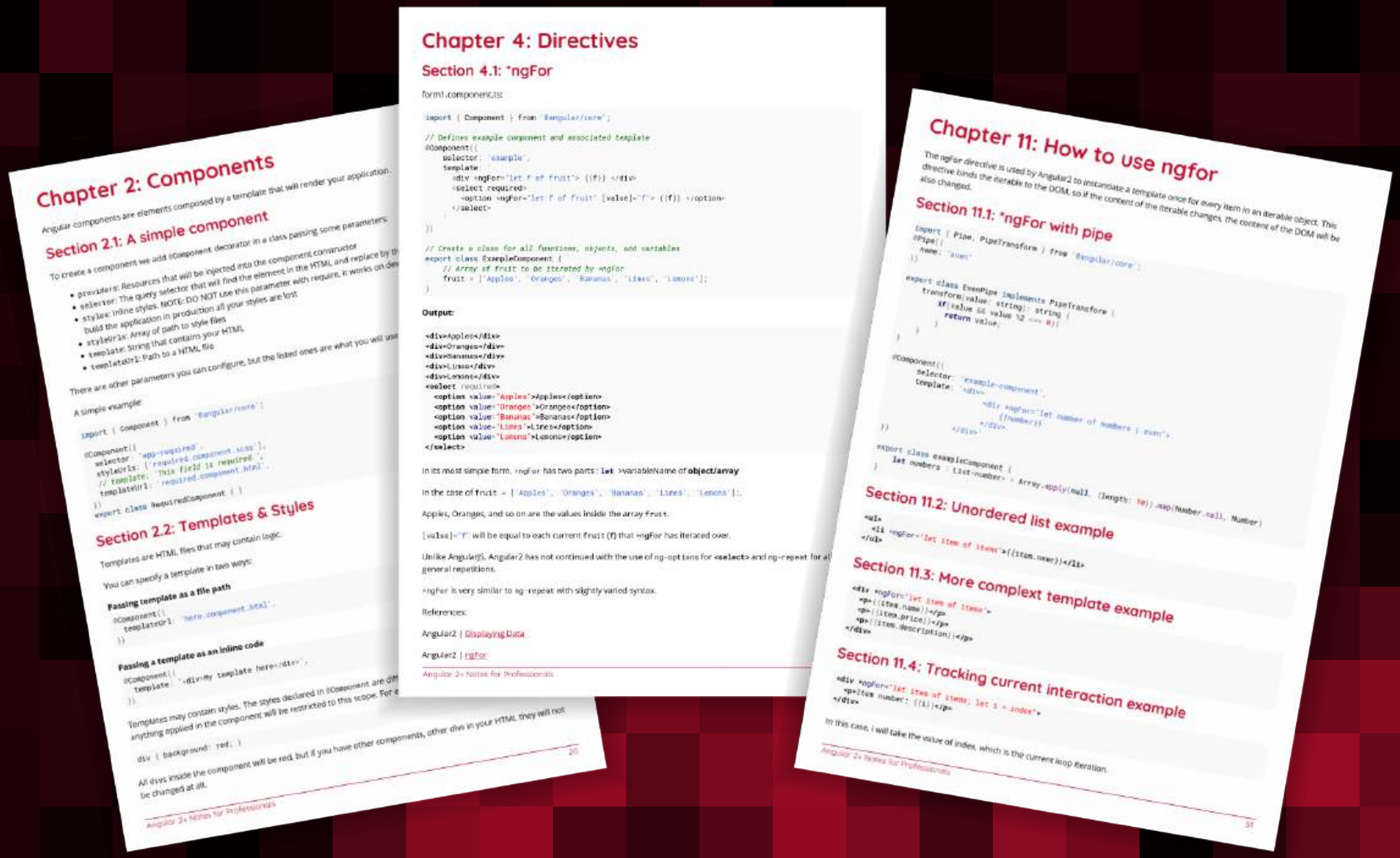
专业人士笔记

Angular 2+

专业人士笔记



Chapter 2: Components

Section 2.1: A simple component

Section 2.2: Templates & Styles

Chapter 4: Directives

Section 4.1: *ngFor

Chapter 11: How to use ngfor

Section 11.1: *ngFor with pipe

Section 11.2: Unordered list example

Section 11.3: More complex template example

Section 11.4: Tracking current interaction example

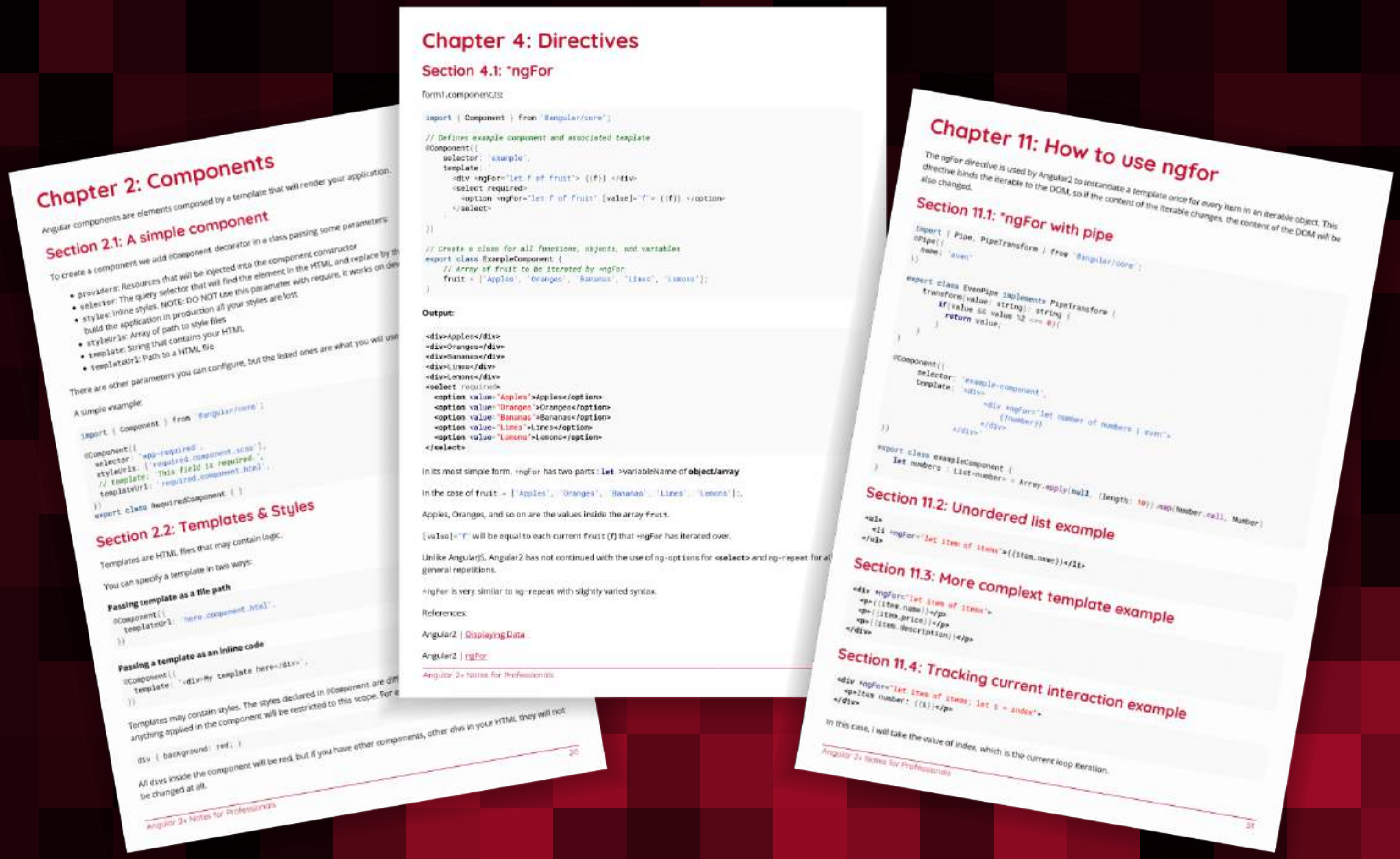
200多页

专业提示和技巧

GoalKicker.com
免费编程书籍

Angular 2+

Notes for Professionals



Chapter 2: Components

Section 2.1: A simple component

Section 2.2: Templates & Styles

Chapter 4: Directives

Section 4.1: *ngFor

Chapter 11: How to use ngfor

Section 11.1: *ngFor with pipe

Section 11.2: Unordered list example

Section 11.3: More complex template example

Section 11.4: Tracking current interaction example

200+ pages

of professional hints and tricks

GoalKicker.com
Free Programming Books

目录

Contents

关于

About

第1章：Angular 2+入门

版本发布日期	
6.0.0	2018-05-04
6.0.0-rc.5	2018-04-14
6.0.0-beta.0	2018-01-25
5.0.0	2017-11-01
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
2.0.0-rc.4	2016-06-30
2.0.0-rc.3	2016-06-21
2.0.0-rc.2	2016-06-15
2.0.0-rc.1	2016-05-03
2.0.0-rc.0	2016-05-02

第1.1节：使用带有node.js/expressjs后端的Angular 2入门（包含http示例）

我们将使用Angular2 2.4.1（@NgModule变更）和node.js（expressjs）后端创建一个简单的“Hello World!”应用。

先决条件

- Node.js v4.x.x或更高版本
- npm v3.x.x或更高版本或yarn

然后运行npm**install**-gtypescript或`yarn global add typescript`以全局安装typescript

路线图
步骤 1

为我们的应用创建一个新文件夹（也是后端的根目录）。我们称它为Angular2-express。

命令行:

Chapter 1: Getting started with Angular 2+

Version	Release Date
6.0.0	2018-05-04
6.0.0-rc.5	2018-04-14
6.0.0-beta.0	2018-01-25
5.0.0	2017-11-01
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
2.0.0-rc.4	2016-06-30
2.0.0-rc.3	2016-06-21
2.0.0-rc.2	2016-06-15
2.0.0-rc.1	2016-05-03
2.0.0-rc.0	2016-05-02

Section 1.1: Getting started with Angular 2 with node.js/expressjs backend (http example included)

We will create a simple "Hello World!" app with Angular2 2.4.1 (`@NgModule` change) with a node.js (expressjs) backend.

Prerequisites

- Node.js v4.x.x or higher
- npm v3.x.x or higher or yarn

Then run `npm install -g typescript` or `yarn global add typescript`to install typescript globally

Roadmap
Step 1

Create a new folder (and the root dir of our back-end) for our app. Let's call it `Angular2-express`.

command line:

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

步骤 2

为我们的node.js应用创建package.json（用于依赖）和app.js（用于引导）。

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){ //示例 HTTP 请求接收器
  return res.send(myTestVar);
});

//在每次页面刷新时发送 index.html，由 Angular 处理路由
app.get('/*',  function(req, res, next) {
    console.log("Reloading");
res.sendFile('index.html', { root: __dirname });
});
```

然后运行 npm install 或 yarn 来安装依赖。

现在我们的后端结构已经完成。让我们继续前端部分。

步骤3

我们的前端应该放在 front 文件夹内，该文件夹位于 Angular2-express 文件夹中。

命令行：

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

Step2

Create the `package.json` (for dependencies) and `app.js` (for bootstrapping) for our `node.js` app.

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 });
});
```

Then run an `npm install` or `yarn` to install the dependencies.

Now our back-end structure is complete. Let's move on to the front-end.

Step3

Our front-end should be in a folder named `front` inside our `Angular2-express` folder.

command line:

```
mkdir front
cd front
```

就像我们对后端所做的那样，前端也需要依赖文件。让我们继续创建以下文件：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:

```
/**
* Angular 示例的系统配置
 * 根据您的应用需求进行相应调整。
 */
(function (global) {
  System.config({
    defaultJSExtensions:true,
    paths: {
      // 路径作为别名
      'npm:': 'node_modules/'
    },
    // map 告诉 System 加载器去哪里查找资源
map: {
```

```
mkdir front
cd front
```

Just like we did with our back-end our front-end needs the dependency files too. Let's go ahead and create the following files: `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: {
      // 我们的应用在 app 文件夹内
app: 'app',
      // angular 包
      '@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/*"
  ]
}
```

然后运行 npm install 或 yarn 来安装依赖。

现在我们的依赖文件已经完成。让我们继续进行我们的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">
```

```
      // 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/*"
  ]
}
```

Then run an `npm install` or `yarn` to install the dependencies.

Now that our dependency files are complete. Let's move on to our `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. 加载库 -->
     <!-- 旧浏览器的 Polyfill(s) -->
    <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. 配置 SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>

  </head>
  <!-- 3. 显示应用程序 -->
  <body>
    <my-app>加载中...</my-app>
  </body>
</html>
```

现在我们准备创建第一个组件。在我们的front文件夹内创建一个名为app的文件夹。

命令行：

```
mkdir app
cd app
```

让我们创建以下文件，分别命名为`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 { HttpModule } from "@angular/http";

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

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

app.component.ts:

```
    <!-- 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>
```

Now we're ready to create our first component. Create a folder named `app` inside our `front` folder.

command line:

```
mkdir app
cd app
```

Let's make the following files named `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 { HttpModule } from "@angular/http";

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

@NgModule({
  imports:      [
    BrowserModule,
    HttpModule
  ],
  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 示例
    this.http.get('/test')
     .subscribe((res)=>{
       console.log(res);
     });
  }
}
```

之后，将 TypeScript 文件编译为 JavaScript 文件。从当前目录（Angular2-express
文件夹内）向上返回两级，然后运行以下命令。

命令行：

```
cd ..
cd ..
tsc -p front
```

我们的文件夹结构应如下所示：

```
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
```

最后，在 Angular2-express 文件夹内，在命令行运行 `node app.js` 命令。打开你喜欢的
浏览器，访问 `localhost:9999` 查看你的应用。

第1.2节：使用 angular-cli 安装 angular2

本示例是 Angular 2 的快速安装及如何生成一个快速示例项目。

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

After this, compile the typescript files to javascript files. Go 2 levels up from the current dir (inside Angular2-express folder) and run the command below.

command line:

```
cd ..
cd ..
tsc -p front
```

Our folder structure should look like:

```
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
```

Finally, inside Angular2-express folder, run `node app.js` command in the command line. Open your favorite browser and check `localhost:9999` to see your app.

Section 1.2: Install angular2 with angular-cli

This example is a quick setup of Angular 2 and how to generate a quick example project.

先决条件：

- Node.js v4 或更高版本。
- npm v3 或更高版本，或 yarn。

打开终端，依次运行以下命令：

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

或

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

取决于您选择的包管理器。

前面的命令全局安装了 @angular/cli， 并将可执行文件 ng 添加到 PATH 中。

设置一个新项目

使用终端导航到您想要设置新项目的文件夹。

运行以下命令：

```
ng new PROJECT_NAME
cd PROJECT_NAME
ng serve
```

就是这样，您现在有了一个用 Angular 2 制作的简单示例项目。您现在可以导航到终端中显示的链接，查看其运行情况。

添加到现有项目

导航到当前项目的根目录。

运行命令：

```
ng init
```

这将为你的项目添加必要的脚手架。文件将创建在当前目录中，因此请确保
在空目录中运行此命令。

本地运行项目

为了在浏览器中查看和交互你的应用程序，你必须启动一个本地
开发服务器来托管你的项目文件。

```
ng serve
```

如果服务器启动成功，应该会显示服务器运行的地址。通常是：

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

开箱即用的本地开发服务器支持热模块重载，因此对 html、
typescript 或 css 的任何更改都会触发浏览器自动重新加载（如果需要，也可以禁用）。

Prerequisites:

- Node.js v4 or greater.
- npm v3 or greater or yarn.

Open a terminal and run the commands one by one:

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

or

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

depending on your choice of package manager.

The previous command installs **@angular/cli** globally, adding the executable `ng` to PATH.

To setup a new project

Navigate with the terminal to a folder where you want to set up the new project.

Run the commands:

```
ng new PROJECT_NAME
cd PROJECT_NAME
ng serve
```

That is it, you now have a simple example project made with Angular 2. You can now navigate to the link displayed in terminal and see what it is running.

To add to an existing project

Navigate to the root of your current project.

Run the command:

```
ng init
```

This will add the necessary scaffolding to your project. The files will be created in the current directory so be sure to run this in an empty directory.

Running The Project Locally

In order to see and interact with your application while it's running in the browser you must start a local development server hosting the files for your project.

```
ng serve
```

If the server started successfully it should display an address at which the server is running. Usually is this:

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

Out of the box this local development server is hooked up with Hot Module Reloading, so any changes to the html, typescript, or css, will trigger the browser to be automatically reloaded (but can be disabled if desired).

生成组件、指令、管道和服务

ng generate <脚手架类型> <名称>（或简写为ng g <脚手架类型> <名称>）命令允许你自动生成Angular组件：

```
# 下面的命令将在你当前所在的文件夹中生成一个组件
ng generate component my-generated-component
# 使用别名（效果同上）
ng g component my-generated-component
```

angular-cli可以生成多种类型的脚手架：

脚手架类型	用法
模块	`ng g module my-new-module`
组件	`ng g 组件 my-new-component`
指令	`ng g 指令 my-new-directive`
管道	`ng g 管道 my-new-pipe`
服务	`ng g 服务 my-new-service`
类	`ng g 类 my-new-class`
接口	`ng g interface my-new-interface`
枚举	`ng g enum my-new-enum`

你也可以用类型名称的首字母来替代。例如：

`ng g m my-new-module` 用于生成一个新模块，或者`ng g c my-new-component` 用于创建一个组件。

构建/打包

当你完成构建你的 Angular 2 网络应用，并且想要将其安装到像
Apache Tomcat 这样的网络服务器上时，你只需运行构建命令，无论是否设置生产标志。
生产模式会压缩代码并针对生产环境进行优化。

```
ng build
```

或

```
ng build --prod
```

然后在项目根目录下查看一个/dist文件夹，里面包含了构建结果。

如果您想要更小的生产包体积，也可以使用预编译模板（Ahead-of-Time template compilation），
这会将模板编译器从最终构建中移除：

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

单元测试

Angular 2 提供了内置的单元测试功能，且每个由 angular-cli 创建的项目都会生成一个基础的单元测试，可以在此基础上进行扩展。单元测试使用 jasmine 编写，并通过 Karma 执行。要开始测试，请执行以下命令：

```
ng test
```

Generating Components, Directives, Pipes and Services

The `ng generate <scaffold-type> <name>` (or simply `ng g <scaffold-type> <name>`) command allows you to automatically generate Angular components:

```
# 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
```

There are several possible types of scaffolds angular-cli can generate:

Scaffold Type	Usage
Module	`ng g module my-new-module`
Component	`ng g component my-new-component`
Directive	`ng g directive my-new-directive`
Pipe	`ng g pipe my-new-pipe`
Service	`ng g service my-new-service`
Class	`ng g class my-new-class`
Interface	`ng g interface my-new-interface`
Enum	`ng g enum my-new-enum`

You can also replace the type name by its first letter. For example:

`ng g m my-new-module` to generate a new module or `ng g c my-new-component` to create a component.

Building/Bundling

When you are all finished building your Angular 2 web app and you would like to install it on a web server like Apache Tomcat, all you need to do is run the build command either with or without the production flag set. Production will minifiy the code and optimize for a production setting.

```
ng build
```

or

```
ng build --prod
```

Then look in the projects root directory for a `/dist` folder, which contains the build.

If you'd like the benefits of a smaller production bundle, you can also use Ahead-of-Time template compilation, which removes the template compiler from the final build:

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

Unit Testing

Angular 2 provides built-in unit testing, and every item created by angular-cli generates a basic unit test, that can be expanded. The unit tests are written using jasmine, and executed through Karma. In order to start testing execute the following command:

```
ng test
```

该命令将执行项目中的所有测试，并且每当源文件发生更改时（无论是测试文件还是应用代码），都会重新执行测试。

更多信息请访问：angular-cli github 页面

第1.3节：不使用 angular-cli 开始使用 Angular 2

Angular 2.0.0-rc.4

在本例中，我们将创建一个只有一个根组件（AppComponent）的“Hello World!”应用，以简化示例。

先决条件：

- Node.js v5 或更高版本
- npm v3 或更高版本

> 注意： 你可以通过在控制台/终端运行 node -v 和 npm -v 来检查版本。

步骤 1

创建并进入一个新的项目文件夹。我们称它为 angular2-example。

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

步骤 2

在开始编写应用代码之前，我们将添加下面提供的4个文件：package.json、tsconfig.json、typings.json 和 systemjs.config.js。

> 免责声明： 这些文件也可以在 官方 5 分钟快速入门 中找到。

package.json - 允许我们使用 npm 下载所有依赖，并提供简单的脚本执行，使简单项目的开发更轻松。（你以后可以考虑使用类似 Gulp 的工具来自动化任务）。

```
{
  "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",
```

This command will execute all the tests in the project, and will re-execute them every time a source file changes, whether it is a test or code from the application.

For more info also visit: angular-cli github page

Section 1.3: Getting started with Angular 2 without angular-cli

Angular 2.0.0-rc.4

In this example we'll create a "Hello World!" app with only one root component (`AppComponent`) for the sake of simplicity.

Prerequisites:

- Node.js v5 or later
- npm v3 or later

> **Note:** You can check versions by running `node -v` and `npm -v` in the console/terminal.

Step 1

Create and enter a new folder for your project. Let's call it `angular2-example`.

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

Step 2

Before we start writing our app code, we'll add the 4 files provided below: `package.json`, `tsconfig.json`, `typings.json`, and `systemjs.config.js`.

> **Disclaimer:** The same files can be found in the Official 5 Minute Quickstart.

`package.json` - Allows us to download all dependencies with npm and provides simple script execution to make life easier for simple projects. (You should consider using something like Gulp in the future to automate tasks).

```
{
  "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 - 配置 TypeScript 转译器。

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

typings.json - 使 TypeScript 识别我们使用的库。

```
{
  "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 - 配置SystemJS（你也可以使用webpack）。

```
/**
 * Angular 2 示例的系统配置
  * 根据你的应用需求进行调整。
  */
(function(global) {
  // map 告诉 System 加载器去哪里查找资源
  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'
  };
```

```
    "@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` - Configures the TypeScript transpiler.

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

`typings.json` - Makes TypeScript recognize libraries we're using.

```
{
  "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` - Configures SystemJS (you can also use 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',
  ];
  // 单个文件（约300个请求）：
  function packIndex(pkgName) {
packages['@angular/'+pkgName] = { main: 'index.js', defaultExtension: 'js' };
  }
  // 打包文件（约40个请求）：
  function packUmd(pkgName) {
packages['@angular/'+pkgName] = { main: '/bundles/' + pkgName + '.umd.js', defaultExtension:
'js' };
  }
  // 大多数环境应使用UMD；某些环境（如Karma）需要单独的索引文件
  var setPackageConfig = System.packageWithIndex ? packIndex : packUmd;
  // 为 Angular 包添加包条目
ngPackageNames.forEach(setPackageConfig);
  var config = {
map: map,
    packages: packages
  };
System.config(config);
})(this);
```

步骤 3

让我们通过输入来安装依赖项

```
npm install
```

在控制台/终端中。

步骤 4

在angular2-example文件夹内创建index.html文件。

```
<html>
  <head>
    <title>Angular2 示例</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 1. 加载库 -->
    <!-- 旧浏览器的 Polyfill(s) -->
    <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>
```

```
  // 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);
```

Step 3

Let's install the dependencies by typing

```
npm install
```

in the console/terminal.

Step 4

Create `index.html` inside of the `angular2-example` folder.

```
<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. 配置 SystemJS -->
    <script src="systemjs.config.js"></script>
    <script>
      System.import('app').catch(function(err){ console.error(err); });
    </script>
  </head>
  <!-- 3. 显示应用程序 -->
  <body>
    <my-app></my-app>
  </body>
</html>
```

您的应用程序将渲染在 my-app 标签之间。

但是，Angular 仍然不知道渲染什么。为此，我们将定义AppComponent。

步骤 5

创建一个名为app的子文件夹，在这里我们可以定义构成应用程序的组件和服务。（在本例中，它只包含AppComponent代码和main.ts。）

```
mkdir app
```

步骤 6

创建文件 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>
  `
})
导出类 AppComponent {
    标题 = "Angular2 示例";
    消息 = [
        "你好，世界！",
        "另一个字符串",
        "再一个"
    ];
}
```

发生了什么？首先，我们导入了@Component装饰器，用来给Angular提供该组件的HTML标签和模板。然后，我们创建了AppComponent类，包含title和messages变量，这些变量可以在模板中使用。

现在让我们看看那个模板：

```
<h1>{{title}}</h1>
<ul>
```

```
    <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>
```

Your application will be rendered between the my-app tags.

However, Angular still doesn't know *what* to render. To tell it that, we'll define AppComponent.

Step 5

Create a subfolder called app where we can define the components and services that make up our app. (In this case, it'll just contain the AppComponent code and main.ts.)

```
mkdir app
```

Step 6

Create the file 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"
    ];
}
```

What's happening? First, we're importing the @Component decorator which we use to give Angular the HTML tag and template for this component. Then, we're creating the class AppComponent with title and messages variables that we can use in the template.

Now let's look at that template:

```
<h1>{{title}}</h1>
<ul>
```

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

我们在h1标签中显示了title变量，然后使用*ngFor指令创建一个列表，显示messages数组中的每个元素。对于数组中的每个元素，*ngFor都会创建一个message变量，我们在li元素中使用它。结果将是：

```
<h1>Angular 2 示例</h1>
<ul>
    <li>你好，世界！</li>
    <li>另一个字符串</li>
    <li>再一个</li>
</ul>
```

第7步

现在我们创建一个main.ts文件，这将是Angular首先查看的文件。

创建文件app/main.ts。

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

bootstrap(AppComponent);
```

我们导入了bootstrap函数和AppComponent类，然后使用bootstrap告诉Angular使用哪个组件作为根组件。

第8步

现在是启动你的第一个应用程序的时候了。输入

```
npm start
```

在你的控制台/终端中。这将运行package.json中预设的脚本，启动lite-server，打开浏览器窗口中的应用程序，并以监听模式运行TypeScript转译器（因此.ts文件将在保存更改时被转译，浏览器也会刷新）。

现在怎么办？

查看官方的 Angular 2 指南以及 StackOverflow 文档中的其他主题。

你也可以编辑AppComponent以使用外部模板、样式或添加/编辑组件变量。保存文件后，你应该能立即看到更改效果。

第 1.4 节：突破那个讨厌的公司代理

如果你试图在 XYZ MegaCorp 的 Windows 工作电脑上运行 Angular2 网站，很可能你遇到了通过公司代理的问题。

至少有两个包管理器需要通过代理：

1.NPM
2.Typings

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

We're displaying the `title` variable in an `h1` tag and then making a list showing each element of the `messages` array by using the `*ngFor` directive. For each element in the array, `*ngFor` creates a `message` variable that we use within the `li` element. The result will be:

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

Step 7

Now we create a `main.ts` file, which will be the first file that Angular looks at.

Create the file `app/main.ts`.

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

bootstrap(AppComponent);
```

We're importing the `bootstrap` function and `AppComponent` class, then using `bootstrap` to tell Angular which component to use as the root.

Step 8

It's time to fire up your first app. Type

```
npm start
```

in your console/terminal. This will run a prepared script from `package.json` that starts lite-server, opens your app in a browser window, and runs the TypeScript transpiler in watch mode (so `.ts` files will be transpiled and the browser will refresh when you save changes).

What now?

Check out the official Angular 2 guide and the other topics on StackOverflow's documentation.

You can also edit `AppComponent` to use external templates, styles or add/edit component variables. You should see your changes immediately after saving files.

Section 1.4: Getting through that pesky company proxy

If you are attempting to get an Angular2 site running on your Windows work computer at XYZ MegaCorp the chances are that you are having problems getting through the company proxy.

There are (at least) two package managers that need to get through the proxy:

1. NPM
2. Typings

对于 NPM，你需要在.npmrc文件中添加以下行：

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

对于 Typings，你需要在.typingsrc文件中添加以下行：

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

这些文件可能还不存在，所以你可以将它们创建为空白文本文件。它们可以添加到项目根目录（与`package.json`相同的位置），或者你可以将它们放在`%HOMEPATH%`中，这样它们将对你所有的项目可用。

不明显且是人们认为代理设置不起作用的主要原因是%5C，它是\\的URL编码，用于分隔域名和用户名。感谢Steve Roberts提供的信息：在企业代理.pac后使用npm

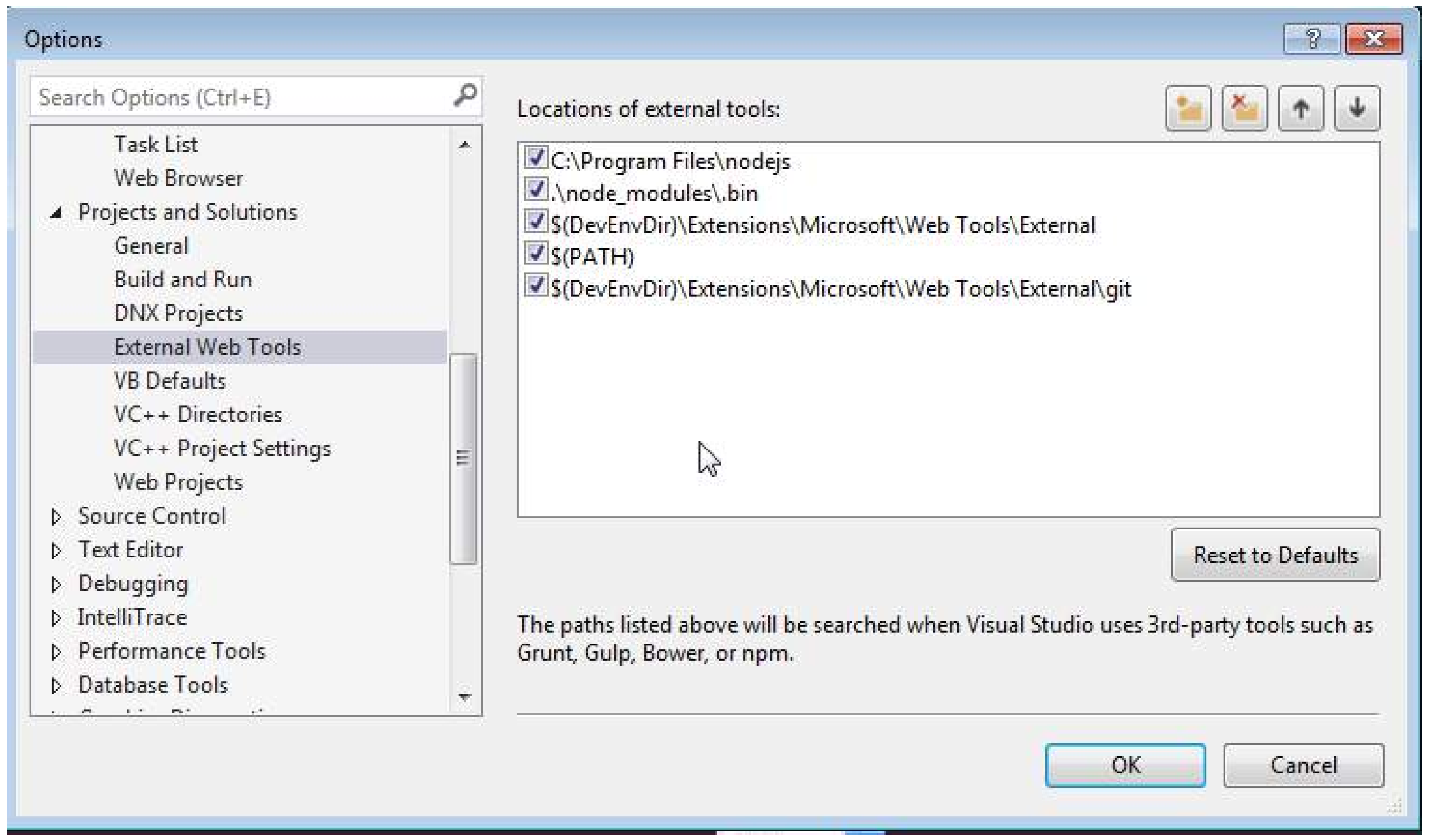
第1.5节：保持Visual Studios与NPM和NODE更新同步

步骤1：定位你的Node.js下载位置，通常安装在C:/program files/nodejs下

步骤2：打开Visual Studios并导航到“工具>选项”

步骤3：在选项窗口中导航到“项目和解决方案>外部Web工具”

步骤4：添加新的条目，路径为你的Node.js文件位置（C:/program files/nodejs），重要的是使用菜单上的箭头按钮将你的引用移动到列表顶部。


Options

Search Options (Ctrl+E)

Task List
Web Browser
Projects and Solutions
General
Build and Run
DNX Projects
External Web Tools
VB Defaults
VC++ Directories
VC++ Project Settings
Web Projects
Source Control
Text Editor
Debugging
IntelliTrace
Performance Tools
Database Tools

Locations of external tools:

☑ C:\Program Files\nodejs
☑ .\node_modules\.bin
☑ $(DevEnvDir)\Extensions\Microsoft\Web Tools\External
☑ $(PATH)
☑ $(DevEnvDir)\Extensions\Microsoft\Web Tools\External\git

Reset to Defaults

The paths listed above will be searched when Visual Studio uses 3rd-party tools such as Grunt, Gulp, Bower, or npm.

OK Cancel

步骤5：重启Visual Studios，并从npm命令窗口针对你的项目运行npm install

For NPM you need to add the following lines to the `.npmrc` file:

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

For Typings you need to add the following lines to the `.typingsrc` file:

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

These files probably don't exist yet, so you can create them as blank text files. They can be added to the project root (same place as `package.json` or you can put them in `%HOMEPATH%` and they will be available to all your projects.

The bit that isn't obvious and is the main reason people think the proxy settings aren't working is the `%5C` which is the URL encode of the `\` to separate the domain and user names. Thanks to Steve Roberts for that one: Using npm behind corporate proxy .pac

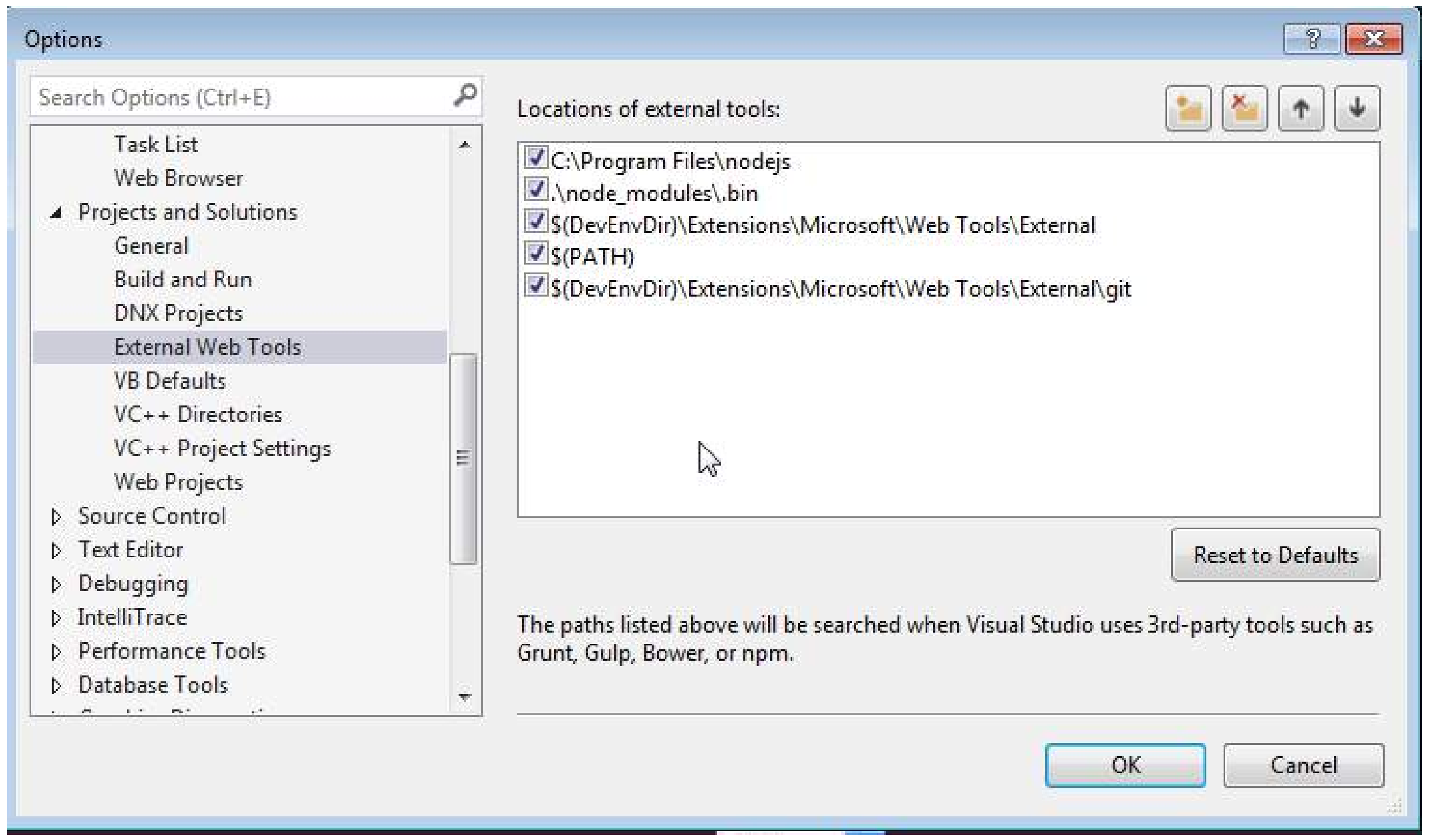
Section 1.5: Keeping Visual Studios in sync with NPM and NODE Updates

Step 1: Locate your download of Node.js, typically it is installed under C:/program files/nodejs

Step 2: Open Visual Studios and navigate to "Tools>Options"

Step 3: In the options window navigate to "Projects and Solutions>External Web Tools"

Step 4: Add new entry with you Node.js file location (C:/program files/nodejs), IMPORTANT use the arrow buttons on menu to move your reference to the top of the list.


Options

Search Options (Ctrl+E)

Task List
Web Browser
Projects and Solutions
General
Build and Run
DNX Projects
External Web Tools
VB Defaults
VC++ Directories
VC++ Project Settings
Web Projects
Source Control
Text Editor
Debugging
IntelliTrace
Performance Tools
Database Tools

Locations of external tools:

☑ C:\Program Files\nodejs
☑ .\node_modules\.bin
☑ $(DevEnvDir)\Extensions\Microsoft\Web Tools\External
☑ $(PATH)
☑ $(DevEnvDir)\Extensions\Microsoft\Web Tools\External\git

Reset to Defaults

The paths listed above will be searched when Visual Studio uses 3rd-party tools such as Grunt, Gulp, Bower, or npm.

OK Cancel

Step 5: Restart Visual Studios and Run an npm install, against your project, from npm command window

第1.6节：让我们深入了解Angular 4！

Angular 4现已发布！实际上，自Angular 2起，Angular采用了语义化版本控制（semver），当引入破坏性更改时，主版本号必须增加。Angular团队推迟了会导致破坏性更改的功能，这些功能将在Angular 4中发布。为了使核心模块的版本号保持一致，跳过了Angular 3，因为路由器已经是版本3。

根据Angular团队的说法，Angular 4应用程序将比以前占用更少空间且运行更快。他们将动画包从@angular/core包中分离出来。如果有人不使用动画包，那么额外的代码空间将不会出现在生产环境中。模板绑定语法现在支持if/else风格的语法。Angular 4现已兼容最新版本的Typescript 2.1和2.2。因此，Angular 4将更加令人兴奋。

现在我将向你展示如何在你的项目中设置 Angular 4。

让我们开始用三种不同的方法设置Angular：

你可以使用Angular-CLI（命令行界面），它会为你安装所有依赖。

- 你可以从Angular 2迁移到Angular 4。
- 你可以使用GitHub并克隆Angular4-boilerplate。（这是最简单的方法。????）
- 使用Angular-CLI（命令行界面）进行Angular设置。

在开始使用Angular-CLI之前，确保你的电脑上已安装node。这里我使用的是node v7.8.0。现在，打开终端并输入以下命令来安装Angular-CLI。

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

或

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

取决于你使用的包管理器。

让我们使用Angular-CLI安装Angular 4。

```
ng new Angular4-boilerplate
```

cd Angular4-boilerplate 我们已经为Angular 4做好了准备。这是一个非常简单直接的方法。????

通过从Angular 2迁移到Angular 4进行Angular设置

现在让我们来看第二种方法。我将向你展示如何将 Angular 2 迁移到 Angular 4。为此，你需要克隆任何一个 Angular 2 项目，并在你的 package.json 中将 Angular 2 的依赖更新为 Angular 4 的依赖，如下所示：

```
"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",
```

Section 1.6: Let's dive into Angular 4!

Angular 4 is now available! Actually Angular uses semver since Angular 2, which requires the major number being increased when breaking changes were introduced. The Angular team postponed features that cause breaking changes, which will be released with Angular 4. Angular 3 was skipped to be able to align the version numbers of the core modules, because the Router already had version 3.

As per the Angular team, Angular 4 applications are going to be less space consuming and faster than before. They have separated animation package from @angular/core package. If anybody is not using animation package so extra space of code will not end up in the production. The template binding syntax now supports if/else style syntax. Angular 4 is now compatible with most recent version of Typescript 2.1 and 2.2. So, Angular 4 is going to be more exciting.

Now I'll show you how to do setup of Angular 4 in your project.

Let's start Angular setup with three different ways:

You can use Angular-CLI (Command Line Interface) , It will install all dependencies for you.

- You can migrate from Angular 2 to Angular 4.
- You can use github and clone the Angular4-boilerplate. (It is the easiest one.????)
- Angular Setup using Angular-CLI(command Line Interface).

Before You start using Angular-CLI , make sure You have node installed in your machine. Here, I am using node v7.8.0. Now, Open your terminal and type the following command for Angular-CLI.

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

or

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

depending on the package manager you use.

Let's install Angular 4 using Angular-CLI.

```
ng new Angular4-boilerplate
```

cd Angular4-boilerplate We are all set for Angular 4. Its pretty easy and straightforward method.????

Angular Setup by migrating from Angular 2 to Angular 4

Now Let's see the second approach. I ll show you how to migrate Angular 2 to Angular 4. For that You need clone any Angular 2 project and update Angular 2 dependencies with the Angular 4 Dependency in your package.json as following:

```
"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"
  }
```

这些是 Angular 4 的主要依赖。现在你可以运行 npm install，然后用 npm start 来启动应用程序。供参考，这是我的 package.json。

来自 GitHub 项目的 Angular 设置

开始这一步之前，请确保你的电脑已安装 git。打开终端，使用以下命令克隆 angular4-boilerplate：

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

然后安装所有依赖并运行它。

```
npm install

npm start
```

Angular 4 设置完成。所有步骤都非常简单，您可以选择其中任何一个。

angular4-boilerplate 的目录结构

```
Angular4-boilerplate
-karma
-node_modules
-src
   -mocks
   -models
      -loginform.ts
      -index.ts
   - 模块
      - 应用
        - 应用。组件。ts
     - 应用。组件。html
      - 登录
    - 登录。组件。ts
    - 登录。组件。html
    - 登录。组件。css
      - 小部件
     - 小部件。组件。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
```

```
    "@angular/platform-browser": "4.0.2",
    "@angular/platform-browser-dynamic": "4.0.2",
    "@angular/router": "4.0.2",
    "typescript": "2.2.2"
  }
```

These are the main dependencies for Angular 4. Now You can npm install and then npm start to run the application. For reference my package.json.

Angular setup from github project

Before starting this step make sure you have git installed in your machine. Open your terminal and clone the angular4-boilerplate using below command:

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

Then install all dependencies and run it.

```
npm install

npm start
```

And you are done with the Angular 4 setup. All the steps are very straightforward so you can opt any of them.

Directory Structure of the angular4-boilerplate

```
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
```

目录结构的基本理解：

所有代码都位于src文件夹中。

mocks文件夹用于测试目的的模拟数据。

model文件夹包含组件中使用的类和接口。

modules 文件夹包含 app、login、widget 等组件列表。所有组件都包含 typescript、html 和 css 文件。index.ts 用于导出所有类。

services 文件夹包含应用程序中使用的服务列表。我将 rest 服务和不同的组件服务分开了。rest 服务中包含不同的 http 方法。登录服务作为登录组件和 rest 服务之间的中介。

app.routing.ts 文件描述了应用程序的所有可能路由。

app.module.ts 描述了作为根组件的应用模块。

bootstrap.ts 将运行整个应用程序。

webpack 文件夹包含 webpack 配置文件。

package.json 文件包含所有依赖项列表。

karma 包含单元测试的 karma 配置。

node_modules 包含包的列表。

让我们从登录组件开始。在 login.component.html 中

```
<form>Dreamfactory - 地址簿 2.0
 <label>邮箱</label> <input id="email" form="" name="email" type="email" />
 <label>密码</label> <input id="password" form="" name="password"
type="password" />
 <button form="">登录</button>
</form>
```

在 login.component.ts 中

```
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: 登录服务, private router: 路由器, form: 登录表单) { }
```

```
-tslint.json
-typings.json
```

Basic understanding for Directory structure:

All the code resides in src folder.

mocks folder is for mock data that is used in testing purpose.

model folder contains the class and interface that used in component.

modules folder contains list of components such as app, login, widget etc. All component contains typescript, html and css file. index.ts is for exporting all the class.

services folder contains list of services used in application. I have separated rest service and different component service. In rest service contains different http methods. Login service works as mediator between login component and rest service.

app.routing.ts file describes all possible routes for the application.

app.module.ts describes app module as root component.

bootstrap.ts will run the whole application.

webpack folder contains webpack configuration file.

package.json file is for all list of dependencies.

karma contains karma configuration for unit test.

node_modules contains list of package bundles.

Lets start with Login component. In 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>
```

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

我们需要在 index.ts 中导出该组件。

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

我们需要在 app.routes.ts 中为登录设置路由

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

在根组件 app.module.ts 文件中，你只需要导入该组件。

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

然后运行 npm install 和 npm start。好了！你可以在本地主机上查看登录界面。如果遇到任何困难，可以参考 angular4-boilerplate。

基本上，我感觉使用 Angular 4 应用程序时构建包更小，响应更快，尽管我发现代码与 Angular 2 完全相似。

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

We need to export this component to in index.ts.

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

we need to set routes for login in app.routes.ts

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

In root component, app.module.ts file you just need to import that component.

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

and after that npm install and npm start. Here, you go! You can check login screen in your localhost. In case of any difficulty, You can refer the angular4-boilerplate.

Basically I can feel less building package and more faster response with Angular 4 application and Although I found Exactly similar to Angular 2 in coding.

第2章：组件

Angular组件是由模板组成的元素，用于渲染您的应用程序。

第2.1节：一个简单的组件

要创建一个组件，我们在类中添加@Component装饰器，并传入一些参数：

- `providers`：将注入到组件构造函数中的资源
- `selector`：用于在HTML中查找元素并被组件替换的查询选择器
- `styles`：内联样式。注意：不要将此参数与require一起使用，它在开发时有效，但当您在生产环境构建应用程序时，所有样式都会丢失
- `styleUrls`：样式文件路径数组
- `template`：包含HTML的字符串
- `templateUrl`：HTML文件的路径

还有其他参数可以配置，但上述列出的参数是您最常用的。

一个简单的示例：

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

@Component({
selector: 'app-required',
  styleUrls: ['required.component.scss'],
  // template: `此字段为必填。`,
templateUrl: 'required.component.html',
})
export class RequiredComponent { }
```

第2.2节：模板与样式

模板是可能包含逻辑的HTML文件。

你可以通过两种方式指定模板：

以文件路径传递模板

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

以内联代码传递模板

```
@Component({
template: `<div>我的模板内容</div>`,
})
```

模板可能包含样式。在`@Component`中声明的样式与您的应用程序样式文件不同，组件中应用的任何内容都将限制在此范围内。例如，假设您添加了：

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

组件内的所有 div 都将是红色的，但如果你的 HTML 中有其他组件或其他 div，它们将完全不会被改变。

Chapter 2: Components

Angular components are elements composed by a template that will render your application.

Section 2.1: A simple component

To create a component we add `@Component` decorator in a class passing some parameters:

- `providers`: Resources that will be injected into the component constructor
- `selector`: The query selector that will find the element in the HTML and replace by the component
- `styles`: Inline styles. NOTE: DO NOT use this parameter with require, it works on development but when you build the application in production all your styles are lost
- `styleUrls`: Array of path to style files
- `template`: String that contains your HTML
- `templateUrl`: Path to a HTML file

There are other parameters you can configure, but the listed ones are what you will use the most.

A simple example:

```
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 { }
```

Section 2.2: Templates & Styles

Templates are HTML files that may contain logic.

You can specify a template in two ways:

Passing template as a file path

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

Passing a template as an inline code

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

Templates may contain styles. The styles declared in `@Component` are different from your application style file, anything applied in the component will be restricted to this scope. For example, say you add:

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

All `divs` inside the component will be red, but if you have other components, other divs in your HTML they will not be changed at all.

生成的代码将如下所示：

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

你可以通过两种方式向组件添加样式：

传递文件路径数组

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

传递内联代码数组

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

你不应该将 styles 与 require 一起使用，因为在构建生产环境应用时它将无法工作。

第2.3节：测试组件

hero.component.html

```
<form (ngSubmit)="submit($event)" [formGroup]="form" novalidate>
  <input type="text" formControlName="name" />
  <button type="submit">显示英雄名称</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(() => {
```

The generated code will look like this:

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

You can add styles to a component in two ways:

Passing an array of file paths

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

Passing an array of inline codes

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

You shouldn't use `styles` with `require` as it will not work when you build your application to production.

Section 2.3: Testing a Component

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('应该被创建', () => {
    expect(component).toBeTruthy();
  });

it('当用户提交表单时，应该在控制台记录英雄名称', 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('应该验证名称字段为必填项', () => {
    component.form.controls['name'].setValue('');
    expect(component.form.invalid).toBeTruthy();
  });
});
```

第2.4节：嵌套组件

组件将渲染在各自的selector中，因此你可以利用这一点来嵌套组件。

如果你有一个显示消息的组件：

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

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

你可以在另一个组件中使用app-required（该组件的选择器）：

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

@Component({
selector: 'app-sample',
  template: `
```

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

Section 2.4: Nesting components

Components will render in their respective `selector`, so you can use that to nest components.

If you have a component that shows a message:

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

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

You can use it inside another component using `app-required` (this component's selector):

```
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 = '';
}
```

```
    <input type="text" name="heroName" />
    <app-required name="Hero Name"></app-required>
  `
})
export class RequiredComponent {
  @Input()
  public name: String = '';
}
```

第3章：组件交互

名称	值
pageCount	用于告诉子组件要创建的页数。
pageNumberClicked	子组件中输出变量的名称。
pageChanged	父组件中监听子组件输出的函数。

第3.1节：通过输入绑定从父组件传递数据到子组件

HeroChildComponent 有两个输入属性，通常使用 @Input 装饰器标注。

```
import { Component, Input } from '@angular/core';
import { Hero } from './hero';
@Component({
selector: 'hero-child',
  template: `
    <h3>{{hero.name}} 说:</h3>
    <p>我，{{hero.name}}，为您服务，{{masterName}}。</\p>
  `
})
export class HeroChildComponent {
  @Input() hero: Hero;
  @输入('master') masterName: 字符串;
}
```

使用 setter 拦截输入属性的变化

使用输入属性的 setter 来拦截并处理来自父组件的值。

子组件 NameChildComponent 中 name 输入属性的 setter 会去除名字的空白字符，并将空值替换为默认文本。

```
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; }
}
```

下面是 NameParentComponent，演示了包括全空格名字在内的名字变体：

```
import { Component } from '@angular/core';
@Component({
selector: 'name-parent',
  template: `
  <h2>主控 {{names.length}} 个名字</h2>
  <name-child *ngFor="let name of names" [name]="name"></name-child>
  `
})
导出类 NameParentComponent {
```

Chapter 3: Component interactions

Name	Value
pageCount	Used to tell number of pages to be created to the child component.
pageNumberClicked	Name of output variable in the child component.
pageChanged	Function at parent component that listening for child components output.

Section 3.1: Pass data from parent to child with input binding

HeroChildComponent has two input properties, typically adorned with @Input decorations.

```
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;
}
```

Intercept input property changes with a setter

Use an input property setter to intercept and act upon a value from the parent.

The setter of the name input property in the child NameChildComponent trims the whitespace from a name and replaces an empty value with default text.

```
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; }
}
```

Here's the NameParentComponent demonstrating name variations including a name with all spaces:

```
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 {
  // 显示 'Mr. IQ', '<未设置名字>', 'Bombasto'
names = ['先生 IQ', '   ', '  Bombasto  '];
}
```

父组件监听子组件事件

子组件暴露了一个 EventEmitter 属性，当发生某些事情时会发出事件。
父组件绑定该事件属性并对这些事件做出响应。

子组件的 EventEmitter 属性是一个输出属性，通常用 @Output 装饰器标注，如下 VoterComponent 所示：

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

点击按钮会触发一个布尔值（true 或 false）的事件发射。

父组件 VoteTakerComponent 绑定了一个事件处理器（onVoted），用于响应子组件事件负载
($event) 并更新计数器。

```
import { Component } from '@angular/core';
@Component({
selector: 'vote-taker',
  template: `
    <h2>人类应该殖民宇宙吗？</h2>
    <h3>同意: {{agreed}}, 不同意: {{disagreed}}</h3>
    <my-voter *ngFor="let voter of voters"
      [name]="voter"
      (onVoted)="onVoted($event)">
    </my-voter>
  `
})
export class VoteTakerComponent {
  agreed = 0;
disagreed = 0;
voters = ['智商先生', '宇宙小姐', '炸弹手'];
  onVoted(agreed: boolean) {
agreed ? this.agreed++ : this.disagreed++;
  }
}
```

父组件通过局部变量与子组件交互

```
  // Displays 'Mr. IQ', '<no name set>', 'Bombasto'
  names = ['Mr. IQ', '   ', '  Bombasto  '];
}
```

Parent listens for child event

The child component exposes an EventEmitter property with which it emits events when something happens. The parent binds to that event property and reacts to those events.

The child's EventEmitter property is an output property, typically adorned with an @Output decoration as seen in this 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(agreed: boolean) {
    this.onVoted.emit(agreed);
    this.voted = true;
  }
}
```

Clicking a button triggers emission of a true or false (the boolean payload).

The parent VoteTakerComponent binds an event handler (onVoted) that responds to the child event payload ($event) and updates a counter.

```
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++;
  }
}
```

Parent interacts with child via local variable

父组件不能通过数据绑定读取子组件属性或调用子组件方法。我们可以通过为子元素创建模板引用变量，然后在父模板中引用该变量，来实现这两者，如以下示例所示。

我们有一个子组件CountdownTimerComponent，它会反复倒计时至零并发射火箭。它有控制时钟的开始和停止方法，并在自己的模板中显示倒计时状态消息。

```
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-${this.seconds} seconds`;
  }
private countDown() {
    this.clearTimer();
    this.intervalId = window.setInterval(() => {
      this.seconds -= 1;
      if (this.seconds === 0) {
        this.message = '发射！';
      } else {
        if (this.seconds < 0) { this.seconds = 10; } // 重置
        this.message = `T-${this.seconds} 秒并继续计数`;
      }
    }, 1000);
  }
}
```

让我们来看一下托管计时器组件的 CountdownLocalVarParentComponent。

```
import { Component }                from '@angular/core';
import { CountdownTimerComponent }  from './countdown-timer.component';
@Component({
selector: 'countdown-parent-lv',
  template: `
  <h3>倒计时发射 (通过局部变量)</h3>
  <button (click)="timer.start()">开始</button>
  <button (click)="timer.stop()">停止</button>
  <div class="seconds">{{timer.seconds}}</div>
  <countdown-timer #timer></countdown-timer>
  `,
styleUrls: ['demo.css']
})
export class CountdownLocalVarParentComponent { }
```

父组件无法对子组件的 start 和 stop 方法以及其 seconds 属性进行数据绑定。

我们可以在表示子组件的标签（）上放置一个局部变量（#timer）。这样我们就可以引用子组件本身，并能够从父组件内部访问其任何属性或方法。

A parent component cannot use data binding to read child properties or invoke child methods. We can do both by creating a template reference variable for the child element and then reference that variable within the parent template as seen in the following example.

We have a child CountdownTimerComponent that repeatedly counts down to zero and launches a rocket. It has start and stop methods that control the clock and it displays a countdown status message in its own template.

```
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);
  }
}
```

Let's see the CountdownLocalVarParentComponent that hosts the timer component.

```
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 { }
```

The parent component cannot data bind to the child's start and stop methods nor to its seconds property.

We can place a local variable (#timer) on the tag () representing the child component. That gives us a reference to the child component itself and the ability to access any of its properties or methods from within the parent

模板。

在这个例子中，我们将父组件的按钮连接到子组件的开始和停止方法，并使用插值显示子组件的seconds属性。

这里我们看到父组件和子组件协同工作。

父组件调用ViewChild

局部变量方法简单易用。但它有局限性，因为父子组件的连接必须完全在父组件模板内完成。父组件本身无法访问子组件。

如果父组件类的实例必须读取或写入子组件的值，或者必须调用子组件的方法，就不能使用局部变量技术。

当父组件类需要这种访问时，我们将子组件作为ViewChild注入到父组件中。

我们将用同一个倒计时器示例来说明这种技术。我们不会改变它的外观或行为。子组件CountdownTimerComponent也保持不变。

我们仅为了演示目的，从局部变量切换到 ViewChild 技术。这里是父组件 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>倒计时发射 (通过 ViewChild)</h3>
  <button (click)="start()">开始</button>
  <button (click)="stop()">停止</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() {
    // 重新定义 `seconds()`，从 `CountdownTimerComponent.seconds` 获取 ...
    // 但先等一下，以避免一次性 devMode
    // 单向数据流违规错误
setTimeout(() => this.seconds = () => this.timerComponent.seconds, 0);
  }
start() { this.timerComponent.start(); }
  stop() { this.timerComponent.stop(); }
}
```

将子视图引入父组件类需要更多的工作。

我们导入了 ViewChild 装饰器和 AfterViewInit 生命周期钩子的引用。

我们通过 @ViewChild 属性装饰，将子 CountdownTimerComponent 注入到私有的 timerComponent 属性中。

template.

In this example, we wire parent buttons to the child's start and stop and use interpolation to display the child's seconds property.

Here we see the parent and child working together.

Parent calls a ViewChild

The local variable approach is simple and easy. But it is limited because the parent-child wiring must be done entirely within the parent template. The parent component itself has no access to the child.

We can't use the local variable technique if an instance of the parent component class must read or write child component values or must call child component methods.

When the parent component class requires that kind of access, we inject the child component into the parent as a ViewChild.

We'll illustrate this technique with the same Countdown Timer example. We won't change its appearance or behavior. The child CountdownTimerComponent is the same as well.

We are switching from the local variable to the ViewChild technique solely for the purpose of demonstration. Here is the parent, 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(); }
}
```

It takes a bit more work to get the child view into the parent component class.

We import references to the ViewChild decorator and the AfterViewInit lifecycle hook.

We inject the child CountdownTimerComponent into the private timerComponent property via the @ViewChild property decoration.

组件元数据中的 #timer 局部变量已被移除。取而代之的是，我们将按钮绑定到父组件自身的 start 和 stop 方法，并通过插值表达式显示父组件的 seconds 方法中的计时秒数。

这些方法直接访问被注入的计时器组件。

ngAfterViewInit 生命周期钩子是一个重要的细节。计时器组件直到 Angular 显示父视图后才可用。因此，我们最初显示 0 秒。

然后 Angular 调用 ngAfterViewInit 生命周期钩子，此时更新父视图中倒计时秒数的显示已经太晚。Angular 的单向数据流规则阻止我们在同一周期内更新父视图。我们必须等待一个周期后才能显示秒数。

我们使用 setTimeout 等待一个时钟周期，然后修改 seconds 方法，使其从计时器组件获取未来的值。

父组件和子组件通过服务进行通信

父组件及其子组件共享一个服务，该服务的接口支持家庭内部的双向通信。

服务实例的作用域是父组件及其子组件。该组件子树之外的组件无法访问该服务或其通信内容。

该 MissionService 连接 MissionControlComponent 与多个 AstronautComponent 子组件。

```
import { Injectable } from '@angular/core';
import { Subject }    from 'rxjs/Subject';
@Injectable()
export class MissionService {
  // 可观察的字符串源
private missionAnnouncedSource = new Subject<string>();
  private missionConfirmedSource = new Subject<string>();
  // 可观察的字符串流
missionAnnounced$ = this.missionAnnouncedSource.asObservable();
  missionConfirmed$ = this.missionConfirmedSource.asObservable();
  // 服务消息命令
announceMission(mission: string) {
    this.missionAnnouncedSource.next(mission);
  }
confirmMission(astronaut: string) {
    this.missionConfirmedSource.next(astronaut);
  }
}
```

MissionControlComponent 既提供它与子组件共享的服务实例（通过 providers 元数据数组），又通过构造函数将该实例注入自身：

```
import { Component }          from '@angular/core';
import { MissionService }     from './mission.service';
@Component({
selector: 'mission-control',
  template: `
    <h2>任务控制</h2>
    <button (click)="announce()">宣布任务</button>
    <my-astronaut *ngFor="let astronaut of astronauts"
      [astronaut]="astronaut">
    <my-astronaut>
```

The #timer local variable is gone from the component metadata. Instead we bind the buttons to the parent component's own start and stop methods and present the ticking seconds in an interpolation around the parent component's seconds method.

These methods access the injected timer component directly.

The ngAfterViewInit lifecycle hook is an important wrinkle. The timer component isn't available until after Angular displays the parent view. So we display 0 seconds initially.

Then Angular calls the ngAfterViewInit lifecycle hook at which time it is too late to update the parent view's display of the countdown seconds. Angular's unidirectional data flow rule prevents us from updating the parent view's in the same cycle. We have to wait one turn before we can display the seconds.

We use setTimeout to wait one tick and then revise the seconds method so that it takes future values from the timer component.

Parent and children communicate via a service

A parent component and its children share a service whose interface enables bi-directional communication within the family.

The scope of the service instance is the parent component and its children. Components outside this component subtree have no access to the service or their communications.

This MissionService connects the MissionControlComponent to multiple AstronautComponent children.

```
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);
  }
}
```

The MissionControlComponent both provides the instance of the service that it shares with its children (through the providers metadata array) and injects that instance into itself through its constructor:

```
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>历史</h3>
    <ul>
      <li *ngFor="let event of history">{{event}}</li>
    </ul>
`,
providers: [MissionService]
})
export class 任务控制组件 {
  宇航员 = ['洛威尔', '斯威格特', '海斯'];
  历史: string[] = [];
任务 = ['飞往月球！',
              '飞往火星！',
              '飞往拉斯维加斯！'];
下一个任务 = 0;
constructor(private missionService: MissionService) {
    missionService.missionConfirmed$.subscribe(
宇航员 => {
        this.history.push(`${astronaut} 确认了任务`);
      });
  }
announce() {
    let mission = this.missions[this.nextMission++];
    this.missionService.announceMission(mission);
    this.history.push(`任务 "${mission}" 已宣布`);
    if (this.nextMission >= this.missions.length) { this.nextMission = 0; }
  }
}
```

AstronautComponent 也在其构造函数中注入了该服务。每个 AstronautComponent 都是 MissionControlComponent 的子组件，因此会接收其父组件的服务实例：

```
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">
       确认
      </button>
    </p>
  `
})
export class 宇航员组件 implements OnDestroy {
  @Input() 宇航员: string;
mission = '<未宣布任务>';
  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;
    });
  }
```

```
    <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; }
  }
}
```

The AstronautComponent also injects the service in its constructor. Each AstronautComponent is a child of the MissionControlComponent and therefore receives its parent's service instance:

```
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() {
     // 防止组件销毁时内存泄漏
     this.subscription.unsubscribe();
  }
}
```

请注意，我们捕获了订阅并在 AstronautComponent 销毁时取消订阅。这是一个防止内存泄漏的步骤。这个应用中实际上没有风险，因为 AstronautComponent 的生命周期与应用本身的生命周期相同。但在更复杂的应用中情况未必如此。

我们没有在 MissionControlComponent 中添加此防护，因为作为父组件，它控制 MissionService 的生命周期。历史日志显示，消息在父组件
MissionControlComponent 和 AstronautComponent 子组件之间双向传递，这通过服务实现：

第3.2节：使用 @Input 和 @Output 属性的父子交互

我们有一个 DataListComponent，用于显示从服务获取的数据。DataListComponent 还有一个 PagerCo
mponent 作为其子组件。

PagerComponent 根据从 DataListComponent 获取的总页数创建页码列表。
PagerComponent 还通过 Output 属性通知 DataListComponent 用户何时点击了任意页码。

```
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>
    `
})
导出类 DataListComponent {
   私有 personsData = null;
   私有 pageCount: 数字;

   构造函数(私有 dataListService: DataListService) {
        变量 response = this.dataListService.getData(1); //从服务请求第一页数据
        this.personsData = response.persons;
        this.pageCount = response.totalCount / 10;//每页显示10条记录。
    }

页面改变(pageNumber: 数字){
        变量 response = this.dataListService.getData(pageNumber); //根据新页码从服务请求数据

        this.personsData = response.persons;
```

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

Notice that we capture the subscription and unsubscribe when the AstronautComponent is destroyed. This is a memory-leak guard step. There is no actual risk in this app because the lifetime of a AstronautComponent is the same as the lifetime of the app itself. That would not always be true in a more complex application.

We do not add this guard to the MissionControlComponent because, as the parent, it controls the lifetime of the MissionService. The History log demonstrates that messages travel in both directions between the parent MissionControlComponent and the AstronautComponent children, facilitated by the service:

Section 3.2: Parent - Child interaction using @Input & @Output properties

We have a DataListComponent that shows a data we pull from a service. DataListComponent also has a PagerComponent as it's child.

PagerComponent creates page number list based on total number of pages it gets from the DataListComponent. PagerComponent also lets the DataListComponent know when user clicks any page number via Output property.

```
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({
导入: [CommonModule],
    导出: [],
声明: [DataListComponent, PagerComponent],
    提供者: [DataListService],
})
导出类 DataListModule { }
```

PagerComponent 列出所有页码。我们在每个页码上设置点击事件，以便让父组件知道被点击的页码。

```
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
    }
}
```

第3.3节：使用ViewChild的父子组件交互

Viewchild 提供了从父组件到子组件的单向交互。当使用 ViewChild 时，子组件不会有反馈或输出。

我们有一个显示一些信息的数据列表组件（DataListComponent）。DataListComponent 有一个作为其子组件的分页组件（PagerComponent）。当用户在 DataListComponent 上进行搜索时，它会从服务获取数据，并请求 PagerComponent 根据新的页数刷新分页布局。

```
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()">搜索</button>
        <table>
        <tr *ngFor="let person of personsData">
            <td>{{person.name}}</td>
            <td>{{person.姓}}</td>
        </tr>
        </table>
```

```
    }
}

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

PagerComponent lists all the page numbers. We set click event on each of them so we can let the parent know about the clicked page number.

```
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
    }
}
```

Section 3.3: Parent - Child interaction using ViewChild

Viewchild offers one way interaction from parent to child. There is no feedback or output from child when ViewChild is used.

We have a DataListComponent that shows some information. DataListComponent has PagerComponent as it's child. When user makes a search on DataListComponent, it gets a data from a service and ask PagerComponent to refresh paging layout based on new number of pages.

```
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); //每页显示10条记录
    }
}

@NgModule({
imports: [CommonModule],
exports: [],
声明: [DataListComponent, PagerComponent],
   提供者: [DataListService],
})
导出类 DataListModule { }
```

通过这种方式，你可以调用子组件中定义的函数。

子组件在父组件渲染之前不可用。尝试在父组件的AfterViewInit生命周期钩子之前访问子组件会导致异常。

第3.4节：通过

service实现双向父子交互

用于通信的服务：

```
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);
    }
}
```

父组件：

```
        <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 { }
```

In this way you can call functions defined at child components.

Child component is not available until parent component is rendered. Attempting to access to the child before parents `AfterViewInit` life cyle hook will cause exception.

Section 3.4: Bidirectional parent-child interaction through a service

Service that is used for communication:

```
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);
    }
}
```

Parent component:

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

@Component({
selector: 'parent',
    template: `
    <button (click)="refreshSubsribed()">刷新</button>
    <h1>从子组件接收到的最后日期: {{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;
        });
    }

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

子组件：

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

@Component({
selector: 'child-component',
    template: `
    <h1>来自父组件的最后刷新时间: {{lastRefreshed}}</h1>
    <button (click)="sendNewDate()">发送新日期</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();
    }
}
```

应用模块：

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

@Component({
    selector: 'parent',
    template: `
    <button (click)="refreshSubsribed()">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;
        });
    }

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

Child component:

```
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({
声明: [
        父组件,
        子组件
     ],
提供者: [组件通信服务],
    启动: [应用组件] // 示例中未包含
})
export class AppModule {}
```

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

第4章：指令

第4.1节：*ngFor

form1.component.ts：

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

// 定义示例组件及其关联模板
@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>
`
})

// 创建一个类，用于所有函数、对象和变量
export class ExampleComponent {
    // 用于 *ngFor 遍历的水果数组
fruit = ['苹果', '橙子', '香蕉', '酸橙', '柠檬'];
}
```

输出：

```
<div>苹果</div>
<div>橙子</div>
<div>香蕉</div>
<div>青柠</div>
<div>柠檬</div>
<select required>
  <option value="Apples">苹果</option>
  <option value="Oranges">橙子</option>
  <option value="Bananas">香蕉</option>
  <option value="Limes">青柠</option>
  <option value="Lemons">柠檬</option>
</select>
```

在最简单的形式中，`*ngFor` 有两个部分：**`let`** >变量名 of **对象/数组**

以 `fruit = ['苹果', '橙子', '香蕉', '青柠', '柠檬'];`为例，

苹果、橙子等是数组 `fruit` 中的值。

[value]="f" 将等于 *ngFor 迭代的每个当前 fruit（f）。

与 AngularJS 不同，Angular2 没有继续使用 ng-options 来处理 <select>，也没有使用 ng-repeat 来处理所有其他一般的重复操作。

*ngFor 非常类似于 ng-repeat，只是语法略有不同。

参考资料：

Angular2 | 显示数据

Angular2 | ngFor

Chapter 4: Directives

Section 4.1: *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'];
}
```

Output:

```
<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>
```

In its most simple form, `*ngFor` has two parts : **`let`** >variableName of **object/array**

In the case of `fruit = ['Apples', 'Oranges', 'Bananas', 'Limes', 'Lemons'];`,

Apples, Oranges, and so on are the values inside the array `fruit`.

`[value]="f"` will be equal to each current `fruit` (f) that `*ngFor` has iterated over.

Unlike AngularJS, Angular2 has not continued with the use of `ng-options` for **`<select>`** and `ng-repeat` for all other general repetitions.

`*ngFor` is very similar to `ng-repeat` with slightly varied syntax.

References:

Angular2 | Displaying Data

Angular2 | ngFor

第4.2节：属性指令

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

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

<p [attr.data-note]="'这是 data-note 属性的值'">这里有很多文本</p>
```

第4.3节：组件是带模板的指令

```
import { Component } from '@angular/core';
@Component({
selector: 'my-app',
  template: `
    <h1>Angular 2 应用</h1>
    <p>组件是带有模板的指令</p>
  `
})
export class AppComponent {
}
```

第4.4节：结构型指令

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

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

第4.5节：自定义指令

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

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

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

用法：

```
<p green>这里有很多绿色文本</p>
```

第4.6节：复制到剪贴板指令

在本例中，我们将创建一个指令，通过点击元素将文本复制到剪贴板

copy-text.directive.ts

```
import {
    Directive,
```

Section 4.2: Attribute directive

```
<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>
```

Section 4.3: Component is a directive with template

```
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 {
}
```

Section 4.4: Structural directives

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

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

Section 4.5: Custom directive

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

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

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

Usage:

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

Section 4.6: Copy to Clipboard directive

In this example we are going to create a directive to copy a text into the clipboard by clicking on an element

copy-text.directive.ts

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

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

    // 将属性值解析为 'text' 变量
    @Input('text-copy') text:string;

constructor() {
    }


    // HostListener 将监听点击事件并运行以下函数，HostListener 支持其他标准事件，如 mouseenter、mouseleave 等
。
    @HostListener('click') copyText() {

        // 我们需要在 DOM 中创建一个包含要复制文本的虚拟 textarea
        var textArea = document.createElement("textarea");

        // 隐藏 textarea，防止其实际显示
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';

        // 将 textarea 的内容设置为我们在 [text-copy] 属性中定义的值
textArea.value = this.text;
        document.body.appendChild(textArea);

        // 这将选中 textarea
textArea.select();

        try {
            // 大多数现代浏览器支持 execCommand('copy'|'cut'|'paste')，如果不支持则
应抛出错误
            var successful = document.execCommand('copy');
            var msg = successful ? '成功' : '失败';
            // 让用户知道文本已被复制，例如使用 toast、alert 等提示
console.log(msg);
        } catch (err) {
            // 告诉用户复制不被支持，并提供替代方案，例如弹窗显示需复制的文本

console.log('无法复制');
        }

        // 最后我们从 DOM 中移除 textarea 元素
document.body.removeChild(textArea);
    }
}

export const TEXT_COPY_DIRECTIVES = [TextCopyDirective];
```

```
    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];
```

some-page.component.html

请记得将 TEXT_COPY_DIRECTIVES 注入到组件的 directives 数组中

```
...
    <!-- 将变量作为属性的值插入，let textToBeCopied = 'http://facebook.com/' -->
    <button [text-copy]="textToBeCopied">复制 URL</button>
    <button [text-copy]="'https://www.google.com/'">复制 URL</button>
...
```

第4.7节：测试自定义指令

给定一个在鼠标事件时高亮文本的指令

```
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 || '红色');
  }

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

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

可以这样测试

```
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>红色文本</span>
      <span id="green" [appHighlight]="'green'">绿色文本</span>
      <span id="no">无颜色</span>
    </div>
  `
})
class ContainerComponent { }

const mouseEvents = {
  get enter() {
    const mouseenter = document.createEvent('MouseEvent');
```

some-page.component.html

Remember to inject TEXT_COPY_DIRECTIVES into the directives array of your component

```
...
    <!-- 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>
...
```

Section 4.7: Testing a custom directive

Given a directive that highlights text on mouse events

```
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;
  }
}
```

It can be tested like this

```
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 ("当鼠标离开且指令无参数时，应将背景颜色设置为空",  ()
=> {
    const targetElement = <HTMLSpanElement>element.querySelector('#red');

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

it('当鼠标离开时，带有参数的指令应将背景色设置为空', () =>
{
    const targetElement = <HTMLSpanElement>element.querySelector('#green');

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

it('当未传递参数时，应将背景色设置为红色', () => {
    const targetElement = <HTMLSpanElement>element.querySelector('#red');

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

it('传递绿色参数时，应将背景色设置为绿色', () => {
    const targetElement = <HTMLSpanElement>element.querySelector('#green');

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

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

第5章：页面标题

如何更改页面标题

第5.1节：更改页面标题

1.首先我们需要提供标题服务。
2.使用 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('页面标题已更改');
   }
}
```

Chapter 5: Page title

How can you change the title of the page

Section 5.1: changing the page title

1. First we need to provide Title service.
2. Using 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');
   }
}
```

第6章：模板

模板与Angular 1中的模板非常相似，尽管有许多小的语法变化，使得发生的事情更加清晰。

第6.1节：Angular 2 模板

一个简单的模板

让我们从一个非常简单的模板开始，展示我们的名字和我们喜欢的东西：

```
<div>
你好，我的名字是 {{name}}，我非常喜欢 {{thing}}。
</div>
```

{}： 渲染

要渲染一个值，我们可以使用标准的双大括号语法：

```
我的名字是 {{name}}
```

管道，之前称为“过滤器”，将一个值转换成一个新值，比如本地化字符串或将浮点数值转换成货币表示：

[]： 绑定属性

要解析并绑定变量到组件，使用 [] 语法。如果我们的组件中有 this.currentVolume，
我们将通过组件传递它，值将保持同步：

```
<video-control [volume]="currentVolume"></video-control>
(): 事件处理
```

(): 事件处理 监听组件上的事件时，我们使用 () 语法

```
<my-component (click)="onClick($event)"></my-component>
```

[()]: 双向数据绑定

为了根据用户输入和其他事件保持绑定的更新，使用 [()] 语法。可以将其视为事件处理和属性绑定的结合：

<input [(ngModel)]="myName"> 你的组件中的 this.myName 值将与输入值保持同步。

*: 星号

表示该指令将此组件视为模板，而不会按原样渲染它。例如，ngFor 会遍历 items 中的每个项目并生成对应内容，但它从不渲染我们最初的，因为它是一个模板：

```
<my-component *ngFor="#item of items">
</my-component>
```

其他类似的指令是作用于模板而非渲染组件的 *ngIf 和 *ngSwitch。

Chapter 6: Templates

Templates are very similar to templates in Angular 1, though there are many small syntactical changes that make it more clear what is happening.

Section 6.1: Angular 2 Templates

A SIMPLE TEMPLATE

Let's start with a very simple template that shows our name and our favorite thing:

```
<div>
  Hello my name is {{name}} and I like {{thing}} quite a lot.
</div>
```

{}: RENDERING

To render a value, we can use the standard double-curly syntax:

```
My name is {{name}}
```

Pipes, previously known as "Filters," transform a value into a new value, like localizing a string or converting a floating point value into a currency representation:

[]: BINDING PROPERTIES

To resolve and bind a variable to a component, use the [] syntax. If we have this.currentVolume in our component, we will pass this through to our component and the values will stay in sync:

```
<video-control [volume]="currentVolume"></video-control>
(): HANDLING EVENTS
```

(): HANDLING EVENTS To listen for an event on a component, we use the () syntax

```
<my-component (click)="onClick($event)"></my-component>
```

[()]: TWO-WAY DATA BINDING

To keep a binding up to date given user input and other events, use the [()] syntax. Think of it as a combination of handling an event and binding a property:

<input [(ngModel)]="myName"> The this.myName value of your component will stay in sync with the input value.

*: THE ASTERISK

Indicates that this directive treats this component as a template and will not draw it as-is. For example, ngFor takes our and stamps it out for each item in items, but it never renders our initial since it's a template:

```
<my-component *ngFor="#item of items">
</my-component>
```

Other similar directives that work on templates rather than rendered components are *ngIf and *ngSwitch.

第7章：常用内置指令和服务

@angular/common - 常用指令和服务 @angular/core - Angular核心框架

第7.1节：Location类

Location 是一个服务，应用程序可以用它与浏览器的URL进行交互。根据使用的LocationStrategy，Location要么将持久化到URL的路径，要么持久化到URL的哈希段。

Location负责根据应用程序的基础href规范化URL。

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

@Component({
selector: 'app-component'
})
class AppCmp {

constructor(_location: Location) {

    //将浏览器的URL更改为给定URL的规范化版本，
    //并将新项推入平台的历史记录中。
_location.go('/foo');

  }

backClicked() {
    // 在平台的历史记录中后退。
    this._location.back();
  }

forwardClicked() {
    // 在平台的历史记录中前进。
    this._location.back();
  }
}
```

第7.2节：AsyncPipe

async管道订阅一个Observable或Promise，并返回它发出的最新值。当发出新值时，async管道会标记组件以检查更改。当组件被销毁时，async管道会自动取消订阅，以避免潜在的内存泄漏。

```
@Component({
selector: 'async-observable-pipe',
template: '<div><code>observable|async</code>: 时间: {{ time | async }}</div>'
})
导出类 AsyncObservablePipeComponent {
  time = new Observable<string>((observer: Subscriber<string>) => {
    setInterval(() => observer.next(new Date().toString()), 1000);
  });
}
```

Chapter 7: Commonly built-in directives and services

@angular/common - commonly needed directives and services @angular/core - the angular core framework

Section 7.1: Location Class

Location is a service that applications can use to interact with a browser's URL. Depending on which LocationStrategy is used, Location will either persist to the URL's path or the URL's hash segment.

Location is responsible for normalizing the URL against the application's base href.

```
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();
  }
}
```

Section 7.2: AsyncPipe

The async pipe subscribes to an Observable or Promise and returns the latest value it has emitted. When a new value is emitted, the async pipe marks the component to be checked for changes. When the component gets destroyed, the async pipe unsubscribes automatically to avoid potential memory leaks.

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

第7.3节：显示项目中使用的当前Angular 2版本

要显示当前版本，我们可以使用来自 @angular/core 包的VERSION。

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

@Component({
selector: 'my-app',
  template: `<h1>Hello {{name}}</h1>
  <h2>当前版本: {{ver}}</h2>
  `,
})
export class AppComponent  {
   name = 'Angular2';
ver = VERSION.full;
}
```

第7.4节：货币管道

货币管道允许你将数据作为普通数字处理，但在视图中以标准货币格式（货币符号、小数位数等）显示。

```
@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;
}
```

该管道接受三个可选参数：

- currencyCode: 允许您指定 ISO 4217 货币代码。
- **symbolDisplay**: 布尔值，指示是否使用货币符号
- **digitInfo**: 允许您指定小数位的显示方式。

关于货币管道的更多文档：
https://angular.io/docs/ts/latest/api/common/index/CurrencyPipe-pipe.html

Section 7.3: Displaying current Angular 2 version used in your project

To display current version, we can use **VERSION** from @angular/core package.

```
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;
}
```

Section 7.4: Currency Pipe

The currency pipe allows you to work with you data as regular numbers but display it with standard currency formatting (currency symbol, decimal places, etc.) in the view.

```
@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;
}
```

The pipe takes three optional parameters:

- **currencyCode**: Allows you to specify the ISO 4217 currency code.
- **symbolDisplay**: Boolean indicating whether to use the currency symbol
- **digitInfo**: Allows you to specify how the decimal places should be displayed.

More documentation on the currency pipe:
https://angular.io/docs/ts/latest/api/common/index/CurrencyPipe-pipe.html

第8章：指令与组件：@Input @Output

第8.1节：Angular 2 中嵌套组件的 @Input 和 @Output

一个按钮指令，接受一个 @Input() 用于指定点击次数限制，达到限制后按钮将被禁用。父组件可以监听一个事件，当点击次数达到限制时通过 @Output 发出该事件：

```
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);
        }
    }
}
```

使用 Button 指令的父组件，当点击次数达到限制时弹出提示消息：

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

@Component({
selector: '我的父组件',
    template: `<limited-button [clickLimit]="2"
                                   (limitReached)="onLimitReached($event)">
                 你只能点击我两次
               </limited-button>`,
directives: [LimitedButton]
})

export class 我的父组件 {
onLimitReached(clickCount: number) {
        alert('按钮点击 ' + clickCount + ' 次后被禁用.');
    }
}
```

Chapter 8: Directives & components : @Input @Output

Section 8.1: Angular 2 @Input and @Output in a nested component

A Button directive which accepts an `@Input()` to specify a click limit until the button gets disabled. The parent component can listen to an event which will be emitted when the click limit is reached via `@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);
        }
    }
}
```

Parent component which uses the Button directive and alerts a message when the click limit is reached:

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

第8.2节：输入示例

@input 用于在组件之间绑定数据

首先，在你的组件中导入它

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

然后，将输入作为组件类的属性添加

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

假设你的组件选择器是 'car-component'，调用组件时添加属性
'car'

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

现在您的汽车可以作为对象中的一个属性访问（this.car）

完整示例：

1. car.entity.ts

```
    导出类 CarEntity {
构造函数(public brand : string, public color : string) {
      }
    }
```

2. car.component.ts

```
    导入 { Component, Input } 来自 '@angular/core';
    导入 {CarEntity} 来自 "./car.entity";

    @Component({
选择器: 'car-component',
      模板: require('./templates/car.html'),
    })

    导出类 CarComponent {
        @Input() car: CarEntity;

构造函数() {
            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 {
```

Section 8.2: Input example

@input is useful to bind data between components

First, import it in your component

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

Then, add the input as a property of your component class

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

Let's say that the selector of your component is 'car-component', when you call the component, add the attribute 'car'

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

Now your car is accessible as an attribute in your object (this.car)

Full Example :

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('雷诺', '蓝色');
            var carTwo : CarEntity = new CarEntity('菲亚特', '绿色');
            var carThree : CarEntity = new CarEntity('雪铁龙', '黄色');
            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>
```

第8.3节：Angular 2 @Input 与异步数据

有时你需要异步获取数据，然后再传递给子组件使用。如果子组件在数据尚未接收之前尝试使用它，就会抛出错误。你可以使用ngOnChanges来检测组件的@Input的变化，并在它们被定义后再进行操作。

带有异步调用端点的父组件

```
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>
    `
})
导出类 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 || { };
    }
```

```
        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>
```

Section 8.3: Angular 2 @Input with asynchronous data

Sometimes you need to fetch data asynchronously before passing it to a child component to use. If the child component tries to use the data before it has been received, it will throw an error. You can use `ngOnChanges` to detect changes in a components' `@Inputs` and wait until they are defined before acting upon them.

Parent component with async call to an endpoint

```
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);
    }
}
```

具有异步数据作为输入的子组件

该子组件以异步数据作为输入。因此，它必须等待数据存在后才能使用。
我们使用 ngOnChanges，当组件的输入发生变化时触发，检查数据是否存在，若存在则使用它。
请注意，如果依赖传入数据的属性不为真，子组件的模板将不会显示。

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

@Component ({
selector : 'child-component',
    template : `
        <p *ngIf="doesDataExist">你好，子组件</p>
    `
})
导出类 ChildComponent {

    doesDataExist: boolean = false;

    @Input('data') data : any;

    // 每当组件的 @Inputs 发生变化时运行
ngOnChanges () {
        // 使用数据前先检查数据是否存在
        if (this.data) {
            this.useData(data);
        {
    }

    // 设计示例，将数据赋值给 reliesOnData
useData (data) {
        this.doesDataExist = true;
    }
}
```

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

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

Child component which has async data as input

This child component takes the async data as input. Therefore it must wait for the data to exist before Using it. We use ngOnChanges which fires whenever a component's input changes, check if the data exists and use it if it does. Notice that the template for the child will not show if a property that relies on the data being passed in is not true.

```
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;
    }
}
```

第9章：使用@HostBinding装饰器将属性指令应用于宿主节点的属性值。

第9.1节：@HostBinding

@HostBinding 装饰器允许我们以编程方式设置指令宿主元素上的属性值。它的工作方式类似于模板中定义的属性绑定，但它专门针对宿主元素。该绑定会在每个变更检测周期中进行检查，因此如果需要，可以动态更改。例如，假设我们想创建一个指令，用于按钮在按下时动态添加一个类。代码可能如下所示：

```
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;
  }
}
```

请注意，对于 @HostBinding 的两种用例，我们都传入了一个字符串值，表示我们想要影响的属性。如果我们不向装饰器提供字符串，则会使用类成员的名称代替。在第一个 @HostBinding 中，我们静态地将 role 属性设置为 button。对于第二个示例，当 isPressed 为 true 时，将应用 pressed 类。

Chapter 9: Attribute directives to affect the value of properties on the host node by using the @HostBinding decorator.

Section 9.1: @HostBinding

The @HostBinding decorator allows us to programmatically set a property value on the directive's host element. It works similarly to a property binding defined in a template, except it specifically targets the host element. The binding is checked for every change detection cycle, so it can change dynamically if desired. For example, lets say that we want to create a directive for buttons that dynamically adds a class when we press on it. That could look something like:

```
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;
  }
}
```

Notice that for both use cases of @HostBinding we are passing in a string value for which property we want to affect. If we don't supply a string to the decorator, then the name of the class member will be used instead. In the first @HostBinding, we are statically setting the role attribute to button. For the second example, the pressed class will be applied when isPressed is true

第10章：如何使用ngif

*NgIf：它根据表达式的求值结果移除或重新创建DOM树的一部分。它是一种结构型指令
，结构型指令通过添加、替换和移除元素来改变DOM的布局。

第10.1节：使用*ngIf在*ngFor循环开始或结束时运行函数

NgFor提供一些可以别名为局部变量的值

- index -（变量）当前项在可迭代对象中的位置，从0开始
- first -（布尔值）如果当前项是可迭代对象中的第一项，则为true
- last -（布尔值）如果当前项是可迭代对象中的最后一项，则为true
- even -（布尔值）如果当前索引是偶数，则为true
- odd -（布尔值）如果当前索引是奇数，则为true

```
<div *ngFor="let note of csvdata; let i=index; let lastcall=last">
      <h3>{{i}}</h3> <!-- 显示索引位置
     <h3>{{note}}</h3>
      <span *ngIf="lastcall">{{anyfunction()}} </span><-- 这个 lastcall 布尔值只有在循环的最后一个时才为真

    // anyfunction() 将在循环结束时运行，同样我们也可以在开始时运行
   </div>
```

第10.2节：显示加载消息

如果我们的组件尚未准备好并且正在等待来自服务器的数据，那么我们可以使用 *ngIf 添加加载器。步骤：

首先声明一个布尔值：

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

接下来，在您的组件中添加一个名为ngOnInit的生命周期钩子

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

当你从服务器获取完整数据后，将加载状态布尔值设为假。

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

在你的 HTML 模板中使用带有loading属性的 *ngIf：

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

第10.3节：根据条件显示警告信息

```
<p class="alert alert-success" *ngIf="names.length > 2">当前名字数量超过2个！</p>
```

Chapter 10: How to Use ngif

***NgIf**: It removes or recreates a part of DOM tree depending on an expression evaluation. It is a structural directive and structural directives alter the layout of the DOM by adding, replacing and removing its elements.

Section 10.1: To run a function at the start or end of *ngFor loop Using *ngIf

NgFor provides Some values that can be aliased to local variables

- **index** -(variable) position of the current item in the iterable starting at 0
- **first** -(boolean) true if the current item is the first item in the iterable
- **last** -(boolean) true if the current item is the last item in the iterable
- **even** -(boolean) true if the current index is an even number
- **odd** -(boolean) true if the current index is an odd number

```
<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>
```

Section 10.2: Display a loading message

If our component is not ready and waiting for data from server, then we can add loader using *ngIf. **Steps:**

First declare a boolean:

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

Next, in your component add a lifecycle hook called `ngOnInit`

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

and after you get complete data from server set you loading boolean to false.

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

In your html template use *ngIf with the `loading` property:

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

Section 10.3: Show Alert Message on a condition

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

第10.4节：将 *ngIf 与 *ngFor 一起使用

虽然不允许在同一个 div 中同时使用*ngIf和*ngFor（运行时会报错），但你可以将*ngIf嵌套在*ngFor中以实现所需的行为。

示例1：通用语法

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

   <!-- 如果条件为真，则执行此处代码 -->

  </div>
</div>
```

示例 2：显示偶数索引的元素

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

缺点是需要添加一个额外的外层div元素。

但考虑这样一种用例：需要对一个div元素进行迭代（使用 *ngFor），同时还要检查该元素是否需要被移除（使用 *ngIf），但不希望添加额外的div。在这种情况下，可以使用template标签来实现 *ngFor：

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

这样就不需要添加额外的外层div，而且<template>元素也不会被添加到DOM中。上述示例中，唯一被添加到DOM中的元素是被迭代的div元素。

注意：在 Angular v4 中，<template>已被弃用，改用<ng-template>，并将在 v5 中移除。在 Angular v2.x 版本中，<template>仍然有效。

Section 10.4: Use *ngIf with*ngFor

While you are not allowed to use *ngIf and *ngFor in the same div (it will gives an error in the runtime) you can nest the *ngIf in the *ngFor to get the desired behavior.

Example 1: General syntax

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

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

  </div>
</div>
```

Example 2: Display elements with even index

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

The downside is that an additional outer div element needs to be added.

But consider this use case where a div element needs to be iterated (using *ngFor) and also includes a check whether the element need to be removed or not (using *ngIf), but adding an additional div is not preferred. In this case you can use the template tag for the *ngFor:

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

This way adding an additional outer div is not needed and furthermore the **<template>** element won't be added to the DOM. The only elements added in the DOM from the above example are the iterated div elements.

Note: In Angular v4 **<template>** has been deprecated in favour of **<ng-template>** and will be removed in v5. In Angular v2.x releases **<template>** is still valid.

第11章：如何使用ngfor

Angular2 使用 ngFor 指令为可迭代对象中的每个项目实例化一次模板。该指令将可迭代对象绑定到 DOM，因此如果可迭代对象的内容发生变化，DOM 的内容也会相应变化。

第11.1节：带管道的 *ngFor

```
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)
}
```

第11.2节：无序列表示例

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

第11.3节：更复杂的模板示例

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

第11.4节：跟踪当前交互示例

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

在这种情况下，我将取索引的值，即当前循环的迭代次数。

Chapter 11: How to use ngfor

The ngFor directive is used by Angular2 to instantiate a template once for every item in an iterable object. This directive binds the iterable to the DOM, so if the content of the iterable changes, the content of the DOM will be also changed.

Section 11.1: *ngFor with pipe

```
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)
}
```

Section 11.2: Unordered list example

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

Section 11.3: More complext template example

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

Section 11.4: Tracking current interaction example

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

In this case, i will take the value of index, which is the current loop iteration.

第11.5节：Angular 2 别名导出值

Angular2 提供了几个可以别名为本地变量的导出值。这些是：

- index
- first
- last
- even
- odd

除了index，其他的都接受一个Boolean值。和之前使用 index 的例子一样，可以使用这些导出值中的任意一个：

```
<div *ngFor="let item of items; let firstItem = first; let lastItem = last">
  <p *ngIf="firstItem">我是第一个项目，我会被显示</p>
  <p *ngIf="firstItem">我不是第一个项目，我不会显示 :(</p>
  <p *ngIf="lastItem">但我是最后一个项目，我会被显示 :)</p>
</div>
```

Section 11.5: Angular 2 aliased exported values

Angular2 provides several exported values that can be aliased to local variables. These are:

- index
- first
- last
- even
- odd

Except index, the other ones take a Boolean value. As the previous example using index, it can be used any of these exported values:

```
<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>
```

第12章：Angular - For 循环

第12.1节：NgFor - 标记循环

NgFor 指令为可迭代对象中的每个项目实例化一次模板。每个实例化模板的上下文继承自外部上下文，并将给定的循环变量设置为来自可迭代对象的当前项目。

为了自定义默认的跟踪算法，NgFor 支持trackBy选项。 trackBy 接受一个有两个参数的函数：索引和项。如果提供了trackBy，Angular 会通过该函数的返回值来跟踪变化。

```
<li *ngFor="let item of items; let i = index; trackBy: trackByFn">
    {{i}} - {{item.name}}
</li>
```

附加选项: NgFor 提供了几个导出值，可以别名为局部变量：

- index 将被设置为每个模板上下文的当前循环迭代索引。
- first 将被设置为一个布尔值，表示该项是否为迭代中的第一个。
- last 将被设置为一个布尔值，表示该项是否为迭代中的最后一个。
- even 将被设置为一个布尔值，表示该项的索引是否为偶数。
- odd 将被设置为一个布尔值，表示该项的索引是否为奇数。

第12.2节：带组件的*ngFor

```
   @Component({
selector: '主组件',
     template: '<示例组件
                   *ngFor="let hero of heroes"
                [hero]="hero"></示例组件>'
   })


   @Component({
selector: 'example-component',
      template: '<div>{{hero?.name}}</div>'
   })

   export class ExampleComponent {
     @Input() hero : Hero = null;
   }
```

第12.3节：Angular 2 for循环

实时 plnkr 点击...

```
<!doctype html>
<html>
<head>
    <title>Angular 2 中使用 ES5 的 ng for 循环。</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 = "Angular 2 中使用 ES5 的 ng for 循环。";
            this.users = ["Anil Singh", "Sunil Singh", "Sushil Singh", "Aradhya", 'Reena'];
        };
```

Chapter 12: Angular - ForLoop

Section 12.1: NgFor - Markup For Loop

The **NgFor** directive instantiates a template once per item from an iterable. The context for each instantiated template inherits from the outer context with the given loop variable set to the current item from the iterable.

To customize the default tracking algorithm, NgFor supports **trackBy** option. **trackBy** takes a function which has two arguments: index and item. If **trackBy** is given, Angular tracks changes by the return value of the function.

```
<li *ngFor="let item of items; let i = index; trackBy: trackByFn">
    {{i}} - {{item.name}}
</li>
```

Additional Options: NgFor provides several exported values that can be aliased to local variables:

- **index** will be set to the current loop iteration for each template context.
- **first** will be set to a boolean value indicating whether the item is the first one in the iteration.
- **last** will be set to a boolean value indicating whether the item is the last one in the iteration.
- **even** will be set to a boolean value indicating whether this item has an even index.
- **odd** will be set to a boolean value indicating whether this item has an odd index.

Section 12.2: *ngFor with component

```
   @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;
   }
```

Section 12.3: Angular 2 for-loop

For live plnkr click...

```
<!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> 用户列表 : </p>' +
                                   '<ul>' +
'<li *ng-for="let user of users">' +
                                   '{{user}}' +
'</li>' +
                                          '</ul>',
                         directives: [angular.NgFor]
                    })
            ];

document.addEventListener("DOMContentLoaded", function () {
               angular.bootstrap(ngForLoop);
            });
      </script>
</head>
<body>
      <ngforloop></ngforloop>
      <h2>
         <a href="http://www.code-sample.com/" target="_blank">更多详情...</a>
      </h2>
</body>
</html>
```

第12.4节：*ngFor 每行显示X个项目

示例显示每行5个项目：

```
<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>
```

第12.5节：表格行中的 *ngFor

```
<table>
    <thead>
        <th>名称</th>
        <th>索引</th>
    </thead>
    <tbody>
        <tr *ngFor="let hero of heroes">
            <td>{{hero.name}}</td>
        </tr>
    </tbody>
</table>
```

```
        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>
<body>
    <ngforloop></ngforloop>
    <h2>
      <a href="http://www.code-sample.com/" target="_blank">For more detail...</a>
    </h2>
</body>
</html>
```

Section 12.4: *ngFor X amount of items per row

Example shows 5 items per row:

```
<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>
```

Section 12.5: *ngFor in the Table Rows

```
<table>
    <thead>
        <th>Name</th>
        <th>Index</th>
    </thead>
    <tbody>
        <tr *ngFor="let hero of heroes">
            <td>{{hero.name}}</td>
        </tr>
    </tbody>
</table>
```

第13章：模块

Angular模块是应用程序不同部分的容器。

你可以有嵌套模块，你的`app.module`实际上已经嵌套了其他模块，比如`BrowserModule`
你还可以添加`RouterModule`等等。

第13.1节：一个简单的模块

模块是带有@NgModule装饰器的类。要创建一个模块，我们添加@NgModule并传入一些参数：

- bootstrap：将作为应用程序根组件的组件。此配置仅存在于你的根模块中
- declarations：模块声明的资源。当你添加一个新组件时，必须更新declarations（ng generate component 会自动完成此操作）
- exports：模块导出的资源，可以在其他模块中使用
- imports：模块从其他模块使用的资源（只接受模块类）
- 提供者：可以注入（依赖注入）到组件中的资源

一个简单的示例：

```
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 { }
```

第13.2节：模块嵌套

可以通过@NgModule装饰器的imports参数来嵌套模块。

我们可以在应用中创建一个core.模块，包含通用内容，比如一个ReservePipe（一个反转字符串的管道），并将它们打包到该模块中：

```
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { ReversePipe } from '../reverse.pipe';

@NgModule({
imports: [
    CommonModule
  ],
exports: [ReversePipe], // 导出内容以供其他模块导入
  declarations: [ReversePipe],
})
export class CoreModule { }
```

然后在app.module中：

Chapter 13: Modules

Angular modules are containers for different parts of your app.

You can have nested modules, your `app.module` is already actually nesting other modules such as `BrowserModule` and you can add `RouterModule` and so on.

Section 13.1: A simple module

A module is a class with the `@NgModule` decorator. To create a module we add `@NgModule` passing some parameters:

- `bootstrap`: The component that will be the root of your application. This configuration is only present on your root module
- `declarations`: Resources the module declares. When you add a new component you have to update the declarations (`ng generate component` does it automatically)
- `exports`: Resources the module exports that can be used in other modules
- `imports`: Resources the module uses from other modules (only module classes are accepted)
- `providers`: Resources that can be injected (di) in a component

A simple example:

```
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 { }
```

Section 13.2: Nesting modules

Modules can be nested by using the `imports` parameter of `@NgModule` decorator.

We can create a `core.module` in our application that will contain generic things, like a `ReservePipe` (a pipe that reverse a string) and bundle those in this module:

```
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 { }
```

Then in the `app.module`:

```
import { CoreModule } from 'app/core/core.module';

@NgModule({
declarations: [...], // ReversePipe 可用，无需在此声明
                       // 因为 CoreModule 已导出它
imports: [
CoreModule,       // 从 CoreModule 导入内容
    ...
  ],
providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

```
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 { }
```

第14章：管道

函数/参数	解释
@Pipe({name, pure})	管道的元数据，必须紧跟在管道类之前
名称：字符串	你将在模板中使用的名称
纯净：布尔值	默认为 true，将此标记为 false 可使管道更频繁地重新计算
transform(value, args[]?)	用于在模板中转换值的函数
value: *任意类型*	你想要转换的值
参数：任意数组	你可能需要包含在转换中的参数。使用 ? 操作符标记可选参数，如 transform(value, arg 1, arg2?)

管道符号 | 用于在 Angular 2 中应用管道。管道与 AngularJS 中的过滤器非常相似，它们都帮助将数据转换为指定格式。

第14.1节：自定义管道

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; // 在此实现你的转换逻辑
    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,
  ],
```

Chapter 14: Pipes

Function/Parameter	Explanation
@Pipe({name, pure})	metadata for pipe, must immediately precede pipe class
name: *string*	what you will use inside the template
pure: *boolean*	defaults to true, mark this as false to have your pipe re-evaluated more often
transform(value, args[]?)	the function that is called to transform the values in the template
value: *any*	the value that you want to transform
args: *any[]*	the arguments that you may need included in your transform. Mark optional args with the ? operator like so transform(value, arg1, arg2?)

The pipe | character is used to apply pipes in Angular 2. Pipes are very similar to filters in AngularJS in that they both help to transform the data into a specified format.

Section 14.1: Custom Pipes

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 { }
```

第14.2节：内置管道

Angular2自带一些内置管道：

管道	用法	示例
DatePipe（日期管道）	date（日期）	`{{ dateObj | date }} // 输出为 '2015年6月15日'`
UpperCasePipe	大写	`{{ value | uppercase }} // 输出为 'SOMETEXT'`
LowerCasePipe	小写	`{{ value | lowercase }} // 输出为 'sometext'`
CurrencyPipe	货币	`{{ 31.00 | currency:'USD':true }} // 输出为 '$31'`
PercentPipe（百分比管道）	percent（百分比）	`{{ 0.03 | percent }} //输出为 %3`

还有其他的。请在 here 查找他们的文档。

示例

hotel-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;
}
```

hotel-reservation.template.html

```
<div>
    <h1>欢迎回来 {{fName | uppercase}} {{lName | lowercase}}</h1>
    <p>
您于{reservationMade | date} {reservationMade | date:'shortTime'}预订了205号房，入住日期为{re
servationDate | date}，总费用为{cost | currency}。

    </p>
</div>
```

输出

```
欢迎回来，JOE schmo
2016年6月26日7:18，您预订了205号房间，入住日期为2025年11月14日，总费用为
$99.99。
```

第14.3节：管道链式调用

管道可以链式调用。

```
<p>今天是{{ today | date:'fullDate' | uppercase}}。</p>
```

```
  declarations: [
    MyComponent,
    MyPipe
  ],
})
export class MyModule { }
```

Section 14.2: Built-in Pipes

Angular2 comes with a few built-in pipes:

Pipe	Usage	Example
DatePipe	date	`{{ dateObj | date }} // output is 'Jun 15, 2015'`
UpperCasePipe	uppercase	`{{ value | uppercase }} // output is 'SOMETEXT'`
LowerCasePipe	lowercase	`{{ value | lowercase }} // output is 'sometext'`
CurrencyPipe	currency	`{{ 31.00 | currency:'USD':true }} // output is '$31'`
PercentPipe	percent	`{{ 0.03 | percent }} //output is %3`

There are others. Look here for their documentation.

Example

hotel-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;
}
```

hotel-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>
```

Output

```
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.
```

Section 14.3: Chaining Pipes

Pipes may be chained.

```
<p>Today is {{ today | date:'fullDate' | uppercase}}.</p>
```

第14.4节：使用JsonPipe调试

JsonPipe可用于调试任何给定内部状态。

代码

```
@Component({
selector: 'json-example',
  template: `<div>
    <p>无 JSON 管道:</p>
    <pre>{{object}}</pre>
    <p>使用 JSON 管道:</p>
    <pre>{{object | json}}</pre>
  </div>`
})
导出类 JsonPipeExample {
object: Object = {foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}};
}
```

输出

```
无 JSON 管道:
object
使用 JSON 管道:
{object:{foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}}
```

第14.5节：动态管道

使用场景：一个表格视图包含不同列，不同列的数据格式不同，需要用不同的管道进行转换。

table.component.ts

```
...
import { DYNAMIC_PIPES } from '../pipes/dynamic.pipe.ts';

@Component({
    ...
    pipes: [DYNAMIC_PIPES]
})
export class TableComponent {
   ...

    // 每列使用的管道
table.pipes = [ null, null, null, 'humanizeDate', 'statusFromBoolean' ],
    table.header = [ 'id', 'title', 'url', 'created', 'status' ],
table.rows = [
        [ 1, '首页', 'home', '2016-08-27T17:48:32', true ],
        [ 2, '关于我们', 'about', '2016-08-28T08:42:09', true ],
        [ 4, '联系我们', 'contact', '2016-08-28T13:28:18', false ],
        ...
    ]
    ...

}
```

dynamic.pipe.ts

```
import {
    Pipe,
```

Section 14.4: Debugging With JsonPipe

The JsonPipe can be used for debugging the state of any given internal.

Code

```
@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]}};
}
```

Output

```
Without JSON Pipe:
object
With JSON pipe:
{object:{foo: 'bar', baz: 'qux', nested: {xyz: 3, numbers: [1, 2, 3, 4, 5]}}
```

Section 14.5: Dynamic Pipe

Use case scenario: A table view consists of different columns with different data format that needs to be transformed with different pipes.

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
} 来自 '@angular/core';
// 本示例中用于将日期人性化的库
import * 作为 moment 从 'moment' 导入；

@Pipe({name: 'dynamic'})
导出类 DynamicPipe 实现 PipeTransform {

    transform(value:string, modifier:string) {
        如果 (!modifier) 返回 value;
        // 评估管道字符串
        返回 eval('this.' + modifier + '(\'' + value + '\')')
    }

    // 根据输入值返回 'enabled' 或 'disabled'
    statusFromBoolean(value:string):string {
        切换 (value) {
            情况 'true':
            情况 '1':
                返回 'enabled';
            默认:
                return 'disabled';
        }
    }

    // 返回人性化的时间格式，例如：'14分钟前'，'昨天'
    humanizeDate(value:string):string {
        // 如果日期差在一周内则人性化显示，否则返回 '2016年12月20日'

        if (moment().diff(moment(value), '天') < 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>
```

结果

```
| ID | 页面标题     | 页面URL    | 创建时间           | 状态     |
-------------------------------------------------------------------------------------
|  1 | 首页         | home        | 4分钟前            | 启用      |
|  2 | 关于我们       | about      | 昨天               | 启用      |
|  4 | 联系我们     | contact     | 昨天               | 禁用      |
-------------------------------------------------------------------------------------
```

第14.6节：使用async管道解包异步值

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

```
    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>
```

Result

```
| 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   |
---------------------------------------------------------------------
```

Section 14.6: Unwrap async values with async pipe

```
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>
   你的朋友们是:
    <ul>
      <li *ngFor="let friend of friends | async">
        {{friend}}
      </li>
    </ul>
  `
})
class AsyncStuffComponent {
  name = Promise.resolve('Misko');
  friends = Observable.of(['Igor']);
}
```

变为：

```
<h1>Hello, Misko</h1>
你的朋友们是：
<ul>
  <li>
伊戈尔
  </li>
</ul>
```

第14.7节：有状态管道

Angular 2 提供了两种不同类型的管道——无状态和有状态。管道默认是无状态的。然而，我们可以通过将pure属性设置为false来实现有状态管道。如参数部分所示，你可以指定一个name并声明该管道是否为纯管道，也就是有状态或无状态。当数据流经无状态管道（即纯函数）时，该管道不会记住任何内容，而有状态管道则可以管理并记住数据。一个有状态管道的好例子是 Angular 2 提供的AsyncPipe。

重要

请注意，大多数管道应属于无状态管道类别。这对于性能来说非常重要，因为 Angular 可以针对变更检测器优化无状态管道。因此，请谨慎使用有状态管道。总体来说，Angular 2 中管道的优化相比 Angular 1.x 中的过滤器带来了显著的性能提升。在 Angular 1 中，即使数据完全未改变，digest 循环也必须重新运行所有过滤器。而在 Angular 2 中，一旦管道的值被计算出来，变更检测器就知道除非输入发生变化，否则不会再次运行该管道。

有状态管道的实现

```
import {Pipe, PipeTransform, OnDestroy} from '@angular/core';

@Pipe({
name: 'countdown',
  pure: false
})
export class CountdownPipe implements PipeTransform, OnDestroy  {
  private interval: any;
private remainingTime: number;
```

```
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']);
}
```

Becomes:

```
<h1>Hello, Misko</h1>
Your Friends are:
<ul>
  <li>
    Igor
  </li>
</ul>
```

Section 14.7: Stateful Pipes

Angular 2 offers two different types of pipes - stateless and stateful. Pipes are stateless by default. However, we can implement stateful pipes by setting the `pure` property to **`false`**. As you can see in the parameter section, you can specify a `name` and declare whether the pipe should be pure or not, meaning stateful or stateless. While data flows through a stateless pipe (which is a pure function) that **does not** remember anything, data can be managed and remembered by stateful pipes. A good example of a stateful pipe is the `AsyncPipe` that is provided by Angular 2.

Important

Notice that most pipes should fall into the category of stateless pipes. That's important for performance reasons since Angular can optimize stateless pipes for the change detector. So use stateful pipes cautiously. In general, the optimization of pipes in Angular 2 have a major performance enhancement over filters in Angular 1.x. In Angular 1 the digest cycle always had to re-run all filters even though the data hasn't changed at all. In Angular 2, once a pipe's value has been computed, the change detector knows not to run this pipe again unless the input changes.

Implementation of a stateful pipe

```
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);
    }
  }
}
```

然后你可以像平常一样使用这个管道：

```
{{ 1000 | countdown:50 }}
{{ 300 | countdown }}
```

你的管道同样需要实现OnDestroy接口，以便在管道被销毁时进行清理。在上面的示例中，必须清除定时器以避免内存泄漏。

第14.8节：创建自定义管道

app/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
  }
}
```

app/my-component.component.ts

```
import { Truthy} from './pipes.pipe';

@Component({
selector: 'my-component',
```

```
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);
    }
  }
}
```

You can then use the pipe as usual:

```
{{ 1000 | countdown:50 }}
{{ 300 | countdown }}
```

It's important that your pipe also implements the `OnDestroy` interface so you can clean up once your pipe gets destroyed. In the example above, it's necessary to clear the interval to avoid memory leaks.

Section 14.8: Creating Custom Pipe

app/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
  }
}
```

app/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{ }
```

第14.9节：全局可用的自定义管道

为了使自定义管道在整个应用程序中可用，在应用程序启动期间，扩展 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'; // 你的自定义管道

bootstrap(AppComponent, [
provide(PLATFORM_PIPES, {
            useValue: [
              MyPipe
            ],
multi: true
        })
]);
```

教程链接：https://scotch.io/tutorials/create-a-globally-available-custom-pipe-in-angular-2

第14.10节：扩展现有管道

```
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
    }
  }
}
```

第14.11节：测试管道

给定一个反转字符串的管道

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'reverse' })
export class ReversePipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
```

```
  template: `
    <p>{{value | truthy:'enabled':'disabled' }}</p>
  `,
  pipes: [Truthy]
})
export class MyComponent{ }
```

Section 14.9: Globally Available Custom Pipe

To make a custom pipe available application wide, During application bootstrap, extending 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
        })
]);
```

Tutorial here: https://scotch.io/tutorials/create-a-globally-available-custom-pipe-in-angular-2

Section 14.10: Extending an Existing Pipe

```
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
    }
  }
}
```

Section 14.11: Testing a pipe

Given a pipe that reverse a string

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({ name: 'reverse' })
export class ReversePipe implements PipeTransform {
  transform(value: string): string {
    return value.split('').reverse().join('');
  }
}
```

可以通过如下配置 spec 文件进行测试

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

import { ReversePipe } from './reverse.pipe';

describe('ReversePipe', () => {
  beforeEach(() => {
TestBed.configureTestingModule({
      providers: [ReversePipe],
    });
  });

it('应该被创建', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe).toBeTruthy();
  }));

it('应该反转字符串', inject([ReversePipe], (reversePipe: ReversePipe) => {
    expect(reversePipe.transform('abc')).toEqual('cba');
  }));
});
```

```
  }
}
```

It can be tested configuring the spec file like this

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

第15章：OrderBy管道

如何编写 order pipe 及其使用方法。

第15.1节：管道

管道的实现

```
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))){
      //不是数字，因此将字符串转换为小写以便正确比较
      if(a.toLowerCase() < b.toLowerCase()) return -1;
      if(a.toLowerCase() > b.toLowerCase()) return 1;
    }else{
      //将字符串解析为数字以便正确比较
      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){
```

Chapter 15: OrderBy Pipe

How to write order pipe and use it.

Section 15.1: The Pipe

The Pipe implementation

```
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 {
      //按顺序遍历数组的属性并排序
      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]);

          //还不能返回0，以防需要按下一个属性排序
          if(comparison !== 0) return comparison;
        }

        return 0; //彼此相等
      });
    }
  }
}
```

如何在HTML中使用管道 - 按名字升序排序

```
<table>
   <thead>
    <tr>
     <th>名字</th>
     <th>姓氏</th>
    <th>年龄</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>

</table>
```

如何在HTML中使用管道 - 按名字降序排列

```
<table>
   <thead>
    <tr>
     <th>名字</th>
     <th>姓氏</th>
    <th>年龄</th>
    </tr>
</thead>
<tbody>
```

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

How to use the pipe in the HTML - order ascending by first name

```
<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>
```

How to use the pipe in the HTML - order descending by first name

```
<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>
```

```
    <tr *ngFor="let user of users | orderBy : ['-firstName']>
        <td>{{user.firstName}}</td>
        <td>{{user.lastName}}</td>
        <td>{{user.age}}</td>
    </tr>
</tbody>
</table>
```

第16章：Angular 2 自定义验证

参数	描述
控件	这是正在验证的控件。通常你会想查看 control.value 是否满足某些条件。

第16.1节：获取/设置 formBuilder 控件参数

设置 formBuilder 控件参数有两种方式。

1. 初始化时：

```
exampleForm : FormGroup;
constructor(fb: FormBuilder){
  this.exampleForm = fb.group({
name : new FormControl({value: '默认名称'}, Validators.compose([Validators.required,
Validators.maxLength(15)]))
   });
}
```

2. 初始化后：

```
this.exampleForm.controls['name'].setValue('默认名称');
```

获取 formBuilder 控件的值：

```
let name = this.exampleForm.controls['name'].value();
```

第16.2节：自定义验证器示例：

Angular 2 有两种自定义验证器。第一种示例中的同步验证器会直接在客户端运行，第二种示例中的异步验证器可以调用远程服务为你进行验证。在此示例中，验证器应调用服务器以检查某个值是否唯一。

```
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) => {
        // 模拟远程验证器。
setTimeout(function () {
            if (control.value == "exisitingUser")
                resolve({ shouldBeUnique: true });
            else
resolve(null);
        }, 1000);
    });
}}
```

如果您的控件值有效，您只需返回 null 给调用者。否则，您可以返回一个描述错误的对象。

Chapter 16: Angular 2 Custom Validations

parameter	description
control	This is the control that is being validated. Typically you will want to see if control.value meets some criteria.

Section 16.1: get/set formBuilder controls parameters

There are 2 ways to set formBuilder controls parameters.

1. On initialize:

```
exampleForm : FormGroup;
constructor(fb: FormBuilder){
  this.exampleForm = fb.group({
      name : new FormControl({value: 'default name'}, Validators.compose([Validators.required,
Validators.maxLength(15)]))
   });
}
```

2.After initialize:

```
this.exampleForm.controls['name'].setValue('default name');
```

Get formBuilder control value:

```
let name = this.exampleForm.controls['name'].value();
```

Section 16.2: Custom validator examples:

Angular 2 has two kinds of custom validators. Synchronous validators as in the first example that will run directly on the client and asynchronous validators (the second example) that you can use to call a remote service to do the validation for you. In this example the validator should call the server to see if a value is unique.

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

If your control value is valid you simply return null to the caller. Otherwise you can return an object which describes

错误。

第16.3节：在Formbuilder中使用验证器

```
constructor(fb: FormBuilder) {
     this.form = fb.group({
firstInput: ['', Validators.compose([Validators.required,
CustomValidators.cannotContainSpace]), CustomValidators.shouldBeUnique],
        secondInput: ['', Validators.required]
     });
}
```

这里我们使用FormBuilder创建了一个非常基础的表单，包含两个输入框。FormBuilder为每个输入控件接受一个包含三个参数的数组。

1. 控件的默认值。
2. 将在客户端运行的验证器。您可以使用Validators.compose([验证器数组])来对控件应用多个验证器。

3. 一个或多个异步验证器，使用方式类似于第二个参数。

the error.

Section 16.3: Using validators in the Formbuilder

```
   constructor(fb: FormBuilder) {
    this.form = fb.group({
        firstInput: ['', Validators.compose([Validators.required,
CustomValidators.cannotContainSpace]), CustomValidators.shouldBeUnique],
        secondInput: ['', Validators.required]
    });
}
```

Here we use the FormBuilder to create a very basic form with two input boxes. The FromBuilder takes an array for three arguments for each input control.

1. The default value of the control.
2. The validators that will run on the client. You can use Validators.compose([arrayOfValidators]) to apply multiple validators on your control.
3. One or more async validators in a similar fashion as the second argument.

第17章：路由

第17.1节：ResolveData

本示例将向您展示如何在渲染应用程序的视图之前，解析从服务获取的数据。

使用 angular/router 3.0.0-beta.2（撰写时）

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) {}

    /**
* 返回所有用户
     * @returns {Observable<User[]>}
     */
index():Observable<User[]> {

        return this.http.get('http://mywebsite.com/api/v1/users')
            .map((res:Response) => res.json());
    }

    /**
* 通过ID返回用户
     * @param id
* @returns {Observable<User>}
    */
    get(id:数字|字符串):Observable<用户> {

        return this.http.get('http://mywebsite.com/api/v1/users/' + id)
            .map((res:Response) => res.json());
    }
}
```

users.resolver.ts

```
...
import { 用户服务 } from './users.service.ts';
import { Observable } from 'rxjs/Rx';
import {
Resolve,
ActivatedRouteSnapshot,
   RouterStateSnapshot
} from "@angular/router";
```

Chapter 17: Routing

Section 17.1: ResolveData

This example will show you how you can resolve data fetched from a service before rendering your application's view.

Uses angular/router 3.0.0-beta.2 at the time of writing

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()
导出类 UsersResolver 实现 Resolve<User[] | User> {

    // 将 UsersService 注入解析器
constructor(private service:UsersService) {}

    resolve(route:ActivatedRouteSnapshot, state:RouterStateSnapshot):Observable<User[] | User> {
        // 如果当前 URL 中存在 userId 参数，则返回单个用户，否则返回所有用户// 使用括号表示法访问 `id` 以抑
        制编辑器警告，如果你创建了一个带有可选 id? 属性的继承自 ActivatedRoute 的接口，也可以使用点表示法

        if (route.params['id']) return this.service.get(route.params['id']);
        return this.service.index();
    }
}
```

users.component.ts

这是一个包含所有用户列表的页面组件。用户详情页面组件的工作方式类似，替换 data.users 为 data.user 或者在 app.routes.ts 中定义的任何键（见下文）

```
...
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;
        });
    }

    /**
* 不需要取消订阅解析器，因为 Angular 的 HTTP
     * 在从服务器接收到数据后会自动完成订阅
     */
}
```

app.routes.ts

```
...
import { UsersResolver } from './resolvers/users.resolver';



export const routes:RouterConfig = <RouterConfig>[
    …
    {
path: 'user/:id',
        component: UserComponent,
        resolve: {
            // 因此在 UserComponent 中使用 data.user
user: UsersResolver
        }
```

```
@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

This is a page component with a list of all users. It will work similarly for User detail page component, replace `data.users` with `data.user` or whatever key defined in *app.routes.ts*(see below)

```
...
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: {
            // 因此在 UsersComponent 中使用 data.users，注意复数形式
users: UsersResolver
        }
    },
    ...
]
...
```

app.resolver.ts

可选地将多个解析器捆绑在一起。

重要：在解析器中使用的服务必须先导入，否则会出现"没有提供者 ..Resolver 错误"。
请记住，这些服务将全局可用，您无需在任何组件的
providers 中声明它们。请确保取消订阅任何订阅以防止内存泄漏

```
...
import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';

export const ROUTE_RESOLVERS = [
...,
UsersService,
    UsersResolver
]
```

main.browser.ts

解析器必须在引导期间注入。

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

bootstrap(<Type>App, [
    ...
...ROUTE_RESOLVERS
])
.catch(err => console.error(err));
```

第17.2节：带子路由的路由

与原始文档相反，我发现这是在app.routing.ts或app.module.ts文件中正确嵌套子路由的方法（取决于你的偏好）。这种方法在使用WebPack或SystemJS时都有效。

下面的示例展示了home、home/counter和home/counter/fetch-data的路由。第一个和最后一个路由是重定向的示例。最后示例末尾展示了导出路由的正确方式，以便在单独的文件中导入，例如app.module.ts。

进一步说明，Angular要求在children数组中有一个无路径的路由，该路由包含父组件，用以表示父路由。这有点令人困惑，但如果你考虑子路由的空白URL，

```
    },
    {
        path: 'users',
        component: UsersComponent,
        resolve: {
            // hence data.users in UsersComponent, note the pluralisation
            users: UsersResolver
        }
    },
    ...
]
...
```

app.resolver.ts

Optionally bundle multiple resolvers together.

IMPORTANT: *Services used in resolver must be imported first or you will get a 'No provider for ..Resolver error'. Remember that these services will be available globally and you will not need to declare them in any component's* `providers` *anymore. Be sure to unsubscribe from any subscription to prevent memory leak*

```
...
import { UsersService } from './users.service';
import { UsersResolver } from './users.resolver';

export const ROUTE_RESOLVERS = [
    ...,
    UsersService,
    UsersResolver
]
```

main.browser.ts

Resolvers have to be injected during bootstrapping.

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

bootstrap(<Type>App, [
    ...
    ...ROUTE_RESOLVERS
])
.catch(err => console.error(err));
```

Section 17.2: Routing with Children

Contrary to original documentation, I found this to be the way to properly nest children routes inside the app.routing.ts or app.module.ts file (depending on your preference). This approach works when using either WebPack or SystemJS.

The example below shows routes for home, home/counter, and home/counter/fetch-data. The first and last routes being examples of redirects. Finally at the end of the example is a proper way to export the Route to be imported in a separate file. For ex. app.module.ts

To further explain, Angular requires that you have a pathless route in the children array that includes the parent component, to represent the parent route. It's a little confusing but if you think about a blank URL for a child route,

它本质上等同于父路由的相同URL。

```
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 { }
```

Siraj 提供的优秀示例和说明

第17.3节：基础路由

路由器使得基于用户与应用的交互，从一个视图导航到另一个视图成为可能。

以下是在 Angular 2 中实现基础路由的步骤 -

it would essentially equal the same URL as the parent route.

```
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 { }
```

Great Example and Description via Siraj

Section 17.3: Basic Routing

Router enables navigation from one view to another based on user interactions with the application.

Following are the steps in implementing basic routing in Angular 2 -

基本注意事项：确保你有标签

```
<base href='/'>
```

作为 index.html 文件中 head 标签下的第一个子元素。该标签表明你的 app 文件夹是应用程序根目录。Angular 2 会据此组织你的链接。

第一步是检查你是否在package.json中指向了正确/最新的路由依赖项 -

```
"dependencies": {
  ......
  "@angular/router": "3.0.0-beta.1",
  ......
}
```

第二步是根据其类定义来定义路由 -

```
class Route {
path : string
  pathMatch : 'full'|'prefix'
  component : Type|string
  .........
}
```

在路由文件（route/routes.ts）中，导入所有需要配置不同路由路径的组件。空路径表示默认加载该视图。路径中的":"表示传递给加载组件的动态参数。

路由通过依赖注入提供给应用。调用ProviderRouter方法并传入RouterConfig作为参数，以便注入到组件中执行特定的路由任务。

```
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)];
```

第三步是引导路由提供者。

在你的 main.ts 文件中（文件名可以是任意的，基本上应该是你在 systemjs.config 中定义的主文件）

```
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));
```

Basic precaution: Ensure you have the tag

```
<base href='/'>
```

as the first child under your head tag in your index.html file. This tag tells that your app folder is the application root. Angular 2 would then know to organize your links.

First step is to check if you are pointing to correct/latest routing dependencies in package.json -

```
"dependencies": {
  ......
  "@angular/router": "3.0.0-beta.1",
  ......
}
```

Second step is to define the route as per it's class definition -

```
class Route {
  path : string
  pathMatch : 'full'|'prefix'
  component : Type|string
  .........
}
```

In a routes file (`route/routes.ts`), import all the components which you need to configure for different routing paths. Empty path means that view is loaded by default. ":" in the path indicates dynamic parameter passed to the loaded component.

Routes are made available to application via dependency injection. ProviderRouter method is called with RouterConfig as parameter so that it can be injected to the components for calling routing specific tasks.

```
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)];
```

Third step is to bootstrap the route provider.

In your `main.ts` (It can be any name. basically, it should your main file defined in 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));
```

第四步是根据访问的路径加载/显示路由组件。directive 用于告诉 Angular 在哪里加载组件。要使用它，需要导入 ROUTER_DIRECTIVES。

```
import { ROUTER_DIRECTIVES } from '@angular/router';

@Component({
selector: 'demo-app',
  template: `
    ....................................
    <div>
      <router-outlet></router-outlet>
    </div>
    ....................................
`,
  // 添加我们将使用的路由指令
directives: [ROUTER_DIRECTIVES]
})
```

第五步是链接其他路由。默认情况下，RouterOutlet 会加载 RouterConfig 中为空路径指定的组件。RouterLink 指令与 HTML 锚点标签一起使用，用于加载附加到路由的组件。RouterLink 会生成 href 属性，用于生成链接。例如：

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

@Component({
selector: 'demo-app',
  template: `
    <a [routerLink]="['/login']">登录</a>
    <a [routerLink]="['/signup']">注册</a>
    <a [routerLink]="['/dashboard']">仪表盘</a>
    <div>
      <router-outlet></router-outlet>
    </div>
`,
  // 添加我们将使用的路由指令
directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }
```

现在，我们已经可以路由到静态路径。RouterLink 也支持动态路径，通过传递额外参数与路径一起使用。

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

```
@Component({
  selector: 'demo-app',
  template: `
        <ul>
          <li *ngFor="let bar of bars | async">
            <a [routerLink]="['/bars', bar.id]">
              {{bar.name}}
            <original></a>
          </li>
        </ul></original>
    <div>
      <router-outlet></router-outlet>
    </div>
`,
```

Fourth step is to load/display the router components based on path accessed. directive is used to tell angular where to load the component. To use import the 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]
})
```

Fifth step is to link the other routes. By default, RouterOutlet will load the component for which empty path is specified in the RouterConfig. RouterLink directive is used with html anchor tag to load the components attached to routes. RouterLink generates the href attribute which is used to generate links. For 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 { }
```

Now, we are good with routing to static path. RouterLink support dynamic path also by passing extra parameters along with the path.

import { Component } from '@angular/core'; import { ROUTER_DIRECTIVES } from '@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>
  `,
  // 添加我们将使用的路由指令
directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }
```

RouterLink 接受一个数组，数组的第一个元素是路由路径，后续元素是动态路由参数。

第17.4节：子路由

有时将视图或路由嵌套在彼此之中是有意义的。例如，在仪表盘上你想要几个子视图，类似于标签页，但通过路由系统实现，用来显示用户的项目、联系人、消息等。为了支持这种场景，路由器允许我们定义子路由。

首先我们调整上面的RouterConfig，添加子路由：

```
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 }
];
```

现在我们已经定义了子路由，我们必须确保这些子路由可以在我们的DashboardComponent，因为我们是在这里添加子组件的。之前我们已经了解到，组件是显示在<router-outlet></router-outlet>标签中的。类似地，我们在DashboardComponent中声明了另一个RouterOutlet：

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

@Component({
selector: 'dashboard',
  template: `
    <a [routerLink]="['projects']">项目</a>
    <a [routerLink]="['messages']">消息</a>
    <div>
      <router-outlet></router-outlet>
    </div>
  `
})
export class DashboardComponent { }
```

如你所见，我们添加了另一个RouterOutlet，用于显示子路由。通常会显示路径为空的路由，然而，我们设置了重定向到projects路由，因为我们希望在加载dashboard路由时立即显示它。也就是说，我们需要一个空路由，否则你会遇到如下错误：

```
无法匹配任何路由: 'dashboard'
```

```
  // Add our router directives we will be using
  directives: [ROUTER_DIRECTIVES]
})
export class AppComponent { }
```

RouterLink takes an array where first element is the path for routing and subsequent elements are for the dynamic routing parameters.

Section 17.4: Child Routes

Sometimes it makes sense to nest view's or routes within one another. For example on the dashboard you want several sub views, similar to tabs but implemented via the routing system, to show the users' projects, contacts, messages ets. In order to support such scenarios the router allows us to define child routes.

First we adjust our `RouterConfig` from above and add the child routes:

```
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 }
];
```

Now that we have our child routes defined we have to make sure those child routes can be displayed within our `DashboardComponent`, since that's where we have added the childs to. Previously we have learned that the components are displayed in a **`<router-outlet></router-outlet>`** tag. Similar we declare another `RouterOutlet` in the `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 { }
```

As you can see, we have added another `RouterOutlet` in which the child routes will be displayed. Usually the route with an empty path will be shown, however, we set up a redirect to the `projects` route, because we want that to be shown immediately when the `dashboard` route is loaded. That being said, we need an empty route, otherwise you'll get an error like this:

```
Cannot match any routes: 'dashboard'
```

因此，通过添加empty路由，即路径为空的路由，我们为路由器定义了一个入口点。

So by adding the *empty* route, meaning a route with an empty path, we have defined an entry point for the router.

第18章：路由（3.0.0及以上）

第18.1节：控制路由的访问权限

默认的 Angular 路由器允许无条件地导航到任何路由及从任何路由导航。这并不总是期望的行为。

在某些情况下，用户可能有条件地被允许导航到某个路由或从某个路由导航，此时可以使用路由守卫（Route Guard
）来限制这种行为。

如果您的场景符合以下任一情况，建议使用路由守卫，

- 用户需要经过身份验证才能导航到目标组件。
- 用户需要经过授权才能导航到目标组件。
- 组件在初始化前需要异步请求。
- 组件在导航离开前需要用户输入。

路由守卫的工作原理

路由守卫通过返回布尔值来控制路由导航的行为。如果返回true，路由器将继续导航到目标组件。如果返回false，路由器将拒绝导航到目标组件。

路由守卫接口

路由器支持多个保护接口：

- CanActivate：发生在路由导航之间。
- CanActivateChild：发生在导航到子路由之间。
- CanDeactivate：发生在离开当前路由时。
- CanLoad：发生在导航到异步加载的功能模块之间。
- Resolve：用于在路由激活之前执行数据获取。

这些接口可以在你的守卫中实现，以授予或移除对导航某些过程的访问权限。

同步与异步路由守卫

路由守卫允许同步和异步操作有条件地控制导航。

同步路由守卫

同步路由守卫返回一个布尔值，例如通过计算即时结果，以有条件地控制导航。

```
import { Injectable }     from '@angular/core';
import { CanActivate }    from '@angular/router';

@Injectable()
export class 同步守卫 implements CanActivate {
  canActivate() {
```

Chapter 18: Routing (3.0.0+)

Section 18.1: Controlling Access to or from a Route

The default Angular router allows navigation to and from any route unconditionally. This is not always the desired behavior.

In a scenario where a user may conditionally be allowed to navigate to or from a route, a **Route Guard** may be used to restrict this behavior.

If your scenario fits one of the following, consider using a Route Guard,

- User is required to be authenticated to navigate to the target component.
- User is required to be authorized to navigate to the target component.
- Component requires asynchronous request before initialization.
- Component requires user input before navigated away from.

How Route Guards work

Route Guards work by returning a boolean value to control the behavior of router navigation. If *true* is returned, the router will continue with navigation to the target component. If *false* is returned, the router will deny navigation to the target component.

Route Guard Interfaces

The router supports multiple guard interfaces:

- *CanActivate*: occurs between route navigation.
- *CanActivateChild*: occurs between route navigation to a child route.
- *CanDeactivate*: occurs when navigating away from the current route.
- *CanLoad*: occurs between route navigation to a feature module loaded asynchronously.
- *Resolve*: used to perform data retrieval before route activation.

These interfaces can be implemented in your guard to grant or remove access to certain processes of the navigation.

Synchronous vs. Asynchronous Route Guards

Route Guards allow synchronous and asynchronous operations to conditionally control navigation.

Synchronous Route Guard

A synchronous route guard returns a boolean, such as by computing an immediate result, in order to conditionally control navigation.

```
import { Injectable }     from '@angular/core';
import { CanActivate }    from '@angular/router';

@Injectable()
export class SynchronousGuard implements CanActivate {
  canActivate() {
```

```
console.log('SynchronousGuard#canActivate called');
    return true;
  }
}
```

异步路由守卫

对于更复杂的行为，路由守卫可以异步阻止导航。异步路由守卫可以
返回一个 Observable 或 Promise。

这对于等待用户输入回答问题、等待成功保存更改到服务器，或等待接收从远程服务器获取的数据等情况非常有用。

```
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()
导出类 AsynchronousGuard 实现 CanActivate {
   构造函数(私有 router: Router, 私有 auth: MockAuthenticationService) {}

    canActivate(路由:ActivatedRouteSnapshot,
state:RouterStateSnapshot):Observable<boolean>|boolean {
        this.auth.subscribe((authenticated) => {
            if (authenticated) {
                return true;
            }
            this.router.navigateByUrl('/login');
            return false;
        });
    }
}
```

第18.2节：向路由配置添加守卫

文件 app.routes

受保护的路由绑定了 canActivate 到 Guard

```
import { provideRouter, Router, RouterConfig, CanActivate } from '@angular/router';

//组件
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] }
}
```

导出APP_ROUTER_PROVIDERS以供应用启动时使用

```
export const APP_ROUTER_PROVIDERS = [
    AuthGuard,
provideRouter(routes)
];
```

```
    console.log('SynchronousGuard#canActivate called');
    return true;
  }
}
```

Asynchronous Route Guard

For more complex behavior, a route guard can asynchronously block navigation. An asynchronous route guard can return an Observable or Promise.

This is useful for situations like waiting for user input to answer a question, waiting to successfully save changes to the server, or waiting to receive data fetched from a remote server.

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

Section 18.2: Add guard to route configuration

File *app.routes*

Protected routes have `canActivate` binded to `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] }
}
```

Export the **APP_ROUTER_PROVIDERS** to be used in app bootstrap

```
export const APP_ROUTER_PROVIDERS = [
    AuthGuard,
    provideRouter(routes)
];
```

第18.3节：使用解析器和守卫

我们在路由配置中使用了顶层守卫，以在首次加载页面时获取当前用户，并使用解析器存储currentUser的值，即来自后端的已认证用户。

我们实现的简化版本如下：

这是我们的顶层路由：

```
export const routes = [
{
path: 'Dash',
  pathMatch : 'prefix',
  component: DashCmp,
  canActivate: [AuthGuard],
  resolve: {
currentUser: CurrentUserResolver
  },
children: [...[
    path: '',
component: ProfileCmp,
    resolve: {
currentUser: currentUser
    }
  ]]
  }
];
```

这是我们的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);
 }
}
```

这是我们的AuthGuard：

```
import { Injectable } from '@angular/core';
import { CanActivate } from "@angular/router";
import { Observable } from 'rxjs/Rx';

import { AuthService } from '../services/AuthService';

@Injectable()
```

Section 18.3: Using Resolvers and Guards

We're using a toplevel guard in our route config to catch the current user on first page load, and a resolver to store the value of the `currentUser`, which is our authenticated user from the backend.

A simplified version of our implementation looks as follows:

Here is our top level route:

```
export const routes = [
{
  path: 'Dash',
  pathMatch : 'prefix',
  component: DashCmp,
  canActivate: [AuthGuard],
  resolve: {
      currentUser: CurrentUserResolver
  },
  children: [...[
    path: '',
    component: ProfileCmp,
    resolve: {
        currentUser: currentUser
    }
  ]]
  }
];
```

Here is our `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);
 }
}
```

Here is our `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);
  }
}
```

这是我们的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;
  }
}
```

第18.4节：在应用启动时使用守卫

文件 *main.ts*（或 *boot.ts*）

参考上述示例：

1. 创建守卫（守卫创建的位置）和
2. 将守卫添加到路由配置，（守卫配置到路由的位置，然后
   APP_ROUTER_PROVIDERS 已导出），
   我们可以将引导程序与守卫（Guard）结合如下

```
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));
```

第18.5节：引导程序（Bootstrapping）

现在路由已经定义，我们需要让应用程序知道这些路由。为此，引导我们在前面示例中导出的提供者（provider）。

找到你的引导配置（通常在main.ts中，但具体情况可能有所不同）。

```
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);
  }
}
```

Here is our `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;
  }
}
```

Section 18.4: Use Guard in app bootstrap

File *main.ts* (or *boot.ts*)

Consider the examples above:

1. **Create the guard** (where the Guard is created) and
2. **Add guard to route configuration**, (where the Guard is configured for route, then
   APP_ROUTER_PROVIDERS is exported),
   we can couple the bootstrap to Guard as follows

```
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));
```

Section 18.5: Bootstrapping

Now that the routes are defined, we need to let our application know about the routes. To do this, bootstrap the provider we exported in the previous example.

Find your bootstrap configuration (should be in `main.ts`, but **your mileage may vary**).

```
//main.ts

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

//导入App组件（根组件）
import { 应用程序 } 来自 './app/app';

// 同时导入应用程序路由
import { APP_ROUTES_PROVIDER } 来自 './app/app.routes';

bootstrap(应用程序, [
APP_ROUTES_PROVIDER,
])
.catch(err => console.error(err));
```

第18.6节：配置router-outlet

现在路由器已配置好，我们的应用程序知道如何处理路由，我们需要显示实际
配置的组件。

为此，请像下面这样配置顶层（应用程序）组件的HTML模板/文件：

```
//app.ts

import {组件} 来自 '@angular/core';
import {路由器, ROUTER_DIRECTIVES} 来自 '@angular/router';

@组件({
选择器: 'app',
    模板URL: 'app.html',
    样式URL: ['app.css'],
    指令: [
ROUTER_DIRECTIVES,
     ]
})
导出类 App {
    构造函数() {
     }
}

<!-- app.html -->

<!-- 你所有的 '视图' 都将在这里 -->
<router-outlet></router-outlet>
```

元素 <router-outlet></router-outlet> 会根据路由切换内容。这个元素的另一个优点是它 不 必须是你HTML中的唯一元素。

例如：假设你想在每个页面上都有一个工具栏，在路由之间保持不变，类似于Stack Overflow的样子。你可以将 <ro
uter-outlet> 嵌套在其他元素下，这样页面的只有某些部分会改变。

第18.7节：更改路由（使用模板和指令）

现在路由已经设置好了，我们需要某种方式来实际更改路由。

这个示例将展示如何使用模板更改路由，但也可以在TypeScript中更改路由。

```
//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));
```

Section 18.6: Configuring router-outlet

Now that the router is configured and our app knows how to handle the routes, we need to show the actual components that we configured.

To do so, configure your HTML template/file for your **top-level (app)** component like so:

```
//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>
```

The **`<router-outlet></router-outlet>`** element will switch the content given the route. Another good aspect about this element is that it *does not* have to be the only element in your HTML.

For example: Lets say you wanted a a toolbar on every page that stays constant between routes, similar to how Stack Overflow looks. You can nest the **`<router-outlet>`** under elements so that only certain parts of the page change.

Section 18.7: Changing routes (using templates & directives)

Now that the routes are set up, we need some way to actually change routes.

This example will show how to change routes using the template, but it is possible to change routes in TypeScript.

这里有一个示例（无绑定）：

```
<a routerLink="/home">首页</a>
```

如果用户点击该链接，它将路由到/home。路由器知道如何处理/home，因此它会显示Home组件。

这里是一个带数据绑定的示例：

```
<a *ngFor="let link of links" [routerLink]="link">{{link}}</a>
```

这需要一个名为links的数组存在，所以请将以下内容添加到app.ts中：

```
public links[] = [
    'home',
    'login'
]
```

这将遍历数组，并添加一个带有routerLink指令的<a>元素，指令的值为数组中当前元素，生成如下内容：

```
 <a routerLink="home">home</a>
 <a routerLink="login">login</a>
```

如果你有很多链接，或者链接需要不断更改，这尤其有用。我们让Angular处理添加链接的繁琐工作，只需提供它所需的信息。

目前，links[]是静态的，但也可以从其他来源提供数据。

第18.8节：设置路由

注意：此示例基于 @angular/router 的 3.0.0-beta.2 版本。撰写时，该版本是路由器的最新版本。

要使用路由器，请在一个新的 TypeScript 文件中定义路由，如下所示

```
//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);
```

在第一行，我们导入了 provideRouter，以便在应用启动阶段告知应用程序路由信息。

Home 和 Profile 只是两个示例组件。你需要导入每个作为路由所需的 Component 组件。

Here is one example (without binding):

```
<a routerLink="/home">Home</a>
```

If the user clicks on that link, it will route to /home. The router knows how to handle /home, so it will display the Home Component.

Here is an example with data binding:

```
<a *ngFor="let link of links" [routerLink]="link">{{link}}</a>
```

Which would require an array called links to exist, so add this to app.ts:

```
public links[] = [
    'home',
    'login'
]
```

This will loop through the array and add an <a> element with the routerLink directive = the value of the current element in the array, creating this:

```
 <a routerLink="home">home</a>
 <a routerLink="login">login</a>
```

This is particularly helpful if you have a lot of links, or maybe the links need to be constantly changed. We let Angular handle the busy work of adding links by just feeding it the info it requires.

Right now, links[] is static, but it is possible to feed it data from another source.

Section 18.8: Setting the Routes

NOTE: This example is based on the 3.0.0.-beta.2 release of the @angular/router. At the time of writing, this is the latest version of the router.

To use the router, define routes in a new TypeScript file like such

```
//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);
```

In the first line, we import provideRouter so we can let our application know what the routes are during the bootstrap phase.

Home and Profile are just two components as an example. You will need to import each Component you need as a route.

然后，导出路由数组。

`path`：组件的路径。**你不需要使用 '/........'** Angular 会自动处理这个

`component`：访问该路由时加载的组件

redirectTo：可选。如果你需要在用户访问特定路由时自动重定向，提供此项。

最后，我们导出配置好的路由器。`provideRouter` 会返回一个提供者，我们可以引导它，使我们的应用程序知道如何处理每个路由。

Then, export the array of routes.

`path`: The path to the component. **YOU DO NOT NEED TO USE '/........'** Angular will do this automatically

`component`: The component to load when the route is accessed

`redirectTo`: *Optional*. If you need to automatically redirect a user when they access a particular route, supply this.

Finally, we export the configured router. `provideRouter` will return a provider that we can boostrap so our application knows how to handle each route.

第19章：使用 ViewContainerRef.createComponent 动态添加组件

第19.1节：一个声明式添加动态组件的包装组件

一个自定义组件，接收组件类型作为输入，并在自身内部创建该组件类型的实例。当输入更新时，之前添加的动态组件会被移除，替换为新的组件。

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

这使您能够创建动态组件，例如

Chapter 19: Dynamically add components using ViewContainerRef.createComponent

Section 19.1: A wrapper component that adds dynamic components declaratively

A custom component that takes the type of a component as input and creates an instance of that component type inside itself. When the input is updated, the previously added dynamic component is removed and the new one added instead.

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

This allows you to create dynamic components like

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

Plunker 示例

第19.2节：在特定事件（点击）上动态添加组件

主组件文件：

```
//我们的根应用组件
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>你好 {{name}}</h2>
                  <input type="button" value="点击我添加元素" (click) = addElement()> // 调用
按钮点击时的函数
      <div #parent> </div> // 动态组件将加载在这里
    </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>这是子组件</p>
  `,
})
export class ChildComponent {
  constructor(){

  }
}
```

app.module.ts :

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

Plunker example

Section 19.2: Dynamically add component on specific event(click)

Main Component File:

```
//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] // 在 module.ts 中定义动态组件
})
export class AppModule {}
```

Plunker 示例

第19.3节：在 Angular 2 模板 HTML 中动态渲染创建的组件数组

我们可以创建动态组件，并将组件实例放入数组，最后在模板中渲染它们。

例如，我们可以考虑两个小部件组件，ChartWidget 和 PatientWidget，它们继承了我想添加到容器中的 WidgetComponent 类。

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.compoment.html (使用 primeng 面板)

```
<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">图表控件</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>
一些数据
```

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

Plunker example

Section 19.3: Rendered dynamically created component array on template HTML in Angular 2

We can create dynamic component and get the instances of component into an array and finally rendered it on template.

For example, we can can consider two widget component, ChartWidget and PatientWidget which extended the class WidgetComponent that I wanted to add in the container.

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.compoment.html (using primeng Panel)

```
<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.compoment.html（与使用 primeng Panel 的 chart-widget 相同）

WidgetComponent.ts

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

我们可以通过选择预先存在的组件来创建动态组件实例。例如，

```
@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;
  }
}
```

最后我们在应用组件中使用它。app.component.ts

```
@Component({
  selector: 'app-root',
```

```
</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.compoment.html (same as chart-widget using primeng Panel)

WidgetComponent.ts

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

we can creat dynamic component instances by selecting the pre-existing components. For example,

```
@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;
  }
}
```

Finally we use it in app component. 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')">添加 ChartWidget</button>
<button (click)="addComponent('DataWidget')">添加 DataWidget</button>

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

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

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

@yurzui 对小部件使用鼠标事件进行了一些修改

view.directive.ts

import { ViewRef, Directive, Input, ViewContainerRef } from '@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()
```

```
  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

Some modification by @yurzui to use mouse event on the widgets

view.directive.ts

import { ViewRef, Directive, Input, ViewContainerRef } from '@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 组件 = this.dynamicComponent.addComponent(this.WidgetClasses[widget]);
  let 视图: ViewRef = this.dynamicComponent.container.detach(0);
  this.elements.push({视图,组件});

  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

```
  }
}
```

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

第20章：使用 angular-cli@1.0.0-beta.10 安装第三方插件

第20.1节：添加没有类型定义的第三方库

> 注意，这仅适用于 angular-cli 最高到 1.0.0-beta.10 版本！

有些库或插件可能没有类型定义。没有这些，TypeScript 无法进行类型检查，因此会导致编译错误。这些库仍然可以使用，但方式不同于导入的模块。

1. 在你的页面（index.html）中包含对该库的脚本引用

```
<script src="//cdn.somewhe.re/lib.min.js" type="text/javascript"></script>
<script src="/local/path/to/lib.min.js" type="text/javascript"></script>
```

   - 这些脚本应添加一个全局变量（例如 THREE、mapbox、$ 等）或附加到一个全局变量上

2. 在需要这些脚本的组件中，使用 declare 来初始化一个与所用全局变量名称匹配的变量该库。这让 TypeScript 知道它已经被初始化。1

```
declare var <globalname>: any;
```

有些库会附加到window，需要扩展它才能在应用中访问。

```
interface WindowIntercom extends Window { Intercom: any; }
declare var window: WindowIntercom;
```

3.根据需要在你的组件中使用该库。

```
@Component { ... }
export class AppComponent implements AfterViewInit {
    ...
    ngAfterViewInit() {
        var geometry = new THREE.BoxGeometry( 1, 1, 1 );
        window.Intercom('boot', { ... }
    }
}
```

   - 注意：有些库可能会与 DOM 交互，应在合适的组件生命周期方法中使用。

第20.2节：在angular-cli项目中添加jquery库

1.通过npm安装jquery：

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

> 安装该库的类型定义：

要为库添加类型定义，请执行以下操作：

Chapter 20: Installing 3rd party plugins with angular-cli@1.0.0-beta.10

Section 20.1: Add 3rd party library that does not have typings

> Notice, this is only for angular-cli up to 1.0.0-beta.10 version !

Some libraries or plugins may not have typings. Without these, TypeScript can't type check them and therefore causes compilation errors. These libraries can still be used but differently than imported modules.

1. Include a script reference to the library on your page (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>
```

   - These scripts should add a global (eg. THREE, mapbox, $, etc.) or attach to a global

2. In the component that requires these, use declare to initialize a variable matching the global name used by the lib. This lets TypeScript know that it has already been initialized. 1

```
declare var <globalname>: any;
```

Some libs attach to window, which would need to be extended in order to be accessible in the app.

```
interface WindowIntercom extends Window { Intercom: any; }
declare var window: WindowIntercom;
```

3. Use the lib in your components as needed.

```
@Component { ... }
export class AppComponent implements AfterViewInit {
    ...
    ngAfterViewInit() {
        var geometry = new THREE.BoxGeometry( 1, 1, 1 );
        window.Intercom('boot', { ... }
    }
}
```

   - NOTE: Some libs may interact with the DOM and should be used in the appropriate component lifecycle method.

Section 20.2: Adding jquery library in angular-cli project

1. Install jquery via npm :

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

> Install typings for the library:

To add typings for a library, do the following:

```
typings install jquery --global --save
```

2.将jquery添加到angular-cli-build.js文件的vendorNpmFiles数组中：

这是必需的，以便构建系统能够识别该文件。设置完成后，angular-cli-build.js应如下所示
：

> 浏览 node_modules，查找要添加到vendor文件夹中的文件和文件夹。

```
var Angular2App = require('angular-cli/lib/broccoli/angular2-app');

module.exports = function(defaults) {
  return new Angular2App(defaults, {
    vendorNpmFiles: [
      // ...
      'jquery/dist/*.js'


    ]
  });
};
```

3. 配置 SystemJS 映射以确定 jquery 的查找位置：

SystemJS 配置位于 system-config.ts 中，自定义配置完成后，相关
部分应如下所示：

```
/** 将相对路径映射到 URL。*/
const map: any = {
  'jquery': 'vendor/jquery'
};

/** 用户包配置。*/
const packages: any = {

// jquery 不需要在这里添加任何内容

};
```

4. 在你的 src/index.html 中添加以下行

```
<script src="vendor/jquery/dist/jquery.min.js" type="text/javascript"></script>
```

> 你的其他选项有：

```
<script src="vendor/jquery/dist/jquery.js" type="text/javascript"></script>
```

```
typings install jquery --global --save
```

2. Add jquery to angular-cli-build.js file to vendorNpmFiles array:

This is required so the build system will pick up the file. After setup the angular-cli-build.js should look like this:

> Browse the `node_modules` and look for files and folders you want to add to the vendor folder.

```
var Angular2App = require('angular-cli/lib/broccoli/angular2-app');

module.exports = function(defaults) {
  return new Angular2App(defaults, {
    vendorNpmFiles: [
      // ...
      'jquery/dist/*.js'


    ]
  });
};
```

3. Configure SystemJS mappings to know where to look for jquery :

SystemJS configuration is located in system-config.ts and after the custom configuration is done the related section should look like:

```
/** 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. In your src/index.html add this line

```
<script src="vendor/jquery/dist/jquery.min.js" type="text/javascript"></script>
```

> Your other options are:

```
<script src="vendor/jquery/dist/jquery.js" type="text/javascript"></script>
```

或

```
<script src="/vendor/jquery/dist/jquery.slim.js" type="text/javascript"></script>
```

以及

```
<script src="/vendor/jquery/dist/jquery.slim.min.js" type="text/javascript"></script>
```

5. 在你的项目源文件中导入并使用 jquery 库：

   在你的源 .ts 文件中这样导入 jquery 库：

```
声明 var $:any;

@组件({
})
export class YourComponent {
  ngOnInit() {
$.("button").click(function(){
       // 现在你可以做任何你想做的事情
     });
console.log();
  }
}
```

如果你按照步骤操作正确，现在你的项目中应该已经可以使用jquery库了。祝你使用愉快！

or

```
<script src="/vendor/jquery/dist/jquery.slim.js" type="text/javascript"></script>
```

and

```
<script src="/vendor/jquery/dist/jquery.slim.min.js" type="text/javascript"></script>
```

5. Importing and using jquery library in your project source files:

   Import jquery library in your source .ts files like this:

```
declare var $:any;

@Component({
})
export class YourComponent {
  ngOnInit() {
    $.("button").click(function(){
       // now you can DO, what ever you want
     });
     console.log();
  }
}
```

If you followed the steps correctly you should now have jquery library working in your project. Enjoy!

第21章：生命周期钩子

第21.1节：OnChanges

当一个或多个组件或指令的属性发生变化时触发。

```
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);
    }
}
```

变更事件将被记录

```
name: {
currentValue: '新名称值',
    previousValue: '旧名称值'
},
message: {
currentValue: '新消息值',
    previousValue: '旧消息值'
}
```

第21.2节：OnInit

当组件或指令的属性被初始化时触发。

（在子指令的属性初始化之前）

```
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('组件已准备好！');
    }
}
```

第21.3节：OnDestroy

当组件或指令实例被销毁时触发。

Chapter 21: Lifecycle Hooks

Section 21.1: OnChanges

Fired when one or more of the component or directive properties have been changed.

```
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);
    }
}
```

On change event will log

```
name: {
    currentValue: 'new name value',
    previousValue: 'old name value'
},
message: {
    currentValue: 'new message value',
    previousValue: 'old message value'
}
```

Section 21.2: OnInit

Fired when component or directive properties have been initialized.

(Before those of the child directives)

```
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 !');
    }
}
```

Section 21.3: OnDestroy

Fired when the component or directive instance is destroyed.

```
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('组件已被销毁！');
    }
}
```

第21.4节：AfterContentInit

组件或指令的内容初始化完成后触发。

（紧接 OnInit 之后）

```
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('组件内容已加载！');
    }
}
```

第21.5节：AfterContentChecked

视图完全初始化后触发。

（仅适用于组件）

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

@Component({
selector: 'so-aftercontentchecked-component',
    templateUrl: 'aftercontentchecked-component.html',
    styleUrls: ['aftercontentchecked-component.']
})
class AfterContentCheckedComponent 实现 AfterContentChecked {

    ngAfterContentChecked(): void {
console.log('组件内容已被检查！');
    }
}
```

第21.6节：AfterViewInit

在初始化组件视图及其任何子视图后触发。这是Angular 2生态系统外插件的一个有用生命周期钩子。例如，您可以使用此方法基于Angular 2渲染的标记初始化jQuery日期选择器

```
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 !');
    }
}
```

Section 21.4: AfterContentInit

Fire after the initialization of the content of the component or directive has finished.

(Right after 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!');
    }
}
```

Section 21.5: AfterContentChecked

Fire after the view has been fully initialized.

(Only available for components)

```
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!');
    }
}
```

Section 21.6: AfterViewInit

Fires after initializing both the component view and any of its child views. This is a useful lifecycle hook for plugins outside of the Angular 2 ecosystem. For example, you could use this method to initialize a jQuery date picker based

。

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

@Component({
selector: 'so-afterviewinit-component',
    templateUrl: 'afterviewinit-component.html',
    styleUrls: ['afterviewinit-component.']
})
class AfterViewInitComponent 实现 AfterViewInit {

    ngAfterViewInit(): void {
console.log('内容初始化加载后触发此事件！');
    }
}
```

第21.7节：AfterViewChecked

视图组件检查完成后触发。

（仅适用于组件）

```
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('内容检查完成后触发此事件！');
    }
}
```

第21.8节：DoCheck

允许仅监听指定属性的变化

```
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 {
        // 获取 elements 属性的值
        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));
        }
    }
```

on the markup that Angular 2 has rendered.

```
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!');
    }
}
```

Section 21.7: AfterViewChecked

Fire after the check of the view, of the component, has finished.

(Only available for components)

```
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!');
    }
}
```

Section 21.8: DoCheck

Allows to listen for changes only on specified properties

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

```
}
```

第22章：Angular RXJS Subjects 和带有API请求的 Observables

第22.1节：等待多个请求

一个常见的场景是等待多个请求完成后再继续。这可以通过使用 forkJoin 方法来实现。

在下面的示例中，forkJoin 用于调用两个返回 Observables 的方法。指定在 .subscribe 方法中的回调将在两个 Observable 都完成时调用。由 .subscribe 提供的参数顺序与调用 .forkJoin 时的顺序相匹配。在本例中，先是 posts 然后是 tags。

```
loadData() : void {
    Observable.forkJoin(
        this.blogApi.getPosts(),
        this.blogApi.getTags()
    ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
        this.posts = posts;
        this.tags = tags;
    }));
}
```

第22.2节：基本请求

下面的示例演示了一个简单的HTTP GET请求。 `http.get()` 返回一个 `Observable` 对象，该对象具有方法 `subscribe`。此方法将返回的数据追加到 `posts` 数组中。

```
var posts = []

getPosts(http: Http):void {
    this.http.get(`https://jsonplaceholder.typicode.com/posts`)
        .map(response => response.json())
        .subscribe(post => posts.push(post));
}
```

第22.3节：封装API请求

将 HTTP 处理逻辑封装到自己的类中可能是个好主意。以下类公开了一个获取帖子的方法。它调用了 http.get() 方法，并在返回的 Observable 上调用 .map，将 Response 对象转换为 Post 对象。

```
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)
```

Chapter 22: Angular RXJS Subjects and Observables with API requests

Section 22.1: Wait for multiple requests

One common scenario is to wait for a number of requests to finish before continuing. This can be accomplished using the `forkJoin` method.

In the following example, `forkJoin` is used to call two methods that return `Observables`. The callback specified in the `.subscribe` method will be called when both Observables complete. The parameters supplied by `.subscribe` match the order given in the call to `.forkJoin`. In this case, first `posts` then `tags`.

```
loadData() : void {
    Observable.forkJoin(
        this.blogApi.getPosts(),
        this.blogApi.getTags()
    ).subscribe((([posts, tags]: [Post[], Tag[]]) => {
        this.posts = posts;
        this.tags = tags;
    }));
}
```

Section 22.2: Basic request

The following example demonstrates a simple HTTP GET request. `http.get()` returns an `Observable` which has the method `subscribe`. This one appends the returned data to the `posts` array.

```
var posts = []

getPosts(http: Http):void {
    this.http.get(`https://jsonplaceholder.typicode.com/posts`)
        .map(response => response.json())
        .subscribe(post => posts.push(post));
}
```

Section 22.3: Encapsulating API requests

It may be a good idea to encapsulate the HTTP handling logic in its own class. The following class exposes a method for getting Posts. It calls the `http.get()` method and calls `.map` on the returned `Observable` to convert the `Response` object to a `Post` object.

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

前面的示例使用了一个Post类来保存返回的数据，该类可能如下所示：

```
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;
  }
}
```

组件现在可以使用BlogApi类轻松获取Post数据，而无需关心Http类的具体实现。

```
        });
    }
}
```

The previous example uses a `Post` class to hold the returned data, which could look as follows:

```
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;
  }
}
```

A component now can use the `BlogApi` class to easily retrieve `Post` data without concerning itself with the workings of the `Http` class.

第23章：服务与依赖注入

第23.1节：示例服务

services/my.service.ts

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

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return this.data;
  }
}
```

在引导方法中注册服务提供者将使该服务在全局范围内可用。

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from 'app.component.ts';
import { MyService } from 'services/my.service';

bootstrap(AppComponent, [MyService]);
```

在 RC5 版本中，全局服务提供者的注册可以在模块文件内完成。为了在整个应用程序中获得服务的单一实例，服务应在应用程序的 ngmodule 的 providers 列表中声明。app_module.ts

```
import { NgModule }       from '@angular/core';
import { BrowserModule  } from '@angular/platform-browser';
import { routing, appRoutingProviders } from './app-routes/app.routes';
import { HttpModule} 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 {}
```

在 MyComponent 中的使用

Chapter 23: Services and Dependency Injection

Section 23.1: Example service

services/my.service.ts

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

@Injectable()
export class MyService {
  data: any = [1, 2, 3];

  getData() {
    return this.data;
  }
}
```

The service provider registration in the bootstrap method will make the service available globally.

main.ts

```
import { bootstrap } from '@angular/platform-browser-dynamic';
import { AppComponent } from 'app.component.ts';
import { MyService } from 'services/my.service';

bootstrap(AppComponent, [MyService]);
```

In version RC5 global service provider registration can be done inside the module file. In order to get a single instance of your service for your whole application the service should be declared in the providers list in the ngmodule of your application. *app_module.ts*

```
import { NgModule }       from '@angular/core';
import { BrowserModule  } from '@angular/platform-browser';
import { routing, appRoutingProviders } from './app-routes/app.routes';
import { HttpModule} 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 {}
```

Usage in `MyComponent`

components/my.component.ts

在应用组件中注册应用提供者的另一种方法。如果我们在组件级别添加提供者，每当组件被渲染时，都会创建服务的新实例。

```
import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({
    ...
    ...
    providers:[MyService] //
 })
export class MyComponent implements OnInit {
    data: any[];
    // 创建一个私有变量 myService，类型为 MyService，用于使用
constructor(private myService: MyService) { }

    ngOnInit() {
        this.data = this.myService.getData();
    }
}
```

第23.2节：使用 Promise.resolve 的示例

services/my.service.ts

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

@Injectable()
export class MyService {
    data: any = [1, 2, 3];

    getData() {
        return Promise.resolve(this.data);
    }
}
```

getData() 现在表现得像一个 REST 调用，创建了一个 Promise，并立即被解决。结果可以在 .then() 中处理，错误也可以被检测到。这是异步方法的良好实践和惯例。

components/my.component.ts

```
import { Component, OnInit } from '@angular/core';
import { MyService } from '../services/my.service';

@Component({...})
export class MyComponent implements OnInit {
    data: any[];
    // 创建一个私有变量 myService，类型为 MyService，用于使用
constructor(private myService: MyService) { }

    ngOnInit() {
        // 使用"箭头"函数设置数据
        this.myService.getData().then(data => this.data = data);
    }
}
```

components/my.component.ts

Alternative approach to register application providers in application components. If we add providers at component level whenever the component is rendered it will create a new instance of the service.

```
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();
    }
}
```

Section 23.2: Example with Promise.resolve

services/my.service.ts

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

@Injectable()
export class MyService {
    data: any = [1, 2, 3];

    getData() {
        return Promise.resolve(this.data);
    }
}
```

`getData()` now acts likes a REST call that creates a Promise, which gets resolved immediately. The results can be handheld inside `.then()` and errors can also be detected. This is good practice and convention for asynchronous methods.

components/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);
    }
}
```

第23.3节：测试服务

给定一个可以登录用户的服务：

```
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();
  }
}
```

可以这样测试：

```
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('应该被创建', inject([AuthService], (service: AuthService) => {
   expect(service).toBeTruthy();
  }));

  // 备选方案 1
它（'如果传递了正确的凭据，则应登录用户'， 异步 (
   注入 ([AuthService]， 异步  (authService)  => {
      const backend: MockBackend = TestBed.get(ConnectionBackend);
      const http: Http = TestBed.get(Http);

backend.connections.subscribe((c: MockConnection) => {
c.mockRespond(
```

Section 23.3: Testing a Service

Given a service that can login a user:

```
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();
  }
}
```

It can be tested like this:

```
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',
      });
    }))
  );

  // 备选方案 2
it('如果传入正确的凭证，应登录用户', 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',
    });
  });

  // 备选方案 3
it('如果传入正确的凭证，应登录用户', 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',
            },
          }),
        ),
      );
    });
```

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

```
    try {
      const result = await authService.signIn({ password: 'ok', user: 'bruno' });

      expect(result).toEqual({
        accessToken: 'abcdef',
      });

      done();
    } catch (err) {
      fail(err);
      done();
    }
  });
});
```

第24章：服务工作者

我们将学习如何在Angular中设置服务工作者，使我们的网页应用具备离线功能。

服务工作者是一种特殊的脚本，在浏览器后台运行，管理对特定源的网络请求。它最初由应用安装，并常驻于用户的机器/设备上。当浏览器加载来自该源的页面时，它会被激活，并且可以选择在页面加载期间响应HTTP请求。

第24.1节：向我们的应用添加服务工作者

首先，如果你正在查看mobile.angular.io，标志--mobile已不再有效。

因此，开始时我们可以使用Angular CLI创建一个普通项目。

```
ng new serviceWorking-example
cd serviceWorking-example
```

现在重要的是，要告诉 Angular CLI 我们想使用服务工作者，需要执行：

ng set apps.0.serviceWorker=true

如果由于某种原因你没有安装 @angular/service-worker，你会看到一条消息：

> 你的项目配置了 serviceWorker = true，但未安装 @angular/service-worker。请运行 `npm install --save-dev @angular/service-worker` 并重试，或者在你的 .angular-cli.json 中运行 `ng set apps.0.serviceWorker=false`。

检查 .angular-cli.json，你现在应该看到："serviceWorker": true

当此标志为 true 时，生产构建将配置服务工作者。

将生成一个 ngsw-manifest.json 文件（如果我们在项目根目录下已经创建了 ngsw-manifest.json，则会进行增强，通常这样做是为了指定路由，未来这可能会自动完成），该文件位于 dist/ 根目录，服务工作者脚本也会被复制到那里。一个简短的脚本将被添加到 index.html 以注册服务工作者。

现在如果我们以生产模式构建应用 ng build --prod

并检查 dist/ 文件夹。

你会看到那里有三个新文件：

- worker-basic.min.js
- sw-register.HASH.bundle.js
- ngsw-manifest.json

此外，index.html 现在包含了这个 sw-register 脚本，它为我们注册了一个 Angular 服务工作者（ASW）。

在浏览器中刷新页面（由Chrome的Web服务器提供）

打开开发者工具。进入应用程序 -> 服务工作线程

Chapter 24: Service Worker

We will see how to set up a service working on angular, to allow our web app to have offline capabilities.

A Service worker is a special script which runs in the background in the browser and manages network requests to a given origin. It's originally installed by an app and stays resident on the user's machine/device. It's activated by the browser when a page from its origin is loaded and has the option to respond to HTTP requests during the page loading

Section 24.1: Add Service Worker to our app

First in case you are consulting mobile.angular.io the flag --mobile doesn't work anymore.

So to start , we can create a normal project with angular cli.

```
ng new serviceWorking-example
cd serviceWorking-example
```

Now the important thing, to said to angular cli that we want to use service worker we need to do:

ng set apps.0.serviceWorker=true

If for some reason you don't have @angular/service-worker installed, you will see a message:

> Your project is configured with serviceWorker = true, but @angular/service-worker is not installed. Run `npm install --save-dev @angular/service-worker` and try again, or run `ng set apps.0.serviceWorker=false` in your .angular-cli.json.

Check the .angular-cli.json and you now should see this: "serviceWorker": true

When this flag is true, production builds will be set up with a service worker.

A ngsw-manifest.json file will be generated (or augmented in case we have create a ngsw-manifest.json in the root of the project, usually this is done to specify the routing ,in a future this will probably be done automatic) in the dist/ root, and the service worker script will be copied there. A short script will be added to index.html to register the service worker.

Now if we build the app in production mode ng build --prod

And check dist/ folder.

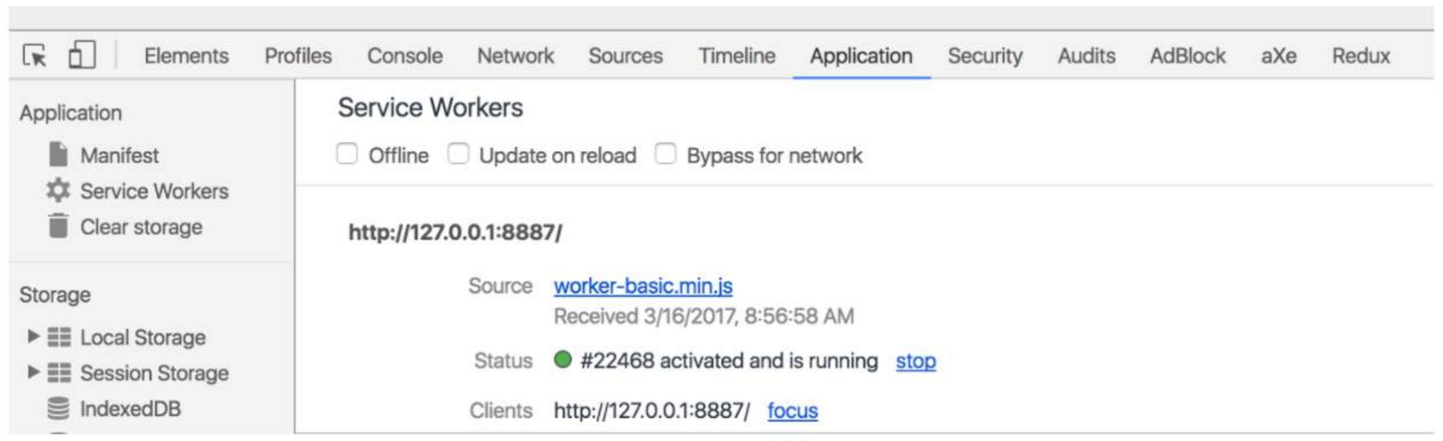
You will see three new files there :

- worker-basic.min.js
- sw-register.HASH.bundle.js
- ngsw-manifest.json

Also, index.html now includes this sw-register script, which registers a Angular Service Worker (ASW) for us.

Refresh the page in your browser (served by the Web Server for Chrome)

Open Developer Tools. Go to the Application -> Service Workers


Elements Profiles Console Network Sources Timeline Application Security Audits AdBlock aXe Redux
Application
Manifest
Service Workers
Clear storage
Storage
Local Storage
Session Storage
IndexedDB

Service Workers

Offline Update on reload Bypass for network

http://127.0.0.1:8887/

Source worker-basic.min.js
Received 3/16/2017, 8:56:58 AM

Status #22468 activated and is running stop

Clients http://127.0.0.1:8887/ focus

很好，现在服务工作线程已启动并运行！

现在我们的应用程序应该加载更快，并且我们应该能够离线使用该应用。

现在如果你在Chrome控制台启用离线模式，你应该会看到我们的应用在 http://localhost:4200/index.html 无需连接互联网也能正常工作。

但在 http://localhost:4200/ 我们遇到了问题，页面无法加载，这是因为静态内容缓存只提供清单中列出的文件。

例如，如果清单声明了 /index.html 的URL，针对 /index.html 的请求将由缓存响应，但对 / 或 /some/route 的请求将会访问网络。

这就是路由重定向插件的作用。它从清单中读取路由配置，并将配置的路由重定向到指定的索引路由。

目前，这部分配置必须手动编写（2017年7月19日）。最终，它将从应用程序源代码中的路由配置自动生成。

所以现在如果我们在项目根目录创建 ngsw-manifest.json 文件

```
{
  "routing": {
    "routes": {
      "/": {
        "prefix": false
      }
    },
    "index": "/index.html"
  }
}
```

然后我们重新构建应用，现在当我们访问 http://localhost:4200/ 时，应该会被重定向到 http://localhost:4200/index.html。

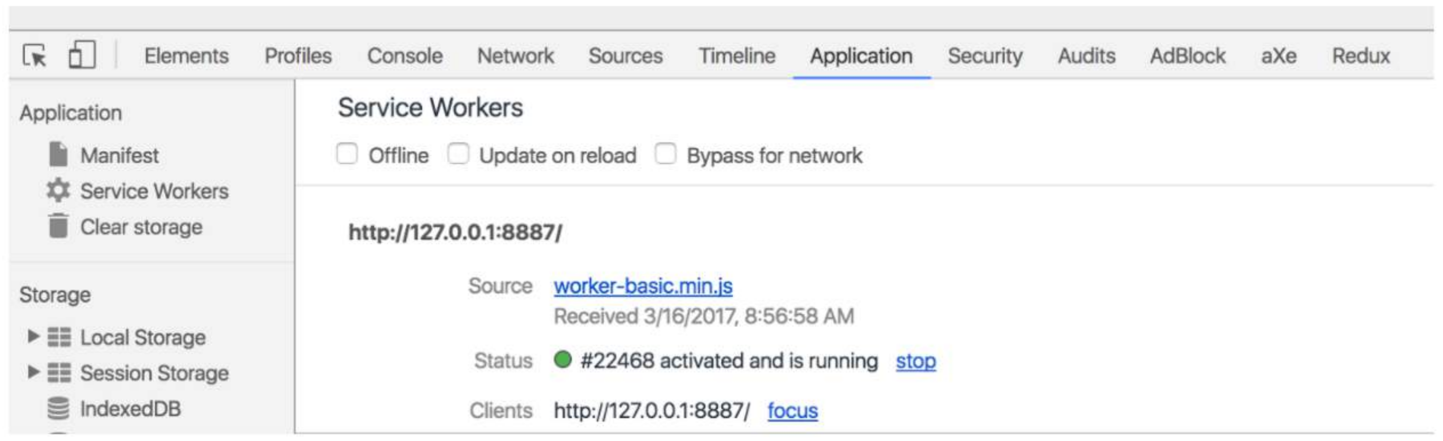
有关路由的更多信息，请阅读 官方文档，链接如下

这里你可以找到更多关于服务工作者的文档：

https://developers.google.com/web/fundamentals/getting-started/primers/service-workers

https://docs.google.com/document/d/19S5ozevWighny788nI99worpcIMDnwWVmaJDGf_RoDY/edit#

这里你可以看到使用 SW precache 库实现服务工作者的另一种方法：


Elements Profiles Console Network Sources Timeline Application Security Audits AdBlock aXe Redux
Application
Manifest
Service Workers
Clear storage
Storage
Local Storage
Session Storage
IndexedDB

Service Workers

Offline Update on reload Bypass for network

http://127.0.0.1:8887/

Source worker-basic.min.js
Received 3/16/2017, 8:56:58 AM

Status #22468 activated and is running stop

Clients http://127.0.0.1:8887/ focus

Good now the Service Worker is up and running!

Now our application, should load faster and we should be able to use the app offline.

Now if you enable the offline mode in the chrome console , you should see that our app in http://localhost:4200/index.html is working without connection to internet.

But in http://localhost:4200/ we have a problem and it doesn't load, this is due to the static content cache only serves files listed in the manifest.

For example, if the manifest declares a URL of /index.html, requests to /index.html will be answered by the cache, but a request to / or /some/route will go to the network.

That's where the route redirection plugin comes in. It reads a routing config from the manifest and redirects configured routes to a specified index route.

Currently, this section of configuration must be hand-written (19-7-2017). Eventually, it will be generated from the route configuration present in the application source.

So if now we create or ngsw-manifest.json in the root of the project

```
{
  "routing": {
    "routes": {
      "/": {
        "prefix": false
      }
    },
    "index": "/index.html"
  }
}
```

And we build again our app, now when we go to http://localhost:4200/, we should be redirected to http://localhost:4200/index.html.

For further information about routing read the official documentation here

Here you can find more documentation about service workers:

https://developers.google.com/web/fundamentals/getting-started/primers/service-workers

https://docs.google.com/document/d/19S5ozevWighny788nI99worpcIMDnwWVmaJDGf_RoDY/edit#

And here you can see an alternative way to implement the service working using SW precache library :

https://coryrylan.com/blog/fast-offline-angular-apps-with-service-workers

https://coryrylan.com/blog/fast-offline-angular-apps-with-service-workers

第25章：EventEmitter 服务

第25.1节：捕获事件

创建一个服务-

```
import {EventEmitter} from 'angular2/core';
export class NavService {
navchange: EventEmitter<number> = new EventEmitter();
    constructor() {}
emitNavChangeEvent(number) {
        this.navchange.emit(number);
    }
getNavChangeEmitter() {
        return this.navchange;
    }
}
```

创建一个组件来使用该服务-

```
import {Component} from 'angular2/core';
import {NavService} from '../services/NavService';

@Component({
selector: 'obs-comp',
    template: `obs 组件, 项目: {{item}}`
    })
    export class 观察组件 {
   item: number = 0;
subscription: any;
constructor(private navService:导航服务) {}
   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 导航 {
项目 = 1;
constructor(private navService:NavService) {}
    selectedNavItem(item: number) {
console.log('选中的导航项 ' + item);
        this.navService.emitNavChangeEvent(item);
    }
}
```

Chapter 25: EventEmitter Service

Section 25.1: Catching the event

Create a service-

```
import {EventEmitter} from 'angular2/core';
export class NavService {
    navchange: EventEmitter<number> = new EventEmitter();
    constructor() {}
    emitNavChangeEvent(number) {
        this.navchange.emit(number);
    }
    getNavChangeEmitter() {
        return this.navchange;
    }
}
```

Create a component to use the service-

```
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);
    }
}
```

第25.2节：实时示例

可以在这里找到一个实时示例。

第25.3节：类组件

```
@Component({
  selector: 'zippy',
  template: `
  <div class="zippy">
    <div (click)="toggle()">切换</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);
    }
  }
}
```

第25.4节：类概述

```
class EventEmitter extends Subject {
    constructor(isAsync?: boolean)
    emit(value?: T)
subscribe(generatorOrNext?: any, error?: any, complete?: any) : any
}
```

第25.5节：触发事件

```
<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>
```

Section 25.2: Live example

A live example for this can be found here.

Section 25.3: Class Component

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

Section 25.4: Class Overview

```
class EventEmitter extends Subject {
    constructor(isAsync?: boolean)
    emit(value?: T)
    subscribe(generatorOrNext?: any, error?: any, complete?: any) : any
}
```

Section 25.5: Emmiting Events

```
<zippy (open)="onOpen($event)" (close)="onClose($event)"></zippy>
```

第26章：使用ChangeDetectionStrategy优化渲染

第26.1节：默认与OnPush

考虑以下组件，它有一个输入myInput和一个名为someInternalValue的内部值。它们都在组件的模板中使用。

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

@Component({
template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input() myInput: any;

  someInternalValue: any;

  // ...
}
```

默认情况下，组件装饰器中的changeDetection:属性将设置为ChangeDetectionStrategy
.Default；在示例中是隐式的。在这种情况下，模板中任何值的变化都会触发MyComponent的重新渲染。换句话说，
如果我更改myInput或someInternalValueAngular 2将会消耗资源并重新渲染组件。

但是，假设我们只想在输入变化时才重新渲染。考虑以下将
changeDetection:设置为ChangeDetectionStrategy.OnPush的组件

```
import {组件,变更检测策略,输入} 来自 '@angular/core';

@组件({
变更检测: 变更检测策略.OnPush
 模板:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input() myInput: any;

  someInternalValue: any;

  // ...
}
```

通过设置 变更检测: 为 变更检测策略.OnPush，MyComponent 只有在其输入
发生变化时才会重新渲染。在这种情况下，myInput 需要从其父组件接收一个新值以触发重新渲染。

Chapter 26: Optimizing rendering using ChangeDetectionStrategy

Section 26.1: Default vs OnPush

Consider the following component with one input `myInput` and an internal value called `someInternalValue`. Both of them are used in a component's template.

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

@Component({
  template:`
  <div>
    <p>{{myInput}}</p>
    <p>{{someInternalValue}}</p>
  </div>
  `
})
class MyComponent {
  @Input() myInput: any;

  someInternalValue: any;

  // ...
}
```

By default, the `changeDetection:` property in the component decorator will be set to `ChangeDetectionStrategy.Default`; implicit in the example. In this situation, any changes to any of the values in the template will trigger a re-render of `MyComponent`. In other words, if I change `myInput` or `someInternalValue` angular 2 will exert energy and re-render the component.

Suppose, however, that we only want to re-render when the inputs change. Consider the following component with `changeDetection:` set to `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;

  // ...
}
```

By setting `changeDetection:` to `ChangeDetectionStrategy.OnPush`, `MyComponent` will only re-render when its inputs change. In this case, `myInput` will need to receive a new value from its parent to trigger a re-render.

第27章：Angular 2 表单更新

第27.1节：Angular 2 ：模板驱动表单

```
import { 组件 } 来自 '@angular/core' ;
import { 路由器 , ROUTER_DIRECTIVES} 来自 '@angular/router' ;
import { NgForm }   来自 '@angular/forms' ;

@组件({
    选择器: 'login',
    模板: `
<h2>登录</h2>
<form #f="ngForm" (ngSubmit)="login(f.value,f.valid)" 无验证>
    <div>
        <标签>用户名</标签>
        <input type="text" [(ngModel)]="username" placeholder="请输入用户名" required>
    </div>
    <div>
        <label>密码</label>
        <input type="password" name="password" [(ngModel)]="password" placeholder="请输入密码"
required>
    </div>
    <input class="btn-primary" type="submit" value="登录">
</form>`
   //对于长表单，我们可以使用 **templateUrl** 替代 template
})

export class LoginComponent{

    constructor(private router : Router){ }

    login (formValue: any, valid: boolean){
        console.log(formValue);

        if(valid){
console.log(valid);
        }
    }
}
```

第27.2节：Angular 2 表单 - 自定义邮箱/密码验证

实时演示 点击..

应用索引 ts

```
import {bootstrap} from '@angular/platform-browser-dynamic';
import {MyForm} from './my-form.component.ts';

bootstrap(MyForm);
```

自定义验证器

```
import {Control} from @'angular/common';

export class CustomValidators {
  static emailFormat(control: Control): [[key: string]: boolean] {
```

Chapter 27: Angular 2 Forms Update

Section 27.1: Angular 2 : Template Driven Forms

```
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);
        }
    }
}
```

Section 27.2: Angular 2 Form - Custom Email/Password Validation

For live demo click..

App index ts

```
import {bootstrap} from '@angular/platform-browser-dynamic';
import {MyForm} from './my-form.component.ts';

bootstrap(MyForm);
```

Custom validator

```
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};
  }
}
```

表单组件 ts

```
import {Component} from '@angular/core';
import {FORM_DIRECTIVES, NgForm, FormBuilder, Control, ControlGroup, Validators} from
'@angular/common';
import {CustomValidators} from './custom-validators';

@Component({
selector: '我的表单',
  templateUrl: 'app/my-form.component.html',
  directives: [FORM_DIRECTIVES],
styleUrls: ['styles.css']
})
export class 我的表单 {
  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);
  }
}
```

表单组件 HTML

```
<form [ngFormModel]="group" (ngSubmit)="onSubmit()" novalidate>

  <div>
    <label for="email">邮箱:</label>
    <input type="email" id="email" [ngFormControl]="email">

    <ul *ngIf="email.dirty && !email.valid">
      <li *ngIf="email.hasError('required')">邮箱为必填项</li>
    </ul>
  </div>

  <div>
    <label for="password">密码:</label>
    <input type="password" id="password" [ngFormControl]="password">

    <ul *ngIf="password.dirty && !password.valid">
```

```
    let pattern:RegExp = /\S+@\S+\.\S+/;
    return pattern.test(control.value) ? null : {"emailFormat": true};
  }
}
```

Form Components 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);
  }
}
```

Form Components 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')">密码为必填项</li><li *ngIf="password.hasErr
        or('minlength')">密码长度至少需要4个字符</li>

    </ul>
  </div>

  <button type="submit">注册</button>
```

第27.3节：带多重控制验证的简单密码更改表单

以下示例使用了在RC3中引入的新表单API。

pw-change.template.html

```
<form class="container" [formGroup]="pwChangeForm">
    <label for="current">当前密码</label>
    <input id="current" formControlName="current" type="password" required><br />

    <label for="newPW">新密码</label>
    <input id="newPW" formControlName="newPW" type="password" required><br/>
    <div *ngIf="newPW.touched && newPW.newIsNotOld">
新密码不能与当前密码相同。


    <label for="confirm">确认新密码</label>
    <input id="confirm" formControlName="confirm" type="password" required><br />
    <div *ngIf="confirm.touched && confirm.errors.newMatchesConfirm">
确认不匹配。


<button type="submit">提交</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;

    // 存储指向 FormControls 路径的属性使我们的模板更简洁
current: AbstractControl;
    newPW: AbstractControl;
    confirm: AbstractControl;

    constructor(private fb: FormBuilder) { }
    ngOnInit() {
        this.pwChangeForm = this.fb.group({
```

```
      <li *ngIf="password.hasError('required')">A password is required</li>
      <li *ngIf="password.hasError('minlength')">A password needs to have at least 4
characterss</li>
    </ul>
  </div>

  <button type="submit">Register</button>

</form>
```

Section 27.3: Simple Password Change Form with Multi Control Validation

The below examples use the new form API introduced in 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]
    }, {
      // 这里我们创建用于整个组的验证器
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;
  }

  // 我们的跨控件验证器如下
  // 注意：它们接收的类型是 FormGroup 而不是 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;
  }
}
```

这里可以找到包含一些 bootstrap 类的 gist。

第27.4节：Angular 2 表单（响应式表单）及注册表单和确认密码验证

app.module.ts

将这些添加到你的 app.module.ts 文件中以使用响应式表单

```
import { NgModule } from '@angular/core';
  import { BrowserModule } from '@angular/platform-browser';
  import { FormsModule, ReactiveFormsModule } from '@angular/forms';
  import { AppComponent } from './app.component';
  @NgModule({
   imports: [
```

```
      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;
  }
}
```

A gist including some bootstrap classes can be found here.

Section 27.4: Angular 2 Forms (Reactive Forms) with registration form and confirm password validation

app.module.ts

Add these into your app.module.ts file to use reactive forms

```
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 变量包含我们所有的表单值。将其存储到你想要的位置
            this.addForm.reset();// 这将重置我们的表单值为 null
        }
    }
}
```

app.component.html

```
<div>
  <form [formGroup]="addForm">
```

```
        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="请输入用户名" formControlName="username" />
    <input type="text" placeholder="请输入邮箱地址" formControlName="email"/>
    <input type="password" placeholder="请输入密码" formControlName="password" />
    <input type="password" placeholder="确认密码" name="password2"
formControlName="password2"/>
    <div class='error' *ngIf="addForm.controls.password2.touched">
     <div class="alert-danger errormessageadduser" *ngIf="addForm.hasError('mismatchedPasswords')">
                                  密码不匹配
   </div>
</div>
<select name="Role" formControlName="role">
     <option value="admin" >管理员</option>
     <option value="Accounts">财务</option>
     <option value="guest">访客</option>
</select>
<br/>
<br/>
<button type="submit" (click)="addUser()"><span><i class="fa fa-user-plus" aria-
hidden="true"></i></span> 添加用户 </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
            };
        }
    }
}
```

第27.5节：Angular 2：响应式表单（又称模型驱动表单）

此示例使用Angular 2.0.0最终版本

registration-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: ['默认值', [<any>Validators.required, <any>Validators.minLength(2)]],
   email: ['default@defa.ult', [<any>Validators.required, <any>Validators.minLength(2)]]
})
```

```
    <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>
<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
            };
        }
    }
}
```

Section 27.5: Angular 2: Reactive Forms (a.k.a Model-driven Forms)

This example uses Angular 2.0.0 Final Release

registration-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)]]
})
```

registration-form.html

```
<form [formGroup]="exampleForm" novalidate (ngSubmit)="submit(exampleForm)">
    <label>姓名: </label>
    <input type="text" formControlName="name"/>
    <label>邮箱: </label>
    <input type="email" formControlName="email"/>
    <button type="submit">提交</button>
```

第27.6节：Angular 2 - 表单构建器

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({
名字 : new FormControl({value: null}, Validators.compose([Validators.必填,
Validators.最大长度(15)])),
姓氏 : new FormControl({value: null}, Validators.compose([Validators.必填,
Validators.最大长度(15)])),
邮箱 : new FormControl({value: null}, Validators.compose([
        Validators.必填,
Validators.最大长度(15),
        Validators.模式(this.emailRegex)]))
    });
  }
}
```

form.component.html

```
<form class="form-details" role="form" [formGroup]="form">
    <div class="row input-label">
      <label class="form-label" for="FirstName">名字</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">姓氏</label>
      <input
       [formControl]="form.controls['LastName']"
       type="text"
       class="form-control"
```

registration-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>
```

Section 27.6: Angular 2 - 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">电子邮件</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">提交</button>
    </div>
  </div>
</form>
```

```
        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>
```

第28章：检测调整大小事件

第28.1节：监听窗口调整大小事件的组件

假设我们有一个组件会在某个窗口宽度时隐藏。

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

@Component({
  ...
  模板: `
    <div>
      <p [hidden]="!visible" (window:resize)="onResize($event)" >现在你看到我了...</p>
      <p>现在你看不到我了！</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 {
// 每当窗口宽度小于768时，隐藏此组件。
this.visible = false;
    }
}
}
```

模板中的一个 p 标签将在 visible 为 false 时隐藏。 visible 的值会在 onResize 事件处理程序被调用时改变。该调用发生在每次 window:resize 触发事件时。

Chapter 28: Detecting resize events

Section 28.1: A component listening in on the window resize event

Suppose we have a component which will hide at a certain window width.

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

@Component({
  ...
  template: `
    <div>
      <p [hidden]="!visible" (window:resize)="onResize($event)" >Now you see me...</p>
      <p>now you don't!</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;
    }
  }
}
```

A p tag in our template will hide whenever `visible` is false. `visible` will change value whenever the `onResize` event handler is invoked. Its call occurs every time `window:resize` fires an event.

第29章：测试ngModel

这是一个关于如何测试Angular2中带有ngModel的组件的示例。

第29.1节：基础测试

```
import { BrowserModule } from '@angular/platform-browser';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpModule } 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 夹具:any;
  let 元素:any;
  let 上下文: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('应该改变组件的值', () => {
        let input = fixture.nativeElement.querySelector("input");
        input.value = "用户名";
        dispatchEvent(input, 'input');
        fixture.detectChanges();

fixture.whenStable().then(() => {
            //此按钮触发事件以保存组件的值到component.value
fixture.nativeElement.querySelectorAll('button')[0].click();
            expect(context.value).toBe("Username");
```

Chapter 29: Testing ngModel

Is a example for how you can test a component in Angular2 that have a ngModel.

Section 29.1: Basic test

```
import { BrowserModule } from '@angular/platform-browser';
import { CommonModule } from '@angular/common';
import { FormsModule } from '@angular/forms';
import { HttpModule } 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");
        });
    });
});
```

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

第30章：功能模块

第30.1节：功能模块

```
// 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 { }
```

现在，在你的根目录（通常是app.module.ts）中：

```
// 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({
  // 在根模块中导入 MyFeatureModule
imports:      [ BrowserModule, MyFeatureModule ],
  declarations: [ AppComponent ],
bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

Chapter 30: Feature Modules

Section 30.1: A Feature Module

```
// 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 { }
```

Now, in your root (usually `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 { }
```

第31章：在 Angular 2 中引导空模块

第31.1节：空模块

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

@NgModule({
declarations: [], // 你的模块拥有的组件。
imports: [], // 你的模块所需的其他模块。
providers: [], // 你的模块可用的提供者。
bootstrap: [] // 引导此根组件。
})
export class MyModule {}
```

这是一个空模块，不包含声明、导入、提供者或引导的组件。将其用作参考。

第31.2节：应用根模块

```
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 { }
```

第31.3节：引导你的模块

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

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

在此示例中，`MyModule` 是包含您的根组件的模块。通过引导`MyModule`，您的 Angular 2 应用程序即可启动。

第31.4节：带有网络功能的网页浏览器模块

```
// app.module.ts

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

@NgModule({
declarations: [MyRootComponent],
  imports: [BrowserModule, HttpModule],
  bootstrap: [MyRootComponent]
```

Chapter 31: Bootstrap Empty module in angular 2

Section 31.1: An empty module

```
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 {}
```

This is an empty module containing no declarations, imports, providers, or components to bootstrap. Use this a reference.

Section 31.2: Application Root Module

```
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 { }
```

Section 31.3: Bootstrapping your module

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

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

In this example, `MyModule` is the module containing your root component. By bootstrapping `MyModule` your Angular 2 app is ready to go.

Section 31.4: A module with networking on the web browser

```
// app.module.ts

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

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

MyRootComponent 是打包在MyModule中的根组件。它是您的 Angular 2 应用程序的入口点。

第31.5节：使用工厂类的静态引导

我们可以通过获取生成的工厂类的纯 ES5 Javascript 输出，来静态引导应用程序。
然后我们可以使用该输出引导应用程序：

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

// 使用应用模块工厂启动。
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

这将使应用程序包更小，因为所有模板编译都已在构建步骤中完成，使用的是 ngc 或直接调用其内部功能。

```
})
export class MyModule {}
```

`MyRootComponent` is the root component packaged in `MyModule`. It is the entry point to your Angular 2 application.

Section 31.5: Static bootstrapping with factory classes

We can statically bootstrap an application by taking the plain ES5 Javascript output of the generated factory classes. Then we can use that output to bootstrap the application:

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

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

This will cause the application bundle to be much smaller, because all the template compilation was already done in a build step, using either ngc or calling its internals directly.

第32章：模块的懒加载

第32.1节：懒加载示例

懒加载模块有助于减少启动时间。通过懒加载，应用程序不需要一次性加载所有内容，只需加载用户在应用首次加载时预期看到的内容。被懒加载的模块只有在用户导航到其路由时才会被加载。

app/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
  ],
声明: [
   AppComponent,
    EagerComponent
  ],
bootstrap: [AppComponent]
})
export class AppModule {}
```

app/app.component.ts

```
import { Component } from '@angular/core';
@Component({
selector: 'my-app',
  template: `<h1>我的应用</h1>    <nav>
      <a routerLink="eager">预加载</a>
      <a routerLink="lazy">懒加载</a>
    </nav>
    <router-outlet></router-outlet>
  `
})
export class AppComponent {}
```

app/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);
```

app/eager.component.ts

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

Chapter 32: Lazy loading a module

Section 32.1: Lazy loading example

Lazy loading modules helps us decrease the startup time. With lazy loading our application does not need to load everything at once, it only needs to load what the user expects to see when the app first loads. Modules that are lazily loaded will only be loaded when the user navigates to their routes.

app/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 {}
```

app/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 {}
```

app/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);
```

app/eager.component.ts

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

```
@Component({
template: '`<p>急切组件</p>`'
})
export class 急切组件 {}
```

LazyModule 没有什么特别的，除了它有自己的路由和一个名为 LazyComponent 的组件（但不一定非要这样命名你的模块或类似的东西）。

app/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 延迟模块 {}
```

app/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);
```

app/lazy.component.ts

```
import { Component } from '@angular/core';
@Component({
template: `<p>Lazy 组件</p>`
})
export class LazyComponent {}
```

```
@Component({
  template: '`<p>Eager Component</p>`'
})
export class EagerComponent {}
```

There's nothing special about LazyModule other than it has its own routing and a component called LazyComponent (but it's not necessary to name your module or simliar so).

app/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 {}
```

app/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);
```

app/lazy.component.ts

```
import { Component } from '@angular/core';
@Component({
  template: `<p>Lazy Component</p>`
})
export class LazyComponent {}
```

第33章：高级组件示例

第33.1节：带预览的图片选择器

在本示例中，我们将创建一个图片选择器，可以在上传前预览您的图片。预览器还支持将文件拖放到输入框中。在本示例中，我只介绍单文件上传，但您可以稍作调整以实现多文件上传功能。

image-preview.html

这是我们图片预览的HTML布局

```
<!-- 当未选择文件时作为占位符的图标 -->
<i class="material-icons">cloud_upload</i>

<!-- 文件输入，仅接受图片。通过Angular的原生
(change)事件监听器检测文件是否被选择/更改 -->
<input type="file" accept="image/*" (change)="updateSource($event)">

<!-- 当文件被选择后显示的图片占位符。仅当'source'不为空时显示 -->
<img *ngIf="source" [src]="source" src="">
```

image-preview.ts

这是我们<image-preview>组件的主文件

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

@Component({
selector: 'image-preview',
    styleUrls: [ './image-preview.css' ],
    templateUrl: './image-preview.html'
})
export class MtImagePreviewComponent {

    // 当选择文件时触发事件。这里返回文件本身
    @Output() onChange: EventEmitter<File> = new EventEmitter<File>();

constructor() {}

    // 如果输入发生变化（选择了文件），我们将文件投影到图片预览器中
updateSource($event: Event) {
        // 我们通过 $event.target['files'][0] 访问文件
        this.projectImage($event.target['files'][0]);
    }

    // 使用 FileReader 从输入中读取文件
source:string = '';
    projectImage(file: File) {
        let reader = new FileReader;
        // TODO: 定义 'e' 的类型
reader.onload = (e: any) => {
            // 简单地将 e.target.result 设置为布局中 <img> 的 src
```

Chapter 33: Advanced Component Examples

Section 33.1: Image Picker with Preview

In this example, we are going to create an image picker that previews your picture before uploading. The previewer also supports drag and dropping files into the input. In this example, I am only going to cover uploading single files, but you can tinker a bit to get multi file upload working.

image-preview.html

This is the html layout of our image preview

```
<!-- 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="">
```

image-preview.ts

This is the main file for our `<image-preview>` component

```
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 he 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);
            };
            // 这将处理我们的文件并获取其属性/数据
          reader.readAsDataURL(file);
        }
}
```

another.component.html

```
<form (ngSubmit)="submitPhoto()">
    <image-preview (onChange)="getFile($event)"></image-preview>
    <button type="submit">上传</button>
```

就是这样。比 AngularJS 1.x 版本简单多了。我实际上是基于我在 AngularJS 1.5.5 中制作的旧版本组件制作的这个组件。

第33.2节：通过输入过滤表格值

导入 ReactiveFormsModule，然后

```
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 我的组件 implements OnInit, OnDestroy {

  public control = new 表单控件('');

public content: { id: 数字; name: 字符串; }[];

  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();
  }
```

```
            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>
```

And that's it. Way more easier than it was in AngularJS 1.x. I actually made this component based on an older version I made in AngularJS 1.5.5.

Section 33.2: Filter out table values by the input

Import ReactiveFormsModule, and then

```
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();
  }
}
```

```
}
```

第34章：绕过对可信值的清理

参数	详情
选择器	在html中引用组件的标签名
template(templateUrl)	是一个字符串，表示html，将插入到<selector>标签所在的位置。templateUrl 是指向html文件的路径，具有相同的行为
管道	该组件使用的管道数组。

第34.1节：使用管道绕过清理（用于代码重用）

项目遵循Angular2快速入门指南中的结构 here。

```
项目根目录
|
+-- 应用程序
|    |-- app.组件.ts
|    |-- main.ts
|    |-- pipeUser.组件.ts
|    \-- sanitize.管道.ts
|
|-- index.html
|-- main.html
|-- pipe.html
```

main.ts

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

bootstrap(AppComponent);
```

这会在项目根目录中找到 index.html 文件，并以此为基础构建。

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 { }
```

这是顶层组件，用于组合其他使用的组件。

pipeUser.component.ts

Chapter 34: Bypassing Sanitizing for trusted values

Params	Details
selector	tag name you reference your component by in the html
template(templateUrl)	a string that represents html which will be inserted wherever the **<selector>** tag is. templateUrl is a path to an html file with the same behavior
pipes	an array of pipes that are used by this component.

Section 34.1: Bypassing Sanitizing with pipes (for code re-use)

Project is following the structure from the Angular2 Quickstart guide here.

```
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);
```

This finds the index.html file in the root of the project, and builds off of that.

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 { }
```

This is the top level component that groups other components that are used.

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";
        }
    }
}
```

该组件提供了管道工作的视图。

> sanitize.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
import { DomSanitizationService } from '@angular/platform-browser';

@Pipe({
name: '卫生管道'
})
导出类 IgnoreSanitize 实现 PipeTransform {

  构造函数(私有 sanitizer: DomSanitizationService){}

  转换(输入: 字符串) : 任意 {
      返回 this.sanitizer.bypassSecurityTrustUrl(输入);
   }

}
```

这是描述管道格式化逻辑的代码。

> index.html

```
<head>
内容放在这里...
</head>
<body>
    <main-app>
main.html 将加载在这里。
    </main-app>
</body>
```

```
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";
        }
    }
}
```

This component provides the view for the Pipe to work with.

> 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);
   }

}
```

This is the logic that describes what the pipe formats.

> 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 将在此处加载。
</pipe-example>

<moretags>
</moretags>
```

pipe.html

```
<img [src]="getUrl('1234') | sanitaryPipe">
<embed [src]="getUrl() | sanitaryPipe">
```

如果你在应用运行时检查 html，你会看到它是这样的：

```
<head>
内容放在这里...
</head>

<body>

    <othertags>
    </othertags>

    <img [src]="getUrl('1234') | sanitaryPipe">
    <embed [src]="getUrl() | sanitaryPipe">

    <moretags>


</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">
```

If you were to inspect the html while the app is running you would see that it looks like this:

```
<head>
    Stuff goes here...
</head>

<body>

    <othertags>
    </othertags>

    <img [src]="getUrl('1234') | sanitaryPipe">
    <embed [src]="getUrl() | sanitaryPipe">

    <moretags>
    </moretags>

</body>
```

第35章：Angular 2 数据驱动表单

第35.1节：数据驱动表单

组件

```
    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 数据驱动表单组件  implements OnInit {
  myForm: 表单组;

constructor(private formBuilder: 表单构建器) {}

  ngOnInit() {
    this.myForm = this.formBuilder.group({
      'loginCredentials': this.formBuilder.group({
        'login': ['', Validators.必填],
        'email': ['',  [Validators.必填, customValidator]],
        'password': ['',  Validators.必填]
      }),
      'hobbies': this.formBuilder.array([
        this.formBuilder.group({
          'hobby': ['', Validators.必填]
        })
      ])
    });
  }

removeHobby(index: 数字){
    (<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);
  }
}
```

Chapter 35: Angular 2 Data Driven Forms

Section 35.1: Data driven form

Component

```
    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])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?")) {
    return {error: true}
  }
}
```

HTML 标记

```
<h3>注册页面</h3>
<form [formGroup]="myForm" (ngSubmit)="onSubmit()">
  <div formGroupName="loginCredentials">
    <div class="form-group">
      <div>
        <label for="login">登录</label>
        <input  id="login" type="text" class="form-control" formControlName="login">
      </div>
      <div>
        <label for="email">邮箱</label>
        <input  id="email" type="text" class="form-control"  formControlName="email">
      </div>
      <div>
        <label for="password">密码</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>爱好数组：</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()">添加爱好</button>
      </div>
    </div>
  </div>
  <button type="submit" [disabled]="!myForm.valid">提交</button>
</form>
```

```
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])?\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?")) {
    return {error: true}
  }
}
```

HTML Markup

```
<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>
```

第36章：Angular 2 内存中的 Web API

第36.1节：设置多个测试 API 路由

mock-data.ts

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

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

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

现在，您可以与 app/mock 和 app/data 交互以提取它们对应的数据。

第 36.2 节：基本设置

mock-data.ts

创建模拟 API 数据

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

    return {mock};
  }
}
```

main.ts

让依赖注入器为 XHRBackend 请求提供 InMemoryBackendService。同时，提供一个包含 createDb() 函数的类（在本例中为 MockData），用于指定 SEED_DATA 请求的模拟 API 路由。

```
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';
```

Chapter 36: Angular 2 In Memory Web API

Section 36.1: Setting Up Multiple Test API Routes

mock-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 };
    }
}
```

Now, you can interact with both app/mock and app/data to extract their corresponding data.

Section 36.2: Basic Setup

mock-data.ts

Create the mock api data

```
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

Have the dependency injector provide the InMemoryBackendService for XHRBackend requests. Also, provide a class that includes a createDb() function (in this case, MockData) specifying the mocked API routes for SEED_DATA requests.

```
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

调用已创建的 API 路由的 GET 请求示例

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

@Injectable()
export class MockService {
  // Web API 的 URL
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}` : '服务器错误';
   console.error(errMsg);
    return Promise.reject(errMsg);
  }
}
```

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

mock.service.ts

Example of calling a get request for the created API route

```
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);
  }
}
```

第37章：使用Angular 2的提前编译（AOT）

第37.1节：为什么我们需要编译，编译的事件流程和示例？

问：为什么我们需要编译？答：我们需要编译以实现Angular应用程序更高效的运行。

请看以下示例，

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

使用上述代码渲染模板，

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

相比之下，速度要慢得多：

```
// ...
this._text_9 = this.renderer.createText(this._el_3, '', null);this._text_10 = this.ren
derer.createText(parentRenderNode, '', null);this._el_11 = this.renderer.createElement(pare
ntRenderNode, 'ul', null);this._text_12 = this.renderer.createText(this._el_11, '  ', 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 im
port17.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);
// …
```

使用预编译（Ahead-of-Time）编译的事件流程

相比之下，使用AoT我们通过以下步骤：

1. 使用TypeScript开发Angular 2应用程序。
2. 使用ngc编译应用程序。

Chapter 37: Ahead-of-time (AOT) compilation with Angular 2

Section 37.1: Why we need compilation, Flow of events compilation and example?

Q. Why we need compilation? Ans. We need compilation for achieving higher level of efficiency of our Angular applications.

Take a look at the following example,

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

Using the code above to render the template,

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

Is much slower compared to:

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

Flow of events with Ahead-of-Time Compilation

In contrast, with AoT we get through the following steps:

1. Development of Angular 2 application with TypeScript.
2. Compilation of the application with ngc.

3. 使用Angular编译器将模板编译为TypeScript。
4. 将TypeScript代码编译为JavaScript。
5. 打包。
6. 压缩。
7.部署。

虽然上述过程看起来稍微复杂一些，但用户只需经历以下步骤：

1. 下载所有资源。
2. Angular 启动。
3. 应用程序被渲染。

如你所见，第三步被省略了，这意味着更快/更好的用户体验，此外，像 angular2-seed 和 angular-cli 这样的工具将极大地自动化构建过程。

希望这能帮到你！谢谢！

第37.2节：使用 Angular CLI 进行 AoT 编译

Angular CLI 命令行界面自 beta 17 版本起支持 AoT 编译。

要使用 AoT 编译构建你的应用，只需运行：

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

第37.3节：安装带编译器的 Angular 2 依赖

注意：为了获得最佳效果，请确保您的项目是使用 Angular-CLI 创建的。

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

如果您的项目已经安装了 Angular 2 及所有这些依赖项，则无需执行此步骤。只需确保其中包含compiler即可。

第37.4节：向您的
`tsconfig.json` 文件添加 `angularCompilerOptions`

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

这是编译器的输出文件夹。

第37.5节：运行 ngc，即 Angular 编译器

从项目根目录运行 ./node_modules/.bin/ngc -p src 其中 src 是您所有 Angular 2 代码所在的位置。这将生成一个名为 ngfactory 的文件夹，所有编译后的代码都将存放在那里。

"node_modules/.bin/ngc" -p src 适用于 Windows

3. Performs compilation of the templates with the Angular compiler to TypeScript.
4. Compilation of the TypeScript code to JavaScript.
5. Bundling.
6. Minification.
7. Deployment.

Although the above process seems lightly more complicated the user goes only through the steps:

1. Download all the assets.
2. Angular bootstraps.
3. The application gets rendered.

As you can see the third step is missing which means faster/better UX and on top of that tools like angular2-seed and angular-cli will automate the build process dramatically.

I hope it might help you! Thank you!

Section 37.2: Using AoT Compilation with Angular CLI

The Angular CLI command-line interface has AoT compilation support since beta 17.

To build your app with AoT compilation, simply run:

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

Section 37.3: Install Angular 2 dependencies with compiler

NOTE: for best results, make sure your project was created using the Angular-CLI.

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

You don't have to do this step if you project already has angular 2 and all of these dependencies installed. Just make sure that the `compiler` is in there.

Section 37.4: Add `angularCompilerOptions` to your `tsconfig.json` file

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

This is the output folder of the compiler.

Section 37.5: Run ngc, the angular compiler

from the root of your project `./node_modules/.bin/ngc -p src` where `src` is where all your angular 2 code lives. This will generate a folder called `ngfactory` where all your compiled code will live.

`"node_modules/.bin/ngc" -p src` for windows

第37.6节：修改 `main.ts` 文件以使用 NgFactory 和静态平台浏览器

```
                    // 这是静态平台浏览器，通常对应的是 @angular/platform-browser-dynamic。
import { platformBrowser } from '@angular/platform-browser';

// 这是由 Angular 编译器生成的
import { AppModuleNgFactory } from './ngfactory/app/app.module.ngfactory';

// 注意这里使用的是 `bootstrapModuleFactory`，而不是 `bootstrapModule`。
platformBrowser().bootstrapModuleFactory(AppModuleNgFactory);
```

此时你应该能够运行你的项目。在本例中，我的项目是使用 Angular-CLI 创建的。

```
> ng serve
```

Section 37.6: Modify `main.ts` file to use NgFactory and static platform browser

```
// 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);
```

At this point you should be able to run your project. In this case, my project was created using the Angular-CLI.

```
> ng serve
```

第38章：使用 Angular 2 和 Restful API 进行 CRUD 操作

第38.1节：在 Angular 2 中从 Restful API 读取数据

为了将 API 逻辑与组件分离，我们将 API 客户端创建为一个独立的类。这个示例类向维基百科 API 发起请求以获取随机的维基文章。

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

And have a component to consume our new API client.

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

Chapter 38: CRUD in Angular 2 with Restful API

Section 38.1: Read from an Restful API in Angular 2

To separate API logic from the component, we are creating the API client as a separate class. This example class makes a request to Wikipedia API to get random wiki articles.

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

And have a component to consume our new API client.

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

第39章：在

Angular 2中使用原生Web组件

第39.1节：在模块中包含自定义元素模式

```
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 { }
```

第39.2节：在模板中使用你的Web组件

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

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

Chapter 39: Use native webcomponents in Angular 2

Section 39.1: Include custom elements schema in your module

```
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 { }
```

Section 39.2: Use your webcomponent in a template

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

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

第40章：更新类型定义

第40.1节：当出现 typings 警告过时时更新类型定义

警告信息：

```
typings WARN deprecated 10/25/2016: "registry:dt/jasmine#2.5.0+20161003201800" 已过时
(已更新,替换或移除)
```

使用以下命令更新引用：

```
npm run typings -- install dt~jasmine --save --global
```

将任何抛出警告的库中的 [jazmine] 替换掉

Chapter 40: Update typings

Section 40.1: Update typings when: typings WARN deprecated

Warning message:

```
typings WARN deprecated 10/25/2016: "registry:dt/jasmine#2.5.0+20161003201800" is deprecated
(updated, replaced or removed)
```

Update the reference with:

```
npm run typings -- install dt~jasmine --save --global
```

Replace [jazmine] for any library that is throwing warning

第41章：模拟 @ngrx/Store

姓名	描述
值	下一个要观察的值
错误	描述
错误	要抛出的错误
超类	描述
action$	模拟观察者，除非在模拟类中定义，否则不执行任何操作
actionReducer$	模拟观察者，除非在模拟类中定义，否则不执行任何操作
obs$	模拟可观察对象

@ngrx/Store 在 Angular 2 项目中变得越来越广泛使用。因此，任何希望使用 Store 的组件或服务都需要将其注入到构造函数中。不过，单元测试 Store 并不像测试一个简单的服务那样容易。像许多问题一样，有多种实现解决方案的方法。然而，基本的做法是为 Observer 接口编写一个模拟类，并为 Store 编写一个模拟类。然后你可以在 TestBed 中将 Store 作为提供者注入。

第 41.1 节：使用模拟 Store 的组件单元测试

这是一个依赖于Store的组件的单元测试。在这里，我们创建了一个名为MockStore的新类，它被注入到我们的组件中，替代了通常的 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('从模拟存储分发！')
  }

public select(obj) {
console.log('从模拟存储中选择！');

    return Observable.of({})
  }
}


describe('AppComponent', () => {
  beforeEach(() => {
TestBed.configureTestingModule({
      declarations: [
AppComponent,
        SmartComponentComponent,
        DumbComponentComponent,
      ],
imports: [
StoreModule.provideStore({mainReducer})
      ],
```

Chapter 41: Mocking @ngrx/Store

name	description
value	next value to be observed
error	description
err	error to be thrown
super	description
action$	mock Observer that does nothing unless defined to do so in the mock class
actionReducer$	mock Observer that does nothing unless defined to do so in the mock class
obs$	mock Observable

@ngrx/Store is becoming more widely used in Angular 2 projects. As such, the Store is required to be injected into the constructor of any Component or Service that wishes to use it. Unit testing Store isn't as easy as testing a simple service though. As with many problems, there are a myriad of ways to implement solutions. However, the basic recipe is to write a mock class for the Observer interface and to write a mock class for Store. Then you can inject Store as a provider in your TestBed.

Section 41.1: Unit Test For Component With Mock Store

This is a unit test of a component that has *Store* as a dependency. Here, we are creating a new class called *MockStore* that is injected into our component instead of the usual 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('应该创建该应用', async(() => {

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

第41.2节：Angular 2 - 模拟Observable（服务 + 组件）

服务

- 我创建了带有postRequest方法的post服务。

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

组件

- 我创建了带有result参数和调用postService的postExample函数的组件。
- 当post请求成功时，result参数应为'Success'，否则为'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) {
```

```
    providers: [
      {provide: Store, useClass: MockStore}
    ]
  });
});

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

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

Section 41.2: Angular 2 - Mock Observable (service + component)

service

- I created post service with postRequest method.

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

Component

- I created component with result parameter and postExample function that call to postService.
- when the post resquest successed than result parameter should be 'Success' else '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 = '成功';
        },
err =>  this.result = '失败'
      )
  }
}
```

测试服务

- 当你想测试使用http的服务时，应该使用mockBackend，并将其注入。
- 你还需要注入postService。

```
describe('测试 PostService', () => {
  beforeEach(() => {
TestBed.configureTestingModule({
      imports: [HttpModule],
提供者: [
        PostService,
        MockBackend,
        BaseRequestOptions,
          {
提供: Http,
依赖: [MockBackend, BaseRequestOptions],
          使用工厂: (backend: XHRBackend, defaultOptions: BaseRequestOptions) => {
            return new Http(backend, defaultOptions);
          }
        }
      ]
    });
  });



它('sendPostRequest 函数返回 Observable', 注入([PostService, MockBackend], (service:
PostService, mockBackend: MockBackend) => {
    让 mockPostModel = PostModel();

mockBackend.connections.订阅((connection: MockConnection) => {
     期望(connection.request.method).等于(RequestMethod.Post);
     期望(connection.request.url.indexOf('postUrl')).不.等于(-1);
     期望(connection.request.headers.获取('Content-Type')).等于('application/json');
    });

服务
.postRequest(PostModel)
      .订阅((响应) => {
        期望(响应).被定义();
      });
  }));
});
```

测试组件

```
describe('测试帖子组件', () => {
  let 组件: PostComponent;
```

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

test service

- when you want to test service that using http you should use mockBackend. and inject it to it.
- you need also to inject 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();
      });
  }));
});
```

test component

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


  let 模拟路由器 = {
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);
          }
        },
        {提供: 路由器, 使用值: 模拟路由器}
      ],
schemas: [ CUSTOM_ELEMENTS_SCHEMA ]
    }).compileComponents();
  }));

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




it('测试 postRequest 成功', 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' : '消息',
      'ok'      : true
    };
mockBackend.connections.subscribe((connection: MockConnection) => {
      postComponent.result = '成功'
connection.mockRespond(new Response(
        new ResponseOptions({
          body: response
        })
      ))
    });
service.postRequest(postModel)
      .subscribe((data) => {
expect(component.result).toBeDefined();
        expect(PostComponent.result).toEqual('成功');
        expect(data).toEqual(response);
      });
  }));
});
```

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

第41.3节：观察者模拟（Observer Mock）

```
class ObserverMock implements Observer<any> {
  closed?: boolean = false; // 继承自Observer
  nextVal: any = ''; // 我自定义的变量

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('组件:Typeahead', () => {
    beforeEach(() => {
TestBed.configureTestingModule({
            imports: [...],
declarations: [Typeahead],
            providers: [
                {provide: Store, useClass: StoreMock} // 注意使用 useClass 而非 useValue
            ]
        }).compileComponents();
    });
});
```

第41.4节：组件对 Store 进行监视的单元测试

这是一个依赖于*Store*的组件的单元测试。在这里，我们能够使用具有默认“初始状态”的 store，同时防止在调用*store.dispatch()*时实际派发动作。

```
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})
      ]
```

Section 41.3: Observer Mock

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

Section 41.4: Unit Test For Component Spying On Store

This is a unit test of a component that has *Store* as a dependency. Here, we are able to use a store with the default "initial state" while preventing it from actually dispatching actions when *store.dispatch()* is called.

```
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('应该创建该应用', 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!');
});


}));


});
```

第41.5节：简单存储

simple.action.ts

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

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

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

simple.efficts.ts

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

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


@Injectable()
export class simpleEffects {

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

            return Observable.of({ type: simpleActionTpye.add_Success, payload: action.payload })
            //   如果你有一个API，可以使用这段代码
            // return this.http.post(url).catch().map(res=>{ type: simpleAction.add_Success,
payload:res})
        });
```

```
    });

  });


  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!');
});


}));


});
```

Section 41.5: Simple Store

simple.action.ts

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

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

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

simple.efficts.ts

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

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


@Injectable()
export class simpleEffects {

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

            return Observable.of({ type: simpleActionTpye.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, simpleActionTpye } from './simple.action';

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

store/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.efficts";


export interface IAppState {
    sum: number;
}

// 所有新的 reducer 应该在这里定义
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
```

```
    constructor(private actions$: Actions) { }
}
```

simple.reducer.ts

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

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

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

store/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.efficts";


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 { 组件（Component） } from '@angular/core';

import { 存储（Store） } from '@ngrx/store';
import { 可观察对象（Observable） } from 'rxjs';

import { 应用状态接口（IAppState） } from './Store/index';
import { simpleActionTpye } from './Store/simple/simple.action';

@组件（Component）({
选择器（selector）: 'app-root',
  模板地址（templateUrl）: './app.component.html',
  样式地址（styleUrls）: ['./app.component.css']
})
导出类 应用组件（AppComponent） {
  标题（title） = 'app';

构造函数(私有的 store: 存储（Store）<应用状态接口（IAppState）>) {
    store.选择（select）(s => s.总和（sum）).订阅（subscribe）((结果（res）) => {
    控制台（console）.日志（log）(结果（res）);
    })
    this.store.派发（dispatch）({
    类型（type）: simpleActionTpye.添加（add）,
    载荷（payload）: 1
    })
    this.store.派发（dispatch）({
    类型（type）: simpleActionTpye.添加（add）,
    载荷（payload）: 2
    })
    this.store.派发（dispatch）({
    类型（type）: simpleActionTpye.添加（add）,
    载荷（payload）: 3
    })
  }
}
```

结果 0 1 3 6

```
    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
    })
  }
}
```

result 0 1 3 6

第42章：ngrx

NgRx 是一个强大的库，可以与 Angular2 一起使用。其背后的理念是将两个配合良好的概念合并，打造一个具有可预测 状态容器 的 响应式应用 ：- [Redux][1] - [RxJs][2] 主要优点：-在应用中组件之间共享数据将更容易 - 测试应用核心逻辑就是测试纯函数，不依赖 Angular2（非常简单！） [1]: http://redux.js.org [2]:

http://reactivex.io/rxjs

第42.1节：完整示例：用户登录/登出

先决条件

本主题 不 涉及 Redux 和/或 Ngrx：

- 你需要熟悉 Redux
- 至少理解 RxJs 和 Observable 模式的基础知识

首先，让我们从头定义一个示例并尝试一些代码：

作为开发者，我想要：

1. 拥有一个 IUser 接口，定义 User 的属性
2. 声明我们稍后将用于操作User的动作，位于Store
3. 定义UserReducer的初始状态
4. 创建UserReducer
5. 将我们的UserReducer导入主模块以构建Store
6. 使用来自Store的数据在视图中显示信息

剧透提醒：如果你想立即尝试演示或在我们开始之前阅读代码，这里有一个Plunkr (embed view 或 run view)。

1) 定义IUser接口

我喜欢将我的接口分成两部分：

- 我们将从服务器获取的属性
- 我们仅为用户界面定义的属性（例如按钮是否应旋转）

这是我们将使用的接口IUser：

user.interface.ts

```
导出接口 IUser {
  // 来自服务器
username: string;
  email: string;

  // 用于用户界面
isConnecting: boolean;
  isConnected: boolean;
};
```

2) 声明操作User的动作

Chapter 42: ngrx

Ngrx is a powerful library that you can use with **Angular2**. The idea behind is to merge two concepts that plays well together to have a **reactive app** with a predictable **state container** : - [Redux][1] - [RxJs][2] The main advantages : - Sharing data in your app between your components is going to easier - Testing your app core logic consists to test pure functions, without any dependency on Angular2 (very easy so !) [1]: http://redux.js.org [2]: http://reactivex.io/rxjs

Section 42.1: Complete example : Login/logout a user

Prerequisites

This topic is **not** about Redux and/or Ngrx :

- You need to be comfortable with Redux
- At least understand the basics of RxJs and Observable pattern

First, let's define an example from the very beginning and play with some code :

As a developer, I want to :

1. Have an `IUser` interface that defines the properties of a `User`
2. Declare the actions that we'll use later to manipulate the `User` in the `Store`
3. Define the initial state of the `UserReducer`
4. Create the reducer `UserReducer`
5. Import our `UserReducer` into our main module to build the `Store`
6. Use data from the `Store` to display information in our view

Spoiler alert : If you want to try the demo right away or read the code before we even get started, here's a Plunkr (embed view or run view).

1) Define `IUser` interface

I like to split my interfaces in two parts :

- The properties we'll get from a server
- The properties we define only for the UI (should a button be spinning for example)

And here's the interface `IUser` we'll be using :

`user.interface.ts`

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

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

2) Declare the actions to manipulate the `User`

现在我们需要考虑我们的reducers应该处理哪种类型的操作。
假设这里：

user.actions.ts

```
export const UserActions = {
  // 当用户点击登录按钮时，在我们发起HTTP请求之前
  // 这将允许我们在请求期间禁用登录按钮
USR_IS_CONNECTING: 'USR_IS_CONNECTING',
  // 这允许我们保存用户的用户名和邮箱
  // 我们假设这些数据是在之前的请求中获取的
USR_IS_CONNECTED: 'USR_IS_CONNECTED',

  // 断开用户连接时采用相同的模式
USR_IS_DISCONNECTING: 'USR_IS_DISCONNECTING',
  USR_IS_DISCONNECTED: 'USR_IS_DISCONNECTED'
};
```

但在我们使用这些操作之前，让我先解释一下为什么我们需要一个服务来为我们分发**一些**这些
操作：

假设我们想要连接一个用户。那么我们将点击登录按钮，接下来会发生以下情况：

- 点击按钮
- 组件捕获事件并调用`userService.login`
- `userService.login`方法dispatch一个事件来更新我们的store属性：`user.isConnecting`
- 发起一个HTTP调用（演示中我们将使用`setTimeout`来模拟**异步行为**）
- 一旦HTTP调用完成，我们将分发另一个操作来通知我们的store用户已登录

user.service.ts

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

  login(username: string) {
    // 首先，分发一个操作表示用户正在尝试连接
    // 这样我们可以锁定按钮直到HTTP请求完成
    this.store$.dispatch({ type: UserActions.USR_IS_CONNECTING });

    // 模拟一些延迟，就像HTTP请求时一样
    // 通过使用超时
setTimeout(() => {
      // 你会作为HTTP响应获取的一些邮箱（或数据）
      let email = `${username}@email.com`;

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

logout() {
    // 首先，分发一个操作表示用户正在尝试连接
    // 这样我们可以锁定按钮直到HTTP请求完成
    this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTING });

    // 模拟一些延迟，就像HTTP请求时一样
    // 通过使用超时
setTimeout(() => {
      this.store$.dispatch({ type: UserActions.USR_IS_DISCONNECTED });
    }, 2000);
```

Now we've got to think about what kind of actions our ***reducers*** are supposed to handle.
Let say here :

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'
};
```

But before we use those actions, let me explain why we're going to need a service to dispatch **some** of those actions for us :

Let say that we want to connect a user. So we'll be clicking on a login button and here's what's going to happen :

- Click on the button
- The component catch the event and call `userService.login`
- `userService.login` method `dispatch` an event to update our store property : `user.isConnecting`
- An HTTP call is fired (we'll use a `setTimeout` in the demo to simulate the **async behaviour**)
- Once the `HTTP` call is finished, we'll dispatch another action to warn our store that a user is logged

user.service.ts

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

  login(username: string) {
    // first, dispatch an action saying that the user's tyring 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 tyring 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) 定义`UserReducer`

user.state.ts 的初始状态

```
export const UserFactory: IUser = () => {
  return {
    // 来自服务器
用户名: null,
   邮箱: null,

    // 用于用户界面
isConnecting: false,
    isConnected: false,
    isDisconnecting: false
  };
};
```

4) 创建 reducer `UserReducer`

一个 reducer 接受两个参数：

- 当前状态
- 一个类型为 `Action` 的 `Action<{type: string, payload: any}>`

提示：reducer 需要在某个时刻被初始化

由于我们在第3部分定义了 reducer 的默认状态，我们可以这样使用它：

user.reducer.ts

```
导出 const UserReducer: ActionReducer<IUser> = (user: IUser, action: Action) => {
  如果 (user === null) {
    返回 userFactory();
  }

  // ...
}
```

希望有更简单的方法来编写它，使用我们的 `factory` 函数返回一个对象，并在 reducer 中使用 (ES6) 默认参数值：

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

然后，我们需要在 reducer 中处理所有的 actions ：***提示***：使用 ES6 `Object.assign` 函数来保持状态的不可变性

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

    case UserActions.USR_IS_CONNECTED:
      return Object.assign({}, user, { isConnecting: false, isConnected: true, username:
```

```
  }
}
```

3) Define the initial state of the `UserReducer`

`user.state.ts`

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

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

4) Create the reducer `UserReducer`

A reducer takes 2 arguments :

- The current state
- An `Action` of type `Action<{type: string, payload: any}>`

Reminder : A reducer needs to be initialized at some point

As we defined the default state of our reducer in part 3), we'll be able to use it like that :

`user.reducer.ts`

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

  // ...
}
```

Hopefully, there's an easier way to write that by using our `factory` function to return an object and within the reducer use an (ES6) default parameters value :

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

Then, we need to handle every actions in our reducer : ***TIP***: Use ES6 `Object.assign` function to keep our state immutable

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

    case UserActions.USR_IS_CONNECTED:
      return Object.assign({}, user, { isConnecting: false, isConnected: true, username:
```

```
action.payload.username });

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

      case UserActions.USR_IS_DISCONNECTED:
        return Object.assign({}, user, { isDisconnecting: false, isConnected: false });

      默认:
        return user;
  }
};
```

5) 将我们的`UserReducer`导入到主模块中以构建`Store`

应用程序。模块。ts

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

    // 通过提供你的 reducers 来声明你的 store
    // (每个 reducer 应该返回一个默认状态)
StoreModule.provideStore({
        user: UserReducer,
        // 当然，你可以在这里放入任意数量的 reducers
        // ...
    }),

    // 其他要导入的模块
    // ...
    ]
});
```

6) 使用Store中的数据在视图中显示信息

逻辑部分现在一切就绪，我们只需在两个组件中显示所需内容：

- `UserComponent`: **[哑组件]** 我们只需通过 `@Input` 属性和 `async` 管道从存储中传递用户对象。这样，组件只会在用户可用时接收用户（且 `user` 的类型将是 `IUser` 而非 `Observable<IUser>`！）
- `LoginComponent` **[智能组件]** 我们将直接将 `Store` 注入此组件，并仅对 `user` 作为 `Observable` 进行操作。

user.component.ts

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

```
action.payload.username });

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

      case UserActions.USR_IS_DISCONNECTED:
        return Object.assign({}, user, { isDisconnecting: false, isConnected: false });

      default:
        return user;
  }
};
```

5) Import our `UserReducer` into our main module to build the `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) Use data from the `Store` to display information in our view

Everything is now ready on logic side and we just have to display what we want in two components :

- `UserComponent`: **[Dumb component]** We'll just pass the user object from the store using `@Input` property and `async` pipe. This way, the component will receive the user only once it's available (and the `user` will be of type `IUser` and not of type `Observable<IUser>` !)
- `LoginComponent` **[Smart component]** We'll directly inject the `Store` into this component and work only on `user` as an `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>属性</th>
        <th>值</th>
      </tr>

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

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

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

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

      <tr>
        <td>是否断开连接中</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"
```

```
    <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="用户名"
        [disabled]="(user | async).isConnecting"
        ngModel
      >

      <button
type="提交"
        [disabled]="(user | async).isConnecting || (user | async).isConnected"
      >登录</button>
    </form>

    <button
      *ngIf="(user | async).isConnected"
      (click)="logout()"
      [disabled]="(user | async).isDisconnecting"
    >登出</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();
  }
}
```

由于 Ngrx 是 Redux 和 RxJs 概念的融合，刚开始理解其细节可能相当困难。但这是一个强大的模式，正如我们在这个例子中看到的，它允许你拥有一个响应式应用，并且可以轻松共享你的数据。别忘了有一个Plunkr可用，你可以分叉它来进行自己的测试！

希望这对你有帮助，尽管这个话题相当长，干杯！

```
        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();
  }
}
```

As Ngrx is a merge of Redux and RxJs concepts, it can be quite hard to understand the ins an outs at the beginning. But this is a powerful pattern that allows you as we've seen in this example to have a *reactive app* and were you can easily share your data. Don't forget that there's a Plunkr available and you can fork it to make your own tests !

I hope it was helpful even tho the topic is quite long, cheers !

第43章：Http 拦截器

第43.1节：使用我们自己的类替代 Angular 的 Http

继承 Http 类后，我们需要告诉 Angular 使用此类来替代 Http 类。

为了实现这一点，在我们的主模块（或者根据需要，仅在某个特定模块）中，我们需要在 providers 部分写入：

```
        export function httpServiceFactory(xhrBackend: XHRBackend, requestOptions: RequestOptions, router:
Router, appConfig: ApplicationConfiguration) {
  return  new HttpServiceLayer(xhrBackend, requestOptions, router, appConfig);
}

import { HttpModule, 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 { }
```

注意：ApplicationConfiguration 只是我用来在应用程序运行期间保存一些值的服务

第43.2节：简单类继承 Angular 的 Http 类

```
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';

/**
* 该类继承自 Angular 的 Http 类，并默认自动添加服务器 URL（如果处于开发模式）和两个请求头：

* 添加的请求头：'Content-Type' 和 'X-AUTH-TOKEN'。
* 'Content-Type' 可以在其他任何服务中设置，如果已设置，则在此类中将不再被覆盖。

*/
export class HttpServiceLayer extends Http {

constructor(backend: ConnectionBackend, defaultOptions: RequestOptions, private _router: Router,
private appConfig: ApplicationConfiguration) {
```

Chapter 43: Http Interceptor

Section 43.1: Using our class instead of Angular's Http

After extending the Http class, we need to tell angular to use this class instead of Http class.

In order to do this, in our main module(or depending on the needs, just a particular module), we need to write in the providers section:

```
export function httpServiceFactory(xhrBackend: XHRBackend, requestOptions: RequestOptions, router:
Router, appConfig: ApplicationConfiguration) {
  return  new HttpServiceLayer(xhrBackend, requestOptions, router, appConfig);
}

import { HttpModule, 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 { }
```

Note: ApplicationConfiguration is just a service I use to hold some values for the duration of the application

Section 43.2: Simple Class Extending angular's Http class

```
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 automaticaly 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));
  }

  /**
* 此方法检查是否添加了任何头部，如果没有则创建头部映射并添加
'Content-Type' 和 'X-AUTH-TOKEN'
* 如果头部映射中已有 'Content-Type'，则不会覆盖
  */
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;
  }

  /**
* 顾名思义，该方法拦截请求并检查是否存在任何错误。
* 如果存在错误，将检查错误类型；如果是通用错误，则
将在此处处理，否则，
* 将在服务层抛出
  */
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);
      }
    });
  }
}
```

第43.3节：简单的HttpClient AuthToken拦截器（Angular 4.3及以上）

```
import { Injectable } from '@angular/core';
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } from '@angular/common/http';
import { UserService } from '../services/user.service';
import { Observable } from 'rxjs/Observable';
```

```
    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 allready 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 sugests 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);
      }
    });
  }
}
```

Section 43.3: Simple HttpClient AuthToken Interceptor (Angular 4.3+)

```
import { Injectable } from '@angular/core';
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest } 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);
  }
}
```

提供拦截器（some-module.module.ts）

```
{provide: HTTP_INTERCEPTORS, useClass: AuthHeaderInterceptor, multi: true},
```

```
@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);
  }
}
```

Providing Interceptor (some-module.module.ts)

```
{provide: HTTP_INTERCEPTORS, useClass: AuthHeaderInterceptor, multi: true},
```

第44章：动画

第44.1节：空状态之间的过渡

```
    @Component({
      ...
        animations: [
触发器('appear', [
            过渡(':enter', [
                样式({
                    //动画开始时应用的样式
                  }),
animate('300ms ease-in', style({
                  //动画结束时应用的样式
                }))
              ])
          ])
      ]
    })
    class AnimComponent {

    }
]
```

第44.2节：在多个状态之间动画切换

此模板中的<div>会在点击按钮时先增长到50px，然后增长到100px，最后缩小回20px。

每个state都有一个在@Component元数据中描述的关联样式。

哪个state处于激活状态的逻辑可以在组件逻辑中管理。在本例中，组件变量size保存字符串值“small”、“medium”或“large”。

该<div>元素通过@Component元数据中指定的trigger响应该值：
[@size]="size"。

```
@Component({
template: '<div [@size]="size">一些文本</div><button (click)="toggleSize()">切换</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;
```

Chapter 44: Animation

Section 44.1: Transition between null states

```
    @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 {

    }
]
```

Section 44.2: Animating between multiple states

The **<div>** in this template grows to `50px` and then `100px` and then shrinks back to `20px` when you click the button.

Each `state` has an associated style described in the `@Component` metadata.

The logic for whichever `state` is active can be managed in the component logic. In this case, the component variable `size` holds the string value "small", "medium" or "large".

The **<div>** element respond to that value through the `trigger` specified in the `@Component` metadata:
`[@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.大小 = '小';
     }
切换大小(){
          切换(this.大小) {
               情况 '小':
                    this.大小 = '中';
                    跳出;
               情况 '中':
                    this.大小 = '大';
                    跳出;
               情况 '大':
                    this.大小 = '小';
          }
     }
}
```

```
    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';
        }
    }
}
```

第45章：Zone.js

第45.1节：获取NgZone引用

NgZone引用可以通过依赖注入（DI）注入。

my.component.ts

```
导入 { 组件, 初始化钩子, NgZone } 来自 '@angular/core';

@组件({...})
export class Mycomponent implements NgOnInit {
  constructor(private _ngZone: NgZone) { }
  ngOnInit() {
    this._ngZone.runOutsideAngular(() => {
      // 在 Angular 外部执行某些操作，这样不会被检测到
    });
  }
}
```

第45.2节：使用 NgZone 在显示数据前执行多个 HTTP 请求

runOutsideAngular 可用于在 Angular 2 外部运行代码，从而避免不必要地触发变更检测。这可以用于例如执行多个 HTTP 请求以获取所有数据，然后再渲染。要在 Angular 2 内部再次执行代码，NgZone 的 run 方法可以被使用。

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) => {
        // 第一个响应返回，可以使用其数据进行后续请求
        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];
            });
          });
        });
      });
    });
  }
}
```

Chapter 45: Zone.js

Section 45.1: Getting reference to NgZone

NgZone reference can be injected via the Dependency Injection (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
    });
  }
}
```

Section 45.2: Using NgZone to do multiple HTTP requests before showing the data

runOutsideAngular can be used to run code outside Angular 2 so that it does not trigger change detection unnecessarily. This can be used to for example run multiple HTTP request to get all the data before rendering it. To execute code again inside Angular 2, run method of NgZone can be used.

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

第46章：Angular 2 动画

Angular 的动画系统让你构建具有与纯 CSS 动画相同本地性能的动画。你还可以将动画逻辑与应用程序代码的其余部分紧密集成，便于控制。

第46.1节：基本动画——由模型属性驱动的元素在两个状态之间的过渡

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> = [];
```

Chapter 46: Angular 2 Animations

Angular's animation system lets you build animations that run with the same kind of native performance found in pure CSS animations. You can also tightly integrate your animation logic with the rest of your application code, for ease of control.

Section 46.1: Basic Animation - Transitions an element between two states driven by a model attribute

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;
  }

}
```

```
  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;
  }

}
```

第47章：创建一个Angular 2+ NPM包

有时我们需要在多个应用之间共享某些组件，而将其发布到npm是实现这一目的的最佳方式之一。

有一些技巧需要了解，才能在不改变结构的情况下（如内联外部样式）将普通组件用作npm包。

你可以在这里看到一个最小示例

第47.1节：最简单的包

这里我们分享一些构建和发布 Angular 2+ npm 包的最简工作流程。

配置文件

我们需要一些配置文件来告诉git、npm、gulp和typescript如何操作。

.gitignore

首先我们创建一个.gitignore文件，以避免版本控制不需要的文件和文件夹。内容如下：

```
npm-debug.log
node_modules
jspm_packages
.idea
构建
```

.npmignore

其次我们创建一个.npmignore文件，以避免发布不需要的文件和文件夹。内容如下：

```
示例
node_modules
src
```

gulpfile.js

我们需要创建一个gulpfile.js来告诉Gulp如何编译我们的应用程序。这部分是必要的，因为我们
需要在发布我们的包之前最小化并内联所有外部模板和样式。内容如下：

```
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') // 也可以使用 *.js 文件
        .pipe(embedTemplates({sourceType:'ts'}))
        .pipe(inlineNg2Styles({ base: '/src' }))
        .pipe(gulp.dest('./dist'));
});
```

index.d.ts

index.d.ts 文件在 TypeScript 导入外部模块时使用。它帮助编辑器实现自动补全
和函数提示。

Chapter 47: Create an Angular 2+ NPM package

Sometimes we need to share some component between some apps and publishing it in npm is one of the best ways of doing this.

There are some tricks that we need to know to be able to use a normal component as npm package without changing the structure as inlining external styles.

You can see a minimal example here

Section 47.1: Simplest package

Here we are sharing some minimal workflow to build and publish an Angular 2+ npm package.

Configuration files

We need some config files to tell `git`, `npm`, `gulp` and `typescript` how to act.

.gitignore

First we create a `.gitignore` file to avoid versioning unwanted files and folders. The content is:

```
npm-debug.log
node_modules
jspm_packages
.idea
build
```

.npmignore

Second we create a `.npmignore` file to avoid publishing unwanted files and folders. The content is:

```
examples
node_modules
src
```

gulpfile.js

We need to create a `gulpfile.js` to tell Gulp how to compile our application. This part is necessary because we need to minimize and inline all the external templates and styles before publishing our package. The content is:

```
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

The `index.d.ts` file is used by typescript when importing an external module. It helps editor with auto-completion and function tips.

```
export * from './lib';
```

index.js

这是包的入口点。当你使用 NPM 安装此包并在应用程序中导入时，你只需传入包名，应用程序就会知道在哪里找到包中导出的任何组件。

```
exports.AngularXMinimalNpmPackageModule = require('./lib').AngularXMinimalNpmPackageModule;
```

我们使用了lib文件夹，因为当我们编译代码时，输出会放在/lib文件夹内。

package.json

此文件用于配置你的 npm 发布，并定义其工作所需的必要包。

```
{
  "name": "angular-x-minimal-npm-package",
  "version": "0.0.18",
  "description": "一个 Angular 2+ 数据表，使用 HTTP 来创建、读取、更新和删除来自外部 API（如 REST）的数据",

  "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"
  }
}
```

dist/tsconfig.json

```
export * from './lib';
```

index.js

This is the package entry point. When you install this package using NPM and import in your application, you just need to pass the package name and your application will learn where to find any EXPORTED component of your package.

```
exports.AngularXMinimalNpmPackageModule = require('./lib').AngularXMinimalNpmPackageModule;
```

We used `lib` folder because when we compile our code, the output is placed inside `/lib` folder.

package.json

This file is used to configure your npm publication and defines the necessary packages to it to work.

```
{
  "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"
  }
}
```

dist/tsconfig.json

创建一个 dist 文件夹并将此文件放入其中。此文件用于告诉 Typescript 如何编译您的应用程序。获取 typescript 文件夹的位置以及放置编译后文件的位置。

```
{
  "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
  }
}
```

创建配置文件后，我们必须创建我们的组件和模块。该组件接收点击并显示一条消息。它的使用方式类似于一个html标签**`<angular-x-minimal-npm-package></angular-x-minimal-npm-package>`**。只需安装此npm包并在你想使用它的模块中加载其模块。

src/angular-x-minimal-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 = "点击我 ...";
onClick() {
        this.message = "Angular 2+ 最简NPM包。带有外部scss和html！";
    }
}
```

src/angular-x-minimal-npm-package.component.html

```
<div>
  <h1 (click)="onClick()">{{message}}</h1>
</div>
```

src/angular-x-data-table.component.css

```
h1{
color: red;
}
```

src/angular-x-minimal-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 ],
```

Create a dist folder and place this file inside. This file is used to tell Typescript how to compile your application. Where to to get the typescript folder and where to put the compiled files.

```
{
  "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
  }
}
```

After create the configuration files, we must create our component and module. This component receives a click and displays a message. It is used like a html tag **`<angular-x-minimal-npm-package></angular-x-minimal-npm-package>`**. Just instal this npm package and load its module in the model you want to use it.

src/angular-x-minimal-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/angular-x-minimal-npm-package.component.html

```
<div>
  <h1 (click)="onClick()">{{message}}</h1>
</div>
```

src/angular-x-data-table.component.css

```
h1{
    color: red;
}
```

src/angular-x-minimal-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 {}
```

之后，您必须编译、构建并发布您的包。

构建和编译

构建时我们使用gulp，编译时使用 tsc。命令设置在package.json文件的 scripts.build选项中。我们设置了gulp js:build && rm -rf lib && tsc -p dist。这是我们的任务链，将为我们完成工作。

要构建和编译，请在包的根目录运行以下命令：

```
npm run build
```

这将触发任务链，最终你会在`/dist`文件夹中得到构建结果，在`/lib`文件夹中得到编译后的包。这就是为什么在`index.js`中我们导出的是`/lib`文件夹中的代码，而不是`/src`中的代码。

发布

现在我们只需发布我们的包，这样就可以通过npm安装。为此，只需运行命令：

```
npm publish
```

就这么简单！！！

```
})
export class AngularXMinimalNpmPackageModule {}
```

After that, you must compile, build and publish your package.

Build and compile

For build we use `gulp` and for compiling we use `tsc`. The command are set in package.json file, at `scripts.build` option. We have this set `gulp js:build && rm -rf lib && tsc -p dist`. This is our chain tasks that will do the job for us.

To build and compile, run the following command at the root of your package:

```
npm run build
```

This will trigger the chain and you will end up with your build in `/dist` folder and the compiled package in your `/lib` folder. This is why in `index.js` we exported the code from `/lib` folder and not from `/src`.

Publish

Now we just need to publish our package so we can install it through npm. For that, just run the command:

```
npm publish
```

That is all!!!

第48章：Angular 2 CanActivate

第48.1节：Angular 2 CanActivate

在路由器中实现：

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

canActivateRoute 文件：

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

Chapter 48: Angular 2 CanActivate

Section 48.1: Angular 2 CanActivate

Implemented in a router:

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

The `canActivateRoute` file:

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

第49章：Angular 2 - Protractor

第49.1节：Angular 2 Protractor - 安装

> 在命令提示符下运行以下命令

- `npm install -g protractor`
- `webdriver-manager update`
- `webdriver-manager start`

> **在主应用根目录创建 protractor.conf.js 文件。

非常重要的是声明 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');
    // 添加 jasmine 规格报告器
jasmine.getEnv().addReporter(new SpecReporter({ displayStacktrace: true }));

    browser.ignoreSynchronization = false;
  },
useAllAngular2AppRoots: true
};

if (process.env.TRAVIS) {
  config.capabilities = {
    browserName: 'firefox'
  };
```

Chapter 49: Angular 2 - Protractor

Section 49.1: Angular 2 Protractor - Installation

> run the follows commands at cmd

- `npm install -g protractor`
- `webdriver-manager update`
- `webdriver-manager start`

> **create protractor.conf.js file in the main app root.

very important to decleare 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;
```

> **在 dev 目录创建基本测试。**

```
describe('基本测试', () => {

  beforeEach(() => {
browser.get('http://google.com');
  });

it('测试基本测试', () => {
    browser.sleep(2000).then(function(){
      browser.getCurrentUrl().then(function(actualUrl){
        expect(actualUrl.indexOf('google') !== -1).toBeTruthy();
      });
    });
  });
});
```

> **在命令行运行**

```
protractor conf.js
```

第49.2节：使用Protractor测试导航栏路由

首先让我们创建一个包含3个选项的基本navbar.html。（首页，列表，创建）

```
<nav class="navbar navbar-default" role="navigation">
<ul class="nav navbar-nav">
  <li>
    <a id="home-navbar" routerLink="/home">首页</a>
  </li>
  <li>
    <a id="list-navbar" routerLink="/create" >列表</a>
  </li>
  <li>
    <a id="create-navbar" routerLink="/create">创建</a>
  </li>
</ul>
```

其次让我们创建navbar.e2e-spec.ts

```
describe('导航栏', () => {

  beforeEach(() => {
browser.get('home'); // 每个测试前导航到主页。
  });

it('测试导航栏', () => {
    browser.sleep(2000).then(function(){
      checkNavbarTexts();
navigateToListPage();
    });
```

```
}

exports.config = config;
```

> **create basic test at dev directory.**

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

> **run in cmd**

```
protractor conf.js
```

Section 49.2: Testing Navbar routing with Protractor

First lets create basic navbar.html with 3 options. (Home, List , Create)

```
<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>
```

second lets create 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('首页');
    });

element(by.id('list-navbar')).getText().then(function(text){ // Promise
      expect(text).toEqual('列表');
    });

element(by.id('create-navbar')).getText().then(function(text){ // Promise
      expect(text).toEqual('创建');
    });
  }

  function navigateToListPage(){
element(by.id('list-home')).click().then(function(){ // 先找到list-home的a标签然后点击

browser.sleep(2000).then(function(){
browser.getCurrentUrl().then(function(actualUrl){ // promise
            expect(actualUrl.indexOf('list') !== -1).toBeTruthy(); // 检查当前 URL 是否包含 list
          });
        });

    });
  }
});
```

```
  });

  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
          });
        });

    });
  }
});
```

第50章：静态 URL 如 /route/subroute 的路由示例

第50.1节：带有子路由树的基本路由示例

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}
    ]
  }
];
```

Chapter 50: Example for routes such as /route/subroute for static urls

Section 50.1: Basic route example with sub routes tree

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}
    ]
  }
];
```

第51章：Angular 2 Input() output()

第51.1节：Input()

父组件：初始化用户列表。

```
@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 ');
  }
}
```

子组件通过 Input() 从父组件获取用户

```
@Component({
selector: 'child-component',
  template: '<div>
<table *ngIf="users !== null">
                <thead>
<th>姓名</th>
                    <th>姓</th>
                    <th>邮箱</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;
  }
```

Chapter 51: Angular 2 Input() output()

Section 51.1: Input()

Parent Component : Initialize users lists.

```
@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');
  }
}
```

Child component get user from parent component with 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;
  }
}
```

第51.2节：输入属性的简单示例

父元素 html

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

父组件 ts

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

子组件 ts：

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

子组件 html：

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

这段代码会将 inputPropValue 从父组件传递到子组件，传递过去的值是我们在父组件中设置的值——在本例中为 false。然后我们可以在子组件中使用该值，例如给某个元素添加一个类。

```
}
```

Section 51.2: Simple example of Input Properties

Parent element html

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

Parent element ts

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

Child component ts:

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

Child component html:

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

This code will send the inputPropValue from the parent component to the child and it will have the value we have set in the parent component when it arrives there - false in our case. We can then use that value in the child component to, for example add a class to an element.

第52章：Angular-cli

这里你将了解到如何使用 angular-cli 启动项目，使用 angular-cli 生成新的组件/服务/管道/模块，添加第三方库如 bootstrap，以及构建 angular 项目。

第52.1节：使用scss/sass作为样式表的新项目

由@angular/cli生成并编译的默认样式文件是css。

如果你想使用scss，请使用以下命令生成项目：

```
ng new project_name --style=scss
```

如果你想使用sass，请使用以下命令生成项目：

```
ng new project_name --style=sass
```

第52.2节：为@angular/cli设置yarn为默认包管理器

Yarn是npm的替代品，npm是@angular/cli的默认包管理器。如果你想使用yarn作为@angular/cli的包管理器，请按照以下步骤操作：

要求

- yarn（npm install --global yarn 或参见 installation page）
- @angular/cli（npm install -g @angular/cli 或 yarn global add @angular/cli）

要将yarn设置为@angular/cli的包管理器：

```
ng set --global packageManager=yarn
```

将 npm 设置回 @angular/cli 包管理器：

```
ng set --global packageManager=npm
```

第 52.3 节：使用 angular-cli 创建空的 Angular 2 应用程序

要求：

- NodeJS ：下载页面
- npm 或 yarn

在新目录文件夹中使用 cmd 运行以下命令：

1. npm install -g @angular/cli 或 yarn global add @angular/cli
2. ng new 项目名称
3. cd 项目名称
4. ng serve

在浏览器中打开 localhost:4200

Chapter 52: Angular-cli

Here you will find how to start with angular-cli , generating new component/service/pipe/module with angular-cli , add 3 party like bootstrap , build angular project.

Section 52.1: New project with scss/sass as stylesheet

The default style files generated and compiled by `@angular/cli` are **css**.

If you want to use **scss** instead, generate your project with:

```
ng new project_name --style=scss
```

If you want to use **sass**, generate your project with:

```
ng new project_name --style=sass
```

Section 52.2: Set yarn as default package manager for @angular/cli

Yarn is an alternative for npm, the default package manager on @angular/cli. If you want to use yarn as package manager for @angular/cli follow this steps:

Requirements

- yarn (`npm install --global yarn` or see the installation page)
- @angular/cli (`npm install -g @angular/cli` or `yarn global add @angular/cli`)

To set yarn as @angular/cli package manager:

```
ng set --global packageManager=yarn
```

To set back npm as @angular/cli package manager:

```
ng set --global packageManager=npm
```

Section 52.3: Create empty Angular 2 application with angular-cli

Requirements:

- NodeJS : Download page
- npm or yarn

Run the following commands with cmd from new directory folder:

1. `npm install -g @angular/cli` or `yarn global add @angular/cli`
2. `ng new PROJECT_NAME`
3. `cd PROJECT_NAME`
4. `ng serve`

Open your browser at localhost:4200

第52.4节：生成组件、指令、管道和服务

只需使用命令行：你可以使用 ng generate（或简写 ng g）命令来生成 Angular 组件：

- 组件：`ng g component my-new-component`
- 指令：`ng g directive my-new-directive`
- 管道：`ng g pipe my-new-pipe`
- 服务：`ng g service my-new-service`
- 类：`ng g class my-new-classt`
- 接口：`ng g interface my-new-interface`
- 枚举：`ng g enum my-new-enum`
- 模块：ng g module my-module

第52.5节：添加第三方库

你可以在 angular-cli.json 中更改应用配置。

例如，如果你想添加 ng2-bootstrap：

1. npm install ng2-bootstrap --save 或 yarn add ng2-bootstrap
2. 在 angular-cli.json 中只需添加 bootstrap 在 node-modules 中的路径。

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

第52.6节：使用angular-cli构建

在angular-cli.json中的outDir键可以定义你的构建目录；

这些是等效的

```
ng build --target=production --environment=prod
ng build --prod --env=prod
ng build --prod
```

以下命令也是等效的

```
ng build --target=development --environment=dev
ng build --dev --e=dev
ng build --dev
ng build
```

构建时，你可以使用 --base-href your-url 选项修改 index.html 中的 base 标签（base tag）。

将 index.html 中的 base 标签 href 设置为 /myUrl/

```
ng build --base-href /myUrl/
ng build --bh /myUrl/
```

Section 52.4: Generating Components, Directives, Pipes and Services

just use your cmd: You can use the ng generate (or just ng g) command to generate Angular components:

- Component: `ng g component my-new-component`
- Directive: `ng g directive my-new-directive`
- Pipe: `ng g pipe my-new-pipe`
- Service: `ng g service my-new-service`
- Class: `ng g class my-new-classt`
- Interface: `ng g interface my-new-interface`
- Enum: `ng g enum my-new-enum`
- Module: `ng g module my-module`

Section 52.5: Adding 3rd party libs

In angular-cli.json you can change the app configuration.

If you want to add ng2-bootstrap for example:

1. `npm install ng2-bootstrap --save` or `yarn add ng2-bootstrap`
2. In angular-cli.json just add the path of the bootstrap at node-modules.

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

Section 52.6: build with angular-cli

In angular-cli.json at outDir key you can define your build directory;

these are equivalent

```
ng build --target=production --environment=prod
ng build --prod --env=prod
ng build --prod
```

and so are these

```
ng build --target=development --environment=dev
ng build --dev --e=dev
ng build --dev
ng build
```

When building you can modify base tag () in your index.html with --base-href your-url option.

Sets base tag href to /myUrl/ in your index.html

```
ng build --base-href /myUrl/
ng build --bh /myUrl/
```

第53章：Angular 2 变更检测及手动触发

第53.1节：基本示例

父组件：

```
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;
  }

}
```

父组件 HTML：

```
<div>
  <child-component [usersDetails]="users"
                  (changeUsersActivation)="changeUsersActivation($event)">
   </child-component>
</div>
```

子组件：

```
import {Component, Input, EventEmitter, Output} from '@angular/core';
import {User} from "./parent.component";

@Component({
```

Chapter 53: Angular 2 Change detection and manual triggering

Section 53.1: Basic example

Parent component :

```
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;
  }

}
```

Parent HTML:

```
<div>
  <child-component [usersDetails]="users"
                  (changeUsersActivation)="changeUsersActivation($event)">
   </child-component>
</div>
```

child component :

```
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 子组件{
  @Input() 用户详情 : Array<User> = null;
  @Output() 用户激活状态变更 = new EventEmitter();

  触发事件(user : User){
    this.用户激活状态变更.emit(user);
  }
}
```

子组件 HTML：

```
<div>
  <div>
    <table>
      <thead>
        <tr>
          <th>姓名</th>
          <th></th>
        </tr>
      </thead>
      <tbody *ngIf="user !== null">
        <tr *ngFor="let user of 用户详情">
          <td>{{user.firstName}}</td>
          <td><button class="btn" (click)="触发事件(user)">{{user.active}}</button></td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

```
  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>
```

第54章：Angular 2 数据绑定

第54.1节：@Input()

父组件：初始化用户列表。

```
@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 ');
  }
}
```

子组件通过 Input() 从父组件获取用户

```
@Component({
selector: 'child-component',
  template: '<div>
<table *ngIf="users !== null">
                <thead>
<th>姓名</th>
                    <th>姓</th>
                    <th>邮箱</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;
  }
```

Chapter 54: Angular 2 Databinding

Section 54.1: @Input()

Parent Component : Initialize users lists.

```
@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'));
  }
}
```

Child component get user from parent component with 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;
  }
}
```

```
}
```

第55章：蛮力升级

如果你想升级项目中的Angular CLI版本，单纯更改项目中的Angular CLI版本号可能会遇到难以修复的错误和漏洞。此外，由于Angular CLI隐藏了构建和打包过程中的许多细节，当出现问题时，你实际上无法做太多处理。

有时，更新项目的Angular CLI版本最简单的方法是直接用你想使用的Angular CLI版本重新搭建一个新项目。

第55.1节：搭建新的Angular CLI项目

```
ng new NewProject
```

或

```
ng init NewProject
```

Chapter 55: Brute Force Upgrading

If you want to upgrade the Angular CLI version of your project you may run into tough-to-fix errors and bugs from simply changing the Angular CLI version number in your project. Also, because the Angular CLI hides a lot of what's going on in the build and bundles process, you can't really do much when things go wrong there.

Sometimes the easiest way to update the Angular CLI version of the project is to just scaffold out a new proejct with the Angular CLI version that you wish to use.

Section 55.1: Scaffolding a New Angular CLI Project

```
ng new NewProject
```

or

```
ng init NewProject
```

第56章：Angular 2在引导前向应用提供外部数据

在这篇文章中，我将演示如何在Angular应用引导之前传递外部数据。这个外部数据可以是配置数据、遗留数据、服务器渲染的数据等。

第56.1节：通过依赖注入

不要直接调用 Angular'的引导代码，而是将引导代码封装到一个函数中并导出该函数。该函数还可以接受参数。

```
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 引导成功"))
      .catch(err => console.error(err));
}
```

然后，在任何服务或组件中我们都可以注入"legacy model"并访问它。

```
import { Injectable } from "@angular/core";
@Injectable()
export class MyService {
constructor(@Inject("legacyModel") private legacyModel) {
      console.log("遗留数据 — ", legacyModel);
   }
 }
```

引入应用程序然后运行它。

```
require(["myAngular2App"], function(app) {
   app.runAngular2App(legacyModel); // 输入到你的应用程序
});
```

Chapter 56: Angular 2 provide external data to App before bootstrap

In this post I will demonstrate how to pass external data to Angular app before the app bootstraps. This external data could be configuration data, legacy data, server rendered etc.

Section 56.1: Via Dependency Injection

Instead of invoking the Angular's bootstrap code directly, wrap the bootstrap code into a function and export the function. This function can also accept parameters.

```
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));
}
```

Then, in any services or components we can inject the “legacy model” and gain access to it.

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

Require the app and then run it.

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

第57章：自定义 ngx-bootstrap 日期选择器 + 输入框

第57.1节：自定义 ngx-bootstrap 日期选择器

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;
```

Chapter 57: custom ngx-bootstrap datepicker + input

Section 57.1: custom 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('点击外部', 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();
```

```
  @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('/','-'));
  }

}
```

```
    this.close();
  }

  onClickedOutside(event: Date): void {
    if (this.showDatepicker) {
      this.close();
    }
  }

  static convertToDate(val : string): Date {
    return new Date(val.replace('/','-'));
  }

}
```

第58章：在Angular 2中使用第三方库如jQuery

在使用 Angular 2.x 构建应用程序时，有时需要使用第三方库，如jQuery、Google Analytics、聊天集成 JavaScript API 等。

第58.1节：使用angular-cli进行配置

NPM

如果使用NPM安装像jQuery这样的外部库

npm install --save jquery

将脚本路径添加到你的angular-cli.json中

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

资源文件夹

你也可以将库文件保存在你的assets/js目录中，并在angular-cli.json中包含相同内容

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

注意

将你的主库jquery及其依赖如jquery-cycle-plugin保存到assets目录中，并将它们都添加到angular-cli.json中，确保依赖的顺序被维护。

第58.2节：在Angular 2.x组件中使用jQuery

要在你的Angular 2.x组件中使用`jquery`，需在顶部声明一个全局变量

如果使用$作为jQuery

声明var $: any;

如果使用jQuery作为jQuery

声明var jQuery: any

这将允许在你的Angular 2.x组件中使用$或jQuery。

Chapter 58: Using third party libraries like jQuery in Angular 2

When building applications using Angular 2.x there are times when it's required to use any third party libraries like jQuery, Google Analytics, Chat Integration JavaScript APIs and etc.

Section 58.1: Configuration using angular-cli

NPM

If external library like `jQuery` is installed using NPM

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

Add script path into your `angular-cli.json`

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

Assets Folder

You can also save the library file in your `assets/js` directory and include the same in `angular-cli.json`

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

Note

Save your main library `jquery` and their dependencies like `jquery-cycle-plugin` into the assets directory and add both of them into `angular-cli.json`, make sure the order is maintained for the dependencies.

Section 58.2: Using jQuery in Angular 2.x components

To use `jquery` in your Angular 2.x components, declare a global variable on the top

If using $ for jQuery

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

If using `jQuery` for jQuery

```
declare var jQuery: any
```

This will allow using $ or `jQuery` into your Angular 2.x component.

第59章：配置ASP.net Core 应用以支持Angular 2和 TypeScript

场景：ASP.NET Core后台 Angular 2前端 Angular 2组件使用Asp.net Core控制器

这种方式可以在Asp.Net Core应用上实现Angular 2。它允许我们从Angular 2组件调用MVC控制器同时支持Angular 2的MVC结果视图。

第59.1节：Asp.Net Core + Angular 2 + 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)
```

Chapter 59: Configuring ASP.net Core application to work with Angular 2 and TypeScript

SCENARIO: ASP.NET Core background Angular 2 Front-End Angular 2 Components using Asp.net Core Controllers

It way can implement Angular 2 over Asp.Net Core app. It let us call MVC Controllers from Angular 2 components too with the MVC result View supporting Angular 2.

Section 59.1: Asp.Net Core + Angular 2 + 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)
    {
        // 添加框架服务。
services.AddDbContext<ApplicationDbContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

        services.AddIdentity<ApplicationUser, IdentityRole>()
            .添加实体框架存储<ApplicationDbContext>()
            .添加默认令牌提供程序();

services.AddMvc();

        // 添加应用服务。
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,
```

```
    {
        // 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 依赖和网页开发包",
 "version": "1.0.0",
 "description": "Angular 2 MVC。Samuel Maícas 模板",
 "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"
    ]
  },
```

```
    "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 Maícas 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
  }
]
```

将 bundleconfig.json 转换为 gulpfile（在解决方案资源管理器中右键点击 bundleconfig.json，选择 Bundler&Minifier > 转换为 Gulp）

Views/Home/Index.cshtml

```
    @{
ViewData["Title"] = "主页";
}
<div>{{ nombre }}</div>
```

对于 wwwroot 文件夹，使用 https://github.com/angular/quickstart 模板。你需要：**index.html main.ts, systemjs-angular-loader.js, systemjs.config.js, tsconfig.json** 以及 **app 文件夹**

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>正在加载 AppComponent ...</my-app>
  </body>
</html>
```

你可以从 templateUrl 调用它到控制器。 wwwroot/app/app.component.ts

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

@Component({
selector: 'my-app',
  templateUrl: '/home/index',
```

```
  {
    "outputFileName": "wwwroot/js/site.min.js",
    "inputFiles": [
      "wwwroot/js/site.js"
    ],
    "minify": {
      "enabled": true,
      "renameLocals": true
    },
    "sourceMap": false
  }
]
```

Convert bundleconfig.json to gulpfile (RightClick bundleconfig.json on solution explorer, Bundler&Minifier > Convert to Gulp

Views/Home/Index.cshtml

```
    @{
    ViewData["Title"] = "Home Page";
}
<div>{{ nombre }}</div>
```

For wwwroot folder use https://github.com/angular/quickstart seed. You need: **index.html main.ts, systemjs-angular-loader.js, systemjs.config.js, tsconfig.json** And the **app folder**

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>
```

You can call as it to Controllers from templateUrl. wwwroot/app/app.component.ts

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

@Component({
  selector: 'my-app',
  templateUrl: '/home/index',
 })
export class AppComponent  { nombre = 'Samuel Maícas'; }
```

第59.2节：[Seed] Asp.Net Core + Angular 2 + Gulp 在 Visual Studio 2017 上

1.下载 seed
2.运行 dotnet restore
3.运行 npm install

永远享受。

https://github.com/SamML/CoreAngular000

第59.3节：MVC <-> Angular 2

如何：从 ASP.NET Core 控制器调用 Angular 2 HTML/JS 组件：

我们调用 HTML 而不是返回 View()

```
return File("~/html/About.html", "text/html");
```

并在 HTML 中加载 Angular 组件。这里我们可以决定是使用相同模块还是不同模块。
视具体情况而定。

wwwroot/html/About.html

```
    <!DOCTYPE html>
<html>
  <head>
    <title>关于页面</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>正在加载 AppComponent ...</aboutpage>
  </body>
</html>
```

(*)此种种子已经需要加载全部资源列表

操作方法：调用 ASP.NET Core 控制器以显示支持 Angular2 的 MVC 视图：

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

```
 })
export class AppComponent  { nombre = 'Samuel Maícas'; }
```

Section 59.2: [Seed] Asp.Net Core + Angular 2 + Gulp on Visual Studio 2017

1. Download seed
2. Run dotnet restore
3. Run npm install

Always. Enjoy.

https://github.com/SamML/CoreAngular000

Section 59.3: MVC <-> Angular 2

How to: CALL ANGULAR 2 HTML/JS COMPONENT FROM ASP.NET Core CONTROLLER:

We call the HTML instead return View()

```
return File("~/html/About.html", "text/html");
```

And load angular component in the html. Here we can decide if we want to work with same or diferent module. Depends on situation.

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>
```

(*)Already this seed needs to load the entire list of resources

How to: CALL ASP.NET Core Controller to show a MVC View with Angular2 support:

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

```
@Component({
selector: 'aboutpage',
  templateUrl: '/home/about',
})
export class AboutComponent  {

}
```

```
@Component({
  selector: 'aboutpage',
  templateUrl: '/home/about',
})
export class AboutComponent  {

}
```

第60章：使用 webpack 的 Angular 2

第60.1节：Angular 2 webpack 设置

webpack.config.js

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

module.exports = {

    // 设置应用模块的入口点
    "entry": {
        "app": helpers.root("app/main.module.ts"),
    },

    // 输出文件到 dist 文件夹
    "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": [

        // 构建完成时通知
        new WebpackNotifierPlugin({title: "构建完成"}),

        // 获取垫片的参考
        new webpack.DllReferencePlugin({
            "context": helpers.root("src/app"),
            "manifest": helpers.root("config/polyfills-manifest.json")
        }),

        // 获取供应商 DLL 的引用
        new webpack.DllReferencePlugin({
            "context": helpers.root("src/app"),
```

Chapter 60: Angular 2 using webpack

Section 60.1: Angular 2 webpack setup

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")
        }),

        // 压缩编译后的 js
        new webpack.optimize.UglifyJsPlugin(),
    ],
}
```

vendor.config.js

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

module.exports = {
    // 指定导入所有供应商的供应商文件
    "entry": {
        // 可选地添加你的 shim
        "polyfills": [helpers.root("src/app/shims.ts")],
        "vendor": [helpers.root("src/app/vendor.ts")],
    },

    // 输出 vendor 到 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": [

        // 为入口创建 DLL
        new webpack.DllPlugin({
            "name": "[name]",
            "context": helpers.root("src/app"),
            "path": helpers.root("config/[name]-manifest.json")
        }),

        // 压缩生成的 js
        new webpack.optimize.UglifyJsPlugin(),
```

```
            "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>加载中...</app>
</body>
</html>
```

package.json

```
{
  "name": "webpack 示例",
  "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",
```

```
    ],
}
```

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": {
  }
}
```

```
    "@angular/platform-browser-dynamic": "2.4.7",
    "@angular/router": "3.4.7",
    "webpack": "^2.2.1",
    "awesome-typescript-loader": "^3.1.2",
  },
  "dependencies": {
  }
}
```

第61章：Angular 材料设计

第61.1节：Md2手风琴和Md2折叠

Md2折叠：折叠是一个指令，允许用户切换该部分的可见性。

示例

> 一个折叠组件的标记如下。

```
<div [collapse]="isCollapsed">
  Lorum Ipsum 内容
</div>
```

Md2手风琴：手风琴允许用户切换多个部分的可见性。

示例

> 一个手风琴组件的标记如下。

```
<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>自定义标题</md2-accordion-header>
   测试内容
  </md2-accordion-tab>
</md2-accordion>
```

第61.2节：Md2选择组件

组件:

```
<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>
```

> 选择允许用户从选项中选择一个选项。

```
<md2-select></md2-select>
<md2-option></md2-option>
<md2-select-header></md2-select-header>
```

Chapter 61: Angular material design

Section 61.1: Md2Accordion and Md2Collapse

Md2Collapse : Collapse is a directive, it's allow the user to toggle visiblity of the section.

Examples

> A collapse would have the following markup.

```
<div [collapse]="isCollapsed">
  Lorum Ipsum Content
</div>
```

Md2Accordion : Accordion it's allow the user to toggle visiblity of the multiple sections.

Examples

> A accordion would have the following markup.

```
<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>
```

Section 61.2: Md2Select

Component:

```
<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>
```

> Select allow the user to select option from options.

```
<md2-select></md2-select>
<md2-option></md2-option>
<md2-select-header></md2-select-header>
```

第61.3节：Md2Toast

Toast 是一个服务，用于在视图中显示通知。

> 创建并显示一个简单的Toast通知。

```
import {Md2Toast} from 'md2/toast/toast';

@Component({
selector: "..."
})

export class ... {

...
constructor(private toast: Md2Toast) { }
toastMe() {
this.toast.show('Toast message...');

--- 或者 ---

this.toast.show('Toast message...', 1000);
}

...

}
```

第61.4节：Md2Datepicker

日期选择器允许用户选择日期和时间。

```
<md2-datepicker [(ngModel)]="date"></md2-datepicker>
```

详情请见 here

第61.5节：Md2Tooltip

Tooltip 是一个指令，它允许用户在鼠标悬停在某个元素上时显示提示文本。

> 一个提示框的标记如下。

```
<span tooltip-direction="left" tooltip="在左侧！">左侧</span>
<button tooltip="some message"
    tooltip-position="below"
    tooltip-delay="1000">悬停我
</button>
```

Section 61.3: Md2Toast

Toast is a service, which show notifications in the view.

> Creates and show a simple toast noticiation.

```
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);
}

...

}
```

Section 61.4: Md2Datepicker

Datepicker allow the user to select date and time.

```
<md2-datepicker [(ngModel)]="date"></md2-datepicker>
```

see for more details here

Section 61.5: Md2Tooltip

Tooltip is a directive, it allows the user to show hint text while the user mouse hover over an element.

> A tooltip would have the following markup.

```
<span tooltip-direction="left" tooltip="On the Left!">Left</span>
<button tooltip="some message"
    tooltip-position="below"
    tooltip-delay="1000">Hover Me
</button>
```

第62章：Angular 2中的拖放区

第62.1节：投放区

Dropzone 的 Angular 2 封装库。

> npm install angular2-dropzone-wrapper --save-dev

为您的应用模块加载该模块

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

const DROPZONE_CONFIG: DropzoneConfigInterface = {
  // 将此更改为您的上传 POST 地址：
server: 'https://example.com/post',
  maxFilesize: 10,
acceptedFiles: 'image/*'
};

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

> 组件使用
>
> 只需将通常传递给Dropzone的元素替换为dropzone组件即可。

```
<dropzone [config]="config" [message]="'点击或拖动图片到此处上传'"
(error)="onUploadError($event)" (success)="onUploadSuccess($event)"></dropzone>
```

创建dropzone组件

```
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);
    }
}
```

Chapter 62: Dropzone in Angular 2

Section 62.1: Dropzone

Angular 2 wrapper library for Dropzone.

> npm install angular2-dropzone-wrapper --save-dev

Load the module for your app-module

```
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)
  ]
})
```

> COMPONENT USAGE
>
> Simply replace the element that would oridinarily be passed to Dropzone with the dropzone component.

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

Create dropzone component

```
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);
    }
}
```

第63章：angular redux

第63.1节：基础

> 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

Chapter 63: angular redux

Section 63.1: Basic

> 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;
}
```

第63.2节：获取当前状态

```
import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
导出类 exampleService {
构造函数(@Inject(AppStore) private store: Redux.Store<AppState>) {}
    getExampleState(){
console.log(this.store.getState().example);
    }
}
```

第63.3节：更改状态

```
import * as Redux from 'redux';
import {Inject, Injectable} from '@angular/core';

@Injectable()
导出类 exampleService {
构造函数(@Inject(AppStore) private store: Redux.Store<AppState>) {}
    setExampleState(){
        this.store.dispatch(updateExample("new value"));
    }
}
```

> actions.ts

```
导出接口 UpdateExapleAction extends Action {
```

```
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;
}
```

Section 63.2: Get current state

```
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);
    }
}
```

Section 63.3: change state

```
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
});
```

第63.4节：添加redux chrome工具

> 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}
    ];
```

> 安装 Redux DevTools Chrome 扩展

```
  example?: string;
}

export const updateExample: ActionCreator<UpdateExapleAction> =
   (newVal) => ({
     type: EXAMPLE_CHANGED,
     example: newVal
});
```

Section 63.4: Add redux chrome tool

> 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}
    ];
```

> install Redux DevTools chrome extention

第64章：创建一个 Angular npm 库

如何将用 TypeScript 编写的 NgModule 发布到 npm 注册表。设置 npm 项目、TypeScript 编译器、rollup 以及持续集成构建。

第64.1节：带有服务类的最小模块

文件结构

```
/
     -src/
awesome.service.ts
           another-awesome.service.ts
           awesome.module.ts
     -index.ts
     -tsconfig.json
     -package.json
     -rollup.config.js
     -.npmignore
```

服务和模块

把你出色的作品放在这里。

src/awesome.service.ts：

```
export class AwesomeService {
    public doSomethingAwesome(): void {
        console.log("我真棒！");
    }
}
```

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 {}
```

使您的模块和服务可在外部访问。

/index.ts:

```
export { AwesomeService } from './src/awesome.service';
export { AnotherAwesomeService } from './src/another-awesome.service';
export { AwesomeModule } from './src/awesome.module';
```

编译

在 compilerOptions.paths 中，您需要指定包中使用的所有外部模块。

/tsconfig.json

Chapter 64: Creating an Angular npm library

How to publish your NgModule, written in TypeScript in npm registry. Setting up npm project, typescript compiler, rollup and continous integration build.

Section 64.1: Minimal module with service class

File structure

```
/
    -src/
        awesome.service.ts
        another-awesome.service.ts
        awesome.module.ts
    -index.ts
    -tsconfig.json
    -package.json
    -rollup.config.js
    -.npmignore
```

Service and module

Place your awesome work here.

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 {}
```

Make your module and service accessible outside.

/index.ts:

```
export { AwesomeService } from './src/awesome.service';
export { AnotherAwesomeService } from './src/another-awesome.service';
export { AwesomeModule } from './src/awesome.module';
```

Compilation

In compilerOptions.paths you need to specify all external modules which you used in your package.

/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
  }
}
```

请再次指定您的外部依赖

/rollup.config.js

```
导出 默认 {
入口: 'dist/index.js',
    目标: 'dist/bundles/awesome.module.umd.js',
    sourceMap: false,
格式: 'umd',
    模块名: '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']
}
```

NPM 设置

现在，让我们为 npm 添加一些指令

/package.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
  }
}
```

Specify your externals again

/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']
}
```

NPM settings

Now, lets place some instructions for 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"
  }
}
```

我们也可以指定 npm 应该忽略哪些文件

/.npmignore

```
node_modules
npm-debug.log
Thumbs.db
.DS_Store
src
!dist/src
plugin
!dist/plugin
*.ngsummary.json
*.iml
rollup.config.js
```

```
{
  "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"
  }
}
```

We can also specify what files, npm should ignore

/.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
```

持续集成

最后你可以设置持续集成构建

.travis.yml

```
language: node_js
node_js:
-节点

部署:
提供者: npm
邮箱: contact@maciejtreder.com
  api_key:
安全: <your api key>
  在:
标签: true
仓库: maciejtreder/awesome-angular-module
```

演示可在此处找到：https://github.com/maciejtreder/awesome-angular-module

```
tsconfig.json
*.ts
!*.d.ts
.idea
```

Continuous integration

Finally you can set up continuous integration build

.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
```

Demo can be found here: https://github.com/maciejtreder/awesome-angular-module

第65章：桶（Barrel）

桶（barrel）是一种将多个 ES2015 模块的导出汇总到单个便捷 ES2015 模块中的方法。桶本身是一个 ES2015 模块文件，重新导出其他 ES2015 模块中选定的导出内容。

第65.1节：使用桶（Barrel）

例如，没有桶时，使用者需要三个导入语句：

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

我们可以通过在同一组件文件夹中创建一个文件来添加桶。在此情况下，该文件夹名为"heroes"，文件名为 index.ts（遵循约定），该文件导出所有这些项目：

```
export * from './hero.model.ts';   // 重新导出其所有导出内容
export * from './hero.service.ts'; // 重新导出其所有导出内容
export { HeroComponent } from './hero.component.ts'; // 重新导出命名的内容
```

现在，使用者可以从桶中导入所需内容。
import { Hero, HeroService } from '../heroes/index';

不过，这仍然可能是一行很长的代码；可以进一步缩短。

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

这就非常简洁了！* as h 导入所有模块并将其别名为 h

Chapter 65: Barrel

A barrel is a way to rollup exports from several ES2015 modules into a single convenience ES2015 module. The barrel itself is an ES2015 module file that re-exports selected exports of other ES2015 modules.

Section 65.1: Using Barrel

For example without a barrel, a consumer would need three import statements:

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

We can add a barrel by creating a file in the same component folder. In this case the folder is called 'heroes' named index.ts (using the conventions) that exports all of these items:

```
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
```

Now a consumer can import what it needs from the barrel.
`import { Hero, HeroService } from '../heroes/index';`

Still, this can become a very long line; which could be reduced further.

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

That's pretty reduced! The `* as h` imports all of the modules and aliases as h

第66章：测试Angular 2应用

第66.1节：使用Gulp、Webpack、Karma和Jasmine设置测试环境

我们首先需要告诉karma使用Webpack来读取我们的测试文件，基于我们为webpack引擎设置的配置。在这里，我使用babel，因为我用ES6编写代码，你可以根据需要更换为其他类型，比如Typescript。或者我使用Pug（前称Jade）模板，但这不是必须的。

不过，策略保持不变。

所以，这是一个webpack配置：

```
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;
```

然后，我们需要一个 karma.config.js 文件来使用该 webpack 配置：

```
const packConfig = require("./webpack.config.js");
module.exports = function (config) {
config.set({
    basePath: '',
    frameworks: ['jasmine'],
    exclude:[],
```

Chapter 66: Testing an Angular 2 App

Section 66.1: Setting up testing with Gulp, Webpack, Karma and Jasmine

The first thing we need is to tell karma to use Webpack to read our tests, under a configuration we set for the webpack engine. Here, I am using babel because I write my code in ES6, you can change that for other flavors, such as Typescript. Or I use Pug (formerly Jade) templates, you don't have to.

Still, the strategy remains the same.

So, this is a webpack config:

```
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;
```

And then, we need a karma.config.js file to use that webpack config:

```
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
});
};
```

到目前为止，我们已经告诉 Karma 使用 webpack，并且告诉它从名为karma.shim.js的文件开始。这个文件将作为 webpack 的起点。webpack 会读取此文件，并使用import和require语句来收集我们所有的依赖并运行测试。

现在，让我们来看一下 karma.shim.js 文件：

```
// ES6 特定内容开始
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/);
```

```
    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
});
};
```

So far, we have told Karma to use webpack, and we have told it to start at a file called **karma.shim.js**. this file will have the job of acting as the starting point for webpack. webpack will read this file and use the **import** and **require** statements to gather all our dependencies and run our tests.

So now, let's look at the karma.shim.js file:

```
// 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);
```

本质上，我们是从 angular core testing 导入 TestBed，并初始化环境，因为它只需为我们所有的测试初始化一次。然后，我们递归遍历 src/app 目录，读取所有以 .spec.js 结尾的文件，并将它们传递给 testContext，以便运行。

我通常尝试将测试放在与类相同的位置。这是个人喜好，这样我更容易导入依赖并与类一起重构测试。但如果你想将测试放在其他地方，比如 src/test 目录下，这里是你的机会。修改 karma.shim.js 文件中倒数第二行。

完美。还剩什么？啊，就是使用我们上面制作的 karma.config.js 文件的 gulp 任务：

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

我现在使用我们创建的配置文件启动服务器，告诉它只运行一次且不监视更改。我发现这样更适合我，因为测试只会在我准备好运行时才执行，当然如果你想要不同的设置，你知道在哪里修改。

作为我的最后一个代码示例，这里是一组针对 Angular 2 教程“英雄之旅”的测试。

```
import {
TestBed,
    ComponentFixture,
   async
} 来自"@angular/core/testing" ;

import {AppComponent}来自"./app.component" ;
import {AppModule}来自"./app.module" ;
import Hero 来自 "./hero/hero" ;

describe("App 组件", function () {

   beforeEach(()=> {
TestBed.configureTestingModule({
            imports: [AppModule]
        });

        this.fixture = TestBed.createComponent(AppComponent);
        this.fixture.detectChanges();
    });

it("应该有一个标题", async(()=> {
        this.fixture.whenStable().then(()=> {
expect(this.fixture.componentInstance.title).toEqual("英雄之旅");
        });
    }));

it("应该有一个英雄", async(()=> {
        this.fixture.whenStable().then(()=> {
expect(this.fixture.componentInstance.selectedHero).toBeNull();
```

```
testContext.keys().forEach(testContext);
```

In essence, we are importing **TestBed** from angular core testing, and initiating the environment, as it needs to be initiated only once for all of our tests. Then, we are going through the **src/app** directory recursively and reading every file that ends with **.spec.js** and feed them to testContext, so they will run.

I usually try to put my tests the same place as the class. Personat taste, it makes it easier for me to import dependencies and refactor tests with classes. But if you want to put your tests somewhere else, like under **src/test** directory for example, here is you chance. change the line before last in the karma.shim.js file.

Perfect. what is left? ah, the gulp task that uses the karma.config.js file we made above:

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

I am now starting the server with the config file we created, telling it to run once and don't watch for changes. I find this to suite me better as the tests will run only if I am ready for them to run, but of course if you want different you know where to change.

And as my final code sample, here is a set of tests for the Angular 2 tutorial, "Tour of Heroes".

```
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("应该有一个英雄数组", async(()=>
        this.fixture.whenStable().then(()=> {
            const cmp = this.fixture.componentInstance;
expect(cmp.heroes).toBeDefined("组件应该有一个英雄列表");
            expect(cmp.heroes.length).toEqual(10, "英雄列表应该有10个成员");
          cmp.heroes.map((h, i)=> {
expect(h instanceof Hero).toBeTruthy(`成员 ${i} 不是 Hero 实例。 ${h}`)
            });
        })));

它("每个英雄应该有一个列表项", 异步(()=>
        这个.fixture.whenStable().then(()=> {
            const ul = 这个.fixture.nativeElement.querySelector("ul.heroes");
            const li = Array.prototype.slice.call(
                这个.fixture.nativeElement.querySelectorAll("ul.heroes>li"));
            const cmp = this.fixture.componentInstance;
期望(ul).toBeTruthy("应该有一个无序列表用于英雄");
          期望(li.length).toEqual(cmp.heroes.length, "每个英雄应该有一个li");
          li.forEach((li, i)=> {
期望(li.querySelector("span.badge"))
                    .toBeTruthy(`英雄 ${i} 必须有一个用于id的span`);
             期望(li.querySelector("span.badge").textContent.trim())
                    .toEqual(cmp.heroes[i].id.toString(), `英雄 ${i} 显示了错误的id`);
             期望(li.textContent)
.toMatch(cmp.heroes[i].name, `英雄 ${i} 显示了错误的名字`);
            });
        })));

它("应该有正确的英雄项目样式", 异步(()=>
        this.fixture.whenStable().then(()=> {
            const hero = 这个.fixture.nativeElement.querySelector("ul.heroes>li");
            const win = hero.ownerDocument.defaultView ||hero.ownerDocument.parentWindow;
            const styles = win.getComputedStyle(hero);
期望(styles["cursor"]).toEqual("pointer", "英雄的光标应该是指针");
          期望(styles["borderRadius"]).toEqual("4px", "borderRadius 应该是4px");
        })));

它("应该为英雄项目设置点击处理程序",异步(()=>
        this.fixture.whenStable().then(()=>{
            const cmp = this.fixture.componentInstance;
expect(cmp.onSelect)
.toBeDefined("应该为英雄设置点击处理程序");
          expect(this.fixture.nativeElement.querySelector("input.heroName"))
             .toBeNull("未选择英雄时不应显示英雄详情");
          expect(this.fixture.nativeElement.querySelector("ul.heroes li.selected"))
.toBeNull("开始时不应有任何选中的英雄");

          spyOn(cmp,"onSelect").and.callThrough();
            this.fixture.nativeElement.querySelectorAll("ul.heroes li")[5].click();

expect(cmp.onSelect)
                .toHaveBeenCalledWith(cmp.heroes[5]);
            expect(cmp.selectedHero)
.toEqual(cmp.heroes[5], "点击英雄应更改英雄");
        })
    ));
});
```

值得注意的是，我们如何在beforeEach()中配置测试模块并创建测试组件，且

```
        });
    }));

    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.querySelectorAll("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");
        })
    ));
});
```

Noteworthy in this is how we have **beforeEach()** configure a test module and create the component in test, and

我们如何调用 detectChanges()，以便 Angular 实际上能够完成双向绑定等操作。

请注意，每个测试都是对async()的调用，并且它总是等待whenStable的Promise解析后才检查测试夹具。然后，它可以通过componentInstance访问组件，通过ativeElement访问元素。

有一个测试是检查正确的样式。作为教程的一部分，Angular团队演示了组件内部样式的使用。在我们的测试中，我们使用getComputedStyle()来检查样式是否来自我们指定的位置，然而我们需要Window对象来实现这一点，正如你在测试中看到的，我们是从元素中获取它的。

第66.2节：安装Jasmine测试框架

测试Angular 2应用程序最常用的方法是使用Jasmine测试框架。Jasmine允许你在浏览器中测试代码。

安装

开始时，你只需要jasmine-core包（不是jasmine）。

```
npm install jasmine-core --save-dev --save-exact
```

验证

为了验证Jasmine是否正确设置，创建文件./src/unit-tests.html，内容如下，然后在浏览器中打开它。

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="content-type" content="text/html;charset=utf-8">
  <title>Ng App 单元测试</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>
   <!-- 单元测试第1章：生命证明。 -->
   <script>
it('true 是 true', function () {
       expect(true).toEqual(true);
     });
   </script>
</body>
</html>
```

第66.3节：测试Http服务

通常，服务会调用远程API来获取/发送数据。但单元测试不应进行网络调用。Angular内部使用XHRBackend类来执行http请求。用户可以重写此类以更改行为。Angular测试模块提供了MockBackend和MockConnection类，可用于测试和断言http请求。

posts.service.ts 该服务调用一个api端点以获取帖子列表。

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

how we call **detectChanges()** so that angular actually goes through the double-binding and all.

Notice that each test is a call to **async()** and it always waits for **whenStable** promise to resolve before examining the fixture. It then has access to the component through **componentInstance** and to the element through **nativeElement**.

There is one test which is checking the correct styling. as part of the Tutorial, Angular team demonstrates use of styles inside components. In our test, we use **getComputedStyle()** to check that styles are coming from where we specified, however we need the Window object for that, and we are getting it from the element as you can see in the test.

Section 66.2: Installing the Jasmine testing framework

The most common way to test Angular 2 apps is with the Jasmine test framework. Jasmine allows you to test your code in the browser.

Install

To get started, all you need is the `jasmine-core` package (not `jasmine`).

```
npm install jasmine-core --save-dev --save-exact
```

Verify

To verify that Jasmine is set up properly, create the file `./src/unit-tests.html` with the following content and open it in the browser.

```
<!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>
```

Section 66.3: Testing Http Service

Usually, services call remote Api to retrieve/send data. But unit tests shouldn't do network calls. Angular internally uses `XHRBackend` class to do http requests. User can override this to change behavior. Angular testing module provides `MockBackend` and `MockConnection` classes which can be used to test and assert http requests.

`posts.service.ts` This service hits an api endpoint to fetch list of posts.

```
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;
}

@Injectable()
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 在这里，我们将通过模拟http API调用来测试上述服务。

```
import { TestBed, inject, fakeAsync } from '@angular/core/testing';
import {
HttpModule,
    XHRBackend,
    ResponseOptions,
    Response,
RequestMethod
} from '@angular/http';
import {
MockBackend,
    MockConnection
}来自'@angular/http/testing';

导入{PostsService}来自'./posts.service';

描述('PostsService', () => {
    // 模拟http响应
    const mockResponse = [
        {
            'userId': 1,
            'id': 1,
            'title': '自动执行排斥提供者异常选择的防护','body': '因为和支持支持拒绝后果加速和

累积责备的痛苦直到几乎全部我们的事情是但是是重现建筑师'},

        {
            'userId': 1,
            'id': 2,
            'title': '这是什么',
            'body': '这是生命的真实时刻紧随其后的是无所畏惧的痛苦和美好痛苦也不存在逃避诱人的快乐或无视应受谴
责的事物谁会揭示不应承担的债务谁既不属于也不拥有任何东西'

        },
```

```
import { Observable }     from 'rxjs/rx';

import 'rxjs/add/operator/map';

export interface IPost {
    userId: number;
    id: number;
    title: string;
    body: string;
}

@Injectable()
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 Here, we will test above service by mocking http api calls.

```
import { TestBed, inject, fakeAsync } from '@angular/core/testing';
import {
    HttpModule,
    XHRBackend,
    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': '几乎是自我执行的烦恼排斥','body': '以及公正地遵守法律选择所有引起痛苦的权利痛苦
            或指控的责任者烦恼来自其憎恨、劳动和欲望'},


        {
            'userId': 1,
            'id': 4,
            'title': '他和它是痛苦的',
            'body': '经常拒绝获得痛苦但必须承担提供事物的过错谁不知道便利，保留痛苦的权利和法律谁是事物的痛苦
欲望'

        }
    ];

beforeEach(() => {
        TestBed.configureTestingModule({
            imports: [HttpModule],
providers: [
                {
provide: XHRBackend,
                    // 这提供了模拟的XHR后端
useClass: MockBackend
                },
PostsService
            ]
        });
    });

它 ("应该返回从Api检索的帖子", fakeAsync (
        inject ([XHRBackend, PostsService],
            (mockBackend, postsService) => {
                mockBackend.connections.subscribe (
                    (connection: MockConnection) => {
                        // 断言服务已使用预期的方法请求了正确的URL
expect (connection.request.method) .toBe (RequestMethod.Get) ;

expect (connection.request.url) .toBe ('http://jsonplaceholder.typicode.com/posts') ;
                        // 发送模拟响应
connection.mockRespond (new Response (new ResponseOptions ({
                            body: mockResponse
                        })));
                    });

postsService.get ()
                    .subscribe ((posts) => {
                        expect (posts) .toBe (mockResponse) ;
                    });

            })));
});
```

第66.4节：测试Angular组件 - 基础

组件代码如下所示。

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

@Component({
```

```
        {
            '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);
                    });

            })));
});
```

Section 66.4: Testing Angular Components - Basic

The component code is given as below.

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

@Component({
```

```
selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
export class MyAppComponent{
  title = '欢迎';
}
```

对于 Angular 测试，Angular 提供了其测试工具以及测试框架，帮助编写良好的 Angular 测试用例。Angular 工具可以从 @angular/core/testing 导入

```
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { MyAppComponent } from './banner-inline.component';

describe('MyAppComponent 的测试', () => {

  let fixture: ComponentFixture<MyAppComponent>;
  let comp: MyAppComponent;

beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [
MyAppComponent
      ]
    });
  });

beforeEach(() => {

fixture = TestBed.createComponent(MyAppComponent);
    comp = fixture.componentInstance;

  });

it('应该创建 MyAppComponent', () => {

      expect(comp).toBeTruthy();

  });

});
```

在上述示例中，只有一个测试用例，用于说明组件存在性的测试用例。在上述示例中，使用了 Angular 测试工具如 `TestBed` 和 `ComponentFixture`。

`TestBed` 用于创建 Angular 测试模块，我们通过 `configureTestingModule` 方法配置该模块，以为我们想测试的类生成模块环境。测试模块需要在每个测试用例执行前配置，因此我们在 `beforeEach` 函数中配置测试模块。

TestBed 的 createComponent 方法用于创建被测试组件的实例。 `createComponent` 返回 `ComponentFixture`。fixture 提供对组件实例本身的访问。

```
  selector: 'my-app',
  template: '<h1>{{title}}</h1>'
})
export class MyAppComponent{
  title = 'welcome';
}
```

For angular testing, angular provide its testing utilities along with the testing framework which helps in writing the good test case in angular. Angular utilities can be imported from `@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();

  });

});
```

In the above example, there is only one test case which explain the test case for component existence. In the above example angular testing utilities like `TestBed` and `ComponentFixture` are used.

`TestBed` is used to create the angular testing module and we configure this module with the `configureTestingModule` method to produce the module environment for the class we want to test. Testing module to be configured before the execution of every test case that's why we configure the testing module in the `beforeEach` function.

`createComponent` method of `TestBed` is used to create the instance of the component under test. `createComponent` return the `ComponentFixture`. The fixture provides access to the component instance itself.

第67章：angular-cli 测试覆盖率

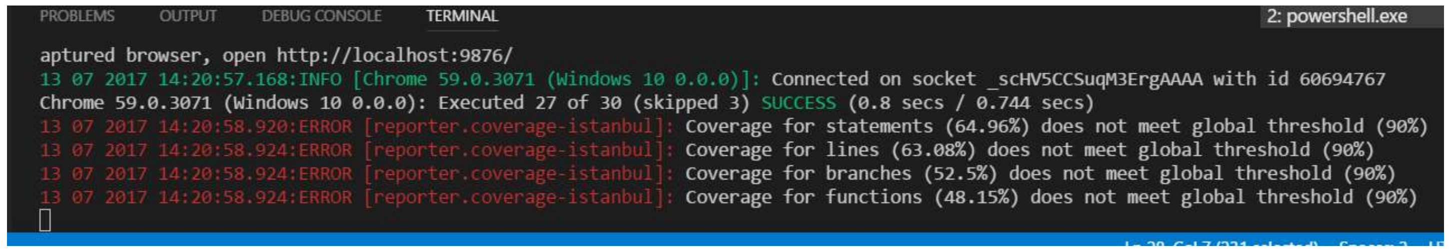
测试覆盖率被定义为一种技术，用于确定我们的测试用例是否真正覆盖了
应用程序代码，以及运行这些测试用例时执行了多少代码。

Angular CLI 内置了代码覆盖功能，只需一个简单命令 ng test --cc

第67.1节：基于简单 angular-cli 命令的测试覆盖率

如果你想查看整体测试覆盖率统计数据，当然在 Angular CLI 中你只需输入以下命令，
并在命令提示符窗口底部查看结果。

```
ng test --cc // 或 --code-coverage
```



```
aptured browser, open http://localhost:9876/
13 07 2017 14:20:57.168:INFO [Chrome 59.0.3071 (Windows 10 0.0.0)]: Connected on socket _scHV5CCSuqM3ErgAAAA with id 60694767
Chrome 59.0.3071 (Windows 10 0.0.0): Executed 27 of 30 (skipped 3) SUCCESS (0.8 secs / 0.744 secs)
13 07 2017 14:20:58.920:ERROR [reporter.coverage-istanbul]: Coverage for statements (64.96%) does not meet global threshold (90%)
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for lines (63.08%) does not meet global threshold (90%)
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for branches (52.5%) does not meet global threshold (90%)
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for functions (48.15%) does not meet global threshold (90%)
```

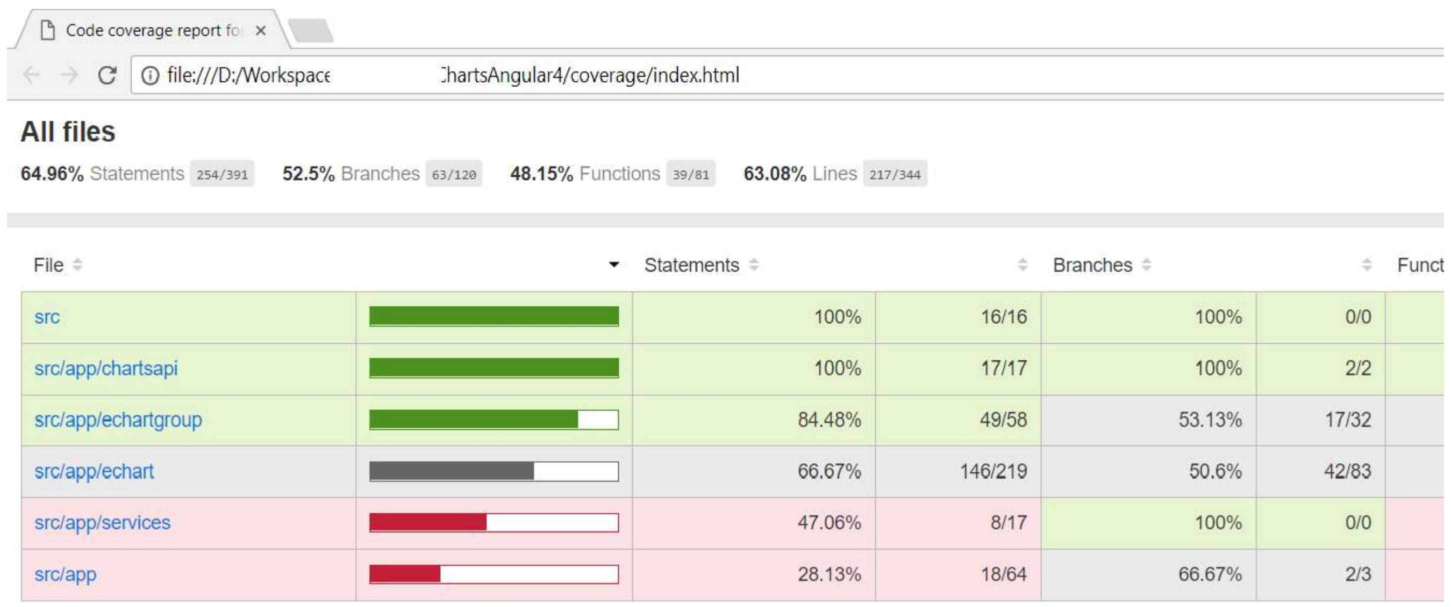
第67.2节：详细的单个组件基础图形测试覆盖率报告

如果您想查看组件的单独测试覆盖率，请按照以下步骤操作。

1. npm install --save-dev karma-teamcity-reporter
2. 在 karma.conf.js 的插件列表中添加 `require('karma-teamcity-reporter')`
3. ng test --code-coverage --reporters=teamcity,coverage-istanbul

请注意，报告器列表是用逗号分隔的，因为我们添加了一个新的报告器 teamcity。

运行此命令后，您可以在目录中看到 coverage 文件夹，并打开 index.html 以图形方式查看测试覆盖率。



All files

64.96% Statements 254/391 52.5% Branches 63/120 48.15% Functions 39/81 63.08% Lines 217/344

File		Statements		Branches		Funct
src		100%	16/16	100%	0/0	
src/app/chartsapi		100%	17/17	100%	2/2	
src/app/echartgroup		84.48%	49/58	53.13%	17/32	
src/app/echart		66.67%	146/219	50.6%	42/83	
src/app/services		47.06%	8/17	100%	0/0	
src/app		28.13%	18/64	66.67%	2/3	

Chapter 67: angular-cli test coverage

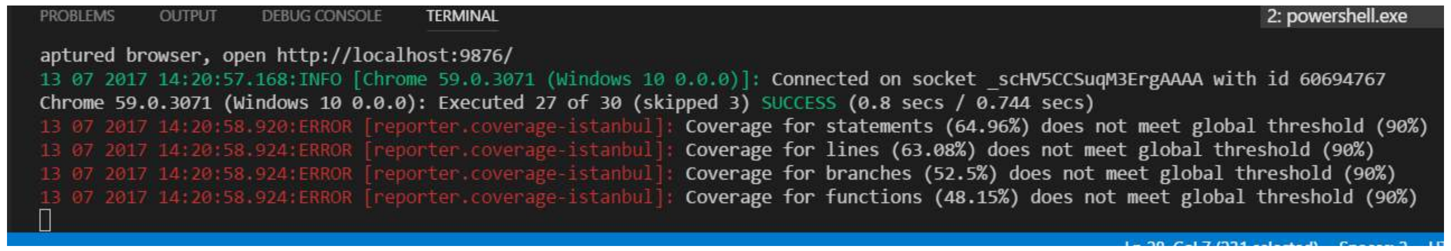
test coverage is defined as a technique which determines whether our test cases are actually covering the application code and how much code is exercised when we run those test cases.

Angular CLI has built in code coverage feature with just a simple command `ng test --cc`

Section 67.1: A simple angular-cli command base test coverage

If you want to see overall test coverage statistics than of course in Angular CLI you can just type below command, and see the bottom of your command prompt window for results.

```
ng test --cc // or --code-coverage
```


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 2: powershell.exe

aptured browser, open http://localhost:9876/
13 07 2017 14:20:57.168:INFO [Chrome 59.0.3071 (Windows 10 0.0.0)]: Connected on socket _scHV5CCSuqM3ErgAAAA with id 60694767
Chrome 59.0.3071 (Windows 10 0.0.0): Executed 27 of 30 (skipped 3) SUCCESS (0.8 secs / 0.744 secs)
13 07 2017 14:20:58.920:ERROR [reporter.coverage-istanbul]: Coverage for statements (64.96%) does not meet global threshold (90%)
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for lines (63.08%) does not meet global threshold (90%)
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for branches (52.5%) does not meet global threshold (90%)
13 07 2017 14:20:58.924:ERROR [reporter.coverage-istanbul]: Coverage for functions (48.15%) does not meet global threshold (90%)

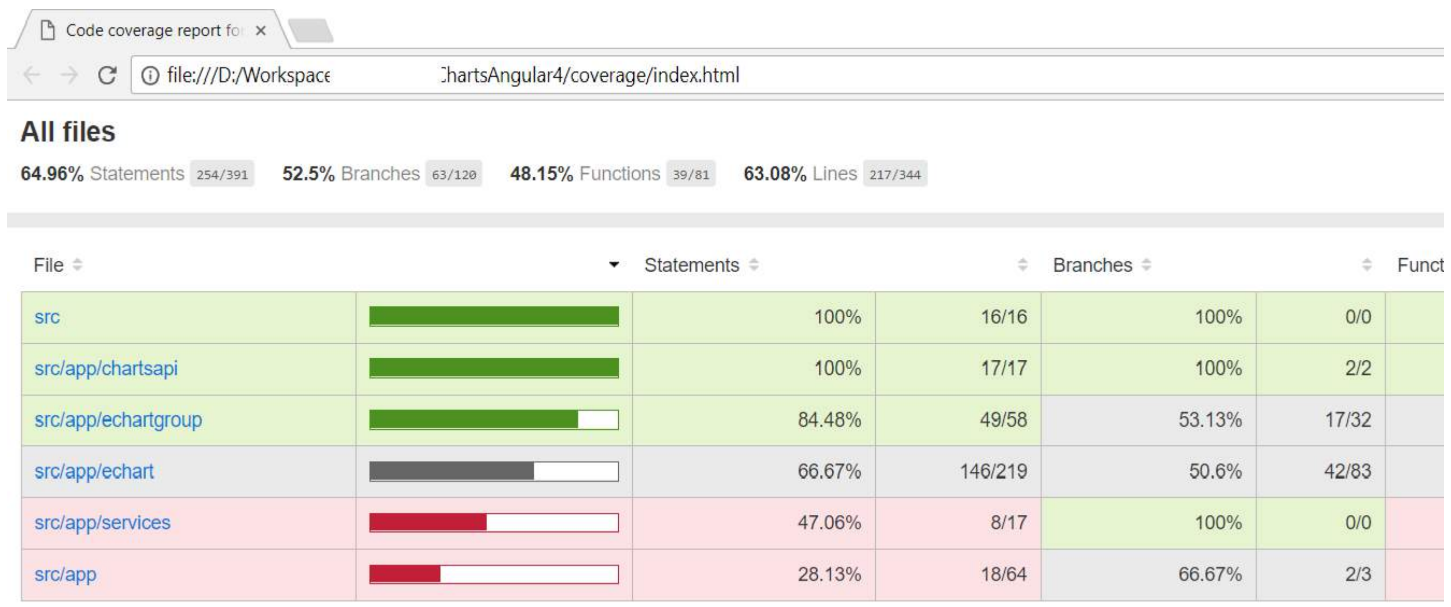
Section 67.2: Detailed individual component base graphical test coverage reporting

if you want to see component's individual coverage of tests follow these steps.

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`

note that list of reporters is comma-separated, as we have added a new reporter, teamcity.

after running this command you can see the folder `coverage` in your dir and open `index.html` for a graphical view of test coverage.


Code coverage report fo
file:///D:/Workspace hartsAngular4/coverage/index.html

All files

64.96% Statements 254/391 52.5% Branches 63/120 48.15% Functions 39/81 63.08% Lines 217/344

File | Statements | Branches | Funct
src | 100% 16/16 | 100% 0/0
src/app/chartsapi | 100% 17/17 | 100% 2/2
src/app/echartgroup | 84.48% 49/58 | 53.13% 17/32
src/app/echart | 66.67% 146/219 | 50.6% 42/83
src/app/services | 47.06% 8/17 | 100% 0/0
src/app | 28.13% 18/64 | 66.67% 2/3

您还可以在 karma.conf.js 中设置您想达到的覆盖率阈值，示例如下。

```
coverageIstanbulReporter: {
      reports: ['html', 'lcovonly'],
      fixWebpackSourcePaths: true,
      thresholds: {
语句: 90,
       行数: 90,
       分支: 90,
       函数: 90
      }
    },
```

You can also set the coverage threshold that you want to achieve, in `karma.conf.js`, like this.

```
coverageIstanbulReporter: {
      reports: ['html', 'lcovonly'],
      fixWebpackSourcePaths: true,
      thresholds: {
        statements: 90,
        lines: 90,
        branches: 90,
        functions: 90
      }
    },
```

第68章：使用Visual Studio Code调试Angular 2 TypeScript应用程序

第68.1节：为你的工作区设置launch.json

1. 从菜单开启调试 - 视图 > 调试
2. 启动调试时出现错误，弹出通知并从该通知中打开launch.json
   这是因为你的工作区未设置launch.json。将以下代码复制粘贴到
   launch.json中 // 新的launch.json

   你之前的launch.json

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "启动扩展",
            "type": "extensionHost",
            "request": "launch",
            "runtimeExecutable": "${execPath}",
            "args": [
                "--extensionDevelopmentPath=${workspaceRoot}"
            ],
            "stopOnEntry": false,
            "sourceMaps": true,
            "outDir": "${workspaceRoot}/out",
            "preLaunchTask": "npm"
        }
    ]
}
```

现在按如下更新你的 launch.json
新的 launch.json
// 请记得在其中注明你的 main.js 路径

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Launch",
            "type": "node",
            "request": "launch",
            "program": "${workspaceRoot}/app/main.js", // 放置你的 main.js 路径
            "stopOnEntry": false,
            "args": [],
            "cwd": "${workspaceRoot}",
            "preLaunchTask": null,
            "runtimeExecutable": null,
            "runtimeArgs": [
                "--nolazy"
            ],
            "env": {
                "NODE_ENV": "development"
            },
            "console": "internalConsole",
```

Chapter 68: Debugging Angular 2 TypeScript application using Visual Studio Code

Section 68.1: Launch.json setup for you workspace

1. Turn on Debug from menu - view > debug
2. it return some error during start debug, show pop out notification and open launch.json from this popup notification It is just because of launch.json not set for your workspace. copy and paste below code in to launch.json //new launch.json

   your old 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"
        }
    ]
}
```

Now update your launch.json as below
new launch.json
// remember please mention your main.js path into it

```
{
    "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. 现在调试功能正常，显示逐步调试的通知弹窗

```
            "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. Now it debug is working, show notification popup for step by step debugging

第69章：单元测试

第69.1节：基本单元测试

组件文件

```
@Component({
selector: 'example-test-compnent',
  template: '<div>
<div>{{user.name}}</div>
                  <div>{{user.fname}}</div>
                  <div>{{user.email}}</div>
              </div>'
})

导出类 ExampleTestComponent 实现 OnInit{

    let user :User = null;
    ngOnInit(): void {
       this.user.name = 'name';
       this.user.fname= 'fname';
       this.user.email= 'email';
    }

}
```

测试文件

```
describe('示例单元测试组件', () => {
  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 应该改变 user 对象的值', () => {
    expect(component.user).toBeNull(); // 检查初始化时 user 是否为 null
    component.ngOnInit(); // 运行 ngOnInit

expect(component.user.name).toEqual('name');
    expect(component.user.fname).toEqual('fname');
    expect(component.user.email).toEqual('email');
  });
});
```

Chapter 69: unit testing

Section 69.1: Basic unit test

component file

```
@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';
    }

}
```

Test file

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

鸣谢

非常感谢所有来自 Stack Overflow Documentation 的人员帮助提供此内容，
更多更改可发送至 web@petercv.com 以发布或更新新内容

Abrar Jahin	第25章
acdcjunior	第1章、第8章和第14章
ahmadalibaloch	第67章
aholtry	第12章和第17章
Ajey	第56章
亚历克斯·莫拉莱斯	第20章
亚历山大·容格斯	第21章
amansoni211	第19章
阿米特·库马尔	第10章和第27章
安德烈·日特凯维奇	第4章
阿尼尔·辛格	第12章、第27章和第37章
Apmis	第17章
阿诺德·维尔斯马	第16章
阿伦·雷杜	第66章
阿里安J	第17、30和31章
阿肖克·维什瓦卡尔马	第58章
Bean0341	第1章
Berseker59	第12章
布米·巴拉尼	第1章
博格丹C	第1章和第52章
borislemke	第4章、第14章、第17章和第33章
布赖恩RT	第41章
brians69	第1章
briantyler	第1章
BrunoLM	第2、4、13、14和23章
cDecker32	第1章
克里斯托弗·泰勒	第14和27章
Chybie	第14章
dafyddPrys	第8章
daniellmb	第21章和第22章
达雷兹克	第20章
echonax	第1章
埃利奥特	第14章
埃里克·希门尼斯	第26章、第28章、第31章和第37章
filoxo	第20章
弗雷德里克·伦丁	第14章
冈特·策鲍尔	第19章
高拉夫·穆克吉	第44章
杰拉德·辛普森	第18章
gerl	第12章
H. 保维林	第1章
哈里	第1章和第37章
哈特姆	第41章
He11ion	第1章
海梅·斯蒂尔	第36章
贾罗德·莫泽	第14章
杰夫·克罗斯	第14章

Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,
more changes can be sent to web@petercv.com for new content to be published or updated

Abrar Jahin	Chapter 25
acdcjunior	Chapters 1, 8 and 14
ahmadalibaloch	Chapter 67
aholtry	Chapters 12 and 17
Ajey	Chapter 56
Alex Morales	Chapter 20
Alexandre Junges	Chapter 21
amansoni211	Chapter 19
Amit kumar	Chapters 10 and 27
Andrei Zhytkevich	Chapter 4
Anil Singh	Chapters 12, 27 and 37
Apmis	Chapter 17
Arnold Wiersma	Chapter 16
Arun Redhu	Chapter 66
AryanJ	Chapters 17, 30 and 31
Ashok Vishwakarma	Chapter 58
Bean0341	Chapter 1
Berseker59	Chapter 12
Bhoomi Bhalani	Chapter 1
BogdanC	Chapters 1 and 52
borislemke	Chapters 4, 14, 17 and 33
BrianRT	Chapter 41
brians69	Chapter 1
briantyler	Chapter 1
BrunoLM	Chapters 2, 4, 13, 14 and 23
cDecker32	Chapter 1
Christopher Taylor	Chapters 14 and 27
Chybie	Chapter 14
dafyddPrys	Chapter 8
daniellmb	Chapters 21 and 22
Daredzik	Chapter 20
echonax	Chapter 1
elliot	Chapter 14
Eric Jimenez	Chapters 26, 28, 31 and 37
filoxo	Chapter 20
Fredrik Lundin	Chapter 14
Günter Zöchbauer	Chapter 19
Gaurav Mukherjee	Chapter 44
Gerard Simpson	Chapter 18
gerl	Chapter 12
H. Pauwelyn	Chapter 1
Harry	Chapters 1 and 37
Hatem	Chapter 41
He11ion	Chapter 1
Jaime Still	Chapter 36
Jarod Moser	Chapter 14
Jeff Cross	Chapter 14

jesussegado	第29章
吉姆	第1、7、41和55章
乔治	第11章
K3v1n	第27章
卡洛扬	第51章
卡斯帕斯·贝尔格斯	第20章和第23章
kEpEx	第40章
凯坦·阿克巴里	第61章和第62章
哈立德	第19章
lexith	第4章和第8章
LLL	第3章
洛根·H	第1章
LordTribual	第14章和第17章
luukgruijs	第60章
M4R1KU	第32章
马切伊·特雷德尔	第22章和第64章
马特里姆	第23章
马特·瓦利戈拉	第35章
马克斯·卡尔波维茨	第6章和第9章
马克西姆	第42章
米奥尔菲	第18章
michaelbahr	第14章和第66章
米哈尔·皮耶特拉什科	第1章
米哈伊	第1章和第43章
迈克·科维茨基	第43章
内特·梅	第44章
nick	第66章
尼古拉斯·伊里萨里	第1章
ob1	第10章和第12章
pd farhad	第20章
彼得	第1章
土豆工程师	第43章
ppovoski	第10章
PSabuwala	第68章
普詹·斯里瓦斯塔瓦	第12章
雷扎	第66章
rivanov	第18章
罗伯托·费尔南德斯	第24章
罗宾·迪克霍夫	第37章
罗纳德·扎rīts	第22章
鲁佩·哈库林恩	第23章和第45章
鲁米特·帕拉基亚	第66章
萨钦·S	第17章和第27章
萨姆	第59章
萨姆·斯托里	第22章
samAlvin	第10章
桑凯特	第7章和第31章
Sbats	第21章
Scrambo	第34章
塞法	第3章和第38章
沙伊莱什·拉杜莫尔	第61章
SlashTag	第18章
smnbbrv	第33章

jesussegado	Chapter 29
Jim	Chapters 1, 7, 41 and 55
Jorge	Chapter 11
K3v1n	Chapter 27
Kaloyan	Chapter 51
Kaspars Bergs	Chapters 20 and 23
kEpEx	Chapter 40
Ketan Akbari	Chapters 61 and 62
Khaled	Chapter 19
lexith	Chapters 4 and 8
LLL	Chapter 3
Logan H	Chapter 1
LordTribual	Chapters 14 and 17
luukgruijs	Chapter 60
M4R1KU	Chapter 32
Maciej Treder	Chapters 22 and 64
Matrim	Chapter 23
MatWaligora	Chapter 35
Max Karpovets	Chapters 6 and 9
Maxime	Chapter 42
meorfi	Chapter 18
michaelbahr	Chapters 14 and 66
Michal Pietraszko	Chapter 1
Mihai	Chapters 1 and 43
Mike Kovetsky	Chapter 43
Nate May	Chapter 44
nick	Chapter 66
Nicolas Irisarri	Chapter 1
ob1	Chapters 10 and 12
pd farhad	Chapter 20
Peter	Chapter 1
PotatoEngineer	Chapter 43
ppovoski	Chapter 10
PSabuwala	Chapter 68
Pujan Srivastava	Chapter 12
Reza	Chapter 66
rivanov	Chapter 18
Roberto Fernandez	Chapter 24
Robin Dijkhof	Chapter 37
Ronald Zarīts	Chapter 22
Roope Hakulinen	Chapters 23 and 45
Rumit Parakhiya	Chapter 66
Sachin S	Chapters 17 and 27
Sam	Chapter 59
Sam Storie	Chapter 22
samAlvin	Chapter 10
Sanket	Chapters 7 and 31
Sbats	Chapter 21
Scrambo	Chapter 34
Sefa	Chapters 3 and 38
Shailesh Ladumor	Chapter 61
SlashTag	Chapter 18
smnbbrv	Chapter 33

斯蒂安·斯坦达尔	第4章
沙姆·普拉迪普	第23章和第32章
TechJhola	第65章
盲眼先知	第4章和第23章
ThomasP1988	第8章
特伦特	第18章
ugreen	第39章
vijaykumar	第27章
vinagreti	第47章
约阿夫·施尼德曼	第5、11、12、15、16、27、41、46、48、49、50、51、52、53、54、57、63和69章

Stian Standahl	Chapter 4
Syam Pradeep	Chapters 23 and 32
TechJhola	Chapter 65
theblindprophet	Chapters 4 and 23
ThomasP1988	Chapter 8
Trent	Chapter 18
ugreen	Chapter 39
vijaykumar	Chapter 27
vinagreti	Chapter 47
Yoav Schniederman	Chapters 5, 11, 12, 15, 16, 27, 41, 46, 48, 49, 50, 51, 52, 53, 54, 57, 63 and 69

You may also like