angular2
中文文档

Emeryao

Published
with GitBook

目錄

Angular2 中文文档

> 翻译说明：
>
> - 关键字和专有名词保持英文
> - 实在不会翻译的保持英文
> - 尽可能做到信达雅
> - 欢迎大家指正和交流

在Gitbook上阅读**Angular2 中文文档**

翻译自Angular2官方文档
使用语言为TypeScript

本文Github地址
PR WELCOME~

目录

    - 组件样式
    - 层级注入器
    - Http客户端
    - 生命周期钩子（狗子 你变了）
    - Npm包
    - 管道
    - 路由和导航
    - 结构化指令
    - 测试
    - TypeScript配置
    - 从1.x升级
- 菜谱
    - 概览
    - Angular 1 和 2 快速对照
    - 组件协作
    - 依赖注入
    - 动态表单
    - 设置文档标题
    - TypeScript到JavaScript
- API参考

上次更新

2016-05-12

五分钟快速上手

我们的快速上手的目标是：用TypeScript构建并运行一个超级简单Angular2应用

下载源码

急着上手？下载本快速上手的源码然后开始Coding吧

查看实例

试试这个plunker上的在线例子，它显示了一些简单的信息：

学习

当然，我们不是为了在plunker上运行而去构建应用。下列是建立本文档所有例子所使用的开发环境，这也可以是真正生产环境的基础。首先，我们将要：

- 准备开发环境
- 编写应用的Angular根组件
- 编写告诉Angular如何展示根组件的main.ts文件
- 编写承载网页（index.html）

> 在我们展开这个话题的时候，我们会看到很多代码块。他们都很可以很容易的被复制和粘贴。

开发环境

我们需要准备我们的开发环境：

- 安装node和npm
- 创建应用项目文件夹
- 添加指导TypeScript编译器所需的tsconfig.json
- 添加typings.json用来定位无法找到的TypeScript定义文件
- 添加package.json用来定义我们需要用到的包和脚本
- 安装npm包和typings文件

如果你的机器上没有**node**和**npm** 请安装

创建一个新的项目文件夹

```
mkdir angular2-quickstart
cd    angular2-quickstart
```

在项目文件夹中添加**tsconfig.json**文件并复制粘贴下面的内容：

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "system",
    "moduleResolution": "node",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "removeComments": false,
    "noImplicitAny": false
  },
  "exclude": [
    "node_modules",
    "typings/main",
    "typings/main.d.ts"
  ]
}
```

`tsconfig.json` 文件用来引导TypeScript编译器。在TypeScript配置章节中可以了解到更多。

在项目文件夹中添加**typings.json**文件并复制粘贴下面的内容：

```
{
  "ambientDependencies": {
    "es6-shim": "registry:dt/es6-shim#0.31.2+20160317120654",
    "jasmine": "registry:dt/jasmine#2.2.0+20160412134438"
  }
}
```

> 许多JavaScript库用特性和语法扩展了JavaScript环境，这些是TypeScript编译器无法本地识别的。我们通过定义在 `typings.json` 文件中的TypeScript类型定义文件—*.d.ts*文件来告诉TypeScript这些扩展功能。

在项目文件夹中添加**package.json**文件并复制粘贴下面的内容：

```
{
  "name": "angular2-quickstart",
  "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": {
    "angular2": "2.0.0-beta.17",
    "systemjs": "0.19.26",
    "es6-shim": "^0.35.0",
    "reflect-metadata": "0.1.2",
    "rxjs": "5.0.0-beta.6",
    "zone.js": "0.6.12"
  },
  "devDependencies": {
    "concurrently": "^2.0.0",
    "lite-server": "^2.2.0",
    "typescript": "^1.8.10",
    "typings":"^0.8.1"
  }
}
```

通过*npm*添加我们需要的库

Angular应用开发者依赖*npm*包管理器来安装应用所需的库。Angular团队推荐定义在 `dependencies` 和 `devDependencies` 中的入门套件包。在npm包章节中查看详细信息。

帮助性脚本

我们在推荐的 `package.json` 文件中包含了许多npm脚本以应对常规的开发任务：

```
{
  "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"
  }
}
```

我们通过以下方式执行大多数npm脚本：`npm run` +脚本名。有些命令（如`start`）不需要`run`关键字。
这些脚本做了这些事情：

- `npm start` - 同时使用监视模式启动编译器和服务器
- `npm run tsc` - 运行一次TypeScript编译器
- `npm run tsc:w` - 使用监视模式运行TypeScript编译器；这个进程始终运行并实时编译TypeScript文件的改动
- `npm run lite` - 启动lite-server，一个拥有对使用路由的Angular应用有很棒支持的轻量级静态文件服务器
- `npm run typings` - 启动typings工具
- `npm run postinsall` - 在*npm*成功完成安装包之后自动被调用，这个脚本安装定义在`typings.json`文件中的TypeScript定义文件

通过在终端窗口（Windows中的命令窗口）输入以下*npm*命令来安装这些包

```
npm install
```

> 安装过程中会出现吓人的红色报错信息。安装过程通常可以从这些错误中恢复并成功完成。
>
> **npm**错误和警告
>
> 如果**npm insall**结束后没有任何以`npm ERR!`开头的控制台信息，那么一切顺利。一些`npm WARN`信息可能会出现，不过不要担心。
>
> 我们经常可以在一系列`gyp ERR!`之后看到一个`npm WARN`信息。忽略他们吧。包可能会尝试使用`node-gyp`重新编译自己。如果重新编译失败，那么这个包将会恢复（通常使用预编译好的版本）然后所有人正常工作。
>
> 只要确保在`npm install`结束后没有`npm ERR!`就行了。

我们已经都准备好了。让我们写一些代码吧。

第一个**Angular**组件

创建一个承载应用的文件夹并添加一个超级简单的Angular组件。 在根路径下创建***app***子文件夹并设置为当前路径

```
mkdir app
cd    app
```

创建名为*app.component.ts*的组件文件并粘贴下面的代码：

```
import {Component} from 'angular2/core';

@Component({
    selector: 'my-app',
    template: '<h1>My First Angular 2 App</h1>'
})
export class AppComponent { }
```

AppComponent是应用的入口

每一个Angular应用都只要有一个根组件，为了方便我们把他命名为 `AppComponent` ，它承载了客户端的用户体验。
组件是Angular应用的基本组成部分。组件通过它关联的模板控制屏幕的一部分—视图。
这个快速上手只有一个超级简单的组件。但是它拥有每一个我们将会编写的组件的必备结构：

- 一个或多个 `import` 语句用来引用我们所需的东西
- `@Component装饰器` 用来告诉Angular应该使用什么模板以及怎么创建这个组件
- `component类` 通过模板控制视图的表现和行为

导入

Angular应用是模块化的。它由许多都专注于单一目的的文件组成。
Angular本身也是模块化的。他是许多库模块的集合，我们在构建应用时将会用到许多相关的特性。

当需要模块内的东西的时候，我们就导入他。这里我们从Angular主库模块中导入Angular `Component` 装饰器函数，因为我们需要它来定义我们的组件。

```
import {Component} from 'angular2/core';
```

@Component装饰器

`Component` 是接受元数据对象的装饰器函数。这些元数据告诉Angular如何去创建和使用这个组件。

我们通过这个函数之前添加@符号来将它应用到组件类上，并通过类上的元数据对象来调用它：

```
@Component({
    selector: 'my-app',
    template: '<h1>My First Angular 2 App</h1>'
})
```

这个指定的元数据对象有两个字段，`selector` 和 `template` 。

`selector` 指定了一个简单的CSS选择器，用来确定以后一个用来展现这个组件的HTML元素。

> 这个组件用到的元素名称为 `my-app` 。Angular在承载HTML中的所有 `my-app` 元素上创建并显示一个 `AppComponent` 实例。

`template` 指定了组件的关联模板，它用一种增强的HTML来告诉Angular如何渲染组件的视图。

> 我们这个模板只有一行HTML输出*"My Fist Angular App"*。更高级的模板应该包含指向组件属性的数据绑定，并可能会指定其他有自己模板的应用组件。这些模板又可能指定别的组件。通过这种方式Angular应用形成了一棵组件树。

组件类

在文件的最底部是一个名为 `AppComponent` 空的什么事情都不做的类。

```
export class AppComponent { }
```

当我们准备要构建一个实质性的应用的时候，我们可以通过属性和业务逻辑来扩展这个类。我们这个 `AppComponent` 是空的，原因是我们这个快速上手不需要它做任何事情。

我们导出（**export**） `AppComponent` ，这样我们就可以在应用的别的地方导入（**import**）它了，在我们创建 `main.ts` 的时候就会看到。

通过*main.ts*展示

现在，我们需要一些东西去告诉Angular加载根组件。

在 `app/` 文件下添加一个新文件， `main.ts` ，内容如下：

```
import {bootstrap}    from 'angular2/platform/browser';
import {AppComponent} from './app.component';

bootstrap(AppComponent);
```

我们导入了两个我们启动应用所需要的东西：

1. Angular的浏览器 `bootstrap` 函数
2. 应用的根组件 `AppComponent`

然后我们使用 `AppComponent` 调用 `bootstrap` 。

启动的平台各异性

注意到，我们是从 `angular2/platform/browser` 导入的 `bootstrap` ，而不是 `angular2/core` 。

这是因为有不止一种方法可以启动应用。诚然，大多数运行在浏览器中的应用都会调用这个库中的启动函数。

但是，在不同的环境中加载组件也是可以的。我们可以使用Apache Cordova或者NativeScript在移动设备中加载它。我们也许希望在服务器上渲染应用的首页，这样可以提高性能并使SEO更容易。

这些目的需要我们从不同的库中导入各种各样的启动函数。

为什么要创建一个独立的*main.ts*文件？

*main.ts*文件很小。这仅仅是个快速上手。我们可以把它的代码整合在 `app.component` 文件中，并给自己留下一点复杂的东西。

我们更愿意展示一个更合适的Angular应用结构。应用启动时一个和展现视图分离的关注点。把关注点混杂在一起，会让后面的路变得难走。我们可能会使用不同的启动器在很多环境中启动 `AppComponent` 。如果组件不去尝试启动整个应用，那么测试组件会变得更容易。让我们为了使用正确的方式而多付出一点吧。

添加 `index.html`

`index.html` 是承载整个应用的网页。 导航到项目根文件夹

```
cd ..
```

在跟文件夹中创建 `index.html` 文件并粘贴以下代码：

```
<html>
  <head>
    <title>Angular 2 QuickStart</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="styles.css">

    <!-- 1. Load libraries -->
    <!-- IE required polyfills, in this exact order -->
    <script src="node_modules/es6-shim/es6-shim.min.js"></script>
    <script src="node_modules/systemjs/dist/system-polyfills.js"></script>
    <script src="node_modules/angular2/es6/dev/src/testing/shims_for_IE.js"></script>

    <script src="node_modules/angular2/bundles/angular2-polyfills.js"></script>
    <script src="node_modules/systemjs/dist/system.src.js"></script>
    <script src="node_modules/rxjs/bundles/Rx.js"></script>
    <script src="node_modules/angular2/bundles/angular2.dev.js"></script>

    <!-- 2. Configure SystemJS -->
    <script>
      System.config({
        packages: {
          app: {
            format: 'register',
            defaultExtension: 'js'
          }
        }
      });
      System.import('app/main')
            .then(null, console.error.bind(console));
    </script>
  </head>

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

HTML中有三段需要注意的地方。

1. JavaScript库
2. SystemJS的配置，也就是我们导入并运行刚刚编写的 `main` 文件的地方、
3. `<body>` 中的my-app标签，那是我们的应用生活的地方。

库

我们加载了下面的脚本

```
<!-- IE required polyfills, in this exact order -->
<script src="node_modules/es6-shim/es6-shim.min.js"></script>
<script src="node_modules/systemjs/dist/system-polyfills.js"></script>
<script src="node_modules/angular2/es6/dev/src/testing/shims_for_IE.js"></script>

<script src="node_modules/angular2/bundles/angular2-polyfills.js"></script>
<script src="node_modules/systemjs/dist/system.src.js"></script>
<script src="node_modules/rxjs/bundles/Rx.js"></script>
<script src="node_modules/angular2/bundles/angular2.dev.js"></script>
```

我们用Internet Explorer填充开始。IE需要填充来运行依赖ES2015 promise和东戴模块加载的应用。大多数应用需要这些功能，并且大多数应用应该在Internet Explorer中运行。

接下来是Angular2的填充，`angualr2-polyfills.js`。 然后是模块加载用的SystemJS库，最后是Reactive Extensions RxJS库。

> 快速上手并不会用到Reactive Extensions，但是任何实质性的应用都会想要用到他们来使用observables。我们在快速上手中添加这个库以免以后忘了。

终于，我们加载了web开发版的Angular2本人。 随着经验的增加和对于产品质量（如加载时间和内存占用等）的更加关注，我们将会做出不同的选择。

SystemJS配置

快速上手使用SystemJS来加载应用和库模块。还有很多别的选项可以使用，包括备受好评的webpack。 SystemJS恰好是一个好的选择，但是我们想要澄清的是，这只是一个选择并不是偏好。

所有的模块加载器都需要配置并且配置会随着文件结构的多样化变得更加复杂，然后我们就要开始考虑产品和性能。

我们建议对你选择的加载器有更多的了解。在这里了解更多有关SystemJS的配置。 有了这么多顾虑在心中，我们到底在快速上手的配置中做了什么呢？

```
<script>
  System.config({
    packages: {
      app: {
        format: 'register',
        defaultExtension: 'js'
      }
    }
  });
  System.import('app/main')
        .then(null, console.error.bind(console));
</script>
`
```

`packages` 节点告诉SystemJS当他看到一个来自 `app/` 文件夹下的请求的时候应该做什么。

在快速上手中，当一个TypeScript文件有类似下面那样的导入语句时就会创建一个这样的请求：

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

注意：`from` 后的模块名并没有提到文件的后缀名。`packages:` 配置告诉SystemJS默认的后缀名为js，一个JavaScript文件。

这是有道理的，因为我们在运行应用之前会把TypeScript转译为JavaScript。

> 浏览器中的转译
> 在plunker上的例子中，我们只在运行时转译为JavaScript。这对于演示是没问题的。但这并不是我们的在开发或者产品中的偏好。 我们建议在运行应用之前的编译阶段转译为JavaScript，原因包括以下几点：
>
> - 在浏览器中我们看不到编译器的警告和报错。
> - 预编译简化了模块加载流程，并使诊断问题变得更简单，因为这是一个独立的外部步骤。
> - 只重新编译改变的文件可以是迭代开发更快。在大量文件前面，我们会随着应用的成长注意到变化。
> - 预编译更适应编译测试部署的持续集成流程。

调用 `System.import` 告诉SystemJS去导入 `main` 文件（`main.ts` 转译后的 `main.js`，还记得吗？）。`main` 是我们告诉Angular启动应用的地方。我们同时也捕获并在控制台记录了启动错误。

所有其他模块都会通过import语句或者Angular自身被加载。

<my-app>

当Angular调用 `main.ts` 中的 `bootstrap` 函数的时候，它读取 `AppComponent` 元数据，找到 `my-app` 选择器，定位一个标签名为 `my-app` 的标签，然后再这些标签之间加载我们的应用。

添加式样

式样并不是必须的，但是是很棒的。`index.html` 假定了一个名为 `styles.css` 的式样表。

在根文件夹中创建一个 `style.css` 文件并开始编写，可以使用下列的样子：

```
/* Master Styles */
h1 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
h2, h3 {
  color: #444;
  font-family: Arial, Helvetica, sans-serif;
  font-weight: lighter;
}
body {
  margin: 2em;
}
body, input[text], button {
  color: #888;
  font-family: Cambria, Georgia;
}

/*
 * See https://github.com/angular/angular.io/blob/master/public/docs/_examples/styles.css
 * for the full set of master styles used by the documentation samples
 */
```

编译并运行！

打开终端窗口并输入下列命令：

```
npm start
```

这个命令启动两个并行的node进程：

1. 监视模式的TypeScript编译器
2. 一个在浏览器中加载 `index.html` 的叫做**lite-server**静态服务器，当应用文件改变的时候，浏览器会刷新。

不一会儿，一个浏览器标签页就会打开并显示：

恭喜你！我们上道了。

改点东西

试试把显示信息改成“My SECOND Angular 2 app”。

TypeScript编译器和 `lite-server` 正在监视。他们会检测到改变，重编译TypeScript为JavaScript，刷新浏览器，显示新的信息。

这是开发应用最漂亮的方式。

当我们杀掉编译器和服务器进程之后，可以关闭终端窗口。

文件结构

最终的项目文件夹结构应该是这样的：

```
angular2-quickstart
├app
│├app.component.ts
│└main.ts
├node_modules ...
├typings ...
├index.html
├package.json
├styles.css
├tsconfig.json
└typings.json
```

总结

我们第一个应用并没有做很多事情。它是指一个基本的Angular2“哈喽，沃德”。

我们在第一步保持简单：我们编写了一个小的Angular组件，我们在 `index.html` 中添加了JavaScript库，然后通过一个静态文件服务器启动。 这就是一个“哈喽，沃德”应用应该做的所有事情。

我们有更伟大的野心

好消息是，预先的准备工作已经（大部分）完成了。我们可能只需要修改 `packages.json` 来更新库。我们可能只会在需要添加库或者css式样的时候才会打开 `index.html` 。

我们该采取下一步行动了，构建一个小应用来展示使用Angular2可以构建出的伟大的东西。

加入我们的超级英雄巡礼教程吧！

教程：超级英雄巡礼