Sample: Angular 6 快速入门

One framework. Mobile & desktop.

WisdomFusion

https://github.com/WisdomFusion

摘要

只是样例填充的文字，内容都不相干的！Laravel 5.6 在 Laravel 5.5 的基础上继续进行优化，包括**日志系统**、单机任务调度、模型序列化优化、动态频率限制、广播频道类、API 资源控制器生成、Eloquent 日期格式化优化、Blade 组件别名、Argon2 密码哈希支持、引入 Collision 扩展包等等等等。此外，所有的前端脚手架代码都已升级到 Bootstrap 4，Laravel 底层使用的 Symfony 组件都已升级到 Symfony ~4.0 版本。

关键字：LISP，PHP，学习环境

1 什么是 Angular?

Angular 是一个开发平台。它能帮你更轻松的构建 Web 应用。Angular 集声明式模板、依赖注入、端到端工具和一些最佳实践于一身，为你解决开发方面的各种挑战。Angular 为开发者提升构建 Web、手机或桌面应用的能力。

2 架构概览

Angular 是一个用 HTML 和 TypeScript 构建客户端应用的平台与框架。Angular 本身使用 TypeScript 写成的。它将核心功能和可选功能作为一组 TypeScript 库进行实现，你可以把它们导入你的应用中。

Angular 的基本构造块是 NgModule，它为组件提供了编译的上下文环境。NgModule 会把相关的代码收集到一些功能集中。Angular 应用就是由一组 NgModule 定义出的。应用至少会有一个用于引导应用的根模块，通常还会有很多特性模块。

- 组件定义**视图**。视图是一组可见的屏幕元素，Angular 可以根据你的程序逻辑和数据来选择和修改它们。每个应用都至少有一个根组件。
- 组件使用**服务**。服务会提供那些与视图不直接相关的功能。服务提供商可以作为依赖被注入到组件中，这能让你的代码更加模块化、可复用，而且高效。

3 @NgModule 元数据

NgModule 是一个带有 @NgModule 装饰器的类。@NgModule 装饰器是一个函数，它接受一个元数据对象，该对象的属性用来描述这个模块。其中最重要的属性如下。

- declarations（可声明对象表）——那些属于本 NgModule 的组件、指令、管道。
- exports（导出表）——那些能在其它模块的组件模板中使用的可声明对象的子集。
- imports（导入表）——那些导出了本模块中的组件模板所需的类的其它模块。
- providers ——本模块向全局服务中贡献的那些服务的创建器。这些服务能被本应用中的任何部分使用。（你也可以在组件级别指定服务提供商，这通常是首选方式。）
- bootstrap ——应用的主视图，称为根组件。它是应用中所有其它视图的宿主。只有根模块才应该设置这个 bootstrap 属性。

下面是一个简单的根 NgModule 定义：

</> CODE 1: src/app/app.module.ts

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
@NgModule({
  imports:      [ BrowserModule ],
  providers:    [ Logger ],
  declarations: [ AppComponent ],
  exports:      [ AppComponent ],
  bootstrap:    [ AppComponent ]
})
export class AppModule { }
```

```
cd my-app
ng serve --open
```

```
blablabla
```

4 快速上手

Angular CLI 是一个命令行界面工具，它可以创建项目、添加文件以及执行一大堆开发任务，比如测试、打包和发布。

本章的目标是构建并运行一个超级简单的 TypeScript Angular 应用。使用 Angular CLI 来让每个 Angular 应用从风格指南的那些建议中获益。

在本章的末尾，你会对用 CLI 进行开发有一个最基本的理解，并将其作为其它文档范例以及真实应用的基础。

4.1 步骤 1. 设置开发环境

在开始工作之前，你必须设置好开发环境。

如果你的电脑里没有 Node.js® 和 npm，请安装它们。

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

4.2 步骤 2. 创建新项目

打开终端窗口。

运行下列命令来生成一个新项目以及默认的应用代码：

```
ng new my-app
```

4.3 步骤 3. 启动开发服务器

进入项目目录，并启动服务器。

```
cd my-app
ng serve --open
```

你的应用代码位于 src 文件夹中。所有的 Angular 组件、模板、样式、图片以及你的应用所需的任何东西都在那里。这个文件夹之外的文件都是为构建应用提供支持用的。

```
src
  app
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
  assets
    .gitkeep
  environments
    environment.prod.ts
    environment.ts
  browserslist
  favicon.ico
  index.html
  ...
```

blabla

表 1: sample table 表

文件	用途
app/app.component.{ts,html,css,spec.ts}	使用 HTML 模板、CSS 样式和单元测试定义 AppComponent 组件。它是根组件，随着应用的成长它会成为一棵组件树的根节点。
app/app.module.ts	定义 AppModule，根模块为 Angular 描述如何组装应用。目前，它只声明了 AppComponent。不久，它将声明更多组件。

5 boxes

\begin{titledBox}{<title>} <content> \end{titledBox}

HTTP/Console 内核

HTTP 内核继承自 Illuminate\Foundation\Http\Kernel 类，该类定义了一个 bootstrappers 数组，这个数组中的类在请求被执行前运行，这些 bootstrappers 配置了错误处理、日志、检测应用环境以及其它在请求被处理前需要执行的任务。

\begin{noteBox} <content> \end{noteBox}

Note

HTTP 内核继承自 Illuminate\Foundation\Http\Kernel 类，该类定义了一个 bootstrappers 数组，这个数组中的类在请求被执行前运行，这些 bootstrappers 配置了错误处理、日志、检测应用环境以及其它在请求被处理前需要执行的任务。

\begin{importantBox} <content> \end{importantBox}

Important

HTTP 内核继承自 Illuminate\Foundation\Http\Kernel 类，该类定义了一个 bootstrappers 数组，这个数组中的类在请求被执行前运行，这些 bootstrappers 配置了错误处理、日志、检测应用环境以及其它在请求被处理前需要执行的任务。

\begin{tipBox} <content> \end{tipBox}

Tip

HTTP 内核继承自 Illuminate\Foundation\Http\Kernel 类，该类定义了一个 bootstrappers 数组，这个数组中的类在请求被执行前运行，这些 bootstrappers 配置了错误处理、日志、检测应用环境以及其它在请求被处理前需要执行的任务。

\begin{warningBox} <content> \end{warningBox}

Warning

HTTP 内核继承自 Illuminate\Foundation\Http\Kernel 类，该类定义了一个 bootstrappers 数组，这个数组中的类在请求被执行前运行，这些 bootstrappers 配置了错误处理、日志、检测应用环境以及其它在请求被处理前需要执行的任务。

\begin{shellBox} <content> \end{shellBox}

```
cd my-app
ng serve --open
```

\begin{invertedShellBox} <content> \end{invertedShellBox}

```
cd my-app
ng serve --open
```

性质 对于任意顶点 u 和 v，$[prev(u), post(u)]$ 和 $[prev(v), post(v)]$ 两个区间要么彼此分离，要么一个包含另外一个。

6 服务容器 3

Laravel 服务容器是一个用于管理类依赖和执行依赖注入的强大工具。依赖注入听上去很花哨，其实质是通过构造函数或者某些情况下通过 setter 方法将类依赖注入到类中。

</> CODE 2: PHP 代码样例

```php
<?php
namespace App\Http\Controllers;

use App\User;
use App\Repositories\UserRepository;
use App\Http\Controllers\Controller;

class UserController extends Controller
{
  /**
  * The user repository implementation.
  *
  * @var UserRepository
  */
  protected $users;

  /**
  * Create a new controller instance.
  *
  * @param  UserRepository  $users
  * @return void
  */
  public function __construct(UserRepository $users)
  {
    $this->users = $users;
  }

  ...
}
```