ru/angular2/topic/3046/директивы-и-компоненты---input-output>

глава 20: Заголовок страницы

Вступление

Как изменить название страницы

Синтаксис

- setTitle(newTitle: string): void;
- getTitle(): string;

Examples

изменение названия страницы

1. Сначала нам нужно предоставить услугу «Название».
2. Использование setTitle

```
import {Title} from "@angular/platform-browser";
@Component({
  selector: 'app',
  templateUrl: './app.component.html',
  providers : [Title]
})

export class AppComponent implements {
  constructor( private title: Title) {
    this.title.setTitle('page title changed');
  }
}
```

Прочитайте Заголовок страницы онлайн: <https://riptutorial.com/ru/angular2/topic/8954/заголовок-страницы>

глава 21: Использование сторонних библиотек, таких как jQuery в Angular 2

Вступление

При создании приложений с использованием Angular 2.x бывают случаи, когда требуется использовать любые сторонние библиотеки, такие как jQuery, Google Analytics, JavaScript API интеграции с чатом и т. Д.

Examples

Конфигурация с использованием угловых кли

NPM

Если внешняя библиотека, такая как `jQuery`, установлена с использованием NPM

```
npm install --save jquery
```

Добавьте путь к скрипту в ваш `angular-cli.json`

```
"scripts": [  
  "../node_modules/jquery/dist/jquery.js"  
]
```

Папка активов

Вы также можете сохранить файл библиотеки в каталоге `assets/js` и включить его в `angular-cli.json`

```
"scripts": [  
  "assets/js/jquery.js"  
]
```

Заметка

Сохраните `jquery` вашей основной библиотеки и их зависимости, такие как `jquery-cycle-plugin` в каталог ресурсов и добавьте оба из них в `angular-cli.json`, убедитесь, что порядок поддерживается для зависимостей.

Использование jQuery в компонентах Angular 2.x

Чтобы использовать `jquery` в ваших компонентах Angular 2.x, объявите глобальную переменную наверху

При использовании `$` for jQuery

```
declare var $: any;
```

Если используется `jQuery` для jQuery

```
declare var jQuery: any
```

Это позволит использовать `$` или `jQuery` в вашем компоненте Angular 2.x.

Прочитайте [Использование сторонних библиотек, таких как jQuery в Angular 2 онлайн](https://riptutorial.com/ru/angular2/topic/9285/использование-сторонних-библиотек-таких-как-jquery-в-angular-2):
<https://riptutorial.com/ru/angular2/topic/9285/использование-сторонних-библиотек-таких-как-jquery-в-angular-2>

глава 22: Используйте собственные веб-компоненты в Angular 2

замечания

Когда вы используете веб-компонент в своем шаблоне Angular 2, угловое будет пытаться найти компонент с селектором, соответствующим пользовательскому тегу веб-компонента, который, конечно же, не может и будет вызывать ошибку.

Решение состоит в том, чтобы импортировать «схему пользовательских элементов» в модуле, который содержит компонент. Это заставит угловой принять любой пользовательский тег, который не соответствует ни одному элементу селектора компонентов.

Examples

Включить в свой модуль схему пользовательских элементов

```
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
import { CommonModule } from '@angular/common';
import { AboutComponent } from './about.component';

@NgModule({
  imports: [ CommonModule ],
  declarations: [ AboutComponent ],
  exports: [ AboutComponent ],
  schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
})

export class AboutModule { }
```

Используйте свой веб-компонент в шаблоне

```
import { Component } from '@angular/core';

@Component({
  selector: 'myapp-about',
  template: `<my-webcomponent></my-webcomponent>`
})
export class AboutComponent { }
```

Прочитайте [Используйте собственные веб-компоненты в Angular 2 онлайн](https://riptutorial.com/ru/angular2/topic/7414/используйте-собственные-веб-компоненты-в-angular-2):

<https://riptutorial.com/ru/angular2/topic/7414/используйте-собственные-веб-компоненты-в-angular-2>

глава 23: Как использовать ngfor

Вступление

Директива `ngFor` используется Angular2 для создания шаблона один раз для каждого элемента в итерируемом объекте. Эта директива связывает итерабельность с DOM, поэтому, если содержимое итеративного изменяется, содержимое DOM также будет изменено.

Examples

Пример неупорядоченного списка

```
<ul>
  <li *ngFor="let item of items">{{item.name}}</li>
</ul>
```

Более подробный пример шаблона

```
<div *ngFor="let item of items">
  <p>{{item.name}}</p>
  <p>{{item.price}}</p>
  <p>{{item.description}}</p>
</div>
```

Пример отслеживания текущего взаимодействия

```
<div *ngFor="let item of items; let i = index">
  <p>Item number: {{i}}</p>
</div>
```

В этом случае я возьму значение индекса, которое является текущей итерацией цикла.

Угловые2 вытесненные экспортированные значения

Angular2 предоставляет несколько экспортированных значений, которые могут быть добавлены к локальным переменным. Это:

- индекс
- первый
- прошлой
- четное
- странный

Кроме `index`, другие берут `Boolean` значение. В качестве предыдущего примера, использующего индекс, он может использоваться любым из этих экспортированных значений:

```
<div *ngFor="let item of items; let firstItem = first; let lastItem = last">
  <p *ngIf="firstItem">I am the first item and I am gonna be showed</p>
  <p *ngIf="firstItem">I am not the first item and I will not show up :(</p>
  <p *ngIf="lastItem">But I'm gonna be showed as I am the last item :)</p>
</div>
```

* ngFor с трубкой

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({
  name: 'even'
})

export class EvenPipe implements PipeTransform {
  transform(value: string): string {
    if(value && value %2 === 0){
      return value;
    }
  }
}

@Component({
  selector: 'example-component',
  template: '<div>
    <div *ngFor="let number of numbers | even">
      {{number}}
    </div>
  </div>'
})

export class exampleComponent {
  let numbers : List<number> = Array.apply(null, {length: 10}).map(Number.call, Number)
}
```

Прочитайте Как использовать ngfor онлайн: <https://riptutorial.com/ru/angular2/topic/8051/как-использовать-ngfor>

глава 24: Как использовать ngIf

Вступление

* **NgIf** : он удаляет или воссоздает часть дерева DOM в зависимости от оценки выражения. Структурная директива и структурные директивы изменяют компоновку DOM путем добавления, замены и удаления его элементов.

Синтаксис

- `<div * ngIf = "false"> test </ div> <! - вычисляет false ->`
- `<div * ngIf = "undefined"> test </ div> <! - вычисляет значение false ->`
- `<div * ngIf = "null"> test </ div> <! - вычисляет false ->`
- `<div * ngIf = "0"> test </ div> <! - вычисляет false ->`
- `<div * ngIf = "NaN"> test </ div> <! - вычисляет false ->`
- `<div * ngIf = ""> test </ div> <! - вычисляет значение false ->`
- Все остальные значения оцениваются как истинные.

Examples

Отображение загрузочного сообщения

Если наш компонент не готов и ждет данных с сервера, мы можем добавить загрузчик, используя * ngIf. **шаги:**

Сначала объявите логическое значение:

```
loading: boolean = false;
```

Затем в вашем компоненте добавьте крюк жизненного цикла, называемый `ngOnInit`

```
ngOnInit() {  
  this.loading = true;  
}
```

и после того, как вы получите полные данные с сервера, вы загружаете `boolean` в `false`.

```
this.loading=false;
```

В вашем шаблоне html используйте * ngIf с свойством `loading` :

```
<div *ngIf="loading" class="progress">  
  <div class="progress-bar info" style="width: 125%;"></div>
```

```
</div>
```

Показать сообщение оповещения о состоянии

```
<p class="alert alert-success" *ngIf="names.length > 2">Currently there are more than 2 names!</p>
```

Запуск функции в начале или конце цикла * ngFor Использование * ngIf

NgFor предоставляет некоторые значения, которые могут быть сглажены локальным переменным

- **index** - (переменная) позиция текущего элемента в истребителе, начинающемся с 0
- **first** - (boolean) true, если текущий элемент является первым элементом в iterable
- **last** - (boolean) true, если текущий элемент является последним элементом в iterable
- **even** - (boolean) true, если текущий индекс является четным числом
- **нечетное** - (логическое) значение true, если текущий индекс является нечетным числом

```
<div *ngFor="let note of csvdata; let i=index; let lastcall=last">
  <h3>{{i}}</h3> <!-- to show index position
  <h3>{{note}}</h3>
  <span *ngIf="lastcall">{{anyfunction()}} </span><!-- this lastcall boolean value will be
true only if this is last in loop
  // anyfunction() will run at the end of loop same way we can do at start
</div>
```

Используйте * ngIf с * ngFor

Хотя вам не разрешено использовать *ngIf и *ngFor в одном и том же div (это даст ошибку во время выполнения), вы можете *ngIf в *ngFor чтобы получить желаемое поведение.

Пример 1: Общий синтаксис

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="<your condition here>">

    <!-- Execute code here if statement true -->

  </div>
</div>
```

Пример 2. Отображение элементов с четным индексом

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="i % 2 == 0">
    {{ item }}
  </div>
</div>
```

Недостатком является то, что необходимо добавить дополнительный внешний элемент `div` .

Но рассмотрите этот случай использования, когда необходимо, чтобы элемент `div` был итерирован (с использованием `* ngFor`), а также проверяет, нужно ли удалить элемент или нет (используя `* ngIf`), но добавление дополнительного `div` не является предпочтительным. В этом случае вы можете использовать тег `template` для `* ngFor`:

```
<template ngFor let-item [ngForOf]="items">
  <div *ngIf="item.price > 100">
    </div>
</template>
```

Таким образом, добавление дополнительного внешнего `div` не требуется, и, кроме того, элемент `<template>` не будет добавлен в DOM. Единственными элементами, добавленными в DOM из приведенного выше примера, являются итерированные элементы `div` .

Примечание. В Angular v4 `<template>` устарел в пользу `<ng-template>` и будет удален в версии 5. В версиях Angular v2.x сохраняется `<template>` .

Прочитайте Как использовать `ngif` онлайн: <https://riptutorial.com/ru/angular2/topic/8346/как-использовать-ngif>

глава 25: Компиляция вовремя (AOT) с помощью Angular 2

Examples

1. Установите зависимости Angular 2 с компилятором

ПРИМЕЧАНИЕ. Для достижения наилучших результатов убедитесь, что ваш проект был создан с помощью Angular-CLI.

```
npm install angular/{core,common,compiler,platform-browser,platform-browser-dynamic,http,router,forms,compiler-cli,tsc-wrapped,platform-server}
```

Вам не нужно делать этот шаг, если у проекта уже есть угловой 2, и все эти зависимости установлены. Просто убедитесь, что `compiler` там.

2. Добавьте `angularCompilerOptions` в файл `tsconfig.json`

```
...  
"angularCompilerOptions": {  
  "genDir": "./ngfactory"  
}  
...
```

Это папка вывода компилятора.

3. Запустите `ngc`, угловой компилятор

из корня вашего проекта `./node_modules/.bin/ngc -p src` где `src` - это место, где живет весь ваш угловой код 2. Это создаст папку с именем `ngfactory` которой будет `ngfactory` весь ваш скомпилированный код.

`"node_modules/.bin/ngc" -p src` для ОКОН

4. Измените файл `main.ts` на использование браузера `NgFactory` и статической платформы.

```
// this is the static platform browser, the usual counterpart is @angular/platform-browser-dynamic.  
import { platformBrowser } from '@angular/platform-browser';  
  
// this is generated by the angular compiler  
import { AppModuleNgFactory } from './ngfactory/app/app.module.ngfactory';  
  
// note the use of `bootstrapModuleFactory`, as opposed to `bootstrapModule`.
```

```
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

На этом этапе вы сможете запустить свой проект. В этом случае мой проект был создан с использованием Angular-CLI.

```
> ng serve
```

Почему нам нужна компиляция, сборник событий и пример?

В. Почему нам нужна компиляция? Отв. Нам нужна компиляция для достижения более высокого уровня эффективности наших угловых применений.

Взгляните на следующий пример:

```
// ...
compile: function (el, scope) {
  var dirs = this._getElDirectives(el);
  var dir;
  var scopeCreated;
  dirs.forEach(function (d) {
    dir = Provider.get(d.name + Provider.DIRECTIVES_SUFFIX);
    if (dir.scope && !scopeCreated) {
      scope = scope.$new();
      scopeCreated = true;
    }
    dir.link(el, scope, d.value);
  });
  Array.prototype.slice.call(el.children).forEach(function (c) {
    this.compile(c, scope);
  }, this);
},
// ...
```

Используя приведенный выше код для визуализации шаблона,

```
<ul>
  <li *ngFor="let name of names"></li>
</ul>
```

Это намного медленнее по сравнению с:

```
// ...
this._text_9 = this.renderer.createText(this._el_3, '\n', null);
this._text_10 = this.renderer.createText(parentRenderNode, '\n\n', null);
this._el_11 = this.renderer.createElement(parentRenderNode, 'ul', null);
this._text_12 = this.renderer.createText(this._el_11, '\n ', null);
this._anchor_13 = this.renderer.createTemplateAnchor(this._el_11, null);
this._appEl_13 = new import2.AppElement(13, 11, this, this._anchor_13);
this._TemplateRef_13_5 = new import17.TemplateRef_(this._appEl_13,
viewFactory_HomeComponent1);
this._NgFor_13_6 = new import15.NgFor(this._appEl_13.vcRef, this._TemplateRef_13_5,
this.parentInjector.get(import18.IterableDiffers), this.ref);
// ...
```

Поток событий с компиляцией «Вперед»

Напротив, с AoT мы выполняем следующие шаги:

1. Разработка приложения Angular 2 с использованием TypeScript.
2. Компиляция приложения с ngc.
3. Выполняет компиляцию шаблонов с помощью Angular-компилятора для TypeScript.
4. Компиляция кода TypeScript для JavaScript.
5. Пакетирование.
6. Минификация.
7. Развертывание.

Хотя вышеописанный процесс кажется более сложным, пользователь идет только по шагам:

1. Загрузите все активы.
2. Угловые бутстрапы.
3. Приложение получает визуализацию.

Как вы видите, отсутствует третий шаг, что означает более быстрый / лучший UX, и помимо таких инструментов, как `angular2-seed` и `angular-cli`, автоматизируется процесс сборки.

Надеюсь, это поможет вам! Спасибо!

Использование компиляции AoT с Угловой CLI

Интерфейс командной строки « **Угловая CLI** » поддерживает AoT-компиляцию с момента выпуска бета-версии 17.

Чтобы создать приложение с помощью компиляции AoT, просто запустите:

```
ng build --prod --aot
```

Прочитайте [Компиляция вовремя \(AOT\) с помощью Angular 2 онлайн](#):

<https://riptutorial.com/ru/angular2/topic/6634/компиляция-вовремя--aot--с-помощью-angular-2>

глава 26: Компонентные взаимодействия

Синтаксис

- `<element [variableName]="value"></element> //Declaring input to child when using @Input() method.`
- `<element (childOutput)="parentFunction($event)"></element> //Declaring output from child when using @Output() method.`
- `@Output() pageNumberClicked = new EventEmitter(); //Used for sending output data from child component when using @Output() method.`
- `this.pageNumberClicked.emit(pageNum); //Used to trigger data output from child component. when using @Output() method.`
- `@ViewChild(ComponentClass) //Property decorator is required when using ViewChild.`

параметры

название	Значение
количество страниц	Используется для указания количества страниц, которые должны быть созданы для дочернего компонента.
pageNumberClicked	Имя выходной переменной в дочернем компоненте.
pageChanged	Функция в родительском компоненте, выполняющая вывод дочерних компонентов.

Examples

Родительский - взаимодействие с ребенком с использованием свойств @Input & @Output

У нас есть `DataListComponent`, который показывает данные, которые мы извлекаем из службы. `DataListComponent` также имеет `PagerComponent`, поскольку он является дочерним.

`PagerComponent` создает список номеров страниц на основе общего количества страниц, которые он получает от `DataListComponent`. `PagerComponent` также позволяет знать `DataListComponent`, когда пользователь нажимает на любой номер страницы через свойство `Output`.

```
import { Component, NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DataListService } from '../dataList.service';
import { PagerComponent } from '../pager.component';

@Component({
```



```

selector: 'datalist',
template: `
  <table>
  <tr *ngFor="let person of personsData">
    <td>{{person.name}}</td>
    <td>{{person.surname}}</td>
  </tr>
</table>

  <pager [pageCount]="pageCount" (pageNumberClicked)="pageChanged($event)"></pager>
`
})
export class DataListComponent {
  private personsData = null;
  private pageCount: number;

  constructor(private dataListService: DataListService) {
    var response = this.dataListService.getData(1); //Request first page from the service
    this.personsData = response.persons;
    this.pageCount = response.totalCount / 10; //We will show 10 records per page.
  }

  pageChanged(pageNumber: number){
    var response = this.dataListService.getData(pageNumber); //Request data from the
service with new page number
    this.personsData = response.persons;
  }
}

@NgModule({
  imports: [CommonModule],
  exports: [],
  declarations: [DataListComponent, PagerComponent],
  providers: [DataListService],
})
export class DataListModule { }

```

PagerComponent перечисляет все номера страниц. Мы устанавливаем событие `click` для каждого из них, чтобы мы могли сообщить родителям о количестве щелкнутых страниц.

```

import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'pager',
  template: `
  <div id="pager-wrapper">
    <span *ngFor="#page of pageCount" (click)="pageClicked(page)">{{page}}</span>
  </div>
  `
})
export class PagerComponent {
  @Input() pageCount: number;
  @Output() pageNumberClicked = new EventEmitter();
  constructor() { }

  pageClicked(pageNum) {
    this.pageNumberClicked.emit(pageNum); //Send clicked page number as output
  }
}

```

Родитель - взаимодействие с ребенком с помощью ViewChild

ViewChild предлагает одностороннее взаимодействие от родителя к ребенку. При использовании ViewChild нет обратной связи или выхода из дочернего элемента.

У нас есть DataListComponent, который показывает некоторую информацию. DataListComponent имеет PagerComponent, поскольку он является дочерним. Когда пользователь выполняет поиск в DataListComponent, он получает данные из службы и просит PagerComponent обновить макет пейджинга на основе нового количества страниц.

```
import { Component, NgModule, ViewChild } from '@angular/core';
import { CommonModule } from '@angular/common';
import { DataListService } from './dataList.service';
import { PagerComponent } from './pager.component';

@Component({
  selector: 'datalist',
  template: `<input type='text' [(ngModel)]="searchText" />
    <button (click)="getData()">Search</button>
    <table>
    <tr *ngFor="let person of personsData">
      <td>{{person.name}}</td>
      <td>{{person.surname}}</td>
    </tr>
    </table>
    <pager></pager>
  `
})
export class DataListComponent {
  private personsData = null;
  private searchText: string;

  @ViewChild(PagerComponent)
  private pagerComponent: PagerComponent;

  constructor(private dataListService: DataListService) {}

  getData(){
    var response = this.dataListService.getData(this.searchText);
    this.personsData = response.data;
    this.pagerComponent.setPaging(this.personsData / 10); //Show 10 records per page
  }
}

@NgModule({
  imports: [CommonModule],
  exports: [],
  declarations: [DataListComponent, PagerComponent],
  providers: [DataListService],
})
export class DataListModule { }
```

Таким образом, вы можете вызывать функции, определенные в дочерних компонентах.

Детский компонент недоступен до отображения родительского компонента. При попытке

доступа к ребенку , прежде чем родители `AfterViewInit` жизни `Cyle` крючок вызовет исключение.

Двунаправленное взаимодействие родитель-ребенок через службу

Сервис, который используется для связи:

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs/Subject';

@Injectable()
export class ComponentCommunicationService {

  private componentChangeSource = new Subject();
  private newDateCreationSource = new Subject<Date>();

  componentChanged$ = this.componentChangeSource.asObservable();
  dateCreated$ = this.newDateCreationSource.asObservable();

  refresh() {
    this.componentChangeSource.next();
  }

  broadcastDate(date: Date) {
    this.newDateCreationSource.next(date);
  }
}
```

Родительский компонент:

```
import { Component, Inject } from '@angular/core';
import { ComponentCommunicationService } from './component-refresh.service';

@Component({
  selector: 'parent',
  template: `
<button (click)="refreshSubscribed()">Refresh</button>
<h1>Last date from child received: {{lastDate}}</h1>
<child-component></child-component>
`
})
export class ParentComponent implements OnInit {

  lastDate: Date;
  constructor(private communicationService: ComponentCommunicationService) { }

  ngOnInit() {
    this.communicationService.dateCreated$.subscribe(newDate => {
      this.lastDate = newDate;
    });
  }

  refreshSubscribed() {
    this.communicationService.refresh();
  }
}
```

Детский КОМПОНЕНТ:

```
import { Component, OnInit, Inject } from '@angular/core';
import { ComponentCommunicationService } from './component-refresh.service';

@Component({
  selector: 'child-component',
  template: `
    <h1>Last refresh from parent: {{lastRefreshed}}</h1>
    <button (click)="sendNewDate()">Send new date</button>
  `
})
export class ChildComponent implements OnInit {

  lastRefreshed: Date;
  constructor(private communicationService: ComponentCommunicationService) { }

  ngOnInit() {
    this.communicationService.componentChanged$.subscribe(event => {
      this.onRefresh();
    });
  }

  sendNewDate() {
    this.communicationService.broadcastDate(new Date());
  }

  onRefresh() {
    this.lastRefreshed = new Date();
  }
}
```

AppModule:

```
@NgModule({
  declarations: [
    ParentComponent,
    ChildComponent
  ],
  providers: [ComponentCommunicationService],
  bootstrap: [AppComponent] // not included in the example
})
export class AppModule {}
```

Прочитайте Компонентные взаимодействия онлайн:

<https://riptutorial.com/ru/angular2/topic/7400/компонентные-взаимодействия>

глава 27: Компонентные взаимодействия

Вступление

Разделяйте информацию между различными директивами и компонентами.

Examples

Передавать данные от родителя к дочернему с привязкой ввода

HeroChildComponent имеет два входных свойства, обычно украшенных украшениями @Input.

```
import { Component, Input } from '@angular/core';
import { Hero } from './hero';
@Component({
  selector: 'hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>
    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
  `
})
export class HeroChildComponent {
  @Input() hero: Hero;
  @Input('master') masterName: string;
}
```

Изменяется входное свойство перехвата с помощью сеттера

Используйте средство ввода свойств входа для перехвата и действия над значением родителя.

Установитель свойства ввода имени в дочернем NameChildComponent обрезает пробел от имени и заменяет пустое значение текстом по умолчанию.

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'name-child',
  template: '<h3>{{name}}</h3>'
})
export class NameChildComponent {
  private _name = '';
  @Input()
  set name(name: string) {
    this._name = (name && name.trim()) || '<no name set>';
  }
  get name(): string { return this._name; }
}
```

Здесь NameParentComponent демонстрирует варианты имен, включая имя со всеми пробелами:

```
import { Component } from '@angular/core';
@Component({
  selector: 'name-parent',
  template: `
    <h2>Master controls {{names.length}} names</h2>
    <name-child *ngFor="let name of names" [name]="name"></name-child>
  `
})
export class NameParentComponent {
  // Displays 'Mr. IQ', '<no name set>', 'Bombasto'
  names = ['Mr. IQ', ' ', ' Bombasto '];
}
```

Родитель слушает дочернее событие

Детский компонент предоставляет свойство EventEmitter, с помощью которого он генерирует события, когда что-то происходит. Родитель связывается с этим свойством события и реагирует на эти события.

Свойство EventEmitter для ребенка является выходным свойством, обычно украшенным украшением @Output, как показано в этом элементе VoterComponent:

```
import { Component, EventEmitter, Input, Output } from '@angular/core';
@Component({
  selector: 'my-voter',
  template: `
    <h4>{{name}}</h4>
    <button (click)="vote(true)" [disabled]="voted">Agree</button>
    <button (click)="vote(false)" [disabled]="voted">Disagree</button>
  `
})
export class VoterComponent {
  @Input() name: string;
  @Output() onVoted = new EventEmitter<boolean>();
  voted = false;
  vote(agree: boolean) {
    this.onVoted.emit(agree);
    this.voted = true;
  }
}
```

Нажатие кнопки вызывает эмиссию истинного или ложного (булева полезная нагрузка).

Родительский VoteTakerComponent связывает обработчик события (onVoted), который отвечает на полезную нагрузку дочернего события (\$ event) и обновляет счетчик.

```
import { Component } from '@angular/core';
@Component({
  selector: 'vote-taker',
  template: `
    <h2>Should mankind colonize the Universe?</h2>
  `
})
```

```

    <h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>
    <my-voter *ngFor="let voter of voters"
      [name]="voter"
      (onVoted)="onVoted($event)">
    </my-voter>
  `
  `
})
export class VoteTakerComponent {
  agreed = 0;
  disagreed = 0;
  voters = ['Mr. IQ', 'Ms. Universe', 'Bombasto'];
  onVoted(agreed: boolean) {
    agreed ? this.agreed++ : this.disagreed++;
  }
}

```

Родитель взаимодействует с дочерним элементом через локальную переменную

Родительский компонент не может использовать привязку данных для чтения дочерних свойств или для вызова дочерних методов. Мы можем сделать это путем создания ссылочной переменной шаблона для дочернего элемента, а затем сослаться на эту переменную в родительском шаблоне, как показано в следующем примере.

У нас есть дочерний `CountdownTimerComponent`, который неоднократно отсчитывает до нуля и запускает ракету. Он имеет методы запуска и остановки, которые управляют часами, и он отображает сообщение о состоянии обратного отсчета в своем собственном шаблоне.

```

import { Component, OnDestroy, OnInit } from '@angular/core';
@Component({
  selector: 'countdown-timer',
  template: '<p>{{message}}</p>'
})
export class CountdownTimerComponent implements OnInit, OnDestroy {
  intervalId = 0;
  message = '';
  seconds = 11;
  clearTimer() { clearInterval(this.intervalId); }
  ngOnInit() { this.start(); }
  ngOnDestroy() { this.clearTimer(); }
  start() { this.countDown(); }
  stop() {
    this.clearTimer();
    this.message = `Holding at T-${this.seconds} seconds`;
  }
  private countDown() {
    this.clearTimer();
    this.intervalId = window.setInterval(() => {
      this.seconds -= 1;
      if (this.seconds === 0) {
        this.message = 'Blast off!';
      } else {
        if (this.seconds < 0) { this.seconds = 10; } // reset
        this.message = `T-${this.seconds} seconds and counting`;
      }
    }, 1000);
  }
}

```

```
}  
}
```

Давайте посмотрим на `CountdownLocalVarParentComponent`, на котором размещен компонент таймера.

```
import { Component }           from '@angular/core';  
import { CountdownTimerComponent } from './countdown-timer.component';  
@Component({  
  selector: 'countdown-parent-lv',  
  template: `  
    <h3>Countdown to Liftoff (via local variable)</h3>  
    <button (click)="timer.start()">Start</button>  
    <button (click)="timer.stop()">Stop</button>  
    <div class="seconds">{{timer.seconds}}</div>  
    <countdown-timer #timer></countdown-timer>  
  `,  
  styleUrls: ['demo.css']  
})  
export class CountdownLocalVarParentComponent { }
```

Родительский компонент не может привязывать данные к методам запуска и остановки дочернего элемента, а также к свойству секунд.

Мы можем поместить локальную переменную (`#timer`) в тег `<countdown-timer>`, представляющий дочерний компонент. Это дает нам ссылку на сам дочерний компонент и возможность доступа к любым его свойствам или методам из родительского шаблона.

В этом примере мы создаем родительские кнопки для запуска и остановки ребенка и используем интерполяцию для отображения свойства секунд дочернего элемента.

Здесь мы видим, как родители и дети работают вместе.

Родитель вызывает ViewChild

Локальный переменный подход прост и прост. Но он ограничен, потому что проводка родитель-ребенок должна выполняться полностью в родительском шаблоне. Сам родительский компонент не имеет доступа к ребенку.

Мы не можем использовать метод локальной переменной, если экземпляр родительского компонента должен читать или записывать значения дочерних компонентов или должен вызывать дочерние компоненты.

Когда класс родительского компонента требует такого доступа, мы добавляем дочерний компонент в родительский элемент как `ViewChild`.

Мы проиллюстрируем эту технику тем же самым примером таймера обратного отсчета. Мы не изменим его внешний вид или поведение. Ребенок `CountdownTimerComponent` тоже такой же.

Мы переходим от локальной переменной к методу ViewChild исключительно для демонстрации. Вот родительский элемент CountdownViewChildParentComponent:

```
import { AfterViewInit, ViewChild } from '@angular/core';
import { Component } from '@angular/core';
import { CountdownTimerComponent } from './countdown-timer.component';
@Component({
  selector: 'countdown-parent-vc',
  template: `
    <h3>Countdown to Liftoff (via ViewChild)</h3>
    <button (click)="start()">Start</button>
    <button (click)="stop()">Stop</button>
    <div class="seconds">{{ seconds() }}</div>
    <countdown-timer></countdown-timer>
  `,
  styleUrls: ['demo.css']
})
export class CountdownViewChildParentComponent implements AfterViewInit {
  @ViewChild(CountdownTimerComponent)
  private timerComponent: CountdownTimerComponent;
  seconds() { return 0; }
  ngAfterViewInit() {
    // Redefine `seconds()` to get from the `CountdownTimerComponent.seconds` ...
    // but wait a tick first to avoid one-time devMode
    // unidirectional-data-flow-violation error
    setTimeout(() => this.seconds = () => this.timerComponent.seconds, 0);
  }
  start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}
```

Требуется немного больше работы, чтобы получить представление child в классе родительских компонентов.

Мы импортируем ссылки на декоратор ViewChild и крючок жизненного цикла AfterViewInit.

Мы добавляем дочерний объект CountdownTimerComponent в свойство private timerComponent через свойство свойства @ViewChild.

Локальная переменная #timer удалена из метаданных компонента. Вместо этого мы привязываем кнопки к собственным методам запуска и остановки родительского компонента и представляем тикающие секунды в интерполяции вокруг метода секунд родительского компонента.

Эти методы напрямую обращаются к инъекционному компоненту таймера.

Ключ жизненного цикла ngAfterViewInit является важной морщиной. Компонент таймера недоступен до тех пор, пока Angular не отобразит родительский вид. Таким образом, мы отображаем сначала 0 секунд.

Затем «Угловой» вызывает крючок жизненного цикла ngAfterViewInit, и в это время слишком поздно обновлять отображение родительского представления секунд обратного отсчета. Правило однонаправленного потока данных Angular не позволяет нам обновлять

родительское представление в том же цикле. Нам нужно подождать один оборот, прежде чем мы сможем отобразить секунды.

Мы используем `setTimeout`, чтобы подождать один тик, а затем пересмотреть метод секунд, чтобы он учитывал будущие значения из компонента таймера.

Родитель и дети общаются через службу

Родительский компонент и его дети совместно используют службу, интерфейс которой обеспечивает двунаправленную связь внутри семьи.

Объем экземпляра службы - это родительский компонент и его дочерние элементы. Компоненты вне этого поддерева компонентов не имеют доступа к службе или их связи.

Этот `MissionService` соединяет `MissionControlComponent` с несколькими детьми `AstronautComponent`.

```
import { Injectable } from '@angular/core';
import { Subject } from 'rxjs/Subject';
@Injectable()
export class MissionService {
  // Observable string sources
  private missionAnnouncedSource = new Subject<string>();
  private missionConfirmedSource = new Subject<string>();
  // Observable string streams
  missionAnnounced$ = this.missionAnnouncedSource.asObservable();
  missionConfirmed$ = this.missionConfirmedSource.asObservable();
  // Service message commands
  announceMission(mission: string) {
    this.missionAnnouncedSource.next(mission);
  }
  confirmMission(astronaut: string) {
    this.missionConfirmedSource.next(astronaut);
  }
}
```

`MissionControlComponent` предоставляет экземпляр службы, которую он разделяет со своими дочерними элементами (через массив метаданных поставщиков) и внедряет этот экземпляр в себя через свой конструктор:

```
import { Component } from '@angular/core';
import { MissionService } from './mission.service';
@Component({
  selector: 'mission-control',
  template: `
    <h2>Mission Control</h2>
    <button (click)="announce()">Announce mission</button>
    <my-astronaut *ngFor="let astronaut of astronauts"
      [astronaut]="astronaut">
    </my-astronaut>
    <h3>History</h3>
    <ul>
      <li *ngFor="let event of history">{{event}}</li>
    </ul>`
})
```

```

    \,
    providers: [MissionService]
  })
export class MissionControlComponent {
  astronauts = ['Lovell', 'Swigert', 'Haise'];
  history: string[] = [];
  missions = ['Fly to the moon!',
              'Fly to mars!',
              'Fly to Vegas!'];
  nextMission = 0;
  constructor(private missionService: MissionService) {
    missionService.missionConfirmed$.subscribe(
      astronaut => {
        this.history.push(`${astronaut} confirmed the mission`);
      });
  }
  announce() {
    let mission = this.missions[this.nextMission++];
    this.missionService.announceMission(mission);
    this.history.push(`Mission "${mission}" announced`);
    if (this.nextMission >= this.missions.length) { this.nextMission = 0; }
  }
}

```

AstronautComponent также вводит службу в свой конструктор. Каждый AstronautComponent является дочерним элементом MissionControlComponent и поэтому получает экземпляр службы родителя:

```

import { Component, Input, OnDestroy } from '@angular/core';
import { MissionService } from './mission.service';
import { Subscription } from 'rxjs/Subscription';
@Component({
  selector: 'my-astronaut',
  template: `
    <p>
      {{astronaut}}: <strong>{{mission}}</strong>
      <button
        (click)="confirm()"
        [disabled]="!announced || confirmed">
        Confirm
      </button>
    </p>
  `
})
export class AstronautComponent implements OnDestroy {
  @Input() astronaut: string;
  mission = '<no mission announced>';
  confirmed = false;
  announced = false;
  subscription: Subscription;
  constructor(private missionService: MissionService) {
    this.subscription = missionService.missionAnnounced$.subscribe(
      mission => {
        this.mission = mission;
        this.announced = true;
        this.confirmed = false;
      });
  }
  confirm() {

```

```
    this.confirmed = true;
    this.missionService.confirmMission(this.astronaut);
  }
  ngOnDestroy() {
    // prevent memory leak when component destroyed
    this.subscription.unsubscribe();
  }
}
```

Обратите внимание, что мы фиксируем подписку и отписываем подписку при уничтожении `AstronautComponent`. Это шаг защиты от утечки памяти. В этом приложении нет реального риска, так как время жизни `AstronautComponent` совпадает с временем жизни самого приложения. Это не всегда было бы правдой в более сложном приложении.

Мы не добавляем этот сторож к `MissionControlComponent`, потому что, как родительский, он контролирует время жизни `MissionService`. Журнал истории показывает, что сообщения перемещаются в обоих направлениях между родительским `MissionControlComponent` и детьми `AstronautComponent`, чему способствует услуга:

Прочитайте [Компонентные взаимодействия онлайн](https://riptutorial.com/ru/angular2/topic/9454/компонентные-взаимодействия):

<https://riptutorial.com/ru/angular2/topic/9454/компонентные-взаимодействия>

глава 28: Компоненты

Вступление

Угловые компоненты - это элементы, составленные с помощью шаблона, который отобразит ваше приложение.

Examples

Простой компонент

Чтобы создать компонент, добавим `@Component` decorator в класс, передающий некоторые параметры:

- `providers` : ресурсы, которые будут вставляться в конструктор компонента
- `selector` : `selector` запросов, который найдет элемент в HTML и заменит компонентом
- `styles` : встроенные стили. ПРИМЕЧАНИЕ. НЕ используйте этот параметр с требованием, он работает над разработкой, но когда вы создаете приложение в процессе производства, все ваши стили теряются
- `styleUrls` : массив путей к файлам стиля
- `template` : Строка, содержащая ваш HTML-код.
- `templateUrl` : путь к файлу HTML

Существуют и другие параметры, которые вы можете настроить, но перечисленные - это то, что вы будете использовать больше всего.

Простой пример:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-required',
  styleUrls: ['required.component.scss'],
  // template: `This field is required.`,
  templateUrl: 'required.component.html',
})
export class RequiredComponent { }
```

Шаблоны и стили

Шаблоны - это файлы HTML, которые могут содержать логику.

Вы можете указать шаблон двумя способами:

Передача шаблона в качестве пути к файлу

```
@Component ({
  templateUrl: 'hero.component.html',
})
```

Передача шаблона в виде встроенного кода

```
@Component ({
  template: `<div>My template here</div>`,
})
```

Шаблоны могут содержать стили. Стили, объявленные в `@Component`, отличаются от вашего файла стиля приложения, все, что применяется в компоненте, будет ограничено этой областью. Например, скажем, вы добавляете:

```
div { background: red; }
```

Все элементы `div` внутри компонента будут красными, но если у вас есть другие компоненты, другие `div` в вашем HTML они вообще не будут изменены.

Сгенерированный код будет выглядеть так:

```
<style>div[_ngcontent-c1] { background: red; }</style>
```

Вы можете добавлять стили к компоненту двумя способами:

Передача массива путей к файлам

```
@Component ({
  styleUrls: ['hero.component.css'],
})
```

Передача массива встроенных кодов

```
styles: [ `div { background: lime; }` ]
```

Вы не должны использовать `styles` с `require` как это не сработает при создании вашего приложения на производство.

Тестирование компонента

hero.component.html

```
<form (ngSubmit)="submit($event)" [formGroup]="form" novalidate>
  <input type="text" FormControlName="name" />
  <button type="submit">Show hero name</button>
</form>
```

hero.component.ts

```
import { FormControl, FormGroup, Validators } from '@angular/forms';

import { Component } from '@angular/core';

@Component({
  selector: 'app-hero',
  templateUrl: 'hero.component.html',
})
export class HeroComponent {
  public form = new FormGroup({
    name: new FormControl('', Validators.required),
  });

  submit(event) {
    console.log(event);
    console.log(this.form.controls.name.value);
  }
}
```

hero.component.spec.ts

```
import { ComponentFixture, TestBed, async } from '@angular/core/testing';

import { HeroComponent } from './hero.component';
import { ReactiveFormsModule } from '@angular/forms';

describe('HeroComponent', () => {
  let component: HeroComponent;
  let fixture: ComponentFixture<HeroComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [HeroComponent],
      imports: [ReactiveFormsModule],
    }).compileComponents();

    fixture = TestBed.createComponent(HeroComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  }));

  it('should be created', () => {
    expect(component).toBeTruthy();
  });

  it('should log hero name in the console when user submit form', async(() => {
    const heroName = 'Saitama';
    const element = <HTMLFormElement>fixture.debugElement.nativeElement.querySelector('form');

    spyOn(console, 'log').and.callThrough();

    component.form.controls['name'].setValue(heroName);
```

```

    element.querySelector('button').click();

    fixture.whenStable().then(() => {
      fixture.detectChanges();
      expect(console.log).toHaveBeenCalledWith(heroName);
    });
  });

  it('should validate name field as required', () => {
    component.form.controls['name'].setValue('');
    expect(component.form.invalid).toBeTruthy();
  });
});

```

Вложенные компоненты

Компоненты будут отображаться в их соответствующем `selector`, поэтому вы можете использовать это для установки компонентов.

Если у вас есть компонент, который показывает сообщение:

```

import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-required',
  template: `{{name}} is required.`
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}

```

Вы можете использовать его внутри другого компонента, используя `app-required` (селектор этого компонента):

```

import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-sample',
  template: `
    <input type="text" name="heroName" />
    <app-required name="Hero Name"></app-required>
  `
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}

```

Прочитайте Компоненты онлайн: <https://riptutorial.com/ru/angular2/topic/10838/компоненты>

глава 29: Конструкция углового материала

Examples

Md2Select

Компонент :

```
<md2-select [(ngModel)]="item" (change)="change($event)" [disabled]="disabled">
<md2-option *ngFor="let i of items" [value]="i.value" [disabled]="i.disabled">
  {{i.name}}</md2-option>
</md2-select>
```

Выберите разрешить пользователю выбирать опцию из опций.

```
<md2-select></md2-select>
<md2-option></md2-option>
<md2-select-header></md2-select-header>
```

Md2Tooltip

Tooltip - это директива, позволяющая пользователю отображать текст подсказки, когда пользовательская мышь наводится на элемент.

Всплывающая подсказка будет иметь следующую разметку.

```
<span tooltip-direction="left" tooltip="On the Left! ">Left</span>
<button tooltip="some message"
  tooltip-position="below"
  tooltip-delay="1000">Hover Me
</button>
```

Md2Toast

Тост - это сервис, который показывает уведомления в представлении.

Создает и демонстрирует простое тост-уведомление.

```
import {Md2Toast} from 'md2/toast/toast';

@Component({
  selector: "...",
})

export class ... {

  ...
  constructor(private toast: Md2Toast) { }
  toastMe() {
```

```
this.toast.show('Toast message...');

--- or ---

this.toast.show('Toast message...', 1000);
}

...

}
```

Md2Datepicker

Datepicker позволяет пользователю выбирать дату и время.

```
<md2-datepicker [(ngModel)]="date"></md2-datepicker>
```

подробнее см. [здесь](#)

Md2Accordion и Md2Collapse

Md2Collapse : Collapse - это директива, позволяющая пользователю переключать видимость раздела.

Примеры

Коллапс будет иметь следующую разметку.

```
<div [collapse]="isCollapsed">
  Lorem Ipsum Content
</div>
```

Md2Accordion : Аккордеон позволяет пользователю переключать видимость нескольких секций.

Примеры

У аккордеона будет следующая разметка.

```
<md2-accordion [multiple]="multiple">
  <md2-accordion-tab *ngFor="let tab of accordions"
    [header]="tab.title"
    [active]="tab.active"
    [disabled]="tab.disabled">
    {{tab.content}}
  </md2-accordion-tab>
  <md2-accordion-tab>
    <md2-accordion-header>Custom Header</md2-accordion-header>
    test content
  </md2-accordion-tab>
</md2-accordion>
```

Прочитайте [Конструкция углового материала онлайн](https://riptutorial.com/ru/angular2/topic/10005/конструкция-углового-материала):

<https://riptutorial.com/ru/angular2/topic/10005/конструкция-углового-материала>

глава 30: Крючки жизненного цикла

замечания

Доступность событий

`AfterViewInit` и `AfterViewChecked` доступны только в компонентах, а не в директивах.

Порядок событий

- `OnChanges` (несколько раз)
- `OnInit` (один раз)
- `DoCheck` (несколько раз)
- `AfterContentInit` (один раз)
- `AfterContentChecked` (несколько раз)
- `AfterViewInit` (один раз) (только компонент)
- `AfterViewChecked` (несколько раз) (только компонент)
- `OnDestroy` (один раз)

Дальнейшее чтение

- [Угловая документация - крючки жизненного цикла](#)

Examples

OnInit

Вызывается при инициализации свойств компонента или директивы.

(До того, как из директив ребенка)

```
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'so-oninit-component',
  templateUrl: 'oninit-component.html',
  styleUrls: ['oninit-component.'],
})
class OnInitComponent implements OnInit {

  ngOnInit(): void {
    console.log('Component is ready !');
  }
}
```

```
}  
}
```

OnDestroy

Вызывается, когда экземпляр компонента или директива уничтожается.

```
import { Component, OnDestroy } from '@angular/core';  
  
@Component({  
  selector: 'so-ondestroy-component',  
  templateUrl: 'ondestroy-component.html',  
  styleUrls: ['ondestroy-component.'],  
})  
class OnDestroyComponent implements OnDestroy {  
  
  ngOnDestroy(): void {  
    console.log('Component was destroyed !');  
  }  
}
```

OnChanges

Вызывается при изменении одного или нескольких свойств компонента или директивы.

```
import { Component, OnChanges, Input } from '@angular/core';  
  
@Component({  
  selector: 'so-onchanges-component',  
  templateUrl: 'onchanges-component.html',  
  styleUrls: ['onchanges-component.'],  
})  
class OnChangesComponent implements OnChanges {  
  @Input() name: string;  
  message: string;  
  
  ngOnChanges(changes: SimpleChanges): void {  
    console.log(changes);  
  }  
}
```

В событии изменения будет записываться журнал

```
name: {  
  currentValue: 'new name value',  
  previousValue: 'old name value'  
},  
message: {  
  currentValue: 'new message value',  
  previousValue: 'old message value'  
}
```

AfterContentInit

Пожар после завершения инициализации содержимого компонента или директивы.

(Сразу после OnInit)

```
import { Component, AfterContentInit } from '@angular/core';

@Component({
  selector: 'so-aftercontentinit-component',
  templateUrl: 'aftercontentinit-component.html',
  styleUrls: ['aftercontentinit-component.'],
})
class AfterContentInitComponent implements AfterContentInit {

  ngAfterContentInit(): void {
    console.log('Component content have been loaded!');
  }
}
```

AfterContentChecked

Пожар после просмотра полностью инициализирован.

(Доступно только для компонентов)

```
import { Component, AfterContentChecked } from '@angular/core';

@Component({
  selector: 'so-aftercontentchecked-component',
  templateUrl: 'aftercontentchecked-component.html',
  styleUrls: ['aftercontentchecked-component.'],
})
class AfterContentCheckedComponent implements AfterContentChecked {

  ngAfterContentChecked(): void {
    console.log('Component content have been checked!');
  }
}
```

AfterViewInit

Пожары после инициализации как представления компонента, так и любого его дочернего вида. Это полезный крючок жизненного цикла для плагинов вне экосистемы Angular 2. Например, вы можете использовать этот метод для инициализации jQuery date picker на основе разметки, которую отобразил Angular 2.

```
import { Component, AfterViewInit } from '@angular/core';

@Component({
  selector: 'so-afterviewinit-component',
  templateUrl: 'afterviewinit-component.html',
  styleUrls: ['afterviewinit-component.'],
})
class AfterViewInitComponent implements AfterViewInit {
```

```
ngAfterViewInit(): void {
    console.log('This event fire after the content init have been loaded!');
}
}
```

AfterViewChecked

Пожар после проверки вида компонента завершен.

(Доступно только для компонентов)

```
import { Component, AfterViewChecked } from '@angular/core';

@Component({
  selector: 'so-afterviewchecked-component',
  templateUrl: 'afterviewchecked-component.html',
  styleUrls: ['afterviewchecked-component.']
})
class AfterViewCheckedComponent implements AfterViewChecked {

  ngAfterViewChecked(): void {
    console.log('This event fire after the content have been checked!');
  }
}
```

DoCheck

Позволяет прослушивать изменения только по указанным свойствам

```
import { Component, DoCheck, Input } from '@angular/core';

@Component({
  selector: 'so-docheck-component',
  templateUrl: 'docheck-component.html',
  styleUrls: ['docheck-component.']
})
class DoCheckComponent implements DoCheck {
  @Input() elements: string[];
  differ: any;
  ngDoCheck(): void {
    // get value for elements property
    const changes = this.differ.diff(this.elements);

    if (changes) {
      changes.forEachAddedItem(res => console.log('Added', r.item));
      changes.forEachRemovedItem(r => console.log('Removed', r.item));
    }
  }
}
```

Прочитайте Крючки жизненного цикла онлайн: <https://riptutorial.com/ru/angular2/topic/2935/крючки-жизненного-цикла>

глава 31: Ленивая загрузка модуля

Examples

Пример ленивой загрузки

Ленивые модули загрузки помогают нам сократить время запуска. При ленивой загрузке нашему приложению не нужно загружать все сразу, ему нужно только загрузить то, что пользователь ожидает увидеть, когда приложение загружается первым. Модули, которые лениво загружаются, загружаются только тогда, когда пользователь переходит к своим маршрутам.

приложение / app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { EagerComponent } from './eager.component';
import { routing } from './app.routing';
@NgModule({
  imports: [
    BrowserModule,
    routing
  ],
  declarations: [
    AppComponent,
    EagerComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

приложение / app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'my-app',
  template: `<h1>My App</h1>    <nav>
    <a routerLink="eager">Eager</a>
    <a routerLink="lazy">Lazy</a>
  </nav>
  <router-outlet></router-outlet>
`
})
export class AppComponent {}
```

приложение / app.routing.ts

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { EagerComponent } from './eager.component';
```



```

const routes: Routes = [
  { path: '', redirectTo: 'eager', pathMatch: 'full' },
  { path: 'eager', component: EagerComponent },
  { path: 'lazy', loadChildren: './lazy.module' }
];
export const routing: ModuleWithProviders = RouterModule.forRoot(routes);

```

приложение / eager.component.ts

```

import { Component } from '@angular/core';
@Component({
  template: `

Eager Component

`
})
export class EagerComponent {}

```

В LazyModule нет ничего особенного, кроме как у него есть собственная маршрутизация и компонент LazyComponent (но нет необходимости называть ваш модуль или simliar так).

приложение / lazy.module.ts

```

import { NgModule } from '@angular/core';
import { LazyComponent } from './lazy.component';
import { routing } from './lazy.routing';
@NgModule({
  imports: [routing],
  declarations: [LazyComponent]
})
export class LazyModule {}

```

приложение / lazy.routing.ts

```

import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { LazyComponent } from './lazy.component';
const routes: Routes = [
  { path: '', component: LazyComponent }
];
export const routing: ModuleWithProviders = RouterModule.forChild(routes);

```

приложение / lazy.component.ts

```

import { Component } from '@angular/core';
@Component({
  template: `

Lazy Component

`
})
export class LazyComponent {}

```

Прочитайте Ленивая загрузка модуля онлайн: <https://riptutorial.com/ru/angular2/topic/7751/ленивая-загрузка-модуля>

глава 32: маршрутизация

Examples

Основная маршрутизация

Маршрутизатор позволяет осуществлять навигацию с одного вида на другой на основе взаимодействия пользователя с приложением.

Ниже приведены шаги по реализации базовой маршрутизации в Angular 2 -

Основные меры предосторожности : убедитесь, что у вас есть тег

```
<base href='/>
```

как первый ребенок под вашим тегом заголовка в вашем файле index.html. Этот тег сообщает, что папка вашего приложения является корнем приложения. Тогда Angular 2 будет знать, как организовать ваши ссылки.

Первый шаг - проверить, указали ли вы на исправление / последние зависимости маршрутизации в package.json -

```
"dependencies": {  
  .....  
  "@angular/router": "3.0.0-beta.1",  
  .....  
}
```

Второй шаг - определить маршрут согласно определению класса -

```
class Route {  
  path : string  
  pathMatch : 'full'|'prefix'  
  component : Type|string  
  .....  
}
```

В файле маршрутов (`route/routes.ts`) импортируйте все компоненты, которые необходимо настроить для разных маршрутов маршрутизации. Пустой путь означает, что представление загружается по умолчанию. «:» в пути указывает динамический параметр, переданный загруженному компоненту.

Маршруты становятся доступными для приложения через инъекцию зависимости. Метод `ProviderRouter` вызывается с параметром `RouterConfig` в качестве параметра, так что он может быть добавлен к компонентам для вызова определенных задач маршрутизации.

```

import { provideRouter, RouterConfig } from '@angular/router';
import { BarDetailComponent } from '../components/bar-detail.component';
import { DashboardComponent } from '../components/dashboard.component';
import { LoginComponent } from '../components/login.component';
import { SignupComponent } from '../components/signup.component';

export const appRoutes: RouterConfig = [
  { path: '', pathMatch: 'full', redirectTo: 'login' },
  { path: 'dashboard', component: DashboardComponent },
  { path: 'bars/:id', component: BarDetailComponent },
  { path: 'login', component: LoginComponent },
  { path: 'signup', component: SignupComponent }
];

export const APP_ROUTER_PROVIDER = [provideRouter(appRoutes)];

```

Третий шаг - загрузить загрузчика маршрута.

В вашем `main.ts` (это может быть любое имя. В основном, это должен ваш основной файл, определенный в `systemjs.config`)

```

import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './components/app.component';
import { APP_ROUTER_PROVIDER } from './routes/routes';

bootstrap(AppComponent, [ APP_ROUTER_PROVIDER ]).catch(err => console.error(err));

```

Четвертый шаг - загрузить / отобразить компоненты маршрутизатора на основе доступа к пути. директива используется для указания угловой нагрузки для загрузки компонента. Использовать импорт `ROUTER_DIRECTIVES`.

```

import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({
  selector: 'demo-app',
  template: `
    .....
    <div>
      <router-outlet></router-outlet>
    </div>
    .....
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})

```

Пятый шаг - связать другие маршруты. По умолчанию `RouterOutlet` загрузит компонент, для которого пуст путь указан в `RouterConfig`. Директива `RouterLink` используется с тэгом `html anchor` для загрузки компонентов, подключенных к маршрутам. `RouterLink` генерирует атрибут `href`, который используется для создания ссылок. Для `Ex`:

```

import { Component } from '@angular/core';
import { ROUTER_DIRECTIVES } from '@angular/router';

```

```

@Component ({
  selector: 'demo-app',
  template: `
    <a [routerLink]="['/login']">Login</a>
    <a [routerLink]="['/signup']">Signup</a>
    <a [routerLink]="['/dashboard']">Dashboard</a>
    <div>
      <router-outlet></router-outlet>
    </div>
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }

```

Теперь мы хорошо справляемся с маршрутизацией в статический путь. RouterLink поддерживает динамический путь, также передавая дополнительные параметры вместе с этим путем.

```
import {Component} из '@ angular / core'; import {ROUTER_DIRECTIVES} из '@ angular / router';
```

```

@Component ({
  selector: 'demo-app',
  template: `
    <ul>
      <li *ngFor="let bar of bars | async">
        <a [routerLink]="['/bars', bar.id]">
          {{bar.name}}
        </a>
      </li>
    </ul>
    <div>
      <router-outlet></router-outlet>
    </div>
  `,
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }

```

RouterLink берет массив, где первый элемент - это путь для маршрутизации, а последующие элементы - для параметров динамической маршрутизации.

Детские маршруты

Иногда имеет смысл вставлять точки зрения или маршруты друг в друга. Например, на панели управления вам нужно несколько подзадач, похожих на вкладки, но реализованные через систему маршрутизации, чтобы показать проекты, контакты, сообщения и т. Д. Пользователей. Чтобы поддерживать такие сценарии, маршрутизатор позволяет нам определять дочерние маршруты.

Сначала мы корректируем RouterConfig сверху и добавляем дочерние маршруты:

```

import { ProjectsComponent } from '../components/projects.component';
import { MessagesComponent } from '../components/messages.component';

export const appRoutes: RouterConfig = [
  { path: '', pathMatch: 'full', redirectTo: 'login' },
  { path: 'dashboard', component: DashboardComponent,
    children: [
      { path: '', redirectTo: 'projects', pathMatch: 'full' },
      { path: 'projects', component: 'ProjectsComponent' },
      { path: 'messages', component: 'MessagesComponent' }
    ] },
  { path: 'bars/:id', component: BarDetailComponent },
  { path: 'login', component: LoginComponent },
  { path: 'signup', component: SignupComponent }
];

```

Теперь, когда мы определили наши дочерние маршруты, мы должны убедиться, что эти дочерние маршруты могут отображаться в нашем `DashboardComponent`, так как именно там мы добавили дочерние элементы. Ранее мы узнали, что компоненты отображаются в `<router-outlet></router-outlet>`. Аналогично, мы объявляем еще один `RouterOutlet` в `DashboardComponent`:

```

import { Component } from '@angular/core';

@Component({
  selector: 'dashboard',
  template: `
    <a [routerLink]="['projects']">Projects</a>
    <a [routerLink]="['messages']">Messages</a>
    <div>
      <router-outlet></router-outlet>
    </div>
  `
})
export class DashboardComponent { }

```

Как вы можете видеть, мы добавили еще один `RouterOutlet` в котором будут отображаться дочерние маршруты. Обычно будет показан маршрут с пустым путём, однако мы перенаправляем маршрут `projects`, потому что мы хотим, чтобы это было показано сразу же после загрузки маршрута `dashboard`. При этом нам нужен пустой маршрут, иначе вы получите сообщение об ошибке:

```
Cannot match any routes: 'dashboard'
```

Таким образом, добавив *пустой* маршрут, то есть маршрут с пустым путем, мы определили точку входа для маршрутизатора.

ResolveData

В этом примере вы покажете, как вы можете разрешать данные, полученные из службы, перед представлением вашего приложения.

Использует угловой / маршрутизатор 3.0.0-beta.2 на момент написания

users.service.ts

```
...
import { Http, Response } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import { User } from './user.ts';

@Injectable()
export class UsersService {

  constructor(public http:Http) {}

  /**
   * Returns all users
   * @returns {Observable<User[]>}
   */
  index():Observable<User[]> {

    return this.http.get('http://mywebsite.com/api/v1/users')
      .map((res:Response) => res.json());
  }

  /**
   * Returns a user by ID
   * @param id
   * @returns {Observable<User>}
   */
  get(id:number|string):Observable<User> {

    return this.http.get('http://mywebsite.com/api/v1/users/' + id)
      .map((res:Response) => res.json());
  }
}
```

users.resolver.ts

```
...
import { UsersService } from './users.service.ts';
import { Observable } from 'rxjs/Rx';
import {
  Resolve,
  ActivatedRouteSnapshot,
  RouterStateSnapshot
} from "@angular/router";

@Injectable()
export class UsersResolver implements Resolve<User[] | User> {

  // Inject UsersService into the resolver
  constructor(private service:UsersService) {}

  resolve(route:ActivatedRouteSnapshot, state:RouterStateSnapshot):Observable<User[] | User>
```

```

{
    // If userId param exists in current URL, return a single user, else return all users
    // Uses brackets notation to access `id` to suppress editor warning, may use dot
notation if you create an interface extending ActivatedRoute with an optional id? attribute
    if (route.params['id']) return this.service.get(route.params['id']);
    return this.service.index();
}
}

```

users.component.ts

Это компонент страницы со списком всех пользователей. Он будет работать аналогично для компонента детали детали пользователя, замените `data.users` на `data.user` или любой другой ключ, определенный в *app.routes.ts* (см. Ниже)

```

...
import { ActivatedRoute } from "@angular/router";

@Component(...)
export class UsersComponent {

    users:User[];

    constructor(route: ActivatedRoute) {
        route.data.subscribe(data => {
            // data['Match key defined in RouterConfig, see below']
            this.users = data.users;
        });
    }

    /**
     * It is not required to unsubscribe from the resolver as Angular's HTTP
     * automatically completes the subscription when data is received from the server
     */
}

```

app.routes.ts

```

...
import { UsersResolver } from '../resolvers/users.resolver';

export const routes:RouterConfig = <RouterConfig>[
    ...
    {
        path: 'user/:id',
        component: UserComponent,
        resolve: {
            // hence data.user in UserComponent
            user: UsersResolver
        }
    },
    {

```

```

    path: 'users',
    component: UsersComponent,
    resolve: {
      // hence data.users in UsersComponent, note the pluralisation
      users: UsersResolver
    }
  },
  ...
]
...

```

app.resolver.ts

Опционально объединить несколько резольверов вместе.

ВАЖНО: сначала необходимо импортировать службы, используемые в *resolver*, или вы получите «Нет провайдера для ошибки *..Resolver*». Помните, что эти службы будут доступны во всем мире, и вам больше не нужно будет объявлять их *providers* любого компонента. Обязательно отмените подписку на любую подписку, чтобы предотвратить утечку памяти

```

...
import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';

export const ROUTE_RESOLVERS = [
  ...,
  UsersService,
  UsersResolver
]

```

main.browser.ts

Резольверы должны вводиться во время начальной загрузки.

```

...
import { bootstrap } from '@angular/platform-browser-dynamic';
import { ROUTE_RESOLVERS } from './app.resolver';

bootstrap(<Type>App, [
  ...,
  ...ROUTE_RESOLVERS
])
.catch(err => console.error(err));

```

Маршрутизация с детьми

В отличие от исходной документации, я нашел, что это способ правильно встраивать маршруты детей в файл *app.routing.ts* или *app.module.ts* (в зависимости от ваших предпочтений). Этот подход работает при использовании WebPack или SystemJS.

В приведенном ниже примере показаны маршруты для дома, дома / счетчика и данных *home / counter / fetch*. Первый и последний маршруты являются примерами

перенаправления. Наконец, в конце примера приведен правильный способ экспорта маршрута, который должен быть импортирован в отдельный файл. Напр. `app.module.ts`

Для дальнейшего объяснения, Угловая требует, чтобы у вас был беспрепятственный маршрут в массиве `children`, который включает в себя родительский компонент, для представления родительского маршрута. Это немного запутанно, но если вы думаете о пустом URL-адресе для детского маршрута, он по существу будет равен тому же URL-адресу, что и родительский маршрут.

```
import { NgModule } from "@angular/core";
import { RouterModule, Routes } from "@angular/router";

import { HomeComponent } from "../components/home/home.component";
import { FetchDataComponent } from "../components/fetchdata/fetchdata.component";
import { CounterComponent } from "../components/counter/counter.component";

const appRoutes: Routes = [
  {
    path: "",
    redirectTo: "home",
    pathMatch: "full"
  },
  {
    path: "home",
    children: [
      {
        path: "",
        component: HomeComponent
      },
      {
        path: "counter",
        children: [
          {
            path: "",
            component: CounterComponent
          },
          {
            path: "fetch-data",
            component: FetchDataComponent
          }
        ]
      }
    ]
  },
  {
    path: "**",
    redirectTo: "home"
  }
];

@NgModule({
  imports: [
    RouterModule.forRoot(appRoutes)
  ],
  exports: [
    RouterModule
  ]
})
```

```
export class AppRoutingModule { }
```

Отличный пример и описание через Siraj

Прочитайте маршрутизация онлайн: <https://riptutorial.com/ru/angular2/topic/2334/>
маршрутизация

глава 33: Маршрутизация (3.0.0+)

замечания

Есть несколько трюков, которые мы можем сделать с маршрутизатором (например, ограничение доступа), но они могут быть рассмотрены в отдельном учебнике.

Если вам нужен новый маршрут, просто измените `app.routes.ts` и выполните следующие действия:

1. Импортировать компонент
2. Добавьте в массив `routes` . Не забудьте включить новый `path` и `component` .

Examples

Бутстрапирование

Теперь, когда маршруты определены, нам нужно сообщить нашему приложению о маршрутах. Для этого загрузите поставщика, который мы экспортировали в предыдущем примере.

Найдите конфигурацию бутстрапа (должно быть в `main.ts` , но **ваш пробег может отличаться**).

```
//main.ts

import {bootstrap} from '@angular/platform-browser-dynamic';

//Import the App component (root component)
import { App } from './app/app';

//Also import the app routes
import { APP_ROUTES_PROVIDER } from './app/app.routes';

bootstrap(App, [
  APP_ROUTES_PROVIDER,
])
.catch(err => console.error(err));
```

Настройка маршрутизатора-выхода

Теперь, когда маршрутизатор настроен, и наше приложение знает, как обрабатывать маршруты, нам нужно показать фактические компоненты, которые мы настроили.

Для этого настройте свой HTML-шаблон / файл для своего **верхнего уровня (приложение)** следующим образом:

```
//app.ts

import {Component} from '@angular/core';
import {Router, ROUTER_DIRECTIVES} from '@angular/router';

@Component({
  selector: 'app',
  templateUrl: 'app.html',
  styleUrls: ['app.css'],
  directives: [
    ROUTER_DIRECTIVES,
  ]
})
export class App {
  constructor() {
  }
}

<!-- app.html -->

<!-- All of your 'views' will go here -->
<router-outlet></router-outlet>
```

Элемент `<router-outlet></router-outlet>` переключит контент, заданный по маршруту. Другим хорошим аспектом этого элемента является то, что он *не* должен быть единственным элементом в вашем HTML.

Например: Допустим, вам нужна панель инструментов `aa` на каждой странице, которая остается постоянной между маршрутами, подобно тому, как выглядит Stack Overflow. Вы можете вставить элементы `<router-outlet>` под элементами, чтобы изменились только определенные части страницы.

Изменение маршрутов (с использованием шаблонов и директив)

Теперь, когда маршруты настроены, нам нужно каким-то образом изменить маршруты.

В этом примере будет показано, как изменить маршруты с помощью шаблона, но можно изменить маршруты в TypeScript.

Вот один пример (без привязки):

```
<a routerLink="/home">Home</a>
```

Если пользователь нажимает на эту ссылку, он будет маршрутизироваться в `/home`. Маршрутизатор знает, как обращаться `/home`, поэтому он отобразит `Home Component`.

Вот пример с привязкой данных:

```
<a *ngFor="let link of links" [routerLink]="link">{{link}}</a>
```

Для этого потребуется массив, называемый `links` чтобы существовать, поэтому добавьте

ЕГО В `app.ts` :

```
public links[] = [
  'home',
  'login'
]
```

Это будет проходить через массив и добавить элемент `<a>` с директивой `routerLink =` значение текущего элемента в массиве, создав это:

```
<a routerLink="home">home</a>
<a routerLink="login">login</a>
```

Это особенно полезно, если у вас много ссылок или, возможно, ссылки должны постоянно меняться. Мы позволяем Angular обрабатывать занятую работу по добавлению ссылок, просто загружая информацию, которую она требует.

Прямо сейчас `links[]` являются статическими, но можно передавать данные из другого источника.

Настройка маршрутов

ПРИМЕЧАНИЕ. Этот пример основан на выпуске `3.0.0-beta.2` для `@angular/router`. На момент написания этой статьи это последняя версия маршрутизатора.

Чтобы использовать маршрутизатор, определите маршруты в новом файле TypeScript, таком как

```
//app.routes.ts

import {provideRouter} from '@angular/router';

import {Home} from './routes/home/home';
import {Profile} from './routes/profile/profile';

export const routes = [
  {path: '', redirectTo: 'home'},
  {path: 'home', component: Home},
  {path: 'login', component: Login},
];

export const APP_ROUTES_PROVIDER = provideRouter(routes);
```

В первой строке мы импортируем `provideRouter` чтобы мы могли сообщить нашему приложению, какие маршруты находятся на этапе загрузки.

Например, «`Home` и `Profile` - это всего лишь два компонента. Вам нужно будет импортировать каждый `Component` вам нужен, в качестве маршрута.

Затем экспортируйте массив маршрутов.

`path` : путь к компоненту. **ВАМ НЕ НУЖНО ИСПОЛЬЗОВАТЬ '/'** « Угловое будет делать это автоматически

`component` : компонент для загрузки при доступе к маршруту

`redirectTo` : *Необязательно* . Если вам необходимо автоматически перенаправить пользователя при доступе к определенному маршруту, поставьте это.

Наконец, мы экспортируем настроенный маршрутизатор. `provideRouter` вернет провайдера, который мы можем увеличить, поэтому наше приложение знает, как обращаться с каждым маршрутом.

Контроль доступа к маршруту или от него

Угловой маршрутизатор по умолчанию позволяет осуществлять навигацию по любому маршруту без каких-либо ограничений. Это не всегда желаемое поведение.

В сценарии, когда пользователю может быть разрешено перемещаться по маршруту или от него, для ограничения этого поведения может использоваться **Guard Route Guard** .

Если ваш сценарий соответствует одному из следующих вариантов, подумайте об использовании Guard Route Guard,

- Пользователь должен пройти аутентификацию для перехода к целевому компоненту.
- Пользователь должен иметь право на переход к целевому компоненту.
- Компонент требует асинхронного запроса перед инициализацией.
- Компонент требует ввода пользователя перед навигацией.

Как работают гвардейцы

Route Guard работают, возвращая логическое значение для управления поведением навигации маршрутизатора. Если *true* , маршрутизатор продолжит навигацию к целевому компоненту. Если возвращается *false* , маршрутизатор отклонит навигацию к целевому компоненту.

Маршрутные защитные интерфейсы

Маршрутизатор поддерживает несколько защитных интерфейсов:

- *CanActivate* : происходит между навигацией по маршруту.
- *CanActivateChild* : происходит между навигацией маршрута к дочернему маршруту.
- *CanDeactivate* : происходит при переходе от текущего маршрута.

- *CanLoad* : происходит между навигацией маршрута к функциональному модулю, загружаемому асинхронно.
- *Resolve* : используется для выполнения поиска данных до активации маршрута.

Эти интерфейсы могут быть реализованы в вашей защите для предоставления или удаления доступа к определенным процессам навигации.

Синхронные и асинхронные маршруты

Маршрутные охранники позволяют выполнять синхронные и асинхронные операции для условного управления навигацией.

Синхронный охранник маршрута

Синхронный защитник маршрута возвращает логическое значение, например, вычисляя немедленный результат, чтобы условно управлять навигацией.

```
import { Injectable }    from '@angular/core';
import { CanActivate }  from '@angular/router';

@Injectable()
export class SynchronousGuard implements CanActivate {
  canActivate() {
    console.log('SynchronousGuard#canActivate called');
    return true;
  }
}
```

Асинхронный охранник маршрута

Для более сложного поведения защитник маршрута может асинхронно блокировать навигацию. Асинхронный защитник маршрута может возвращать Наблюдаемый или Обещающий.

Это полезно для таких ситуаций, как ожидание ввода пользователем для ответа на вопрос, ожидание успешного сохранения изменений на сервере или ожидание получения данных с удаленного сервера.

```
import { Injectable }    from '@angular/core';
import { CanActivate, Router, ActivatedRouteSnapshot, RouterStateSnapshot } from
 '@angular/router';
import { Observable }    from 'rxjs/Rx';
import { MockAuthenticationService } from './authentication/authentication.service';
```

```

@Injectable()
export class AsynchronousGuard implements CanActivate {
  constructor(private router: Router, private auth: MockAuthenticationService) {}

  canActivate(route: ActivatedRouteSnapshot,
state: RouterStateSnapshot): Observable<boolean>|boolean {
    this.auth.subscribe((authenticated) => {
      if (authenticated) {
        return true;
      }
      this.router.navigateByUrl('/login');
      return false;
    });
  }
}

```

Добавить защиту для настройки маршрута

Файл *app.routes*

Защищенные маршруты могут `canActivate` к Guard

```

import { provideRouter, Router, RouterConfig, CanActivate } from '@angular/router';

//components
import { LoginComponent } from './login/login.component';
import { DashboardComponent } from './dashboard/dashboard.component';

export const routes: RouterConfig = [
  { path: 'login', component: LoginComponent },
  { path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] }
]

```

Экспортируйте **APP_ROUTER_PROVIDERS** для использования в начальной загрузке приложения

```

export const APP_ROUTER_PROVIDERS = [
  AuthGuard,
  provideRouter(routes)
];

```

Использовать Guard в бутстрапе приложения

Файл *main.ts* (или *boot.ts*)

Рассмотрим приведенные выше примеры:

1. **Создайте охрану** (где создается Guard) и
2. **Добавьте защиту для настройки маршрута** (где Guard настроен для маршрута, затем экспортируется **APP_ROUTER_PROVIDERS**), мы можем подключить бутстрап к Guard следующим образом


```

import { bootstrap } from '@angular/platform-browser-dynamic';
import { provide } from '@angular/core';

import { APP_ROUTER_PROVIDERS } from './app.routes';
import { AppComponent } from './app.component';

bootstrap(AppComponent, [
  APP_ROUTER_PROVIDERS
])
.then(success => console.log(`Bootstrap success`))
.catch(error => console.log(error));

```

Использование ресольверов и гвардейцев

Мы используем `currentUser` в конфигурации маршрута, чтобы поймать текущего пользователя при загрузке первой страницы и распознаватель, чтобы сохранить значение `currentUser`, который является нашим аутентифицированным пользователем из бэкэнд.

Упрощенная версия нашей реализации выглядит следующим образом:

Вот наш маршрут верхнего уровня:

```

export const routes = [
  {
    path: 'Dash',
    pathMatch: 'prefix',
    component: DashCmp,
    canActivate: [AuthGuard],
    resolve: {
      currentUser: CurrentUserResolver
    },
    children: [...[
      {
        path: '',
        component: ProfileCmp,
        resolve: {
          currentUser: currentUser
        }
      }
    ]]
  }
];

```

Вот наш `AuthService`

```

import { Injectable } from '@angular/core';
import { Http, Headers, RequestOptions } from '@angular/http';
import { Observable } from 'rxjs/Rx';
import 'rxjs/add/operator/do';

@Injectable()
export class AuthService {
  constructor(http: Http) {
    this.http = http;

    let headers = new Headers({ 'Content-Type': 'application/json' });
    this.options = new RequestOptions({ headers: headers });
  }
}

```

```
fetchCurrentUser() {
  return this.http.get('/api/users/me')
    .map(res => res.json())
    .do(val => this.currentUser = val);
}
}
```

Вот наш AuthGuard :

```
import { Injectable } from '@angular/core';
import { CanActivate } from "@angular/router";
import { Observable } from 'rxjs/Rx';

import { AuthService } from '../services/AuthService';

@Injectable()
export class AuthGuard implements CanActivate {
  constructor(auth: AuthService) {
    this.auth = auth;
  }
  canActivate(route, state) {
    return Observable
      .merge(this.auth.fetchCurrentUser(), Observable.of(true))
      .filter(x => x == true);
  }
}
```

Вот наш CurrentUserResolver :

```
import { Injectable } from '@angular/core';
import { Resolve } from "@angular/router";
import { Observable } from 'rxjs/Rx';

import { AuthService } from '../services/AuthService';

@Injectable()
export class CurrentUserResolver implements Resolve {
  constructor(auth: AuthService) {
    this.auth = auth;
  }
  resolve(route, state) {
    return this.auth.currentUser;
  }
}
```

Прочитайте Маршрутизация (3.0.0+) онлайн: <https://riptutorial.com/ru/angular2/topic/1208/маршрутизация--3-0-0plus->

глава 34: Модернизация грубой силы

Вступление

Если вы хотите обновить версию вашего проекта с угловым CLI, вы можете столкнуться с трудными исправлениями ошибок и ошибок, просто изменив номер версии Angular CLI в вашем проекте. Кроме того, поскольку Угловая CLI скрывает многое из того, что происходит в процессе сборки и сборки, вы не можете делать много, если там что-то не так.

Иногда самый простой способ обновить версию проекта «Угловая CLI» - это просто поднять новый проект с помощью версии Angular CLI, которую вы хотите использовать.

замечания

Поскольку Angular 2 является *настолько* модульным и инкапсулированным, вы можете просто скопировать все компоненты, службы, трубы, директивы, а затем заполнить NgModule, как это было в старом проекте.

Examples

Подземный проект нового Углового CLI

```
ng new NewProject
```

или же

```
ng init NewProject
```

Прочитайте Модернизация грубой силы онлайн: <https://riptutorial.com/ru/angular2/topic/9152/модернизация-грубой-силы>

глава 35: Модули

Вступление

Угловые модули - это контейнеры для разных частей вашего приложения.

У вас могут быть вложенные модули, ваш `app.module` уже `app.module` другие модули, такие как `BrowserModule` и вы можете добавить `RouterModule` и так далее.

Examples

Простой модуль

Модуль - это класс с декоратором `@NgModule`. Чтобы создать модуль, мы добавляем `@NgModule` передавая некоторые параметры:

- `bootstrap`: компонент, который будет корнем вашего приложения. Эта конфигурация присутствует только в корневом модуле
- `declarations`: ресурсы, `declarations` модулем. Когда вы добавляете новый компонент, вам нужно обновлять декларации (`ng generate component` делает это автоматически)
- `exports`: Ресурсы экспорта модулей, которые могут использоваться в других модулях
- `imports`: ресурсы, которые модуль использует из других модулей (принимаются только классы модулей)
- `providers`: ресурсы, которые могут быть введены (`di`) в компонент

Простой пример:

```
import { AppComponent } from './app.component';
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

@NgModule({
  bootstrap: [AppComponent]
  declarations: [AppComponent],
  exports: [],
  imports: [BrowserModule],
  providers: [],
})
export class AppModule { }
```

Вложенные модули

Модули могут быть вложенными, используя `imports` параметра `@NgModule` декоратора.

Мы можем создать `core.module` в нашем приложении, которое будет содержать общие вещи, такие как `ReservePipe` (канал, который обращает строку) и связывает их в этом модуле:

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { ReversePipe } from '../reverse.pipe';

@NgModule({
  imports: [
    CommonModule
  ],
  exports: [ReversePipe], // export things to be imported in another module
  declarations: [ReversePipe],
})
export class CoreModule { }
```

Затем в app.module :

```
import { CoreModule } from 'app/core/core.module';

@NgModule({
  declarations: [...], // ReversePipe is available without declaring here
                        // because CoreModule exports it
  imports: [
    CoreModule,        // import things from CoreModule
    ...
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Прочитайте Модули онлайн: <https://riptutorial.com/ru/angular2/topic/10840/модули>

глава 36: модульное тестирование

Examples

Базовый блок-тест

компонентный файл

```
@Component ({
  selector: 'example-test-compnent',
  template: '<div>
    <div>{{user.name}}</div>
    <div>{{user.fname}}</div>
    <div>{{user.email}}</div>
  </div>'
})

export class ExampleTestComponent implements OnInit{

  let user :User = null;
  ngOnInit(): void {
    this.user.name = 'name';
    this.user.fname= 'fname';
    this.user.email= 'email';
  }

}
```

Тестовый файл

```
describe('Example unit test component', () => {
  let component: ExampleTestComponent ;
  let fixture: ComponentFixture<ExampleTestComponent >;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [ExampleTestComponent]
    }).compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(ExampleTestComponent );
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('ngOnInit should change user object values', () => {
    expect(component.user).toBeNull(); // check that user is null on initialize
    component.ngOnInit(); // run ngOnInit

    expect(component.user.name).toEqual('name');
```

```
expect(component.user.fname).toEqual('fname');  
expect(component.user.email).toEqual('email');  
});  
});
```

Прочитайте модульное тестирование онлайн: [https://riptutorial.com/ru/angular2/topic/8955/
модульное-тестирование](https://riptutorial.com/ru/angular2/topic/8955/модульное-тестирование)

глава 37: Настройка приложения ASP.net Core для работы с Angular 2 и TypeScript

Вступление

СКЭНАРИО: ASP.NET Core background Угловые 2 Угловые 2 компоненты с использованием основных контроллеров Asp.net

Он может реализовать Angular 2 над приложением Asp.Net Core. Это позволяет нам называть MVC-контроллеры из компонентов Angular 2 тоже с результатом MVC View, поддерживающим Angular 2.

Examples

Asp.Net Core + Angular2 + Gulp

Startup.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Logging;
using CoreAngular000.Data;
using CoreAngular000.Models;
using CoreAngular000.Services;
using Microsoft.Extensions.FileProviders;
using System.IO;

namespace CoreAngular000
{
    public class Startup
    {
        public Startup(IHostingEnvironment env)
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(env.ContentRootPath)
                .AddJsonFile("appsettings.json", optional: false, reloadOnChange:
true)
                .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional:
true);

            if (env.IsDevelopment())
            {
```



```

        builder.AddUserSecrets<Startup>();
    }

    builder.AddEnvironmentVariables();
    Configuration = builder.Build();
}

public IConfigurationRoot Configuration { get; }

public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole(Configuration.GetSection("Logging"));
    loggerFactory.AddDebug();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
        app.UseDatabaseErrorPage();
        app.UseBrowserLink();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }

    app.UseDefaultFiles();
    app.UseStaticFiles();
    app.UseStaticFiles(new StaticFileOptions
    {
        FileProvider = new
PhysicalFileProvider(Path.Combine(env.ContentRootPath, "node_modules"),
        RequestPath = "/node_modules"
    });

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller=Home}/{action=Index}/{id?}");
    });
}

```

```
}  
}
```

tsConfig.json

```
{  
  "compilerOptions": {  
    "diagnostics": true,  
    "emitDecoratorMetadata": true,  
    "experimentalDecorators": true,  
    "lib": [ "es2015", "dom" ],  
    "listFiles": true,  
    "module": "commonjs",  
    "moduleResolution": "node",  
    "noImplicitAny": true,  
    "outDir": "wwwroot",  
    "removeComments": false,  
    "rootDir": "wwwroot",  
    "sourceMap": true,  
    "suppressImplicitAnyIndexErrors": true,  
    "target": "es5"  
  },  
  "exclude": [  
    "node_modules",  
    "wwwroot/lib/"  
  ]  
}
```

Package.json

```
{  
  "name": "angular dependencies and web dev package",  
  "version": "1.0.0",  
  "description": "Angular 2 MVC. Samuel Maicas Template",  
  "scripts": {},  
  "dependencies": {  
    "@angular/common": "~2.4.0",  
    "@angular/compiler": "~2.4.0",  
    "@angular/core": "~2.4.0",  
    "@angular/forms": "~2.4.0",  
    "@angular/http": "~2.4.0",  
    "@angular/platform-browser": "~2.4.0",  
    "@angular/platform-browser-dynamic": "~2.4.0",  
    "@angular/router": "~3.4.0",  
    "angular-in-memory-web-api": "~0.2.4",  
    "systemjs": "0.19.40",  
    "core-js": "^2.4.1",  
    "rxjs": "5.0.1",  
    "zone.js": "^0.7.4"  
  },  
  "devDependencies": {  
    "del": "^2.2.2",  
    "gulp": "^3.9.1",  
    "gulp-concat": "^2.6.1",  
    "gulp-cssmin": "^0.1.7",  
    "gulp-htmlmin": "^3.0.0",  
    "gulp-uglify": "^2.1.2",  
    "merge-stream": "^1.0.1",  
    "tslint": "^3.15.1",  
  }  
}
```

```
"typescript": "~2.0.10"
},
"repository": {}
}
```

bundleconfig.json

```
[
  {
    "outputFileName": "wwwroot/css/site.min.css",
    "inputFiles": [
      "wwwroot/css/site.css"
    ]
  },
  {
    "outputFileName": "wwwroot/js/site.min.js",
    "inputFiles": [
      "wwwroot/js/site.js"
    ],
    "minify": {
      "enabled": true,
      "renameLocals": true
    },
    "sourceMap": false
  }
]
```

Преобразовать bundleconfig.json в gulpfile (RightClick bundleconfig.json в explorer, Bundler & Minifier> Преобразовать в Gulp

Views / Home / Index.cshtml

```
@{
    ViewData["Title"] = "Home Page";
}
<div>{{ nombre }}</div>
```

Для папки wwwroot используйте <https://github.com/angular/quickstart> seed. Вам нужно: **index.html main.ts, systemjs-angular-loader.js, systemjs.config.js, tsconfig.json** И папка приложения

Wwwroot / Index.html

```
<html>
<head>
  <title>SMTemplate Angular2 & ASP.NET Core</title>
  <base href="/">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <script src="node_modules/core-js/client/shim.min.js"></script>

  <script src="node_modules/zone.js/dist/zone.js"></script>
  <script src="node_modules/systemjs/dist/system.src.js"></script>
```

```

<script src="systemjs.config.js"></script>
<script>
  System.import('main.js').catch(function(err){ console.error(err); });
</script>
</head>

<body>
  <my-app>Loading AppComponent here ...</my-app>
</body>
</html>

```

Вы можете вызвать его для контроллеров из templateUrl. Wwwroot / приложение / app.component.ts

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  templateUrl: '/home/index',
})
export class AppComponent { nombre = 'Samuel Maicas'; }

```

[Seed] Asp.Net Core + Angular2 + Gulp на Visual Studio 2017

1. Скачать семя
2. Запустить восстановление dotnet
3. Запустить npm install

Всегда. Наслаждаться.

<https://github.com/SamML/CoreAngular000>

MVC <-> Угловой 2

Как: вызвать ANGULAR 2 HTML / JS COMPONENT из ASP.NET Core CONTROLLER:

Мы вызываем HTML вместо return View ()

```
return File("~/html/About.html", "text/html");
```

И загрузите угловой компонент в html. Здесь мы можем решить, хотим ли мы работать с тем же или другим модулем. Зависит от ситуации.

Wwwroot / html / about.html

```

<!DOCTYPE html>
<html>
  <head>
    <title>About Page</title>
    <base href="/">

```

```

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
  <link href="../../css/site.min.css" rel="stylesheet" type="text/css"/>

<script src="../../node_modules/core-js/client/shim.min.js"></script>

<script src="../../node_modules/zone.js/dist/zone.js"></script>
<script src="../../node_modules/systemjs/dist/system.src.js"></script>

<script src="../../systemjs.config.js"></script>
<script>
  System.import('../main.js').catch(function(err){ console.error(err); });
</script>
</head>

<body>
  <aboutpage>Loading AppComponent here ...</aboutpage>
</body>
</html>

```

(*) Уже это семя необходимо загрузить весь список ресурсов

Практическое руководство. CALL ASP.NET Core Controller для показа MVC View с поддержкой Angular2:

```

import { Component } from '@angular/core';

@Component({
  selector: 'aboutpage',
  templateUrl: '/home/about',
})
export class AboutComponent {

}

```

Прочитайте [Настройка приложения ASP.net Core для работы с Angular 2 и TypeScript онлайн: https://riptutorial.com/ru/angular2/topic/9543/настройка-приложения-asp-net-core-для-работы-с-angular-2-и-typescript](https://riptutorial.com/ru/angular2/topic/9543/настройка-приложения-asp-net-core-для-работы-с-angular-2-и-typescript)

глава 38: Обнаружение изменений размера

Examples

Компонент, прослушивающий событие изменения размера окна.

Предположим, что у нас есть компонент, который будет скрываться при определенной ширине окна.

```
import { Component } from '@angular/core';

@Component({
  ...
  template: `
    <div>
      <p [hidden]="!visible" (window:resize)="onResize($event)" >Now you see me...</p>
      <p>now you dont!</p>
    </div>
  `,
  ...
})
export class MyComponent {
  visible: boolean = false;
  breakpoint: number = 768;

  constructor() {
  }

  onResize(event) {
    const w = event.target.innerWidth;
    if (w >= this.breakpoint) {
      this.visible = true;
    } else {
      // whenever the window is less than 768, hide this component.
      this.visible = false;
    }
  }
}
```

Тэг `p` в нашем шаблоне будет скрываться всякий раз, когда `visible` `false`. `visible` будет изменять значение всякий раз, когда `onResize` обработчик события `onResize`. Его вызов происходит каждый раз в `window:resize` вызывает событие.

Прочитайте [Обнаружение изменений размера онлайн](https://riptutorial.com/ru/angular2/topic/5276/обнаружение-изменений-размера):

<https://riptutorial.com/ru/angular2/topic/5276/обнаружение-изменений-размера>

глава 39: Обновить титры

Examples

Обновить типизацию, когда: типизация WARN устарела

Предупреждающее сообщение:

```
typings WARN deprecated 10/25/2016: "registry:dt/jasmine#2.5.0+20161003201800" is deprecated  
(updated, replaced or removed)
```

Обновите ссылку:

```
npm run typings -- install dt~jasmine --save --global
```

Заменить [jasmine] для любой библиотеки, которая бросает предупреждение

Прочитайте Обновить титры онлайн: <https://riptutorial.com/ru/angular2/topic/7814/обновить-титры>

глава 40: Обход Sanitizing для доверенных значений

параметры

Params	подробности
селектор	имя тега вы ссылаетесь на свой компонент в html
шаблон (templateUrl)	строка, которая представляет html, который будет вставлен везде, где находится <code><selector></code> . <code>templateUrl</code> - это путь к html-файлу с таким же поведением
трубы	массив труб, которые используются этим компонентом.

замечания

СУПЕР ВАЖНО!

ОТКЛЮЧЕНИЕ САНИТИЗАЦИИ ОСТАВЛЯЕТ ВАС ПРИ РИСКЕ XSS (межсайтовый скриптинг) И ДРУГИХ ВЕКТОРОВ АТТАСК. ПОЖАЛУЙСТА, УБЕДИТЕСЬ, ЧТО ТЫ ХОТИТЕ, ЧТО ВЫ ПОЛУЧАЕТЕ 100%

Использование Pipes позволяет вам изменять только значения атрибутов:

```
<tag [attribute]="expression or variable reference | pipeName">
```

вы не можете использовать трубы таким образом:

```
<tag attribute="expression or variable reference | pipeName">
```

или таким образом

```
<tag attribute={{expression or variable reference | pipeName}}>
```

Examples

Обход Sanitizing с помощью труб (для повторного использования кода)

Проект в соответствии со структурой из направляющей Angular2 Quickstart [здесь](#) .

```
RootOfProject
|
+-- app
|   |-- app.component.ts
|   |-- main.ts
|   |-- pipeUser.component.ts
|   \-- sanitize.pipe.ts
|
|-- index.html
|-- main.html
|-- pipe.html
```

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from './app.component';

bootstrap(AppComponent);
```

Он находит файл index.html в корне проекта и строит его.

app.component.ts

```
import { Component } from '@angular/core';
import { PipeUserComponent } from './pipeUser.component';

@Component({
  selector: 'main-app',
  templateUrl: 'main.html',
  directives: [PipeUserComponent]
})

export class AppComponent { }
```

Это компонент верхнего уровня, который группирует другие компоненты, которые используются.

pipeUser.component.ts

```
import { Component } from '@angular/core';
import { IgnoreSanitize } from './sanitize.pipe';

@Component({
  selector: 'pipe-example',
  templateUrl: 'pipe.html',
  pipes: [IgnoreSanitize]
})

export class PipeUserComponent{
  constructor () { }
```

```

unsafeValue: string = "unsafe/picUrl?id=";
docNum: string;

getUrl(input: string): any {
  if(input !== undefined) {
    return this.unsafeValue.concat(input);
    // returns : "unsafe/picUrl?id=input"
  } else {
    return "fallback/to/something";
  }
}
}
}

```

Этот компонент обеспечивает представление для работы трубки.

sanitize.pipe.ts

```

import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizationService } from '@angular/platform-browser';

@Pipe({
  name: 'sanitaryPipe'
})
export class IgnoreSanitize implements PipeTransform {

  constructor(private sanitizer: DomSanitizationService){}

  transform(input: string) : any {
    return this.sanitizer.bypassSecurityTrustUrl(input);
  }
}

```

Это логика, которая описывает, что форматы трубы.

index.html

```

<head>
  Stuff goes here...
</head>
<body>
  <main-app>
    main.html will load inside here.
  </main-app>
</body>

```

main.html

```

<othertags>
</othertags>

<pipe-example>
  pipe.html will load inside here.
</pipe-example>

<moretags>

```

```
</moretags>
```

pipe.html

```
<img [src]="getUrl('1234') | sanitaryPipe">  
<embed [src]="getUrl() | sanitaryPipe">
```

Если вы хотите проверить html во время работы приложения, вы увидите, что он выглядит так:

```
<head>  
  Stuff goes here...  
</head>  
  
<body>  
  
  <othertags>  
  </othertags>  
  
  <img [src]="getUrl('1234') | sanitaryPipe">  
  <embed [src]="getUrl() | sanitaryPipe">  
  
  <moretags>  
  </moretags>  
  
</body>
```

Прочитайте [Обход Sanitizing для доверенных значений онлайн](https://riptutorial.com/ru/angular2/topic/5942/обход-sanitizing-для-доверенных-значений):

<https://riptutorial.com/ru/angular2/topic/5942/обход-sanitizing-для-доверенных-значений>

глава 41: Обычно встроенные директивы и службы

Вступление

@ угловые / общие - обычно необходимые директивы и услуги @ угловые / ядро - угловая основа сердечника

Examples

Класс местоположения

Местоположение - это служба, которую приложения могут использовать для взаимодействия с URL-адресом браузера. В зависимости от того, какая LocationStrategy используется, Location будет либо сохраняться на пути URL, либо в сегменте хеша URL.

Местоположение отвечает за нормализацию URL-адреса от базового href приложения.

```
import {Component} from '@angular/core';
import {Location} from '@angular/common';

@Component({
  selector: 'app-component'
})
class AppCmp {

  constructor(_location: Location) {

    //Changes the browsers URL to the normalized version of the given URL,
    //and pushes a new item onto the platform's history.
    _location.go('/foo');

  }

  backClicked() {
    //Navigates back in the platform's history.
    this._location.back();
  }

  forwardClicked() {
    //Navigates forward in the platform's history.
    this._location.back();
  }
}
```

AsyncPipe

Асинхронная трубка подписывается на Observable или Promise и возвращает последнее

значение, которое оно выбрало. Когда генерируется новое значение, асинхронный канал отмечает компонент, который нужно проверить для изменений. Когда компонент уничтожается, асинхронный канал автоматически отписывается, чтобы избежать потенциальных утечек памяти.

```
@Component({
  selector: 'async-observable-pipe',
  template: '<div><code>observable|async</code>: Time: {{ time | async }}</div>'
})
export class AsyncObservablePipeComponent {
  time = new Observable<string>((observer: Subscriber<string>) => {
    setInterval(() => observer.next(new Date().toString()), 1000);
  });
}
```

Отображение текущей версии углового2, используемой в вашем проекте

Чтобы отобразить текущую версию, мы можем использовать **VERSION** из @ углового / основного пакета.

```
import { Component, VERSION } from '@angular/core';

@Component({
  selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>
<h2>Current Version: {{ver}}</h2>
`,
})
export class AppComponent {
  name = 'Angular2';
  ver = VERSION.full;
}
```

Валютная труба

Канал валюты позволяет вам работать с вашими данными в виде обычных номеров, но отображать их со стандартным форматированием валюты (символ валюты, десятичные знаки и т. Д.) В представлении.

```
@Component({
  selector: 'currency-pipe',
  template: `<div>
<p>A: {{myMoney | currency:'USD':false}}</p>
<p>B: {{yourMoney | currency:'USD':true:'4.2-2'}}</p>
</div>`
})
export class CurrencyPipeComponent {
  myMoney: number = 100000.653;
  yourMoney: number = 5.3495;
}
```

Труба принимает три необязательных параметра:

- **currencyCode** : Позволяет указать код валюты ISO 4217.
- **symbolDisplay** : Boolean, указывающий, следует ли использовать символ валюты
- **digitInfo** : Позволяет указать способ отображения десятичных знаков.

Дополнительная документация по каналу валюты:

<https://angular.io/docs/ts/latest/api/common/index/CurrencyPipe-pipe.html>

Прочитайте [Обычно встроенные директивы и службы онлайн](#):

<https://riptutorial.com/ru/angular2/topic/8252/обычно-встроенные-директивы-и-службы>

глава 42: Оптимизация рендеринга с помощью ChangeDetectionStrategy

Examples

По умолчанию vs OnPush

Рассмотрим следующий компонент с одним входным `myInput` и внутренним значением, называемым `someInternalValue`. Оба они используются в шаблоне компонента.

```
import {Component, Input} from '@angular/core';

@Component ({
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input () myInput: any;

  someInternalValue: any;

  // ...
}
```

По умолчанию свойство `changeDetection`: в декораторе компонента будет установлено значение `ChangeDetectionStrategy.Default`; неявный в примере. В этой ситуации любые изменения любого из значений в шаблоне `MyComponent` повторную визуализацию `MyComponent`. Другими словами, если я изменю `myInput` или `someInternalValue` угловой 2 будет оказывать энергию и повторно отображать компонент.

Предположим, однако, что мы хотим только повторно визуализировать, когда входы меняются. Рассмотрим следующий компонент с `changeDetection`: установите значение `ChangeDetectionStrategy.OnPush`

```
import {Component, ChangeDetectionStrategy, Input} from '@angular/core';

@Component ({
  changeDetection: ChangeDetectionStrategy.OnPush
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
```

```
class MyComponent {  
  @Input() myInput: any;  
  
  someInternalValue: any;  
  
  // ...  
}
```

Установив `changeDetection: to ChangeDetectionStrategy.OnPush`, `MyComponent` будет повторно отображаться только при изменении его входов. В этом случае `myInput` должен будет получить новое значение от своего родителя, чтобы вызвать повторную визуализацию.

Прочитайте [Оптимизация рендеринга с помощью ChangeDetectionStrategy онлайн](https://riptutorial.com/ru/angular2/topic/2644/оптимизация-рендеринга-с-помощью-changedetectionstrategy):
<https://riptutorial.com/ru/angular2/topic/2644/оптимизация-рендеринга-с-помощью-changedetectionstrategy>

глава 43: Отладка приложений с расширением Angular2 с использованием кода Visual Studio

Examples

Настройка Launch.json для рабочей области

1. Включить Debug из меню - view> debug
2. он возвращает некоторую ошибку во время запуска debug, показывает всплывающее уведомление и открывает launch.json из этого всплывающего уведомления. Это просто из-за того, что launch.json не установлен для вашей рабочей области.

скопируйте и вставьте код ниже в start.json // new launch.json

ваш старый launch.json

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch Extension",
      "type": "extensionHost",
      "request": "launch",
      "runtimeExecutable": "${execPath}",
      "args": [
        "--extensionDevelopmentPath=${workspaceRoot}"
      ],
      "stopOnEntry": false,
      "sourceMaps": true,
      "outDir": "${workspaceRoot}/out",
      "preLaunchTask": "npm"
    }
  ]
}
```

Теперь обновите ваш launch.json, как показано ниже.

новый launch.json

**** // помните, пожалуйста, укажите ваш путь main.js в него ****

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Launch",
      "type": "node",
      "request": "launch",
      "program": "${workspaceRoot}/app/main.js", // put your main.js path
    }
  ]
}
```

```

    "stopOnEntry": false,
    "args": [],
    "cwd": "${workspaceRoot}",
    "preLaunchTask": null,
    "runtimeExecutable": null,
    "runtimeArgs": [
      "--nolazy"
    ],
    "env": {
      "NODE_ENV": "development"
    },
    "console": "internalConsole",
    "sourceMaps": false,
    "outDir": null
  },
  {
    "name": "Attach",
    "type": "node",
    "request": "attach",
    "port": 5858,
    "address": "localhost",
    "restart": false,
    "sourceMaps": false,
    "outDir": null,
    "localRoot": "${workspaceRoot}",
    "remoteRoot": null
  },
  {
    "name": "Attach to Process",
    "type": "node",
    "request": "attach",
    "processId": "${command.PickProcess}",
    "port": 5858,
    "sourceMaps": false,
    "outDir": null
  }
]
}

```

3. Теперь он отлаживается, показывается всплывающее окно уведомления для пошаговой отладки

Прочитайте [Отладка приложений с расширением Angular2 с использованием кода Visual Studio онлайн: https://riptutorial.com/ru/angular2/topic/7139/отладка-приложений-с-расширением-angular2-с-использованием-кода-visual-studio](https://riptutorial.com/ru/angular2/topic/7139/отладка-приложений-с-расширением-angular2-с-использованием-кода-visual-studio-онлайн)

глава 44: Перехватчик Http

замечания

Что мы делаем с классом `HttpServiceLayer`, это расширение класса `Http` от углового и добавление к нему нашей собственной логики.

Затем мы вводим этот класс в класс начальной загрузки приложения и указываем угловое значение, которое мы импортируем в класс `Http`, чтобы вставить `HttpServiceLayer`.

В любом месте кода мы можем просто импортировать

```
import { Http } from '@angular/http';
```

Но наш класс будет использоваться для каждого вызова.

Examples

Простой класс Расширение углового класса Http

```
import { Http, Request, RequestOptionsArgs, Response, RequestOptions, ConnectionBackend, Headers } from '@angular/http';
import { Router } from '@angular/router';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/empty';
import 'rxjs/add/observable/throw';
import 'rxjs/add/operator/catch';
import { ApplicationConfiguration } from '../application-configuration/application-configuration';

/**
 * This class extends the Http class from angular and adds automatically the server URL(if in development mode) and 2 headers by default:
 * Headers added: 'Content-Type' and 'X-AUTH-TOKEN'.
 * 'Content-Type' can be set in any othe service, and if set, it will NOT be overwritten in this class any more.
 */
export class HttpServiceLayer extends Http {

  constructor(backend: ConnectionBackend, defaultOptions: RequestOptions, private _router: Router, private appConfig: ApplicationConfiguration) {
    super(backend, defaultOptions);
  }

  request(url: string | Request, options?: RequestOptionsArgs): Observable<Response> {
    this.getRequestOptionArgs(options);
    return this.intercept(super.request(this.appConfig.getServerAdress() + url, options));
  }

  /**
   * This method checks if there are any headers added and if not created the headers map and
```

```

ads 'Content-Type' and 'X-AUTH-TOKEN'
* 'Content-Type' is not overwritten if it is already available in the headers map
*/
getRequestOptionArgs(options?: RequestOptionsArgs): RequestOptionsArgs {
  if (options == null) {
    options = new RequestOptions();
  }
  if (options.headers == null) {
    options.headers = new Headers();
  }

  if (!options.headers.get('Content-Type')) {
    options.headers.append('Content-Type', 'application/json');
  }

  if (this.appConfig.getAuthToken() != null) {
    options.headers.append('X-AUTH-TOKEN', this.appConfig.getAuthToken());
  }

  return options;
}

/**
* This method as the name suggests intercepts the request and checks if there are any errors.
* If an error is present it will be checked what error there is and if it is a general one
then it will be handled here, otherwise, will be
* thrown up in the service layers
*/
intercept(observable: Observable<Response>): Observable<Response> {

  // return observable;
  return observable.catch((err, source) => {
    if (err.status == 401) {
      this._router.navigate(['/login']);
      //return observable;
      return Observable.empty();
    } else {
      //return observable;
      return Observable.throw(err);
    }
  });
}
}

```

Использование нашего класса вместо Angular's Http

После расширения класса Http нам нужно указывать угловое значение для использования этого класса вместо класса Http.

Чтобы сделать это, в нашем основном модуле (или в зависимости от потребностей, только определенного модуля), нам нужно написать в разделе поставщиков:

```

export function httpServiceFactory(xhrBackend: XHRBackend, requestOptions: RequestOptions,
router: Router, appConfig: ApplicationConfiguration) {
  return new HttpServiceLayer(xhrBackend, requestOptions, router, appConfig);
}

```

```

import { HttpClientModule, Http, Request, RequestOptionsArgs, Response, XHRBackend, RequestOptions,
ConnectionBackend, Headers } from '@angular/http';
import { Router } from '@angular/router';

@NgModule({
  declarations: [ ... ],
  imports: [ ... ],
  exports: [ ... ],
  providers: [
    ApplicationConfiguration,
    {
      provide: Http,
      useFactory: httpServiceFactory,
      deps: [XHRBackend, RequestOptions, Router, ApplicationConfiguration]
    }
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Примечание. ApplicationConfiguration - это просто служба, которую я использую для хранения некоторых значений в течение всего срока действия приложения.

Простой HttpClient AuthToken Interceptor (Angular 4.3+)

```

import { Injectable } from '@angular/core';
import { HttpEvent, HttpRequest, HttpInterceptor, HttpHandler } from '@angular/common/http';
import { UserService } from '../services/user.service';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class AuthHeaderInterceptor implements HttpInterceptor {

  constructor(private userService: UserService) {
  }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    if (this.userService.isAuthenticated()) {
      req = req.clone({
        setHeaders: {
          Authorization: `Bearer ${this.userService.token}`
        }
      });
    }
    return next.handle(req);
  }
}

```

Предоставление Interceptor (some-module.module.ts)

```
{provide: HTTP_INTERCEPTORS, useClass: AuthHeaderInterceptor, multi: true},
```

Прочитайте Перехватчик Http онлайн: <https://riptutorial.com/ru/angular2/topic/1413/перехватчик-http>

глава 45: пользовательский ngx-bootstrap datepicker + input

Examples

пользовательский ngx-bootstrap datepicker

datepicker.component.html

```
<div (clickOutside)="onClickedOutside($event)" (blur)="onClickedOutside($event)">
  <div class="input-group date" [ngClass]="{'disabled-icon': disabledDatePicker == false
  }">
    <input (change)="changedDate()" type="text" [ngModel]="value" class="form-control"
  id="{{id}}" (focus)="openCloseDatepicker()" disabled="{{disabledInput}}" />
    <span id="openCloseDatepicker" class="input-group-addon"
  (click)="openCloseDatepicker()">
      <span class="glyphicon-calendar glyphicon"></span>
    </span>
  </div>

  <div class="dp-popup" *ngIf="showDatePicker">
    <datepicker [startingDay]="1" [startingDay]="dt" [minDate]="min" [(ngModel)]="dt"
  (selectionDone)="onSelectionDone($event)"></datepicker>
  </div>
</div>
```

datepicker.component.ts

```
import {Component, Input, EventEmitter, Output, OnChanges, SimpleChanges, ElementRef, OnInit}
from "@angular/core";
import {DatePipe} from "@angular/common";
import {NgModel} from "@angular/forms";
import * as moment from 'moment';

@Component({
  selector: 'custom-datepicker',
  templateUrl: 'datepicker.component.html',
  providers: [DatePipe, NgModel],
  host: {
    '(document:mousedown)': 'onClick($event)',
  }
})

export class DatepickerComponent implements OnChanges, OnInit{
  ngOnInit(): void {
    this.dt = null;
  }

  inputElement : ElementRef;
  dt: Date = null;
  showDatePicker: boolean = false;

  @Input() disabledInput : boolean = false;
```

```

@Input() disabledDatePicker: boolean = false;
@Input() value: string = null;
@Input() id: string;
@Input() min: Date = null;
@Input() max: Date = null;

@Output() dateModelChange = new EventEmitter();
constructor(el: ElementRef) {
  this.inputElement = el;
}

changedDate(){
  if(this.value === ''){
    this.dateModelChange.emit(null);
  }else if(this.value.split('/').length === 3){
    this.dateModelChange.emit(DatepickerComponent.convertToDate(this.value));
  }
}

clickOutside(event : Event){
  if(this.inputElement.nativeElement !== event.target) {
    console.log('click outside', event);
  }
}

onClick(event) {
  if (!this.inputElement.nativeElement.contains(event.target)) {
    this.close();
  }
}

ngOnChanges(changes: SimpleChanges): void {
  if (this.value !== null && this.value !== undefined && this.value.length > 0) {
    this.value = null;
    this.dt = null;
  }else {
    if(this.value !== null){
      this.dt = new Date(this.value);
      this.value = moment(this.value).format('MM/DD/YYYY');
    }
  }
}

private static transformDate(date: Date): string {
  return new DatePipe('pt-PT').transform(date, 'MM/dd/yyyy');
}

openCloseDatepicker(): void {
  if (!this.disabledDatePicker) {
    this.showDatepicker = !this.showDatepicker;
  }
}

open(): void {
  this.showDatepicker = true;
}

close(): void {
  this.showDatepicker = false;
}

private apply(): void {

```

```
    this.value = DatepickerComponent.transformDate(this.dt);
    this.dateModelChange.emit(this.dt);
  }

  onSelectionDone(event: Date): void {
    this.dt = event;
    this.apply();
    this.close();
  }

  onClickedOutside(event: Date): void {
    if (this.showDatepicker) {
      this.close();
    }
  }

  static convertToDate(val : string): Date {
    return new Date(val.replace('/', '-'));
  }
}
```

Прочитайте пользовательский ngx-bootstrap datepicker + input онлайн:

<https://riptutorial.com/ru/angular2/topic/10549/пользовательский-ngx-bootstrap-datepicker-plus-input>

глава 46: Пример маршрутов, таких как / route / subroute для статических URL-адресов

Examples

Пример базового маршрута с деревом подпунктов

app.module.ts

```
import {routes} from "./app.routes";

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, mainModule.forRoot(), RouterModule.forRoot(routes)],
  providers: [],
  bootstrap: [AppComponent]
})

export class AppModule { }
```

app.routes.ts

```
import { Routes } from '@angular/router';
import {SubTreeRoutes} from "./subTree/subTreeRoutes.routes";

export const routes: Routes = [
  ...SubTreeRoutes,
  { path: '', redirectTo: 'home', pathMatch: 'full' }
];
```

subTreeRoutes.ts

```
import {Route} from '@angular/router';
import {exampleComponent} from "./example.component";

export const SubTreeRoutes: Route[] = [
  {
    path: 'subTree',
    children: [
      {path: '', component: exampleComponent}
    ]
  }
];
```

Прочитайте Пример маршрутов, таких как / route / subroute для статических URL-адресов онлайн: <https://riptutorial.com/ru/angular2/topic/8910/пример-маршрутов--таких-как---route---subroute-для-статических-url-адресов>

глава 47: Примеры расширенных КОМПОНЕНТОВ

замечания

Помните, что Angular 2 - это единственная ответственность. Независимо от того, насколько небольшой ваш компонент, выделите отдельную логику для каждого компонента. Будь то кнопка, причудливая привязка, заголовок диалога или даже подэлемент `sidenav`.

Examples

Image Picker с предварительным просмотром

В этом примере мы собираемся создать подборщик изображений, который предварительно просматривает ваше изображение перед загрузкой. Средство предварительного просмотра также поддерживает перетаскивание файлов во вход. В этом примере я собираюсь охватить только загрузку отдельных файлов, но вы можете немного поработать, чтобы работать с несколькими файлами.

изображения `preview.html`

Это html-макет нашего предварительного просмотра

```
<!-- Icon as placeholder when no file picked -->
<i class="material-icons">cloud_upload</i>

<!-- file input, accepts images only. Detect when file has been picked/changed with Angular's
native (change) event listener -->
<input type="file" accept="image/*" (change)="updateSource($event)">

<!-- img placeholder when a file has been picked. shows only when 'source' is not empty -->
<img *ngIf="source" [src]="source" src="">
```

имидж-`preview.ts`

Это основной файл для нашего компонента `<image-preview>`

```
import {
  Component,
  Output,
  EventEmitter,
} from '@angular/core';

@Component({
  selector: 'image-preview',
  styleUrls: [ './image-preview.css' ],
```

```

    templateUrl: './image-preview.html'
  })
  export class MtImagePreviewComponent {

    // Emit an event when a file has been picked. Here we return the file itself
    @Output() onChange: EventEmitter<File> = new EventEmitter<File>();

    constructor() {}

    // If the input has changed(file picked) we project the file into the img previewer
    updateSource($event: Event) {
      // We access the file with $event.target['files'][0]
      this.projectImage($event.target['files'][0]);
    }

    // Uses FileReader to read the file from the input
    source:string = '';
    projectImage(file: File) {
      let reader = new FileReader;
      // TODO: Define type of 'e'
      reader.onload = (e: any) => {
        // Simply set e.target.result as our <img> src in the layout
        this.source = e.target.result;
        this.onChange.emit(file);
      };
      // This will process our file and get it's attributes/data
      reader.readAsDataURL(file);
    }
  }
}

```

another.component.html

```

<form (ngSubmit)="submitPhoto()">
  <image-preview (onChange)="getFile($event)"></image-preview>
  <button type="submit">Upload</button>
</form>

```

И это все. Путь проще, чем в AngularJS 1.x. Я на самом деле сделал этот компонент на основе старой версии, сделанной в AngularJS 1.5.5.

Отфильтровать значения таблицы с помощью ввода

Импортируйте `ReactiveFormsModule` , а затем

```

import { Component, OnInit, OnDestroy } from '@angular/core';
import { FormControl } from '@angular/forms';
import { Subscription } from 'rxjs';

@Component({
  selector: 'component',
  template: `
    <input [formControl]="control" />
    <div *ngFor="let item of content">
      {{item.id}} - {{item.name}}
    </div>
  `
})

```

```
export class MyComponent implements OnInit, OnDestroy {

  public control = new FormControl('');

  public content: { id: number; name: string; }[];

  private originalContent = [
    { id: 1, name: 'abc' },
    { id: 2, name: 'abce' },
    { id: 3, name: 'ced' }
  ];

  private subscription: Subscription;

  public ngOnInit() {
    this.subscription = this.control.valueChanges.subscribe(value => {
      this.content = this.originalContent.filter(item => item.name.startsWith(value));
    });
  }

  public ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}
```

Прочитайте [Примеры расширенных компонентов онлайн](https://riptutorial.com/ru/angular2/topic/5597/примеры-расширенных-компонентов):

<https://riptutorial.com/ru/angular2/topic/5597/примеры-расширенных-компонентов>

глава 48: Радиально-кли

Вступление

Здесь вы узнаете, как начать с `angular-cli`, создавая новый компонент / сервис / трубу / модуль с угловым `cli`, добавьте 3-стороннюю загрузку, создайте угловой проект.

Examples

Создать пустое приложение Angular2 с угловым `cli`

Требования:

- NodeJS: [страница загрузки](#)
 - `npm` или `пряжа`
-

Выполните следующие команды с `cmd` из папки нового каталога:

1. `npm install -g @angular/cli` **или** `yarn global add @angular/cli`
 2. `ng new PROJECT_NAME`
 3. `cd PROJECT_NAME`
 4. `ng serve`
-

Откройте ваш браузер на `localhost: 4200`

Создание компонентов, директив, труб и услуг

просто используйте ваш `cmd`: вы можете использовать команду `ng generate` (или просто `ng g`) для генерации угловых компонентов:

- **Компонент:** `ng g component my-new-component`
- **Директива:** `ng g directive my-new-directive`
- **Труба:** `ng g pipe my-new-pipe`
- **Сервис: обслуживание** `ng g service my-new-service`
- **Класс:** `ng g class my-new-classt`
- **Интерфейс: интерфейс** `ng g interface my-new-interface`
- **Enum:** `ng g enum my-new-enum`
- **Модуль:** `ng g module my-module` **модуля** `ng g module my-module`

Добавление сторонних библиотек

В `angular-cli.json` вы можете изменить конфигурацию приложения.

Если вы хотите добавить `ng2-bootstrap`, например:

1. `npm install ng2-bootstrap --save` **или** `yarn add ng2-bootstrap`
2. В `angular-cli.json` просто добавьте путь к бутстрапу в узловых модулях.

```
"scripts": [  
  "../node_modules/jquery/dist/jquery.js",  
  "../node_modules/bootstrap/dist/js/bootstrap.js"  
]
```

строить с угловым кли

В `angular-cli.json` по ключу `outDir` вы можете определить свой каталог сборки;

они эквивалентны

```
ng build --target=production --environment=prod  
ng build --prod --env=prod  
ng build --prod
```

и так

```
ng build --target=development --environment=dev  
ng build --dev --e=dev  
ng build --dev  
ng build
```

При создании вы можете изменить `base tag` () в своем `index.html` с помощью опции `-base-href your-url`.

Устанавливает базовый тег `href` в `/myUrl/` в вашем `index.html`

```
ng build --base-href /myUrl/  
ng build --bh /myUrl/
```

Новый проект с `scss` / `sass` как таблица стилей

Файлы стиля по умолчанию, сгенерированные и скомпилированные с помощью `@angular/cli` - это **CSS** .

Если вы хотите использовать **SCSS** вместо этого, сгенерируйте свой проект с помощью:

```
ng new project_name --style=scss
```

Если вы хотите использовать **sass** , сгенерируйте свой проект с помощью:

```
ng new project_name --style=sass
```

Задайте пряжу как менеджер пакетов по умолчанию для @ angular / cli

Пряжа является альтернативой для npm, менеджера пакетов по умолчанию на @ angular / cli. Если вы хотите использовать пряжу в качестве менеджера пакетов для @ angular / cli, выполните следующие действия:

Требования

- [пряжа](#) (`npm install --global yarn` или см. [страницу установки](#))
- [@ angular / cli](#) (`npm install -g @angular/cli` или `yarn global add @angular/cli`)

Чтобы установить пряжу как менеджер пакетов @ angular / cli:

```
ng set --global packageManager=yarn
```

Чтобы вернуть npm в качестве менеджера пакетов @ angular / cli:

```
ng set --global packageManager=npm
```

Прочитайте Радиально-кли онлайн: <https://riptutorial.com/ru/angular2/topic/8956/радиально-кли>

глава 49: Сервисный рабочий

Вступление

Мы увидим, как настроить службу, работающую на угловом уровне, чтобы позволить нашему веб-приложению иметь автономные возможности.

Служебный работник - это специальный скрипт, который работает в фоновом режиме в браузере и управляет сетевыми запросами для данного источника. Он изначально установлен приложением и остается резидентом на машине пользователя / устройстве. Он активируется браузером, когда загружается страница из его источника и имеет возможность отвечать на HTTP-запросы во время загрузки страницы.

Examples

Добавить Service Worker в наше приложение

Во-первых, если вы консультируетесь с mobile.angular.io, флаг `-mobile` больше не работает.

Поэтому, чтобы начать, мы можем создать нормальный проект с угловым кли.

```
ng new serviceWorking-example
cd serviceWorking-example
```

Теперь важно, чтобы сказать угловатому кли, что мы хотим использовать сервисного работника, нам нужно сделать:

```
ng set apps.0.serviceWorker = true
```

Если по какой-либо причине у вас нет установленного `@angular/service-worker`, вы увидите сообщение:

В вашем проекте настроено `serviceWorker = true`, но `@angular/service-worker` не установлен. Запустите `npm install --save-dev @angular/service-worker` и повторите попытку или запустите `ng set apps.0.serviceWorker=false` в вашем `.angular-cli.json`.

Проверьте `.angular-cli.json`, и теперь вы должны увидеть это: `«serviceWorker»: true`

Когда этот флаг верен, производственные сборки будут настроены вместе с работником службы.

Файл `ngsw-manifest.json` будет сгенерирован (или добавлен в случае, если мы создадим `ngsw-manifest.json` в корне проекта, как правило, это делается для указания маршрутизации, в будущем это, вероятно, будет сделано автоматически) в `dist / root`, и там

будет скопирован сценарий рабочего лица. Короткий скрипт будет добавлен в index.html для регистрации рабочего.

Теперь, если мы создадим приложение в режиме производства `ng build --prod`

И проверьте `dist / folder`.

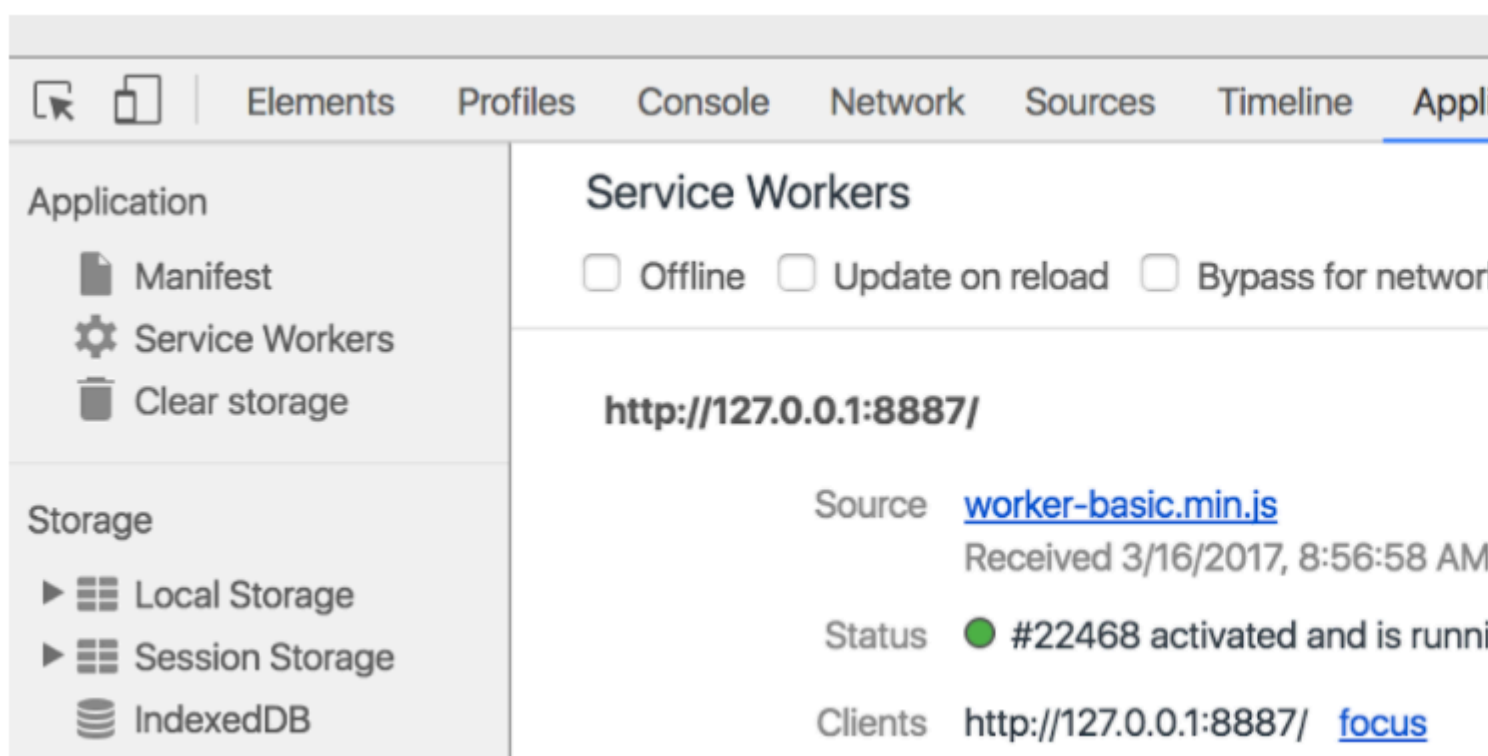
Там вы увидите три новых файла:

- `работник-basic.min.js`
- `SW-register.HASH.bundle.js`
- `ngsw-manifest.json`

Кроме того, `index.html` теперь включает этот скрипт `sw-register`, который регистрирует для нас Угловой Службу Сервиса (ASW).

Обновите страницу в своем браузере (обслуживается веб-сервером для Chrome)

Откройте Инструменты разработчика. Перейдите в приложение -> Служащие службы



Хорошо, теперь рабочий службы работает!

Теперь наше приложение должно загружаться быстрее, и мы должны иметь возможность использовать приложение в автономном режиме.

Теперь, если вы включите автономный режим на консоли Chrome, вы увидите, что наше приложение в <http://localhost:4200/index.html> работает без подключения к Интернету.

Но в <http://localhost:4200/> у нас есть проблема, и она не загружается, это связано с тем,

что кэш статического содержимого поддерживает только файлы, перечисленные в манифесте.

Например, если манифест объявляет URL-адрес /index.html, запросы к /index.html будут отвечать на кэш, но запрос на / или / some / route будет отправлен в сеть.

Именно там входит плагин перенаправления маршрута. Он считывает конфигурацию маршрутизации из манифеста и перенаправляет настроенные маршруты на указанный маршрут индекса.

В настоящее время этот раздел конфигурации должен быть написан вручную (19-7-2017). В конце концов, он будет создан из конфигурации маршрута, присутствующей в источнике приложения.

Итак, если теперь мы создаем или ngsw-manifest.json в корне проекта

```
{
  "routing": {
    "routes": {
      "/": {
        "prefix": false
      }
    },
    "index": "/index.html"
  }
}
```

И мы снова создаем наше приложение, теперь, когда мы идем к <http://localhost:4200/>, мы должны перенаправляться на <http://localhost:4200/index.html>.

Для получения дополнительной информации о маршрутизации прочтите [официальную документацию здесь](#)

Здесь вы можете найти дополнительную документацию об обслуживающих рабочих:

<https://developers.google.com/web/fundamentals/getting-started/primers/service-workers>

https://docs.google.com/document/d/19S5ozevWighny788nI99worpcIMDnwWVmaJDGf_RoDY/edit#

И здесь вы можете увидеть альтернативный способ реализации службы, работающей с использованием библиотеки SW precache:

<https://coryryan.com/blog/fast-offline-angular-apps-with-service-workers>

Прочитайте [Сервисный рабочий онлайн](#): <https://riptutorial.com/ru/angular2/topic/10809/сервисный-рабочий>

глава 50: Служба EventEmitter

Examples

Обзор класса

```
class EventEmitter extends Subject {
  constructor(isAsync?: boolean)
  emit(value?: T)
  subscribe(generatorOrNext?: any, error?: any, complete?: any) : any
}
```

Компонент класса

```
@Component({
  selector: 'zippy',
  template: `
    <div class="zippy">
      <div (click)="toggle()">Toggle</div>
      <div [hidden]="!visible">
        <ng-content></ng-content>
      </div>
    </div>`})
export class Zippy {
  visible: boolean = true;
  @Output() open: EventEmitter<any> = new EventEmitter();
  @Output() close: EventEmitter<any> = new EventEmitter();
  toggle() {
    this.visible = !this.visible;
    if (this.visible) {
      this.open.emit(null);
    } else {
      this.close.emit(null);
    }
  }
}
```

Эмпирирование событий

```
<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>
```

Захват события

Создайте сервис-

```
import {EventEmitter} from 'angular2/core';
export class NavService {
  navchange: EventEmitter<number> = new EventEmitter();
  constructor() {}
  emitNavChangeEvent(number) {
```

```

        this.navchange.emit(number);
    }
    getNavChangeEmitter() {
        return this.navchange;
    }
}

```

Создайте компонент для использования сервис-

```

import {Component} from 'angular2/core';
import {NavService} from '../services/NavService';

@Component({
  selector: 'obs-comp',
  template: `obs component, item: {{item}}`
})
export class ObservingComponent {
  item: number = 0;
  subscription: any;
  constructor(private navService:NavService) {}
  ngOnInit() {
    this.subscription = this.navService.getNavChangeEmitter()
      .subscribe(item => this.selectedNavItem(item));
  }
  selectedNavItem(item: number) {
    this.item = item;
  }
  ngOnDestroy() {
    this.subscription.unsubscribe();
  }
}

@Component({
  selector: 'my-nav',
  template: `
    <div class="nav-item" (click)="selectedNavItem(1)">nav 1 (click me)</div>
    <div class="nav-item" (click)="selectedNavItem(2)">nav 2 (click me)</div>
  `,
})
export class Navigation {
  item = 1;
  constructor(private navService:NavService) {}
  selectedNavItem(item: number) {
    console.log('selected nav item ' + item);
    this.navService.emitNavChangeEvent(item);
  }
}

```

Живой пример

Жесткий пример для этого можно найти [здесь](#) .

Прочитайте [Служба EventEmitter онлайн](https://riptutorial.com/ru/angular2/topic/9159/служба-eventemitter): <https://riptutorial.com/ru/angular2/topic/9159/служба-eventemitter>

глава 51: Создайте пакет с угловым 2+ NPM

Вступление

Иногда нам нужно разделить какой-то компонент между некоторыми приложениями и опубликовать его в npm, это один из лучших способов сделать это.

Есть некоторые трюки, которые нам нужно знать, чтобы иметь возможность использовать обычный компонент в качестве пакета npm без изменения структуры как вставки внешних стилей.

Вы можете увидеть минимальный пример [здесь](#)

Examples

Простейший пакет

Здесь мы используем минимальный рабочий процесс для создания и публикации пакета Angular 2+ npm.

Файлы конфигурации

Нам нужны некоторые файлы конфигурации, чтобы сообщить `git`, `npm`, `gulp` и `typescript` как действовать.

`.gitignore`

Сначала мы создаем файл `.gitignore` чтобы избежать версий нежелательных файлов и папок. Содержание:

```
npm-debug.log
node_modules
jspm_packages
.idea
build
```

`.npmignore`

Во-вторых, мы создаем файл `.npmignore` чтобы избежать публикации нежелательных файлов и папок. Содержание:

```
examples
node_modules
```

```
src
```

gulpfile.js

Нам нужно создать `gulpfile.js` чтобы сообщить Gulp, как скомпилировать наше приложение. Эта часть необходима, потому что перед публикацией нашего пакета нам необходимо свести к минимуму и включить все внешние шаблоны и стили. Содержание:

```
var gulp = require('gulp');
var embedTemplates = require('gulp-angular-embed-templates');
var inlineNg2Styles = require('gulp-inline-ng2-styles');

gulp.task('js:build', function () {
  gulp.src('src/*.ts') // also can use *.js files
    .pipe(embedTemplates({sourceType:'ts'}))
    .pipe(inlineNg2Styles({ base: '/src' }))
    .pipe(gulp.dest('./dist'));
});
```

index.d.ts

При импорте внешнего модуля файл `index.d.ts` используется машинописным `index.d.ts`. Он помогает редактору с автозаполнением и подсказками функций.

```
export * from './lib';
```

index.js

Это точка входа в пакет. Когда вы устанавливаете этот пакет с помощью NPM и импортируете в приложение, вам просто нужно передать имя пакета, и ваше приложение узнает, где найти какой-либо ЭКСПОРТНЫЙ компонент вашего пакета.

```
exports.AngularXMinimalNpmPackageModule = require('./lib').AngularXMinimalNpmPackageModule;
```

Мы использовали папку `lib` потому что когда мы компилируем наш код, вывод помещается в папку `/lib`.

package.json

Этот файл используется для настройки публикации npm и определения необходимых пакетов для работы.

```
{
  "name": "angular-x-minimal-npm-package",
  "version": "0.0.18",
  "description": "An Angular 2+ Data Table that uses HTTP to create, read, update and delete data from an external API such REST.",
  "main": "index.js",
```

```

"scripts": {
  "watch": "tsc -p src -w",
  "build": "gulp js:build && rm -rf lib && tsc -p dist"
},
"repository": {
  "type": "git",
  "url": "git+https://github.com/vinagreti/angular-x-minimal-npm-package.git"
},
"keywords": [
  "Angular",
  "Angular2",
  "Datatable",
  "Rest"
],
"author": "bruno@tzadi.com",
"license": "MIT",
"bugs": {
  "url": "https://github.com/vinagreti/angular-x-minimal-npm-package/issues"
},
"homepage": "https://github.com/vinagreti/angular-x-minimal-npm-package#readme",
"devDependencies": {
  "gulp": "3.9.1",
  "gulp-angular-embed-templates": "2.3.0",
  "gulp-inline-ng2-styles": "0.0.1",
  "typescript": "2.0.0"
},
"dependencies": {
  "@angular/common": "2.4.1",
  "@angular/compiler": "2.4.1",
  "@angular/core": "2.4.1",
  "@angular/http": "2.4.1",
  "@angular/platform-browser": "2.4.1",
  "@angular/platform-browser-dynamic": "2.4.1",
  "rxjs": "5.0.2",
  "zone.js": "0.7.4"
}
}

```

расстояние / tsconfig.json

Создайте папку dist и поместите этот файл внутри. Этот файл используется для указания TypeScript как скомпилировать ваше приложение. Где взять папку с машинописными текстами и куда поместить скомпилированные файлы.

```

{
  "compilerOptions": {
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "mapRoot": "",
    "rootDir": ".",
    "target": "es5",
    "lib": ["es6", "es2015", "dom"],
    "inlineSources": true,
    "stripInternal": true,
    "module": "commonjs",
    "moduleResolution": "node",
    "removeComments": true,
    "sourceMap": true,

```

```
"outDir": "../lib",
"declaration": true
}
}
```

После создания файлов конфигурации мы должны создать наш компонент и модуль. Этот компонент получает клик и отображает сообщение. Он используется как html-тег `<angular-x-minimal-npm-package></angular-x-minimal-npm-package>`. Просто установите этот пакет npm и загрузите его модуль в модель, которую вы хотите использовать.

SRC / углового-x-минимальной NPM-package.component.ts

```
import {Component} from '@angular/core';
@Component({
  selector: 'angular-x-minimal-npm-package',
  styleUrls: ['./angular-x-minimal-npm-package.component.scss'],
  templateUrl: './angular-x-minimal-npm-package.component.html'
})
export class AngularXMinimalNpmPackageComponent {
  message = "Click Me ...";
  onClick() {
    this.message = "Angular 2+ Minimal NPM Package. With external scss and html!";
  }
}
```

SRC / углового-x-минимальной NPM-package.component.html

```
<div>
  <h1 (click)="onClick()">{{message}}</h1>
</div>
```

SRC / Угловое-x-данных table.component.css

```
h1{
  color: red;
}
```

SRC / углового-x-минимальной NPM-package.module.ts

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { AngularXMinimalNpmPackageComponent } from './angular-x-minimal-npm-
package.component';

@NgModule({
  imports: [ CommonModule ],
  declarations: [ AngularXMinimalNpmPackageComponent ],
  exports: [ AngularXMinimalNpmPackageComponent ],
  entryComponents: [ AngularXMinimalNpmPackageComponent ],
})
```



```
export class AngularXMinimalNpmPackageModule {}
```

После этого вы должны скомпилировать, создать и опубликовать свой пакет.

Сборка и компиляция

Для сборки мы используем `gulp` и для компиляции мы используем `tsc`. Команда устанавливается в файле `package.json` по параметру `scripts.build`. У нас есть этот набор `gulp js:build && rm -rf lib && tsc -p dist`. Это наши цепочки, которые будут выполнять эту работу для нас.

Чтобы создать и скомпилировать, запустите в корневом каталоге следующую команду:

```
npm run build
```

Это вызовет цепочку, и в итоге вы получите свою папку сборки `/dist` и скомпилированный пакет в папке `/lib`. Вот почему в `index.js` мы экспортировали код из папки `/lib` а не из `/src`.

Публиковать

Теперь нам просто нужно опубликовать наш пакет, чтобы мы могли установить его через `npm`. Для этого просто запустите команду:

```
npm publish
```

Это все!!!

Прочитайте [Создайте пакет с угловым 2+ NPM онлайн](https://riptutorial.com/ru/angular2/topic/8790/создайте-пакет-с-угловым-2plus-npm):

<https://riptutorial.com/ru/angular2/topic/8790/создайте-пакет-с-угловым-2plus-npm>

глава 52: Создание Угловой библиотеки npm

Вступление

Как опубликовать свой NgModule, написанный в TypeScript в реестре npm. Настройка проекта npm, составление машинописного текста, сборка и непрерывная интеграция.

Examples

Минимальный модуль с классом обслуживания

Файловая структура

```
/
  -src/
    awesome.service.ts
    another-awesome.service.ts
    awesome.module.ts
  -index.ts
  -tsconfig.json
  -package.json
  -rollup.config.js
  -.npmignore
```

Сервис и модуль

Поместите здесь свою удивительную работу.

SRC / awesome.service.ts:

```
export class AwesomeService {
  public doSomethingAwesome(): void {
    console.log("I am so awesome!");
  }
}
```

SRC / awesome.module.ts:

```
import { NgModule } from '@angular/core'
import { AwesomeService } from './awesome.service';
import { AnotherAwesomeService } from './another-awesome.service';

@NgModule({
```

```
    providers: [AwesomeService, AnotherAwesomeService]
  })
  export class AwesomeModule {}
```

Сделайте свой модуль и службу доступными снаружи.

/index.ts:

```
export { AwesomeService } from './src/awesome.service';
export { AnotherAwesomeService } from './src/another-awesome.service';
export { AwesomeModule } from './src/awesome.module';
```

КОМПИЛЯЦИЯ

В `compilerOptions.paths` вам нужно указать все внешние модули, которые вы использовали в своем пакете.

/tsconfig.json

```
{
  "compilerOptions": {
    "baseUrl": ".",
    "declaration": true,
    "stripInternal": true,
    "experimentalDecorators": true,
    "strictNullChecks": false,
    "noImplicitAny": true,
    "module": "es2015",
    "moduleResolution": "node",
    "paths": {
      "@angular/core": ["node_modules/@angular/core"],
      "rxjs/*": ["node_modules/rxjs/*"]
    },
    "rootDir": ".",
    "outDir": "dist",
    "sourceMap": true,
    "inlineSources": true,
    "target": "es5",
    "skipLibCheck": true,
    "lib": [
      "es2015",
      "dom"
    ]
  },
  "files": [
    "index.ts"
  ],
  "angularCompilerOptions": {
    "strictMetadataEmit": true
  }
}
```

Снова укажите свои внешние

/rollup.config.js

```
export default {
  entry: 'dist/index.js',
  dest: 'dist/bundles/awesome.module.umd.js',
  sourceMap: false,
  format: 'umd',
  moduleName: 'ng.awesome.module',
  globals: {
    '@angular/core': 'ng.core',
    'rxjs': 'Rx',
    'rxjs/Observable': 'Rx',
    'rxjs/ReplaySubject': 'Rx',
    'rxjs/add/operator/map': 'Rx.Observable.prototype',
    'rxjs/add/operator/mergeMap': 'Rx.Observable.prototype',
    'rxjs/add/observable/fromEvent': 'Rx.Observable',
    'rxjs/add/observable/of': 'Rx.Observable'
  },
  external: ['@angular/core', 'rxjs']
}
```

Настройки NPM

Теперь давайте укажем некоторые инструкции для npm

/package.json

```
{
  "name": "awesome-angular-module",
  "version": "1.0.4",
  "description": "Awesome angular module",
  "main": "dist/bundles/awesome.module.umd.min.js",
  "module": "dist/index.js",
  "typings": "dist/index.d.ts",
  "scripts": {
    "test": "",
    "transpile": "ngc",
    "package": "rollup -c",
    "minify": "uglifyjs dist/bundles/awesome.module.umd.js --screw-ie8 --compress --mangle --comments --output dist/bundles/awesome.module.umd.min.js",
    "build": "rimraf dist && npm run transpile && npm run package && npm run minify",
    "prepublishOnly": "npm run build"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/maciejtreder/awesome-angular-module.git"
  },
  "keywords": [
    "awesome",
    "angular",
    "module",
    "minimal"
  ],
  "author": "Maciej Treder <contact@maciejtreder.com>",
  "license": "MIT",
  "bugs": {
```

```
  "url": "https://github.com/maciejtreder/awesome-angular-module/issues"
},
"homepage": "https://github.com/maciejtreder/awesome-angular-module#readme",
"devDependencies": {
  "@angular/compiler": "^4.0.0",
  "@angular/compiler-cli": "^4.0.0",
  "rimraf": "^2.6.1",
  "rollup": "^0.43.0",
  "typescript": "^2.3.4",
  "uglify-js": "^3.0.21"
},
"dependencies": {
  "@angular/core": "^4.0.0",
  "rxjs": "^5.3.0"
}
}
```

Мы также можем указать, какие файлы, npm следует игнорировать

/.npmignore

```
node_modules
npm-debug.log
Thumbs.db
.DS_Store
src
!dist/src
plugin
!dist/plugin
*.ngsummary.json
*.iml
rollup.config.js
tsconfig.json
*.ts
!*d.ts
.idea
```

Непрерывная интеграция

Наконец, вы можете настроить непрерывную интеграцию

.travis.yml

```
language: node_js
node_js:
- node

deploy:
  provider: npm
  email: contact@maciejtreder.com
  api_key:
    secure: <your api key>
  on:
    tags: true
    repo: maciejtreder/awesome-angular-module
```

Демо можно найти здесь: <https://github.com/maciejtreder/awesome-angular-module>

Прочитайте [Создание Угловой библиотеки npm онлайн](#):

<https://riptutorial.com/ru/angular2/topic/10704/создание-угловой-библиотеки-npm>

глава 53: Тестирование ngModel

Вступление

Является примером того, как вы можете протестировать компонент в Angular2, у которого есть ngModel.

Examples

Базовый тест

```
import { BrowserModule } from '@angular/platform-browser';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpClientModule } from '@angular/http';

import { Component, DebugElement } from '@angular/core';
import { dispatchEvent } from "@angular/platform-browser/testing/browser_util";
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { By } from "@angular/platform-browser";

import { MyComponentModule } from 'ng2-my-component';
import { MyComponent } from './my-component';

describe('MyComponent:', () => {

  const template = `
    <div>
      <my-component type="text" [(ngModel)]="value" name="TestName" size="9" min="3" max="8"
placeholder="testPlaceholder" disabled=false required=false></my-component>
    </div>
  `;

  let fixture:any;
  let element:any;
  let context:any;

  beforeEach(() => {

    TestBed.configureTestingModule({
      declarations: [InlineEditorComponent],
      imports: [
        FormsModule,
        InlineEditorModule]
    });
    fixture = TestBed.overrideComponent(InlineEditorComponent, {
      set: {
        selector:"inline-editor-test",
        template: template
      }
    });
    .createComponent(InlineEditorComponent);
    context = fixture.componentInstance;
    fixture.detectChanges();
  });
});
```

```
it('should change value of the component', () => {
  let input = fixture.nativeElement.querySelector("input");
  input.value = "Username";
  dispatchEvent(input, 'input');
  fixture.detectChanges();

  fixture.whenStable().then(() => {
    //this button dispatch event for save the text in component.value
    fixture.nativeElement.querySelectorAll('button')[0].click();
    expect(context.value).toBe("Username");
  });
});
```

Прочитайте Тестирование ngModel онлайн: <https://riptutorial.com/ru/angular2/topic/8693/тестирование-ngmodel>

глава 54: Тестирование приложения Angular 2

Examples

Установка схемы тестирования жасмина

Самый распространенный способ тестирования приложений Angular 2 - с помощью тестовой среды Jasmine. Жасмин позволяет вам проверять свой код в браузере.

устанавливать

Чтобы начать работу, вам нужен только пакет `jasmine-core` (не `jasmine`).

```
npm install jasmine-core --save-dev --save-exact
```

проверить

Чтобы проверить правильность настройки Жасмина, создайте файл `./src/unit-tests.html` со следующим содержимым и откройте его в браузере.

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html; charset=utf-8">
  <title>Ng App Unit Tests</title>
  <link rel="stylesheet" href="../node_modules/jasmine-core/lib/jasmine-core/jasmine.css">
  <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine.js"></script>
  <script src="../node_modules/jasmine-core/lib/jasmine-core/jasmine-html.js"></script>
  <script src="../node_modules/jasmine-core/lib/jasmine-core/boot.js"></script>
</head>
<body>
  <!-- Unit Testing Chapter #1: Proof of life. -->
  <script>
    it('true is true', function () {
      expect(true).toEqual(true);
    });
  </script>
</body>
</html>
```

Настройка тестирования с помощью Gulp, Webpack, Karma и Jasmine

Первое, что нам нужно - сказать карме, чтобы использовать Webpack для чтения наших

тестов, в соответствии с конфигурацией, которую мы установили для механизма webpack. Здесь я использую babel, потому что я пишу свой код в ES6, вы можете изменить его для других вкусов, таких как TypeScript. Или я использую шаблоны Pug (ранее Jade), вам это не нужно.

Тем не менее стратегия остается прежней.

Итак, это конфигурация webpack:

```
const webpack = require("webpack");
let packConfig = {
  entry: {},
  output: {},
  plugins:[
    new webpack.DefinePlugin({
      ENVIRONMENT: JSON.stringify('test')
    })
  ],
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude:/(node_modules|bower_components)/,
        loader: "babel",
        query:{
          presets:["es2015", "angular2"]
        }
      },
      {
        test: /\.woff2??$|\.ttf?$|\.eot?$|\.svg$/,
        loader: "file"
      },
      {
        test: /\.scss$/,
        loaders: ["style", "css", "sass"]
      },
      {
        test: /\.pug$/,
        loader: 'pug-html-loader'
      },
    ],
    devtool : 'inline-cheap-source-map'
  };
module.exports = packConfig;
```

И тогда нам нужен файл karma.config.js для использования этой конфигурации webpack:

```
const packConfig = require("./webpack.config.js");
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine'],
    exclude:[],
    files: [
      {pattern: './karma.shim.js', watched: false}
    ],
  },
```

```

preprocessors: {
  "./karma.shim.js": ["webpack"]
},
webpack: packConfig,

webpackServer: {noInfo: true},

port: 9876,

colors: true,

logLevel: config.LOG_INFO,

browsers: ['PhantomJS'],

concurrency: Infinity,

autoWatch: false,
singleRun: true
});
};

```

До сих пор мы сказали Карме использовать webpack, и мы сказали ему начать с файла **karma.shim.js**. Этот файл будет работать как отправная точка для webpack. webpack прочитает этот файл и будет использовать инструкции **import** и **require** для сбора всех наших зависимостей и выполнения наших тестов.

Итак, давайте посмотрим на файл karma.shim.js:

```

// Start of ES6 Specific stuff
import "es6-shim";
import "es6-promise";
import "reflect-metadata";
// End of ES6 Specific stuff

import "zone.js/dist/zone";
import "zone.js/dist/long-stack-trace-zone";
import "zone.js/dist/jasmine-patch";
import "zone.js/dist/async-test";
import "zone.js/dist/fake-async-test";
import "zone.js/dist/sync-test";
import "zone.js/dist/proxy-zone";

import 'rxjs/add/operator/map';
import 'rxjs/add/observable/of';

Error.stackTraceLimit = Infinity;

import { TestBed } from "@angular/core/testing";
import { BrowserDynamicTestingModule, platformBrowserDynamicTesting } from "@angular/platform-browser-dynamic/testing";

TestBed.initTestEnvironment (
  BrowserDynamicTestingModule,
  platformBrowserDynamicTesting());

let testContext = require.context('../src/app', true, /\.spec\.js/);

```

```
testContext.keys().forEach(testContext);
```

По сути, мы импортируем **TestBed** из тестирования угловых ядер и иницируем среду, так как ее нужно инициировать только один раз для всех наших тестов. Затем мы рекурсивно просматриваем каталог **src / app** и читаем каждый файл, который заканчивается на **.spec.js**, и **передаем** их в `testContext`, поэтому они будут запускаться.

Обычно я пытаюсь поставить свои тесты на то же место, что и класс. Чувствительный вкус, мне легче импортировать зависимости и рефакторинг с классами. Но если вы хотите поместить свои тесты в другое место, например, например, в каталог **src / test**, вот вам шанс. измените строку до последнего в файле `karma.shim.js`.

Отлично. Что слева? ah, задача `gulp`, которая использует файл `karma.config.js`, который мы сделали выше:

```
gulp.task("karmaTests", function(done) {
  var Server = require("karma").Server;
  new Server({
    configFile : "./karma.config.js",
    singleRun: true,
    autoWatch: false
  }, function(result){
    return result ? done(new Error(`Karma failed with error code ${result}`)):done();
  }).start();
});
```

Теперь я запускаю сервер с созданным конфигурационным файлом, говоря ему, чтобы он запускался один раз и не следил за изменениями. Я считаю, что это лучше подходит для меня, поскольку тесты будут выполняться только в том случае, если я буду готов к их запуску, но, конечно, если вы захотите по-другому, вы знаете, где меняться.

И как мой последний пример кода, вот набор тестов для Углового 2 учебника «Тур героев».

```
import {
  TestBed,
  ComponentFixture,
  async
} from "@angular/core/testing";

import {AppComponent} from "../app.component";
import {AppModule} from "../app.module";
import Hero from "../hero/hero";

describe("App Component", function () {

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [AppModule]
    });

    this.fixture = TestBed.createComponent(AppComponent);
    this.fixture.detectChanges();
  });
```

```

it("Should have a title", async(()=> {
  this.fixture.whenStable().then(()=> {
    expect(this.fixture.componentInstance.title).toEqual("Tour of Heros");
  });
}));

it("Should have a hero", async(()=> {
  this.fixture.whenStable().then(()=> {
    expect(this.fixture.componentInstance.selectedHero).toBeNull();
  });
}));

it("Should have an array of heros", async(()=>
  this.fixture.whenStable().then(()=> {
    const cmp = this.fixture.componentInstance;
    expect(cmp.heroes).toBeDefined("component should have a list of heroes");
    expect(cmp.heroes.length).toEqual(10, "heroes list should have 10 members");
    cmp.heroes.map((h, i)=> {
      expect(h instanceof Hero).toBeTruthy(`member ${i} is not a Hero instance.
${h}`)
    });
  }));

it("Should have one list item per hero", async(()=>
  this.fixture.whenStable().then(()=> {
    const ul = this.fixture.nativeElement.querySelector("ul.heroes");
    const li = Array.prototype.slice.call(
      this.fixture.nativeElement.querySelectorAll("ul.heroes>li"));
    const cmp = this.fixture.componentInstance;
    expect(ul).toBeTruthy("There should be an unnumbered list for heroes");
    expect(li.length).toEqual(cmp.heroes.length, "there should be one li for each
hero");
    li.forEach((li, i)=> {
      expect(li.querySelector("span.badge"))
        .toBeTruthy(`hero ${i} has to have a span for id`);
      expect(li.querySelector("span.badge").textContent.trim())
        .toEqual(cmp.heroes[i].id.toString(), `hero ${i} had wrong id displayed`);
      expect(li.textContent)
        .toMatch(cmp.heroes[i].name, `hero ${i} has wrong name displayed`);
    });
  }));

it("should have correct styling of hero items", async(()=>
  this.fixture.whenStable().then(()=> {
    const hero = this.fixture.nativeElement.querySelector("ul.heroes>li");
    const win = hero.ownerDocument.defaultView || hero.ownerDocument.parentWindow;
    const styles = win.getComputedStyle(hero);
    expect(styles["cursor"]).toEqual("pointer", "cursor should be pointer on hero");
    expect(styles["borderRadius"]).toEqual("4px", "borderRadius should be 4px");
  }));

it("should have a click handler for hero items", async(()=>
  this.fixture.whenStable().then(()=>{
    const cmp = this.fixture.componentInstance;
    expect(cmp.onSelect)
      .toBeDefined("should have a click handler for heros");
    expect(this.fixture.nativeElement.querySelector("input.heroName"))
      .toBeNull("should not show the hero details when no hero has been selected");
    expect(this.fixture.nativeElement.querySelector("ul.heroes li.selected"))
      .toBeNull("Should not have any selected heroes at start");
  }));

```

```

spyOn(cmp, "onSelect").and.callThrough();
this.fixture.nativeElement.querySelector("ul.heroes li")[5].click();

expect(cmp.onSelect)
  .toHaveBeenCalledWith(cmp.heroes[5]);
expect(cmp.selectedHero)
  .toEqual(cmp.heroes[5], "click on hero should change hero");
  })
  });
});

```

Примечательно, что у нас есть **beforeEach ()** настроить тестовый модуль и создать компонент в тесте, и как мы называем **detectChanges ()**, так что угловой фактически проходит через двойное связывание и все.

Обратите внимание, что каждый тест является вызовом **async ()**, и он всегда ждет, когда команда «**Стабильность**» решит проблему перед исследованием прибора. Затем он имеет доступ к компоненту через **componentInstance** и к элементу через **nativeElement** .

Существует один тест, который проверяет правильный стиль. как часть учебника, команда Angular демонстрирует использование стилей внутри компонентов. В нашем тесте мы используем **getComputedStyle ()**, чтобы проверить, что стили идут от того, где мы указали, однако для этого нам нужен объект Window, и мы получаем его из элемента, как вы можете видеть в тесте.

Тестирование службы Http

Обычно службы вызывают удаленный Api для извлечения / отправки данных. Но модульные тесты не должны выполнять сетевые вызовы. Угловая внутренне использует класс `XHRBackend` для выполнения HTTP-запросов. Пользователь может переопределить это, чтобы изменить поведение. Модуль углового тестирования предоставляет `MockBackend` и `MockConnection` которые могут использоваться для тестирования и утверждения HTTP-запросов.

`posts.service.ts` Эта служба попадает в конечную точку api для получения списка сообщений.

```

import { Http } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/rx';

import 'rxjs/add/operator/map';

export interface IPost {
  userId: number;
  id: number;
  title: string;
  body: string;
}

```

```

@Injectables()
export class PostsService {
  posts: IPost[];

  private postsUri = 'http://jsonplaceholder.typicode.com/posts';

  constructor(private http: Http) {
  }

  get(): Observable<IPost[]> {
    return this.http.get(this.postsUri)
      .map((response) => response.json());
  }
}

```

posts.service.spec.ts **Здесь мы проверим выше сервис, насмехаясь над вызовами http api.**

```

import { TestBed, inject, fakeAsync } from '@angular/core/testing';
import {
  HttpClientModule,
  XMLHttpRequest,
  ResponseOptions,
  Response,
  RequestMethod
} from '@angular/http';
import {
  MockBackend,
  MockConnection
} from '@angular/http/testing';

import { PostsService } from './posts.service';

describe('PostsService', () => {
  // Mock http response
  const mockResponse = [
    {
      'userId': 1,
      'id': 1,
      'title': 'sunt aut facere repellat provident occaecati excepturi optio reprehenderit',
      'body': 'quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto'
    },
    {
      'userId': 1,
      'id': 2,
      'title': 'qui est esse',
      'body': 'est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dolores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui aperiam non debitis possimus qui neque nisi nulla'
    },
    {
      'userId': 1,
      'id': 3,
      'title': 'ea molestias quasi exercitationem repellat qui ipsa sit aut',
      'body': 'et iusto sed quo iure\nvoluptatem occaecati omnis eligendi aut ad\nvoluptatem doloribus vel accusantium quis pariatur\nmolestiae porro eius odio et labore et velit aut'
    },
  ],

```

```

    {
      'userId': 1,
      'id': 4,
      'title': 'eum et est occaecati',
      'body': 'ullam et saepe reiciendis voluptatem adipisci\nsit amet autem assumenda
provident rerum culpa\nquis hic commodi nesciunt rem tenetur doloremque ipsam iure\nquis sunt
voluptatem rerum illo velit'
    }
  ];

  beforeEach(() => {
    TestBed.configureTestingModule({
      imports: [HttpModule],
      providers: [
        {
          provide: XHRBackend,
          // This provides mocked XHR backend
          useClass: MockBackend
        },
        PostsService
      ]
    });
  });

  it('should return posts retrieved from Api', fakeAsync(
    inject([XHRBackend, PostsService],
      (mockBackend, postsService) => {
        mockBackend.connections.subscribe(
          (connection: MockConnection) => {
            // Assert that service has requested correct url with expected method
            expect(connection.request.method).toBe(RequestMethod.Get);

expect(connection.request.url).toBe('http://jsonplaceholder.typicode.com/posts');
            // Send mock response
            connection.mockRespond(new Response(new ResponseOptions({
              body: mockResponse
            })));
          });

        postsService.get()
          .subscribe((posts) => {
            expect(posts).toBe(mockResponse);
          });

        }));
  });
});

```

Тестирование угловых компонентов - базовые

Код компонента приведен ниже.

```

import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
export class MyAppComponent{
  title = 'welcome';
}

```



```
}
```

Для угловых испытаний угловые обеспечивают свои тестовые утилиты вместе с испытательной основой, которая помогает при написании хорошего тестового случая в угловом режиме. Угловые утилиты могут быть импортированы из `@angular/core/testing`

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { MyAppComponent } from './banner-inline.component';

describe('Tests for MyAppComponent', () => {

  let fixture: ComponentFixture<MyAppComponent>;
  let comp: MyAppComponent;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
        MyAppComponent
      ]
    });
  });

  beforeEach(() => {

    fixture = TestBed.createComponent(MyAppComponent);
    comp = fixture.componentInstance;

  });

  it('should create the MyAppComponent', () => {

    expect(comp).toBeTruthy();

  });

});
```

В приведенном выше примере существует только один тестовый пример, который объясняет тестовый пример существования компонента. В приведенном выше примере используются утилиты углового тестирования, такие как `TestBed` и `ComponentFixture` .

`TestBed` используется для создания модуля углового тестирования, и мы настраиваем этот модуль с `configureTestingModule` метода `configureTestingModule` для создания среды модуля для класса, который мы хотим протестировать. Модуль тестирования должен быть настроен перед выполнением каждого тестового примера, поэтому мы настраиваем модуль тестирования в функции `beforeEach` .

`createComponent` метод `TestBed` используется для создания экземпляра компонента в процессе тестирования. `createComponent` возвращает `ComponentFixture` . Устройство обеспечивает доступ к самому экземпляру компонента.

Прочитайте Тестирование приложения Angular 2 онлайн:

<https://riptutorial.com/ru/angular2/topic/2329/тестирование-приложения-angular-2>

глава 55: Труба заказа

Вступление

Как написать заказную трубку и использовать ее.

Examples

Труба

Реализация труб

```
import {Pipe, PipeTransform} from '@angular/core';

@Pipe({
  name: 'orderBy',
  pure: false
})
export class OrderBy implements PipeTransform {

  value:string[] =[];

  static _orderByComparator(a:any, b:any):number{

    if(a === null || typeof a === 'undefined') a = 0;
    if(b === null || typeof b === 'undefined') b = 0;

    if((isNaN(parseFloat(a)) || !isFinite(a)) || (isNaN(parseFloat(b)) || !isFinite(b))){
      //Isn't a number so lowercase the string to properly compare
      if(a.toLowerCase() < b.toLowerCase()) return -1;
      if(a.toLowerCase() > b.toLowerCase()) return 1;
    }else{
      //Parse strings as numbers to compare properly
      if(parseFloat(a) < parseFloat(b)) return -1;
      if(parseFloat(a) > parseFloat(b)) return 1;
    }

    return 0; //equal each other
  }

  transform(input:any, config:string = '+'): any{

    //make a copy of the input's reference
    this.value = [...input];
    let value = this.value;

    if(!Array.isArray(value)) return value;

    if(!Array.isArray(config) || (Array.isArray(config) && config.length === 1)){
      let propertyToCheck:string = !Array.isArray(config) ? config : config[0];
      let desc = propertyToCheck.substr(0, 1) === '-';

      //Basic array
```

```

    if(!propertyToCheck || propertyToCheck === '-' || propertyToCheck === '+'){
      return !desc ? value.sort() : value.sort().reverse();
    }else {
      let property:string = propertyToCheck.substr(0, 1) === '+' ||
propertyToCheck.substr(0, 1) === '-'
        ? propertyToCheck.substr(1)
        : propertyToCheck;

      return value.sort(function(a:any,b:any){
        return !desc
          ? OrderBy._orderByComparator(a[property], b[property])
          : -OrderBy._orderByComparator(a[property], b[property]);
      });
    }
  } else {
    //Loop over property of the array in order and sort
    return value.sort(function(a:any,b:any){
      for(let i:number = 0; i < config.length; i++){
        let desc = config[i].substr(0, 1) === '-';
        let property = config[i].substr(0, 1) === '+' || config[i].substr(0, 1) === '-'
          ? config[i].substr(1)
          : config[i];

        let comparison = !desc
          ? OrderBy._orderByComparator(a[property], b[property])
          : -OrderBy._orderByComparator(a[property], b[property]);

        //Don't return 0 yet in case of needing to sort by next property
        if(comparison !== 0) return comparison;
      }

      return 0; //equal each other
    });
  }
}
}
}
}

```

Как использовать канал в HTML-порядке по возрастанию по имени

```

<table>
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of users | orderBy : ['firstName']">
      <td>{{user.firstName}}</td>
      <td>{{user.lastName}}</td>
      <td>{{user.age}}</td>
    </tr>
  </tbody>
</table>

```

Как использовать трубку в HTML-порядке по убыванию по имени

```
<table>
  <thead>
    <tr>
      <th>First Name</th>
      <th>Last Name</th>
      <th>Age</th>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let user of users | orderBy : ['-firstName']">
      <td>{{user.firstName}}</td>
      <td>{{user.lastName}}</td>
      <td>{{user.age}}</td>
    </tr>
  </tbody>
</table>
```

Прочитайте Труба заказа онлайн: <https://riptutorial.com/ru/angular2/topic/8969/труба-заказа>

глава 56: трубы

Вступление

Труба | character используется для применения труб в Angular 2. Трубы очень похожи на фильтры в AngularJS, поскольку они помогают преобразовать данные в заданный формат.

параметры

Функция / параметр	объяснение
@Pipe ({name, pure})	метаданные для трубы, должны немедленно предшествовать классу труб
имя: <i>строка</i>	что вы будете использовать внутри шаблона
pure: <i>boolean</i>	по умолчанию - true, отметьте это как false, чтобы ваш канал повторно оценивался чаще
transform (значение, args [])	функция, вызываемая для преобразования значений в шаблон
значение: <i>любое</i>	значение, которое вы хотите преобразовать
args: <i>any []</i>	аргументы, которые могут понадобиться вам в вашем преобразовании. Отметить необязательные аргументы с? оператор вроде так преобразует (значение, arg1, arg2?)

замечания

В этом разделе рассматривается [Angular2 Pipes](#) , механизм преобразования и форматирования данных в HTML-шаблонах в приложении Angular2.

Examples

Цепные трубы

Трубы могут быть прикованы.

```
<p>Today is {{ today | date:'fullDate' | uppercase}}.</p>
```

Пользовательские трубы

my.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'myPipe'})
export class MyPipe implements PipeTransform {

  transform(value:any, args?: any):string {
    let transformedValue = value; // implement your transformation logic here
    return transformedValue;
  }

}
```

my.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-component',
  template: `{{ value | myPipe }}`
})
export class MyComponent {

  public value:any;

}
```

my.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { MyComponent } from './my.component';
import { MyPipe } from './my.pipe';

@NgModule({
  imports: [
    BrowserModule,
  ],
  declarations: [
    MyComponent,
    MyPipe
  ],
})
export class MyModule { }
```

Встроенные трубы

Angular2 поставляется с несколькими встроенными

трубами:

труба	использование	пример
<code>DatePipe</code>	<code>date</code>	<code>{{ dateObj date }} // output is 'Jun 15, 2015'</code>
<code>UpperCasePipe</code>	<code>uppercase</code>	<code>{{ value uppercase }} // output is 'SOMETEXT'</code>
<code>LowerCasePipe</code>	<code>lowercase</code>	<code>{{ value lowercase }} // output is 'sometext'</code>
<code>CurrencyPipe</code>	<code>currency</code>	<code>{{ 31.00 currency:'USD':true }} // output is '\$31'</code>
<code>PercentPipe</code>	<code>percent</code>	<code>{{ 0.03 percent }} //output is %3</code>

Есть и другие. Посмотрите [здесь](#) их документацию.

пример

отель-reservation.component.ts

```
import { Component } from '@angular/core';

@Component({
  moduleId: module.id,
  selector: 'hotel-reservation',
  templateUrl: './hotel-reservation.template.html'
})
export class HotelReservationComponent {
  public fName: string = 'Joe';
  public lName: string = 'SCHMO';
  public reservationMade: string = '2016-06-22T07:18-08:00'
  public reservationFor: string = '2025-11-14';
  public cost: number = 99.99;
}
```

Гостинично-reservation.template.html

```
<div>
  <h1>Welcome back {{fName | uppercase}} {{lName | lowercase}}</h1>
  <p>
    On {reservationMade | date} at {reservationMade | date:'shortTime'} you
    reserved room 205 for {reservationDate | date} for a total cost of
    {cost | currency}.
  </p>
</div>
```

Выход

```
Welcome back JOE schmo
```

On Jun 26, 2016 at 7:18 you reserved room 205 for Nov 14, 2025 for a total cost of \$99.99.

Отладка с помощью JsonPipe

JsonPipe может использоваться для отладки состояния любого заданного внутреннего.

Код

```
@Component({
  selector: 'json-example',
  template: `<div>
    <p>Without JSON pipe:</p>
    <pre>{{object}}</pre>
    <p>With JSON pipe:</p>
    <pre>{{object | json}}</pre>
  </div>`
})
export class JsonPipeExample {
  object: Object = {foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}};
}
```

Выход

```
Without JSON Pipe:
object
With JSON pipe:
{object:{foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}}
```

Доступная по всему миру пользовательская труба

Чтобы сделать доступное приложение доступным пользователем, во время загрузки загрузочного диска, PLATFORM_PIPES.

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { provide, PLATFORM_PIPES } from '@angular/core';

import { AppComponent } from './app.component';
import { MyPipe } from './my.pipe'; // your custom pipe

bootstrap(AppComponent, [
  provide(PLATFORM_PIPES, {
    useValue: [
      MyPipe
    ],
    multi: true
  })
]);
```

Учебник здесь: <https://scotch.io/tutorials/create-a-globally-available-custom-pipe-in-angular-2>

Создание пользовательских труб

приложение / pipes.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'truthy'})
export class Truthy implements PipeTransform {
  transform(value: any, truthy: string, falsey: string): any {
    if (typeof value === 'boolean'){return value ? truthy : falsey;}
    else return value
  }
}
```

приложение / My-component.component.ts

```
import { Truthy } from './pipes.pipe';

@Component({
  selector: 'my-component',
  template: `
    <p>{{value | truthy:'enabled':'disabled'}}</p>
  `,
  pipes: [Truthy]
})
export class MyComponent{ }
```

Отменить асинхронные значения с помощью async-канала

```
import { Component } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/observable/of';

@Component({
  selector: 'async-stuff',
  template: `
    <h1>Hello, {{ name | async }}</h1>
    Your Friends are:
    <ul>
      <li *ngFor="let friend of friends | async">
        {{friend}}
      </li>
    </ul>
  `
})
class AsyncStuffComponent {
  name = Promise.resolve('Misko');
  friends = Observable.of(['Igor']);
}
```

СТАНОВИТСЯ:

```
<h1>Hello, Misko</h1>
Your Friends are:
<ul>
```

```
<li>
  Igor
</li>
</ul>
```

Расширение существующей трубы

```
import { Pipe, PipeTransform } from '@angular/core';
import { DatePipe } from '@angular/common'

@Pipe({name: 'ifDate'})
export class IfDate implements PipeTransform {
  private datePipe: DatePipe = new DatePipe();

  transform(value: any, pattern?:string) : any {
    if (typeof value === 'number') {return value}
    try {
      return this.datePipe.transform(value, pattern)
    } catch(err) {
      return value
    }
  }
}
```

Состоятельные трубы

Угловой 2 предлагает два разных типа труб - без гражданства и состояния. По умолчанию они не имеют апатридов. Тем не менее, мы можем реализовать протоколы `stateful`, установив для свойства `pure` значение `false`. Как вы можете видеть в разделе параметров, вы можете указать `name` и объявить, должен ли канал быть чистым или нет, что означает состояние или без состояния. В то время как данные передаются по безстоящей трубе (которая является чистой функцией), которая **ничего не** помнит, данные могут управляться и запоминаться с помощью протоколов `stateful`. Хорошим примером трубы с `AsyncPipe` состояния является `AsyncPipe`, предоставляемый Angular 2.

Важный

Обратите внимание, что большинство труб должно относиться к категории безгосударственных труб. Это важно по соображениям производительности, так как Angular может оптимизировать трубы без гражданства для детектора изменений. Поэтому осторожно используйте стоп-сигналы. В общем, оптимизация труб в Angular 2 имеет значительное повышение производительности по сравнению с фильтрами в Angular 1.x. В Angular 1 цикл дайджеста всегда должен был повторно запускать все фильтры, даже если данные вообще не изменились. В Угловом 2, когда значение трубы было вычислено, детектор изменений знает, что он не запускает этот трубопровод снова, если вход не изменится.

Реализация трубы с отслеживанием состояния

```

import {Pipe, PipeTransform, OnDestroy} from '@angular/core';

@Pipe({
  name: 'countdown',
  pure: false
})
export class CountdownPipe implements PipeTransform, OnDestroy {
  private interval: any;
  private remainingTime: number;

  transform(value: number, interval: number = 1000): number {
    if (!parseInt(value, 10)) {
      return null;
    }

    if (typeof this.remainingTime !== 'number') {
      this.remainingTime = parseInt(value, 10);
    }

    if (!this.interval) {
      this.interval = setInterval(() => {
        this.remainingTime--;

        if (this.remainingTime <= 0) {
          this.remainingTime = 0;
          clearInterval(this.interval);
          delete this.interval;
        }
      }, interval);
    }

    return this.remainingTime;
  }

  ngOnDestroy(): void {
    if (this.interval) {
      clearInterval(this.interval);
    }
  }
}

```

Затем вы можете использовать трубу, как обычно:

```

{{ 1000 | countdown:50 }}
{{ 300 | countdown }}

```

Важно, что ваша труба также реализует интерфейс `OnDestroy` поэтому вы можете очистить ее, как только ваша труба будет уничтожена. В приведенном выше примере необходимо очистить интервал, чтобы избежать утечек памяти.

Динамическая труба

Пример сценария использования: представление таблицы состоит из разных столбцов с различным форматом данных, которые необходимо преобразовать с помощью разных каналов.

table.component.ts

```
...
import { DYNAMIC_PIPES } from '../pipes/dynamic.pipe.ts';

@Component({
  ...
  pipes: [DYNAMIC_PIPES]
})
export class TableComponent {
  ...

  // pipes to be used for each column
  table.pipes = [ null, null, null, 'humanizeDate', 'statusFromBoolean' ],
  table.header = [ 'id', 'title', 'url', 'created', 'status' ],
  table.rows = [
    [ 1, 'Home', 'home', '2016-08-27T17:48:32', true ],
    [ 2, 'About Us', 'about', '2016-08-28T08:42:09', true ],
    [ 4, 'Contact Us', 'contact', '2016-08-28T13:28:18', false ],
    ...
  ]
  ...
}
```

dynamic.pipe.ts

```
import {
  Pipe,
  PipeTransform
} from '@angular/core';
// Library used to humanize a date in this example
import * as moment from 'moment';

@Pipe({name: 'dynamic'})
export class DynamicPipe implements PipeTransform {

  transform(value:string, modifier:string) {
    if (!modifier) return value;
    // Evaluate pipe string
    return eval('this.' + modifier + '(' + value + ')');
  }

  // Returns 'enabled' or 'disabled' based on input value
  statusFromBoolean(value:string):string {
    switch (value) {
      case 'true':
      case '1':
        return 'enabled';
      default:
        return 'disabled';
    }
  }

  // Returns a human friendly time format e.g: '14 minutes ago', 'yesterday'
  humanizeDate(value:string):string {
    // Humanize if date difference is within a week from now else returns 'December 20,
    2016' format
    if (moment().diff(moment(value), 'days') < 8) return moment(value).fromNow();
  }
}
```

```

        return moment(value).format('MMMM Do YYYY');
    }
}

export const DYNAMIC_PIPES = [DynamicPipe];

```

table.component.html

```

<table>
  <thead>
    <td *ngFor="let head of data.header">{{ head }}</td>
  </thead>
  <tr *ngFor="let row of table.rows; let i = index">
    <td *ngFor="let column of row">{{ column | dynamic:table.pipes[i] }}</td>
  </tr>
</table>

```

Результат

ID	Page Title	Page URL	Created	Status
1	Home	home	4 minutes ago	Enabled
2	About Us	about	Yesterday	Enabled
4	Contact Us	contact	Yesterday	Disabled

Тестирование трубы

С учетом трубы, которая обращает строку

```

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'reverse' })
export class ReversePipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
  }
}

```

Его можно протестировать, настроив файл спецификации, как это

```

import { TestBed, inject } from '@angular/core/testing';

import { ReversePipe } from './reverse.pipe';

describe('ReversePipe', () => {
  beforeEach(() => {
    TestBed.configureTestingModule({
      providers: [ReversePipe],
    });
  });

  it('should be created', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe).toBeTruthy();
  }));
});

```

```
it('should reverse a string', inject([ReversePipe], (reversePipe: ReversePipe) => {
  expect(reversePipe.transform('abc')).toEqual('cba');
}));
});
```

Прочитайте трубы онлайн: <https://riptutorial.com/ru/angular2/topic/1165/трубы>

глава 57: Угловая - ForLoop

Синтаксис

1. `<div *ngFor = "let item of items; пусть i = index"> {{i}} {{item}} </div>`

замечания

Структурная директива `*ngFor` работает как цикл в коллекции и повторяет фрагмент html для каждого элемента коллекции.

`@View decorator` теперь устарел. Разработчики должны использовать `template` или свойства `templateUrl` для декоратора `@Component`.

Examples

Угловая 2 для петли

Для live [plnkr](#) нажмите ...

```
<!doctype html>
<html>
<head>
  <title>ng for loop in angular 2 with ES5.</title>
  <script type="text/javascript" src="https://code.angularjs.org/2.0.0-alpha.28/angular2.sfx.dev.js"></script>
  <script>
    var ngForLoop = function () {
      this.msg = "ng for loop in angular 2 with ES5.";
      this.users = ["Anil Singh", "Sunil Singh", "Sushil Singh", "Aradhya", 'Reena'];
    };

    ngForLoop.annotations = [
      new angular.Component({
        selector: 'ngforloop'
      }),
      new angular.View({
        template: '<H1>{{msg}}</H1>' +
          '<p> User List : </p>' +
          '<ul>' +
          '<li *ng-for="let user of users">' +
          '{{user}}' +
          '</li>' +
          '</ul>',
        directives: [angular.NgFor]
      })
    ];

    document.addEventListener("DOMContentLoaded", function () {
      angular.bootstrap(ngForLoop);
    });
  </script>
</head>
</html>
```

```

    });
  </script>
</head>
<body>
  <ngforloop></ngforloop>
  <h2>
    <a href="http://www.code-sample.com/" target="_blank">For more detail...</a>
  </h2>
</body>
</html>

```

NgFor - разметка для цикла

Директива **NgFor** создает экземпляр шаблона один раз для каждого элемента из итерабельного. Контекст для каждого экземпляра-шаблона наследуется от внешнего контекста с заданной переменной цикла, установленной для текущего элемента из итерабельного.

Чтобы настроить алгоритм отслеживания по умолчанию, NgFor поддерживает **функцию trackBy**. **trackBy** принимает функцию, которая имеет два аргумента: `index` и `item`. Если **указано trackBy**, угловые дорожки изменяются по возвращаемому значению функции.

```

<li *ngFor="let item of items; let i = index; trackBy: trackByFn">
  {{i}} - {{item.name}}
</li>

```

Дополнительные параметры : NgFor предоставляет несколько экспортированных значений, которые могут быть сглажены для локальных переменных:

- **index** будет установлен на текущую итерацию цикла для каждого контекста шаблона.
- **сначала** будет установлено логическое значение, указывающее, является ли элемент первым в итерации.
- **last** будет установлен на логическое значение, указывающее, является ли элемент последним в итерации.
- **даже** будет установлено логическое значение, указывающее, имеет ли этот элемент четный индекс.
- **нечетное** будет установлено в логическое значение, указывающее, имеет ли этот элемент нечетный индекс.

* ngFor в строках таблицы

```

<table>
  <thead>
    <th>Name</th>
    <th>Index</th>
  </thead>
  <tbody>
    <tr *ngFor="let hero of heroes">
      <td>{{hero.name}}</td>

```



```
        </tr>
    </tbody>
</table>
```

* ngFor с компонентом

```
@Component({
  selector: 'main-component',
  template: '<example-component
            *ngFor="let hero of heroes"
            [hero]="hero"></example-component>'
})

@Component({
  selector: 'example-component',
  template: '<div>{{hero?.name}}</div>'
})

export class ExampleComponent {
  @Input() hero : Hero = null;
}
```

* ngFor X количество элементов в строке

Пример показывает 5 элементов в строке:

```
<div *ngFor="let item of items; let i = index">
  <div *ngIf="i % 5 == 0" class="row">
    {{ item }}
  <div *ngIf="i + 1 < items.length">{{ items[i + 1] }}</div>
  <div *ngIf="i + 2 < items.length">{{ items[i + 2] }}</div>
  <div *ngIf="i + 3 < items.length">{{ items[i + 3] }}</div>
  <div *ngIf="i + 4 < items.length">{{ items[i + 4] }}</div>
</div>
</div>
```

Прочитайте Угловая - ForLoop онлайн: <https://riptutorial.com/ru/angular2/topic/6543/угловая---forloop>

глава 58: Угловая анимация2

Вступление

Система анимации Angular позволяет создавать анимации, которые работают с той же самой природой, что и в чистых анимациях CSS. Вы легко можете объединить свою анимационную логику с остальной частью кода приложения, чтобы упростить управление.

Examples

Основная анимация - переводит элемент между двумя состояниями, управляемыми атрибутом модели.

app.component.html

```
<div>
  <div>
    <div *ngFor="let user of users">
      <button
        class="btn"
        [ @buttonState ]="user.active"
        (click)="user.changeButtonState()">{{user.firstName}}</button>
    </div>
  </div>
</div>
```

app.component.ts

```
import {Component, trigger, state, transition, animate, style} from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styles: [`
    .btn {
      height: 30px;
      width: 100px;
      border: 1px solid rgba(0, 0, 0, 0.33);
      border-radius: 3px;
      margin-bottom: 5px;
    }
  `],
  animations: [
    trigger('buttonState', [
      state('true', style({
        background: '#04b104',
        transform: 'scale(1)'
      })),
    ]),
  ],
})
```

```

    state('false', style({
      background: '#e40202',
      transform: 'scale(1.1)'
    })),
    transition('true => false', animate('100ms ease-in')),
    transition('false => true', animate('100ms ease-out'))
  ])
]
})
export class AppComponent {
  users : Array<User> = [];
  constructor(){
    this.users.push(new User('Narco', false));
    this.users.push(new User('Bombasto', false));
    this.users.push(new User('Celeritas', false));
    this.users.push(new User('Magneta', false));
  }
}

export class User {
  firstName : string;
  active : boolean;

  changeButtonState(){
    this.active = !this.active;
  }
  constructor(_firstName :string, _active : boolean){
    this.firstName = _firstName;
    this.active = _active;
  }
}

```

Прочитайте Угловая анимация2 онлайн: <https://riptutorial.com/ru/angular2/topic/8970/угловая-анимация2>

глава 59: Угловое обновление 2-х форм

замечания

Угловые 2 позволяют получить доступ к экземпляру ngForm, создав локальную переменную шаблона. Угловой 2 предоставляет директивные экземпляры, такие как ngForm, путем указания свойства exportAs в метаданных директивы. Теперь преимущество здесь в том, что вы не можете много кодировать, вы можете получить доступ к экземпляру ngForm и использовать его для доступа к представленным значениям или для проверки правильности всех полей с использованием свойств (действительных, отправленных, значений и т. Д.).

```
#f = ngForm (creates local template instance "f")
```

ngForm выдает событие «ngSubmit» при его отправке (для получения более подробной информации об эмитенте события см. документацию Check @Output)

```
(ngSubmit)= "login(f.value, f.submitted) "
```

«ngModel» создает элемент управления формой в сочетании с атрибутом «name».

```
<input type="text" [(ngModel)]="username" placeholder="enter username" required>
```

Когда форма отправлена, f.value имеет объект JSON, представляющий представленные значения.

```
{имя пользователя: 'Sachin', пароль: 'Welcome1'}
```

Examples

Простая форма изменения пароля с проверкой нескольких элементов управления

В приведенных ниже примерах используется API новой формы, введенный в RC3.

PW-change.template.html

```
<form class="container" [formGroup]="pwChangeForm">
  <label for="current">Current Password</label>
  <input id="current" formControlName="current" type="password" required><br />

  <label for="newPW">New Password</label>
  <input id="newPW" formControlName="newPW" type="password" required><br />
```

```

<div *ngIf="newPW.touched && newPW.newIsNotOld">
  New password can't be the same as current password.
</div>

<label for="confirm">Confirm new password</label>
<input id="confirm" formControlName="confirm" type="password" required><br />
<div *ngIf="confirm.touched && confirm.errors.newMatchesConfirm">
  The confirmation does not match.
</div>

<button type="submit">Submit</button>
</form>

```

PW-change.component.ts

```

import {Component} from '@angular/core'
import {REACTIVE_FORM_DIRECTIVES, FormBuilder, AbstractControl, FormGroup,
  Validators} from '@angular/forms'
import {PWChangeValidators} from './pw-validators'

@Component({
  moduleId: module.id
  selector: 'pw-change-form',
  templateUrl: './pw-change.template.html`,
  directives: [REACTIVE_FORM_DIRECTIVES]
})

export class PWChangeFormComponent {
  pwChangeForm: FormGroup;

  // Properties that store paths to FormControls makes our template less verbose
  current: AbstractControl;
  newPW: AbstractControl;
  confirm: AbstractControl;

  constructor(private fb: FormBuilder) { }
  ngOnInit() {
    this.pwChangeForm = this.fb.group({
      current: ['', Validators.required],
      newPW: ['', Validators.required],
      confirm: ['', Validators.required]
    }, {
      // Here we create validators to be used for the group as a whole
      validator: Validators.compose([
        PWChangeValidators.newIsNotOld,
        PWChangeValidators.newMatchesConfirm
      ])
    });
    this.current = this.pwChangeForm.controls['current'];
    this.newPW = this.pwChangeForm.controls['newPW'];
    this.confirm = this.pwChangeForm.controls['confirm'];
  }
}

```

PW-validators.ts

```

import {FormControl, FormGroup} from '@angular/forms'
export class PWChangeValidators {

  static OldPasswordMustBeCorrect(control: FormControl) {
    var invalid = false;
    if (control.value != PWChangeValidators.oldPW)
      return { oldPasswordMustBeCorrect: true }
    return null;
  }

  // Our cross control validators are below
  // NOTE: They take in type FormGroup rather than FormControl
  static newIsNotOld(group: FormGroup){
    var newPW = group.controls['newPW'];
    if(group.controls['current'].value == newPW.value)
      newPW.setErrors({ newIsNotOld: true });
    return null;
  }

  static newMatchesConfirm(group: FormGroup){
    var confirm = group.controls['confirm'];
    if(group.controls['newPW'].value != confirm.value)
      confirm.setErrors({ newMatchesConfirm: true });
    return null;
  }
}

```

Сущность включая некоторые классы начальной загрузки можно найти [здесь](#) .

Угловые 2: шаблонные управляемые формы

```

import { Component } from '@angular/core';
import { Router , ROUTER_DIRECTIVES} from '@angular/router';
import { NgForm } from '@angular/forms';

@Component({
  selector: 'login',
  template: `
<h2>Login</h2>
<form #f="ngForm" (ngSubmit)="login(f.value,f.valid)" novalidate>
  <div>
    <label>Username</label>
    <input type="text" [(ngModel)]="username" placeholder="enter username" required>
  </div>
  <div>
    <label>Password</label>
    <input type="password" name="password" [(ngModel)]="password" placeholder="enter
password" required>
  </div>
  <input class="btn-primary" type="submit" value="Login">
</form>`
  //For long form we can use **templateUrl** instead of template
})

export class LoginComponent{

  constructor(private router : Router){ }

  login (formValue: any, valid: boolean){

```

```
    console.log(formValue);

    if (valid) {
        console.log(valid);
    }
}
}
```

Угловая форма 2 - пользовательская проверка электронной почты / пароля

Для демонстрации в реальном времени [нажмите ..](#)

Индекс приложений ts

```
import {bootstrap} from '@angular/platform-browser-dynamic';
import {MyForm} from './my-form.component.ts';

bootstrap(MyForm);
```

Пользовательский валидатор

```
import {Control} from '@angular/common';

export class CustomValidators {
    static emailFormat(control: Control): [[key: string]: boolean] {
        let pattern: RegExp = /\S+@\S+\.\S+/;
        return pattern.test(control.value) ? null : {"emailFormat": true};
    }
}
```

Компоненты формы ts

```
import {Component} from '@angular/core';
import {FORM_DIRECTIVES, NgForm, FormBuilder, Control, ControlGroup, Validators} from '@angular/common';
import {CustomValidators} from './custom-validators';

@Component({
    selector: 'my-form',
    templateUrl: 'app/my-form.component.html',
    directives: [FORM_DIRECTIVES],
    styleUrls: ['styles.css']
})
export class MyForm {
    email: Control;
    password: Control;
    group: ControlGroup;

    constructor(builder: FormBuilder) {
        this.email = new Control('',
            Validators.compose([Validators.required, CustomValidators.emailFormat])
        );

        this.password = new Control('',
```

```

    Validators.compose([Validators.required, Validators.minLength(4)])
  );

  this.group = builder.group({
    email: this.email,
    password: this.password
  });
}

onSubmit() {
  console.log(this.group.value);
}
}

```

Компоненты формы HTML

```

<form [ngFormModel]="group" (ngSubmit)="onSubmit()" novalidate>

  <div>
    <label for="email">Email:</label>
    <input type="email" id="email" [ngFormControl]="email">

    <ul *ngIf="email.dirty && !email.valid">
      <li *ngIf="email.hasError('required')">An email is required</li>
    </ul>
  </div>

  <div>
    <label for="password">Password:</label>
    <input type="password" id="password" [ngFormControl]="password">

    <ul *ngIf="password.dirty && !password.valid">
      <li *ngIf="password.hasError('required')">A password is required</li>
      <li *ngIf="password.hasError('minlength')">A password needs to have at least 4
characters</li>
    </ul>
  </div>

  <button type="submit">Register</button>

</form>

```

Угловая 2: реактивные формы (также называемые образцовыми формами)

В этом примере используется Angular 2.0.0 Final Release

Учетно-form.component.ts

```

import { FormGroup,
  FormControl,
  FormBuilder,
  Validators } from '@angular/forms';

@Component({

```



```

    templateUrl: "./registration-form.html"
  })
  export class ExampleComponent {
    constructor(private _fb: FormBuilder) { }

    exampleForm = this._fb.group({
      name: ['DefaultValue', [<any>Validators.required, <any>Validators.minLength(2)]],
      email: ['default@defa.ult', [<any>Validators.required, <any>Validators.minLength(2)]]
    })
  }

```

учетно-form.html

```

<form [formGroup]="exampleForm" novalidate (ngSubmit)="submit(exampleForm)">
  <label>Name: </label>
  <input type="text" formControlName="name"/>
  <label>Email: </label>
  <input type="email" formControlName="email"/>
  <button type="submit">Submit</button>
</form>

```

Угловые 2 формы (реактивные формы) с регистрационной формой и подтверждение пароля

app.module.ts

Добавьте их в свой файл app.module.ts для использования реактивных форм

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule,
  ],
  declarations: [
    AppComponent
  ]
  providers: [],
  bootstrap: [
    AppComponent
  ]
})
export class AppModule {}

```

app.component.ts

```

import { Component, OnInit } from '@angular/core';
import template from './add.component.html';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';

```

```

import { matchingPasswords } from './validators';
@Component({
  selector: 'app',
  template
})
export class AppComponent implements OnInit {
  addForm: FormGroup;
  constructor(private FormBuilder: FormBuilder) {
  }
  ngOnInit() {

    this.addForm = this.formBuilder.group({
      username: ['', Validators.required],
      email: ['', Validators.required],
      role: ['', Validators.required],
      password: ['', Validators.required],
      password2: ['', Validators.required] },
      { validator: matchingPasswords('password', 'password2')
    })
  };

  addUser() {
    if (this.addForm.valid) {
      var adduser = {
        username: this.addForm.controls['username'].value,
        email: this.addForm.controls['email'].value,
        password: this.addForm.controls['password'].value,
        profile: {
          role: this.addForm.controls['role'].value,
          name: this.addForm.controls['username'].value,
          email: this.addForm.controls['email'].value
        }
      };
      console.log(adduser); // adduser var contains all our form values. store it where you
want
      this.addForm.reset(); // this will reset our form values to null
    }
  }
}

```

app.component.html

```

<div>
  <form [formGroup]="addForm">
    <input type="text" placeholder="Enter username" formControlName="username" />
    <input type="text" placeholder="Enter Email Address" formControlName="email"/>
    <input type="password" placeholder="Enter Password" formControlName="password" />
    <input type="password" placeholder="Confirm Password" name="password2"
formControlName="password2"/>
    <div class='error' *ngIf="addForm.controls.password2.touched">
      <div class="alert-danger errorMessageadduser"
*ngIf="addForm.hasError('mismatchedPasswords') "> Passwords do
not match
    </div>
  </div>
</div>
<select name="Role" formControlName="role">
  <option value="admin" >Admin</option>
  <option value="Accounts">Accounts</option>
  <option value="guest">Guest</option>

```

```

</select>
<br/>
<br/>
<button type="submit" (click)="addUser()"><span><i class="fa fa-user-plus" aria-
hidden="true"></i></span> Add User </button>
</form>
</div>

```

validators.ts

```

export function matchingPasswords(passwordKey: string, confirmPasswordKey: string) {
  return (group: ControlGroup): {
    [key: string]: any
  } => {
    let password = group.controls[passwordKey];
    let confirmPassword = group.controls[confirmPasswordKey];

    if (password.value !== confirmPassword.value) {
      return {
        mismatchedPasswords: true
      };
    }
  }
}

```

Angular2 - Form Builder

FormComponent.ts

```

import {Component} from "@angular/core";
import {FormBuilder} from "@angular/forms";

@Component({
  selector: 'app-form',
  templateUrl: './form.component.html',
  styleUrls: ['./form.component.scss'],
  providers : [FormBuilder]
})

export class FormComponent{
  form : FormGroup;
  emailRegex = /^^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;

  constructor(fb: FormBuilder) {

    this.form = fb.group({
      FirstName : new FormControl({value: null}, Validators.compose([Validators.required,
Validators.maxLength(15)])),
      LastName : new FormControl({value: null}, Validators.compose([Validators.required,
Validators.maxLength(15)])),
      Email : new FormControl({value: null}, Validators.compose([
Validators.required,
Validators.maxLength(15),
Validators.pattern(this.emailRegex)]))
    });
  }
}

```

form.component.html

```
<form class="form-details" role="form" [formGroup]="form">
  <div class="row input-label">
    <label class="form-label" for="FirstName">First name</label>
    <input
      [formControl]="form.controls['FirstName']"
      type="text"
      class="form-control"
      id="FirstName"
      name="FirstName">
  </div>
  <div class="row input-label">
    <label class="form-label" for="LastName">Last name</label>
    <input
      [formControl]="form.controls['LastName']"
      type="text"
      class="form-control"
      id="LastName"
      name="LastName">
  </div>
  <div class="row">
    <label class="form-label" for="Email">Email</label>
    <input
      [formControl]="form.controls['Email']"
      type="email"
      class="form-control"
      id="Email"
      name="Email">
  </div>
  <div class="row">
    <button
      (click)="submit()"
      role="button"
      class="btn btn-primary submit-btn"
      type="button"
      [disabled]="!form.valid">Submit</button>
  </div>
</div>
</form>
```

Прочитайте Угловое обновление 2-х форм онлайн:

<https://riptutorial.com/ru/angular2/topic/4607/угловое-обновление-2-х-форм>

глава 60: угловое покрытие

Вступление

тестовое покрытие определяется как метод, который определяет, действительно ли наши тестовые примеры охватывают код приложения и сколько кода выполняется, когда мы запускаем эти тестовые примеры.

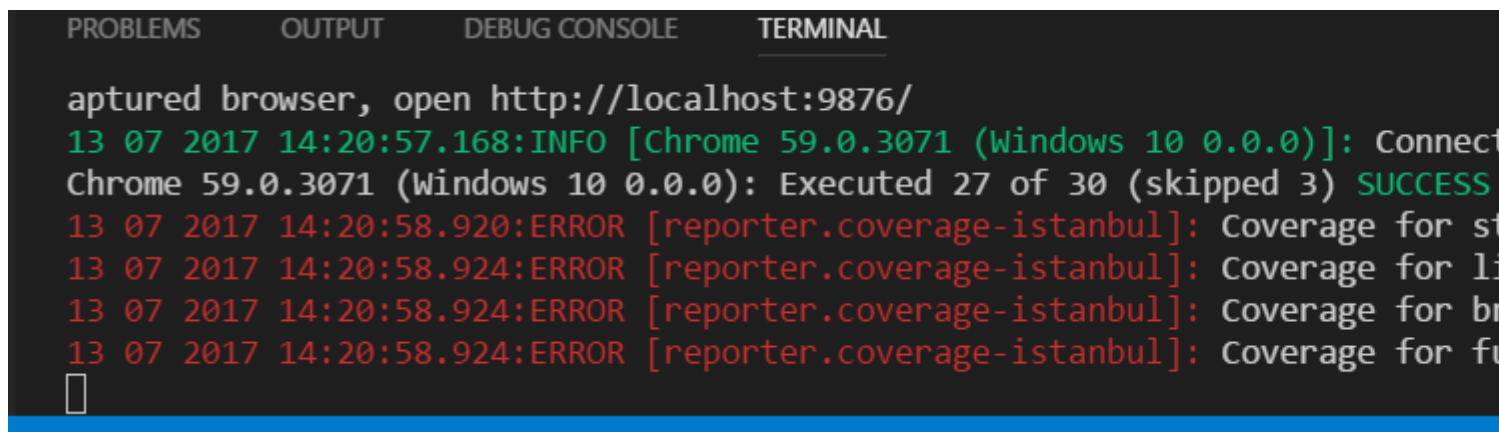
Угловой CLI имеет встроенную функцию покрытия кода с помощью простой команды `ng test --cc`

Examples

Простое тестовое покрытие командной строки с угловым кли

Если вы хотите видеть общую статистику охвата тестирования, чем, конечно, в Angular CLI, вы можете просто ввести команду ниже и увидеть нижнюю часть окна командной строки для получения результатов.

```
ng test --cc // or --code-coverage
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
aptured browser, open http://localhost:9876/
13 07 2017 14:20:57.168:INFO [Chrome 59.0.3071 (Windows 10 0.0.0)]: Connect
Chrome 59.0.3071 (Windows 10 0.0.0): Executed 27 of 30 (skipped 3) SUCCESS
13 07 2017 14:20:58.920:ERROR [reporter.coverage-istanbul]: Coverage for st
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for li
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for br
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for fu
```

Подробная отчетность по графическому тестированию на основе отдельных компонентов

если вы хотите увидеть индивидуальный охват тестов компонентом, выполните следующие действия.

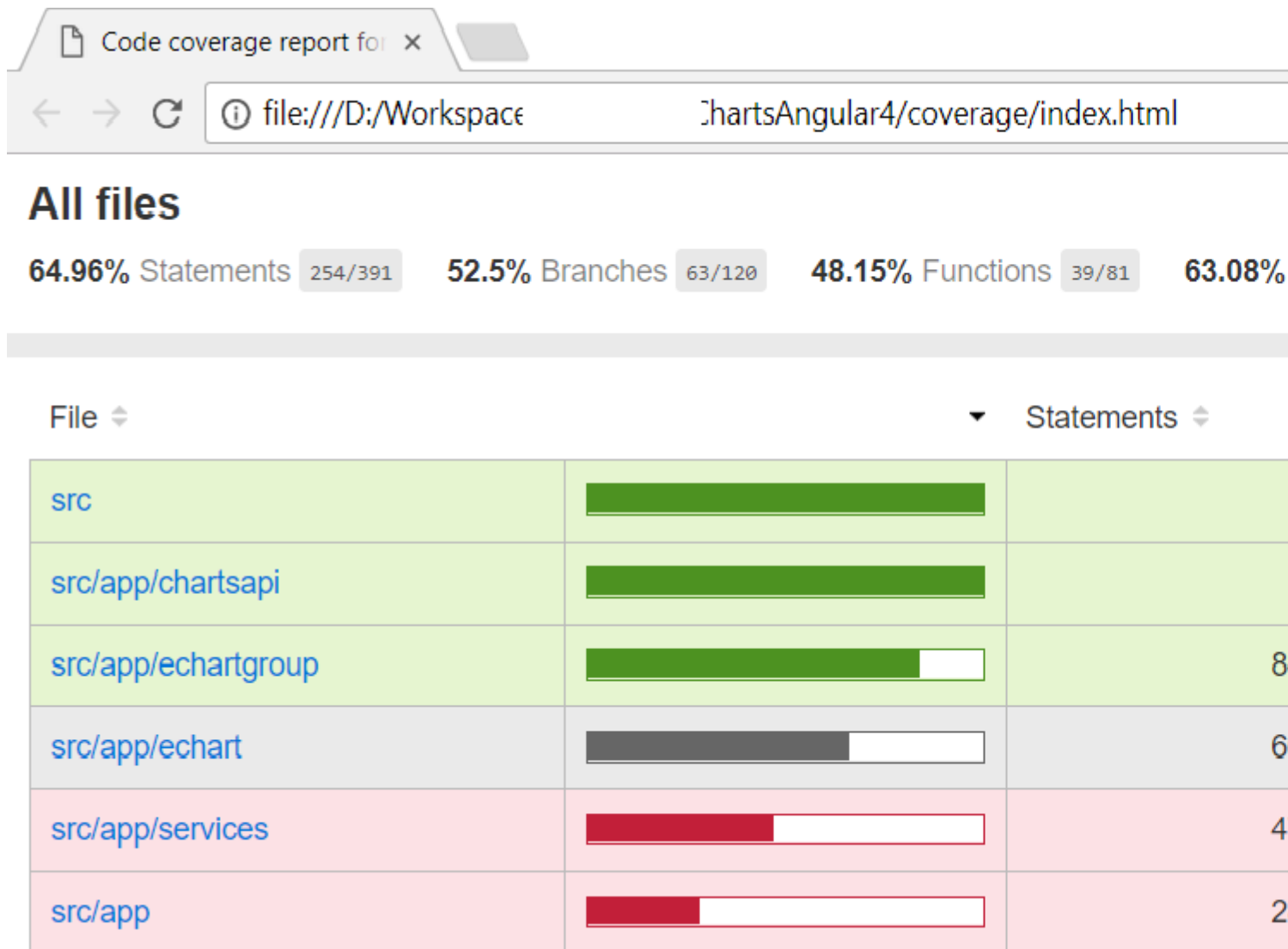
1. `npm install --save-dev karma-teamcity-reporter`

2. Add `require('karma-teamcity-reporter')` to list of plugins in `karma.conf.js`

3. `ng test --code-coverage --reporters=teamcity,coverage-istanbul`

обратите внимание, что список репортеров разделен запятыми, так как мы добавили нового репортера, `teamcity`.

после запуска этой команды вы можете увидеть `coverage` папки в своем каталоге и открыть `index.html` для графического представления тестового покрытия.



Вы также можете установить порог покрытия, который вы хотите достичь, в `karma.conf.js`, как это.

```
coverageIstanbulReporter: {
  reports: ['html', 'lcovonly'],
  fixWebpackSourcePaths: true,
  thresholds: {
    statements: 90,
    lines: 90,
    branches: 90,
    functions: 90
  }
},
```

Прочитайте угловое покрытие онлайн: <https://riptutorial.com/ru/angular2/topic/10764/угловое-покрытие>

глава 61: угловое сокращение

Examples

ОСНОВНОЙ

app.module.ts

```
import {appStoreProviders} from "../app.store";
providers : [
  ...
  appStoreProviders,
  ...
]
```

app.store.ts

```
import {InjectionToken} from '@angular/core';
import {createStore, Store, compose, StoreEnhancer} from 'redux';
import {AppState, default as reducer} from "../app.reducer";

export const AppStore = new InjectionToken('App.store');

const devtools: StoreEnhancer<AppState> =
  window['devToolsExtension'] ?
  window['devToolsExtension']() : f => f;

export function createAppStore(): Store<AppState> {
  return createStore<AppState>(
    reducer,
    compose(devtools)
  );
}

export const appStoreProviders = [
  {provide: AppStore, useFactory: createAppStore}
];
```

app.reducer.ts

```
export interface AppState {
  example : string
}

const rootReducer: Reducer<AppState> = combineReducers<AppState>({
  example : string
});

export default rootReducer;
```

store.ts

```

export interface IAppState {
  example?: string;
}

export const INITIAL_STATE: IAppState = {
  example: null,
};

export function rootReducer(state: IAppState = INITIAL_STATE, action: Action): IAppState {
  switch (action.type) {
    case EXAMPLE_CHANGED:
      return Object.assign(state, state, (<UpdateAction>action));
    default:
      return state;
  }
}

```

actions.ts

```

import {Action} from "redux";
export const EXAMPLE_CHANGED = 'EXAMPLE CHANGED';

export interface UpdateAction extends Action {
  example: string;
}

```

Получить текущее состояние

```

import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
export class exampleService {
  constructor(@Inject(AppStore) private store: Redux.Store<AppState>) {}
  getExampleState() {
    console.log(this.store.getState().example);
  }
}

```

ИЗМЕНИТЬ СОСТОЯНИЕ

```

import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
export class exampleService {
  constructor(@Inject(AppStore) private store: Redux.Store<AppState>) {}
  setExampleState() {
    this.store.dispatch(updateExample("new value"));
  }
}

```

actions.ts

```

export interface UpdateExapleAction extends Action {

```



```
    example?: string;
  }

export const updateExample: ActionCreator<UpdateExapleAction> =
  (newVal) => ({
    type: EXAMPLE_CHANGED,
    example: newVal
  });
```

Добавить редуционный хром-инструмент

app.store.ts

```
import {InjectionToken} from '@angular/core';
import {createStore, Store, compose, StoreEnhancer} from 'redux';
import {AppState, default as reducer} from "../app.reducer";

export const AppStore = new InjectionToken('App.store');

const devtools: StoreEnhancer<AppState> =
  window['devToolsExtension'] ?
  window['devToolsExtension']() : f => f;

export function createAppStore(): Store<AppState> {
  return createStore<AppState>(
    reducer,
    compose(devtools)
  );
}

export const appStoreProviders = [
  {provide: AppStore, useFactory: createAppStore}
];
```

установить хромовое расширение Redux DevTools

Прочитайте угловое сокращение онлайн: <https://riptutorial.com/ru/angular2/topic/10652/угловое-сокращение>

глава 62: Угловой 2 - транспортир

Examples

Тестирование маршрутизации Navbar с помощью Protractor

Сначала давайте создадим базовый файл navbar.html с тремя параметрами. (Главная, Список, Создать)

```
<nav class="navbar navbar-default" role="navigation">
<ul class="nav navbar-nav">
  <li>
    <a id="home-navbar" routerLink="/home">Home</a>
  </li>
  <li>
    <a id="list-navbar" routerLink="/create" >List</a>
  </li>
  <li>
    <a id="create-navbar" routerLink="/create">Create</a>
  </li>
</ul>
```

второй позволяет создать navbar.e2e-spec.ts

```
describe('Navbar', () => {

  beforeEach(() => {
    browser.get('home'); // before each test navigate to home page.
  });

  it('testing Navbar', () => {
    browser.sleep(2000).then(function() {
      checkNavbarTexts();
      navigateToListPage();
    });
  });

  function checkNavbarTexts() {
    element(by.id('home-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('Home');
    });

    element(by.id('list-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('List');
    });

    element(by.id('create-navbar')).getText().then(function(text) { // Promise
      expect(text).toEqual('Create');
    });
  }

  function navigateToListPage() {
    element(by.id('list-home')).click().then(function() { // first find list-home a tag and
      than click
    });
  }
});
```

```
        browser.sleep(2000).then(function() {
            browser.getCurrentUrl().then(function(actualUrl) { // promise
                expect(actualUrl.indexOf('list') !== -1).toBeTruthy(); // check the current url is
list
                });
            });

        });
    }
});
```

Угловой2 Транспорт - Установка

выполните следующие команды в cmd

- `npm install -g protractor`
- `webdriver-manager update`
- `webdriver-manager start`

**** создайте файл `protractor.conf.js` в главном корне приложения.**

очень важно убрать `useAllAngular2AppRoots: true`

```
const config = {
  baseUrl: 'http://localhost:3000/',

  specs: [
    './dev/**/*.e2e-spec.js'
  ],

  exclude: [],
  framework: 'jasmine',

  jasmineNodeOpts: {
    showColors: true,
    isVerbose: false,
    includeStackTrace: false
  },

  directConnect: true,

  capabilities: {
    browserName: 'chrome',
    shardTestFiles: false,
    chromeOptions: {
      'args': ['--disable-web-security', '--no-sandbox', 'disable-extensions', 'start-maximized', 'enable-crash-reporter-for-testing']
    }
  },

  onPrepare: function() {
    const SpecReporter = require('jasmine-spec-reporter');
    // add jasmine spec reporter
    jasmine.getEnv().addReporter(new SpecReporter({ displayStacktrace: true }));

    browser.ignoreSynchronization = false;
  },
  useAllAngular2AppRoots: true
}
```

```
};

if (process.env.TRAVIS) {
  config.capabilities = {
    browserName: 'firefox'
  };
}

exports.config = config;
```

создайте базовый тест в каталоге dev.

```
describe('basic test', () => {

  beforeEach(() => {
    browser.get('http://google.com');
  });

  it('testing basic test', () => {
    browser.sleep(2000).then(function() {
      browser.getCurrentUrl().then(function(actualUrl) {
        expect(actualUrl.indexOf('google') !== -1).toBeTruthy();
      });
    });
  });
});
```

запустить в cmd

```
protractor conf.js
```

Прочитайте Угловой 2 - транспорт онлайн: <https://riptutorial.com/ru/angular2/topic/8900/угловой-2---транспорт>

глава 63: Угловой 2 Обнаружение изменений и ручной запуск

Examples

Основной пример

Родительский компонент:

```
import {Component} from '@angular/core';

@Component({
  selector: 'parent-component',
  templateUrl: './parent-component.html'
})
export class ParentComponent {
  users : Array<User> = [];
  changeUsersActivation(user : User){
    user.changeButtonState();
  }
  constructor(){
    this.users.push(new User('Narco', false));
    this.users.push(new User('Bombasto', false));
    this.users.push(new User('Celeritas', false));
    this.users.push(new User('Magneta', false));
  }
}

export class User {
  firstName : string;
  active : boolean;

  changeButtonState(){
    this.active = !this.active;
  }
  constructor(_firstName :string, _active : boolean){
    this.firstName = _firstName;
    this.active = _active;
  }
}
```

Родительский HTML:

```
<div>
  <child-component [usersDetails]="users"
    (changeUsersActivation)="changeUsersActivation($event)">
  </child-component>
</div>
```

ДОЧЕРНИЙ КОМПОНЕНТ:

```
import {Component, Input, EventEmitter, Output} from '@angular/core';
import {User} from "../parent.component";

@Component({
  selector: 'child-component',
  templateUrl: './child-component.html',
  styles: [`
    .btn {
      height: 30px;
      width: 100px;
      border: 1px solid rgba(0, 0, 0, 0.33);
      border-radius: 3px;
      margin-bottom: 5px;
    }

  `]
})
export class ChildComponent {
  @Input() usersDetails : Array<User> = null;
  @Output() changeUsersActivation = new EventEmitter();

  triggerEvent(user : User) {
    this.changeUsersActivation.emit(user);
  }
}
```

child HTML:

```
<div>
  <div>
    <table>
      <thead>
        <tr>
          <th>Name</th>
          <th></th>
        </tr>
      </thead>
      <tbody *ngIf="user !== null">
        <tr *ngFor="let user of usersDetails">
          <td>{{user.firstName}}</td>
          <td><button class="btn" (click)="triggerEvent(user)">{{user.active}}</button></td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

Прочитайте [Угловой 2 Обнаружение изменений и ручной запуск онлайн:](https://riptutorial.com/ru/angular2/topic/8971/угловой-2-обнаружение-изменений-и-ручной-запуск)

<https://riptutorial.com/ru/angular2/topic/8971/угловой-2-обнаружение-изменений-и-ручной-запуск>

глава 64: Угловые 2 управляемые данными формы

замечания

```
this.myForm = this.formBuilder.group
```

создает объект формы с конфигурацией пользователя и присваивает его переменной `this.myForm`.

```
'loginCredentials': this.formBuilder.group
```

метод создает группу элементов управления, которые состоят из **formControlName**, например. `login` и `value` `['', Validators.required]`, где первым параметром является начальное значение ввода формы, а `secons` - это валидатор или массив валидаторов, как в `'email': ['', [Validators.required, customValidator]]`,

```
'hobbies': this.formBuilder.array
```

Создает массив групп, где индекс группы является **formGroupName** в массиве и доступен как:

```
<div *ngFor="let hobby of myForm.find('hobbies').controls; let i = index">
  <div formGroupName="{{i}}">...</div>
</div>
```

```
onAddHobby() {
  (<FormArray>this.myForm.find('hobbies')).push(new FormGroup({
    'hobby': new FormControl('', Validators.required)
  }))
}
```

этот примерный метод добавляет новую форму `groupGroup` в массив. В настоящее время для доступа требуется указать тип элемента управления, К `<FormArray>` мы хотим получить доступ, в этом примере этот тип: `<FormArray>`

```
removeHobby(index: number) {
  (<FormArray>this.myForm.find('hobbies')).removeAt(index);
}
```

те же правила, что и выше, применяются для удаления определенного элемента управления формы из массива

Examples

Форма данных

Составная часть

```
import {Component, OnInit} from '@angular/core';
import {
  FormGroup,
  FormControl,
  FORM_DIRECTIVES,
  REACTIVE_FORM_DIRECTIVES,
  Validators,
  FormBuilder,
  FormArray
} from "@angular/forms";
import {Control} from "@angular/common";

@Component({
  moduleId: module.id,
  selector: 'app-data-driven-form',
  templateUrl: 'data-driven-form.component.html',
  styleUrls: ['data-driven-form.component.css'],
  directives: [FORM_DIRECTIVES, REACTIVE_FORM_DIRECTIVES]
})
export class DataDrivenFormComponent implements OnInit {
  myForm: FormGroup;

  constructor(private formBuilder: FormBuilder) {}

  ngOnInit() {
    this.myForm = this.formBuilder.group({
      'loginCredentials': this.formBuilder.group({
        'login': ['', Validators.required],
        'email': ['', [Validators.required, customValidator]],
        'password': ['', Validators.required]
      }),
      'hobbies': this.formBuilder.array([
        this.formBuilder.group({
          'hobby': ['', Validators.required]
        })
      ])
    });
  }

  removeHobby(index: number) {
    (<FormArray>this.myForm.find('hobbies')).removeAt(index);
  }

  onAddHobby() {
    (<FormArray>this.myForm.find('hobbies')).push(new FormGroup({
      'hobby': new FormControl('', Validators.required)
    }));
  }

  onSubmit() {
    console.log(this.myForm.value);
  }
}
```



```

}

function customValidator(control: Control): {[s: string]: boolean} {
  if(!control.value.match("[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&'*/+=?^_`{|}~-]+)+*(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)")) {
    return {error: true}
  }
}
}

```

Разметка HTML

```

<h3>Register page</h3>
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
  <div formGroupName="loginCredentials">
    <div class="form-group">
      <div>
        <label for="login">Login</label>
        <input id="login" type="text" class="form-control" formControlName="login">
      </div>
      <div>
        <label for="email">Email</label>
        <input id="email" type="text" class="form-control" formControlName="email">
      </div>
      <div>
        <label for="password">Password</label>
        <input id="password" type="text" class="form-control" formControlName="password">
      </div>
    </div>
  </div>
  <div class="row" >
    <div formGroupName="hobbies">
      <div class="form-group">
        <label>Hobbies array:</label>
        <div *ngFor="let hobby of myForm.find('hobbies').controls; let i = index">
          <div formGroupName="{{i}}">
            <input id="hobby_{{i}}" type="text" class="form-control" formControlName="hobby">
            <button *ngIf="myForm.find('hobbies').length > 1"
(click)="removeHobby(i)">x</button>
          </div>
        </div>
        <button (click)="onAddHobby()">Add hobby</button>
      </div>
    </div>
  </div>
  <button type="submit" [disabled]="!myForm.valid">Submit</button>
</form>

```

Прочитайте Угловые 2 управляемые данными формы онлайн:

<https://riptutorial.com/ru/angular2/topic/6463/угловые-2-управляемые-данными-формы>

глава 65: Угловые объекты RXJS и наблюдения с запросами API

замечания

Выполнение запросов API с помощью функции Angular 2 `Http` и RxJS очень похоже на работу с обещаниями в Angular 1.x.

Для выполнения запросов используйте класс `Http`. Класс `Http` предоставляет методы для выдачи HTTP-запросов `GET`, `POST`, `PUT`, `DELETE`, `PATCH`, `HEAD` помощью соответствующих методов. Он также предоставляет общий метод `request` для выдачи любого HTTP-запроса.

Все методы класса `Http` возвращают `Observable<Response>`, к которым вы можете применить операции RxJS. Вы вызываете метод `.subscribe()` и передаете функцию, которая будет вызываться, когда данные возвращаются в потоке `Observable`.

Поток `Observable` для запроса содержит только одно значение - `Response` и завершается / устанавливается, когда HTTP-запрос завершается успешно или ошибки / ошибки при возникновении ошибки.

Обратите внимание, что наблюдаемые, возвращаемые модулем `Http` являются *ХОЛОДНЫМИ*, что означает, что если вы подписываетесь на наблюдаемое несколько раз, исходный запрос будет *выполняться* один раз для каждой подписки. Это может произойти, если вы хотите использовать результат в нескольких компонентах вашего приложения. Для запросов `GET` это может привести к некоторым дополнительным запросам, но это может привести к неожиданным результатам, если подписываться более одного раза на запросы `PUT` или `POST`.

Examples

Основной запрос

Следующий пример демонстрирует простой HTTP-запрос `GET`. `http.get()` возвращает `Observable`, который имеет метод `subscribe`. Это добавляет возвращаемые данные в массив `posts`.

```
var posts = []

getPosts(http: Http):void {
  this.http.get(`https://jsonplaceholder.typicode.com/posts`)
    .map(response => response.json())
    .subscribe(post => posts.push(post));
}
```

Инкапсулирование запросов API

Может быть хорошей идеей инкапсулировать логику обработки HTTP в свой класс.

Следующий класс предоставляет метод для получения сообщений. Он вызывает метод `http.get()` и вызывает `.map` на возвращаемом `Observable` для преобразования объекта `Response` в объект `Post`.

```
import {Injectable} from "@angular/core";
import {Http, Response} from "@angular/http";

@Injectable()
export class BlogApi {

  constructor(private http: Http) {
  }

  getPost(id: number): Observable<Post> {
    return this.http.get(`https://jsonplaceholder.typicode.com/posts/${id}`)
      .map((response: Response) => {
        const srcData = response.json();
        return new Post(srcData)
      });
  }
}
```

Предыдущий пример использует класс `Post` для хранения возвращаемых данных, который может выглядеть следующим образом:

```
export class Post {
  userId: number;
  id: number;
  title: string;
  body: string;

  constructor(src: any) {
    this.userId = src && src.userId;
    this.id = src && src.id;
    this.title = src && src.title;
    this.body = src && src.body;
  }
}
```

Компонент теперь может использовать класс `BlogApi` для легкого извлечения данных `Post` из `BlogApi` к `BlogApi` класса `Http`.

Подождите несколько запросов

Один из распространенных сценариев - дождаться завершения ряда запросов до продолжения. Это может быть выполнено с использованием метода `forkJoin`.

В следующем примере `forkJoin` используется для вызова двух методов, возвращающих `Observables`. Обратный вызов, указанный в методе `.subscribe` будет вызываться, когда оба

Observables завершены. Параметры, предоставленные `.subscribe` соответствуют порядку, указанному в вызове `.forkJoin`. В этом случае сначала `posts` `tags`.

```
loadData() : void {
  Observable.forkJoin(
    this.blogApi.getPosts(),
    this.blogApi.getTags()
  ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
    this.posts = posts;
    this.tags = tags;
  }));
}
```

Прочитайте [Угловые объекты RXJS и наблюдения с запросами API онлайн](https://riptutorial.com/ru/angular2/topic/3577/угловые-объекты-rxjs-и-наблюдения-с-запросами-api-онлайн):

<https://riptutorial.com/ru/angular2/topic/3577/угловые-объекты-rxjs-и-наблюдения-с-запросами-api>

глава 66: Угловые2 Пользовательские проверки

параметры

параметр	описание
контроль	Это контроль, который проверяется. Обычно вам нужно будет увидеть, соответствует ли <code>control.value</code> некоторые критерии.

Examples

Примеры пользовательских валидаторов:

Угловой 2 имеет два вида пользовательских валидаторов. Синхронные валидаторы, как в первом примере, которые будут выполняться непосредственно на клиенте и асинхронные валидаторы (второй пример), которые вы можете использовать для вызова удаленной службы, чтобы выполнить валидацию для вас. В этом примере валидатор должен вызвать сервер, чтобы узнать, уникально ли значение.

```
export class CustomValidators {  
  
  static cannotContainSpace(control: Control) {  
    if (control.value.indexOf(' ') >= 0)  
      return { cannotContainSpace: true };  
  
    return null;  
  }  
  
  static shouldBeUnique(control: Control) {  
    return new Promise((resolve, reject) => {  
      // Fake a remote validator.  
      setTimeout(function () {  
        if (control.value == "exisitingUser")  
          resolve({ shouldBeUnique: true });  
        else  
          resolve(null);  
      }, 1000);  
    });  
  }  
}
```

Если ваше контрольное значение действительно, вы просто возвращаете `null` для вызывающего. В противном случае вы можете вернуть объект, который описывает ошибку.

Использование валидаторов в FormBuilder

```
constructor(fb: FormBuilder) {
  this.form = fb.group({
    firstInput: ['', Validators.compose([Validators.required,
CustomValidators.cannotContainSpace]), CustomValidators.shouldBeUnique],
    secondInput: ['', Validators.required]
  });
}
```

Здесь мы используем FormBuilder для создания очень простой формы с двумя полями ввода. FormBuilder берет массив для трех аргументов для каждого элемента управления вводом.

1. Значение по умолчанию элемента управления.
2. Валидаторы, которые будут выполняться на клиенте. Вы можете использовать Validators.compose ([arrayOfValidators]) для применения нескольких валидаторов в вашем элементе управления.
3. Один или несколько асинхронных валидаторов аналогично второму аргументу.

get / set FormBuilder управляет параметрами

Существует два способа установки параметров управления FormBuilder.

1. При инициализации:

```
exampleForm : FormGroup;
constructor(fb: FormBuilder){
  this.exampleForm = fb.group({
    name : new FormControl({value: 'default name'}, Validators.compose([Validators.required,
Validators.maxLength(15)]))
  });
}
```

2. После инициализации:

```
this.exampleForm.controls['name'].setValue('default name');
```

Получить управляющее значение FormBuilder:

```
let name = this.exampleForm.controls['name'].value();
```

Прочитайте [Угловые2 Пользовательские проверки онлайн](https://riptutorial.com/ru/angular2/topic/6284/угловые2-пользовательские-проверки):

<https://riptutorial.com/ru/angular2/topic/6284/угловые2-пользовательские-проверки>

глава 67: Услуги и зависимость от инъекций

Examples

Пример сервиса

услуги / my.service.ts

```
import { Injectable } from '@angular/core';

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return this.data;
  }
}
```

Регистрация поставщика услуг в методе начальной загрузки сделает службу доступной по всему миру.

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from 'app.component.ts';
import { MyService } from 'services/my.service';

bootstrap(AppComponent, [MyService]);
```

В версии RC5 регистрация глобального поставщика услуг может быть выполнена внутри файла модуля. Чтобы получить один экземпляр вашей службы для всего вашего приложения, служба должна быть объявлена в списке поставщиков в `ngmodule` вашего приложения. *app_module.ts*

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { routing, appRoutingProviders } from './app-routes/app.routes';
import { HttpClientModule } from '@angular/http';

import { AppComponent } from './app.component';
import { MyService } from 'services/my.service';

import { routing } from './app-resources/app-routes/app.routes';

@NgModule({
  declarations: [ AppComponent ],
  imports: [ BrowserModule,
```

```

        routing,
        RouterModule,
        HttpModule ],
    providers: [
        appRoutingProviders,
        MyService
    ],
    bootstrap: [AppComponent],
})
export class AppModule {}

```

Использование в `MyComponent`

компоненты / my.component.ts

Альтернативный подход к регистрации поставщиков приложений в компонентах приложения. Если мы добавим поставщиков на уровне компонента при визуализации компонента, он создаст новый экземпляр службы.

```

import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({
  ...
  providers:[MyService] //
})
export class MyComponent implements OnInit {
  data: any[];
  // Creates private variable myService to use, of type MyService
  constructor(private myService: MyService) { }

  ngOnInit() {
    this.data = this.myService.getData();
  }
}

```

Пример с `Promise.resolve`

услуги / my.service.ts

```

import { Injectable } from '@angular/core';

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return Promise.resolve(this.data);
  }
}

```

Теперь `getData()` действует как вызов REST, создающий `Promise`, который немедленно разрешается. Результаты могут быть перенесены внутри `.then()` и ошибки также могут быть обнаружены. Это хорошая практика и соглашение для асинхронных методов.

КОМПОНЕНТЫ / *my.component.ts*

```
import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({...})
export class MyComponent implements OnInit {
  data: any[];
  // Creates private variable myService to use, of type MyService
  constructor(private myService: MyService) { }

  ngOnInit() {
    // Uses an "arrow" function to set data
    this.myService.getData().then(data => this.data = data);
  }
}
```

Тестирование службы

Для службы, которая может войти в систему пользователем:

```
import 'rxjs/add/operator/toPromise';

import { Http } from '@angular/http';
import { Injectable } from '@angular/core';

interface LoginCredentials {
  password: string;
  user: string;
}

@Injectable()
export class AuthService {
  constructor(private http: Http) { }

  async signIn({ user, password }: LoginCredentials) {
    const response = await this.http.post('/login', {
      password,
      user,
    }).toPromise();

    return response.json();
  }
}
```

Его можно протестировать следующим образом:

```
import { ConnectionBackend, Http, HttpModule, Response, ResponseOptions } from
 '@angular/http';
import { TestBed, async, inject } from '@angular/core/testing';

import { AuthService } from './auth.service';
import { MockBackend } from '@angular/http/testing';
import { MockConnection } from '@angular/http/testing';

describe('AuthService', () => {
  beforeEach(() => {
```

```

TestBed.configureTestingModule({
  imports: [HttpModule],
  providers: [
    AuthService,
    Http,
    { provide: ConnectionBackend, useClass: MockBackend },
  ]
});
});

it('should be created', inject([AuthService], (service: AuthService) => {
  expect(service).toBeTruthy();
}));

// Alternative 1
it('should login user if right credentials are passed', async(
  inject([AuthService], async (authService) => {
    const backend: MockBackend = TestBed.get(ConnectionBackend);
    const http: Http = TestBed.get(Http);

    backend.connections.subscribe((c: MockConnection) => {
      c.mockRespond(
        new Response(
          new ResponseOptions({
            body: {
              accessToken: 'abcdef',
            },
          }),
        ),
      );
    });

    const result = await authService.signIn({ password: 'ok', user: 'bruno' });

    expect(result).toEqual({
      accessToken: 'abcdef',
    });
  })
);

// Alternative 2
it('should login user if right credentials are passed', async () => {
  const backend: MockBackend = TestBed.get(ConnectionBackend);
  const http: Http = TestBed.get(Http);

  backend.connections.subscribe((c: MockConnection) => {
    c.mockRespond(
      new Response(
        new ResponseOptions({
          body: {
            accessToken: 'abcdef',
          },
        }),
      ),
    );
  });

  const authService: AuthService = TestBed.get(AuthService);

  const result = await authService.signIn({ password: 'ok', user: 'bruno' });

```

```

    expect(result).toEqual({
      accessToken: 'abcdef',
    });
  });

// Alternative 3
it('should login user if right credentials are passed', async (done) => {
  const authService: AuthService = TestBed.get(AuthService);

  const backend: MockBackend = TestBed.get(ConnectionBackend);
  const http: Http = TestBed.get(Http);

  backend.connections.subscribe((c: MockConnection) => {
    c.mockRespond(
      new Response(
        new ResponseOptions({
          body: {
            accessToken: 'abcdef',
          },
        }),
      ),
    );
  });

  try {
    const result = await authService.signIn({ password: 'ok', user: 'bruno' });

    expect(result).toEqual({
      accessToken: 'abcdef',
    });

    done();
  } catch (err) {
    fail(err);
    done();
  }
});
});

```

Прочитайте Услуги и зависимость от инъекций онлайн:

<https://riptutorial.com/ru/angular2/topic/4187/услуги-и-зависимость-от-инъекций>

глава 68: Установка сторонних плагинов с помощью angular-cli@1.0.0-beta.10

замечания

Однако, возможно, установить другие библиотеки, этот подход, однако, может потребоваться указать тип модуля, основной файл и расширение по умолчанию.

```
'lodash': {
  format: 'cjs',
  defaultExtension: 'js',
  main: 'index.js'
}
```

```
'moment': {
  main: 'moment.js'
}
```

Examples

Добавление библиотеки jquery в проект angular-cli

1. Установите jquery через npm:

```
npm install jquery --save
```

Установите типизацию для библиотеки:

Чтобы добавить типизацию для библиотеки, выполните следующие действия:

```
typings install jquery --global --save
```

2. Добавьте файл jquery в файл angular-cli-build.js в массив vendorNpmFiles:

Это требуется, чтобы система сборки забирала файл. После настройки угловое-cli-build.js должно выглядеть так:

Просмотрите `node_modules` и найдите файлы и папки, которые вы хотите добавить в папку поставщика.

```
var Angular2App = require('angular-cli/lib/broccoli/angular2-app');
```

```
module.exports = function(defaults) {
  return new Angular2App(defaults, {
    vendorNpmFiles: [
      // ...
      'jquery/dist/*.js'

    ]
  });
};
```

3. Настройте сопоставления SystemJS, чтобы узнать, где искать jquery:

Конфигурация SystemJS находится в `system-config.ts`, и после выполнения пользовательской настройки соответствующий раздел должен выглядеть так:

```
/** Map relative paths to URLs. */
const map: any = {
  'jquery': 'vendor/jquery'
};

/** User packages configuration. */
const packages: any = {

// no need to add anything here for jquery

};
```

4. В вашем `src / index.html` добавьте эту строку

```
<script src="vendor/jquery/dist/jquery.min.js" type="text/javascript"></script>
```

Другие варианты:

```
<script src="vendor/jquery/dist/jquery.js" type="text/javascript"></script>
```

или же

```
<script src="/vendor/jquery/dist/jquery.slim.js" type="text/javascript"></script>
```

а также

```
<script src="/vendor/jquery/dist/jquery.slim.min.js" type="text/javascript"></script>
```

5. Импорт и использование библиотеки jquery в исходных файлах проекта:

Импортируйте библиотеку jquery в ваши исходные файлы `.ts`:

```

declare var $:any;

@Component({
})
export class YourComponent {
  ngOnInit() {
    $(".button").click(function(){
      // now you can DO, what ever you want
    });
    console.log();
  }
}

```

Если вы правильно выполнили шаги, теперь у вас должна быть библиотека jquery, работающая в вашем проекте. Наслаждайтесь!

Добавить стороннюю библиотеку, которая не имеет типов

Обратите внимание, что это только для angular-cli версии 1.0.0-beta.10!

Некоторые библиотеки или плагины могут не иметь типов. Без них TypeScript не может ввести их проверку и, следовательно, вызывает ошибки компиляции. Эти библиотеки все еще можно использовать, но не так, как импортированные модули.

1. Включите ссылку на скрипт в библиотеку на своей странице (index.html)

```

<script src="//cdn.somewhe.re/lib.min.js" type="text/javascript"></script>
<script src="/local/path/to/lib.min.js" type="text/javascript"></script>

```

- Эти скрипты должны добавлять глобальные (например, THREE , mapbox , \$ и т. Д.) Или присоединяться к глобальному

2. В компоненте, который требует их, используйте declare для инициализации переменной, соответствующей глобальному имени, используемому lib. Это позволяет TypeScript знать, что он уже был инициализирован. ¹

```

declare var <globalname>: any;

```

Некоторые библиотеки присоединяются к window , которые необходимо будет расширить, чтобы быть доступными в приложении.

```

interface WindowIntercom extends Window { Intercom: any; }
declare var window: WindowIntercom;

```

3. Используйте lib в своих компонентах по мере необходимости.

```

@Component { ... }
export class AppComponent implements AfterViewInit {
  ...
  ngAfterViewInit() {

```

```
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
window.Intercom('boot', { ... }
}
}
```

- **ПРИМЕЧАНИЕ.** Некоторые библиотеки могут взаимодействовать с DOM и должны использоваться в соответствующем методе [ЖИЗНЕННОГО ЦИКЛА КОМПОНЕНТА](#) .

Прочитайте [Установка сторонних плагинов с помощью angular-cli@1.0.0-beta.10](#) онлайн:
<https://riptutorial.com/ru/angular2/topic/2328/установка-сторонних-плагинов-с-помощью-angular-cli-1-0-0-beta-10>

глава 69: Функциональные модули

Examples

Функциональный модуль

```
// my-feature.module.ts
import { CommonModule } from '@angular/common';
import { NgModule }      from '@angular/core';

import { MyComponent } from './my.component';
import { MyDirective } from './my.directive';
import { MyPipe }      from './my.pipe';
import { MyService }   from './my.service';

@NgModule({
  imports:      [ CommonModule ],
  declarations: [ MyComponent, MyDirective, MyPipe ],
  exports:     [ MyComponent ],
  providers:   [ MyService ]
})
export class MyFeatureModule { }
```

Теперь в вашем корне (обычно `app.module.ts`):

```
// app.module.ts
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent }  from './app.component';
import { MyFeatureModule } from './my-feature.module';

@NgModule({
  // import MyFeatureModule in root module
  imports:      [ BrowserModule, MyFeatureModule ],
  declarations: [ AppComponent ],
  bootstrap:   [ AppComponent ]
})
export class AppModule { }
```

Прочитайте Функциональные модули онлайн: <https://riptutorial.com/ru/angular2/topic/6551/функциональные-модули>

глава 70: Шаблоны

Вступление

Шаблоны очень похожи на шаблоны в Angular 1, хотя есть много небольших синтаксических изменений, которые делают его более понятным, что происходит.

Examples

Угловые 2 шаблона

ПРОСТОЙ ШАБЛОН

Начнем с очень простого шаблона, который показывает наше имя и нашу любимую вещь:

```
<div>
  Hello my name is {{name}} and I like {{thing}} quite a lot.
</div>
```

{ }: ОКАЗАНИЕ

Чтобы отобразить значение, мы можем использовать стандартный двухслойный синтаксис:

```
My name is {{name}}
```

Трубы, ранее известные как «Фильтры», преобразуют значение в новое значение, например, локализуют строку или преобразуют значение с плавающей запятой в представление валюты:

[]: ОБЯЗАТЕЛЬНЫЕ СВОЙСТВА

Чтобы разрешить и привязать переменную к компоненту, используйте синтаксис []. Если у нас есть `this.currentVolume` в нашем компоненте, мы передадим это через наш компонент и значения останутся в синхронизации:

```
<video-control [volume]="currentVolume"></video-control>
(): HANDLING EVENTS
```

(): ОБРАБОТКА СОБЫТИЙ Для прослушивания события на компоненте мы используем синтаксис ()

```
<my-component (click)="onClick($event)"></my-component>
```

[()]: ДВОЙНАЯ ДАННАЯ СВЯЗЬ

Чтобы сохранить привязку к актуальным данным пользователя и другим событиям, используйте синтаксис `[(())]`. Подумайте об этом как о комбинации обработки события и привязки свойства:

`<input [(ngModel)] = "myName">` Значение `this.myName` вашего компонента будет оставаться в синхронизации с входным значением.

*** : THE ASTERISK**

Указывает, что эта директива рассматривает этот компонент как шаблон и не будет рисовать его как есть. Например, `ngFor` берет наши и печатает его для каждого элемента в элементах, но он никогда не отображает нашу начальную, так как это шаблон:

```
<my-component *ngFor="#item of items">
</my-component>
```

Другие аналогичные директивы, которые работают с шаблонами, а не с отображаемыми компонентами, - это `* ngIf` и `* ngSwitch`.

Прочитайте Шаблоны онлайн: <https://riptutorial.com/ru/angular2/topic/9471/шаблоны>

кредиты

S. No	Главы	Contributors
1	Начало работы с Angular 2	acdcjunior , Alexander Ciesielski , beagleknight , Bean0341 , Bhoomi Bhalani , BogdanC , briantylor , cDecker32 , Christopher Moore , Community , daniellmb , drbishop , echonax , Ekin Yücel , elliot-j , etayluz , ettanany , Everettss , H. Pauwelyn , Harry , He11ion , Janco Boscan , Jim , Kaspars Bergs , Logan H , Madhu Ranjan , michaelbahr , Michal Pietraszko , Mihai , nick , Nicolas Irisarri , Peter , QoP , rickysullivan , Shahzad , spike , theblindprophet , user6939352
2	Angular2 CanActivate	Companjo , Yoav Schniederman
3	Angular2 Databinding	Yoav Schniederman
4	Angular2 В интерфейсе веб-интерфейса Memory	Jaime Still
5	Angular2 предоставляет внешние данные для приложения перед загрузкой	Ajey
6	Angular2 с помощью webpack	luukgruijs
7	CRUD в Angular2 с остальным API	bleakgadfly , Sefa
8	Dropzone в Angular2	Ketan Akbari
9	Mocking @ ngrx / Магазин	BrianRT , Hatem , Jim , Lucas , Yoav Schniederman
10	ngrx	Maxime
11	Zone.js	Roope Hakulinen
12	Анимация	Gaurav Mukherjee , Nate May
13	бочка	TechJhola
14	Бутстрап Пустой модуль в угловом 2	AryanJ-NYC , autoboxer , Berseker59 , Eric Jimenez , Krishan , Sanket , snorkpete

15	Выход Angular2 Input () ()	Kaloyan , Yoav Schniederman
16	Динамически добавлять компоненты с помощью ViewContainerRef.createComponent	amansoni211 , daniellmb , Günter Zöchbauer , jupiter24 , Khaled
17	Директивы	acdcjunior , Andrei Zhytkevich , borislemke , BrunoLM , daniellmb , Everettss , lexith , Stian Standahl , theblindprophet
18	Директивы атрибутов влияют на значение свойств узла узла с помощью декоратора @HostBinding.	Max Karpovets
19	Директивы и компоненты: @Input @Output	acdcjunior , dafyddPrys , Everettss , Joel Almeida , lexith , muetzerich , theblindprophet , ThomasP1988
20	Заголовок страницы	Yoav Schniederman
21	Использование сторонних библиотек, таких как jQuery в Angular 2	Ashok Vishwakarma
22	Используйте собственные веб-компоненты в Angular 2	ugreen
23	Как использовать ngfor	Jorge , Yoav Schniederman
24	Как использовать ngif	Amit kumar , ob1 , ppovoski , samAlvin
25	Компиляция вовремя (AOT) с помощью Angular 2	Anil Singh , Eric Jimenez , Harry , Robin Dijkhof
26	Компонентные взаимодействия	H. Pauwelyn , Janco Boscan , LLL , Sefa
27	Компоненты	BrunoLM
28	Конструкция углового материала	Ketan Akbari , Shailesh Ladumor
29	Крючки жизненного цикла	Alexandre Junges , daniellmb , Deen John , muetzerich , Sbats , theblindprophet
30	Ленивая загрузка модуля	Batajus , M4R1KU , Shannon Young , Syam Pradeep
31	маршрутизация	aholtry , Apmis , AryanJ-NYC , borislemke , camwhite , Kaspars Bergs , LordTribual , Sachin S

		, theblindprophet
32	Маршрутизация (3.0.0+)	Ai_boy , Alexis Le Gal , Everettss , Gerard Simpson , Kaspars Bergs , mast3rd3mon , meorfi , rivanov , SlashTag , smnbbvr , theblindprophet , ThomasP1988 , Trent
33	Модернизация грубой силы	Jim , Treveshan Naidoo
34	Модули	BrunoLM
35	модульное тестирование	Yoav Schniederma
36	Настройка приложения ASP.net Core для работы с Angular 2 и TypeScript	Oleksii Aza , Sam
37	Обнаружение изменений размера	Eric Jimenez
38	Обновить титры	kEpEx
39	Обход Sanitizing для доверенных значений	Scrambo
40	Обычно встроенные директивы и службы	Jim , Sanket
41	Оптимизация рендеринга с помощью ChangeDetectionStrategy	daniellmb , Eric Jimenez , Everettss
42	Отладка приложений с расширением Angular2 с использованием кода Visual Studio	PSabuwala
43	Перехватчик Http	Everettss , Mihai , Mike Kovetsky , Nilz11 , Paul Marshall , peeskillet , theblindprophet
44	пользовательский ngx-bootstrap datepicker + input	Yoav Schniederma
45	Пример маршрутов, таких как / route / subroute для статических URL-адресов	Yoav Schniederma
46	Примеры расширенных компонентов	borislemke , smnbbvr

47	Радиально-кли	BogdanC , Yoav Schniederman
48	Сервисный рабочий	Roberto Fernandez
49	Служба EventEmitter	Abrar Jahin
50	Создайте пакет с угловым 2+ NPM	BogdanC , Janco Boscan , vinagreti
51	Создание Угловой библиотеки npm	Maciej Treder
52	Тестирование ngModel	jesusegado
53	Тестирование приложения Angular 2	Arun Redhu , michaelbahr , nick , Reza , Rumit Parakhiya
54	Труба заказа	Yoav Schniederman
55	трубы	acdcjunior , Boris , borislemke , BrunoLM , Christopher Taylor , Chybie , daniellmb , Daredzik , elliott-j , Everettss , Fredrik Lundin , Jarod Moser , Jeff Cross , Jim , Kaspars Bergs , Leon Adler , Lexi , LordTribual , michaelbahr , Philipp Kief , theblindprophet
56	Угловая - ForLoop	aholtry , Anil Singh , Berseker59 , gerl , Johan Van de Merwe , ob1 , Pujan Srivastava , Stephen Leppik , Yoav Schniederman
57	Угловая анимация2	Yoav Schniederman
58	Угловое обновление 2-х форм	Amit kumar , Anil Singh , Christopher Taylor , Highmastdon , Johan Van de Merwe , K3v1n , Manmeet Gill , mayur , Norsk , Sachin S , victoroniibukun , vijaykumar , Yoav Schniederman
59	угловое покрытие	ahmadalibaloch
60	угловое сокращение	Yoav Schniederman
61	Угловой 2 - транспорт	Yoav Schniederman
62	Угловой 2 Обнаружение изменений и ручной запуск	Yoav Schniederman
63	Угловые 2 управляемые данными формы	MatWaligora , ThunderRoid

64	Угловые объекты RXJS и наблюдения с запросами API	daniellmb , Maciej Treder , Ronald Zarīts , Sam Storie , Sébastien Temprado , willydee
65	Угловые2 Пользовательские проверки	Arnold Wiersma , Norsk , Yoav Schniederman
66	Услуги и зависимость от инъекций	BrunoLM , Eduardo Carísio , Kaspars Bergs , Matrim , Roope Hakulinen , Syam Pradeep , theblindprophet
67	Установка сторонних плагинов с помощью angular-cli@1.0.0-beta.10	Alex Morales , Daredzik , filoxo , Kaspars Bergs , pd farhad
68	Функциональные модули	AryanJ-NYC , gsc
69	Шаблоны	Max Karpovets