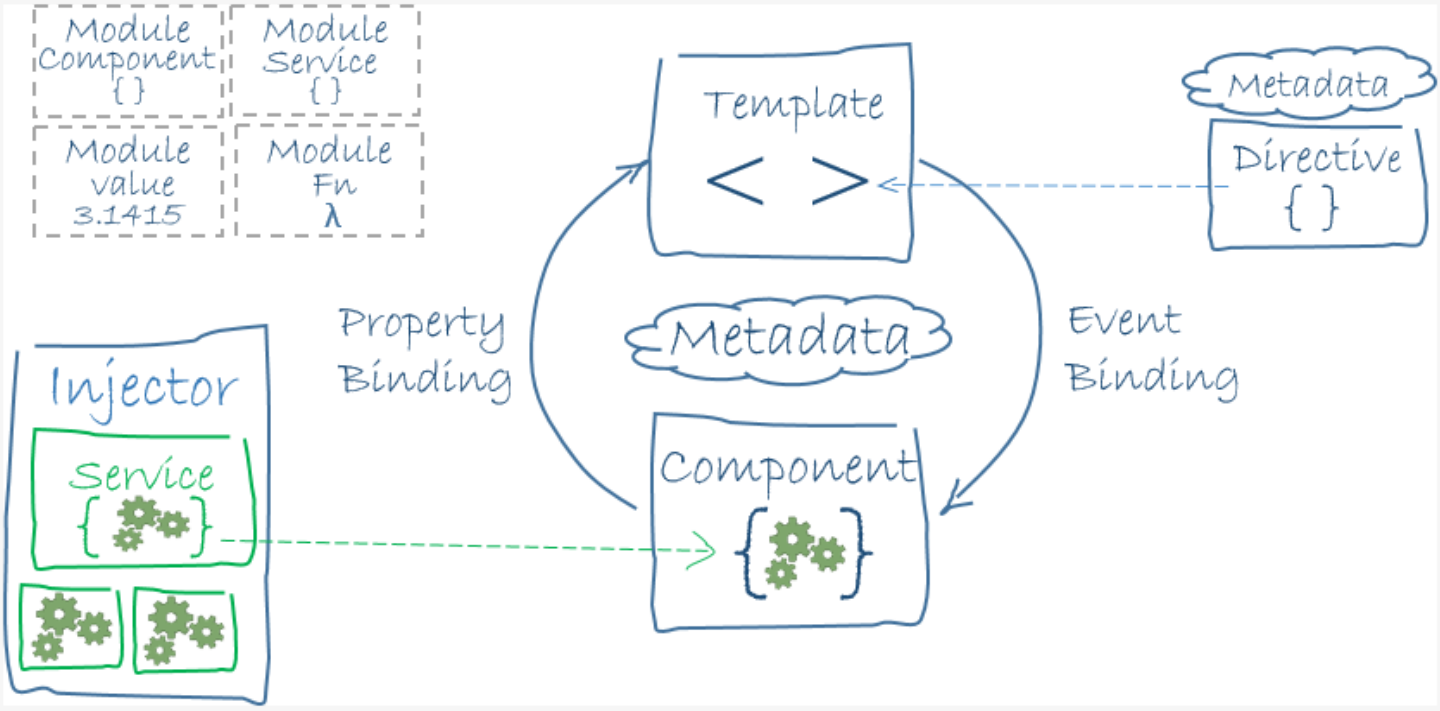
← Angular 2 TypeScript 环境配置 Angular 2 数据显示 →

Angular 2 架构

Angular 2 应用程序应用主要由以下 8 个部分组成：

- 1、模块 (Modules)
- 2、组件 (Components)
- 3、模板 (Templates)
- 4、元数据 (Metadata)
- 5、数据绑定 (Data Binding)
- 6、指令 (Directives)
- 7、服务 (Services)
- 8、依赖注入 (Dependency Injection)

下图展示了每个部分是如何相互工作的：



图中的模板 (Templates)是由 Angular 扩展的 HTML 语法组成，组件 (Components)类用来管理这些模板，应用逻辑部分通过服务 (Services)来完成，然后在模块中打包服务与组件，最后通过引导根模块来启动应用。

接下来我们会对以上 8 个部分分开解析：

模块

模块由一块代码组成，可用于执行一个简单的任务。

Angular 应用是由模块化的，它有自己的模块系统：NgModules。

每个 Angular 应该至少要有一个模块(根模块)，一般可以命名为：AppModule。

Angular 模块是一个带有 @NgModule 装饰器的类，它接收一个用来描述模块属性的元数据对象。

几个重要的属性如下：

- **declarations （声明）** - 视图类属于这个模块。 Angular 有三种类型的视图类： 组件 、 指令 和 管道 。
- **exports** - 声明（ declaration ）的子集，可用于其它模块中的组件模板 。
- **imports** - 本模块组件模板中需要由其它导出类的模块。
- **providers** - 服务的创建者。本模块把它们加入全局的服务表中，让它们在应用中的任何部分都可被访问到。
- **bootstrap** - 应用的主视图，称为根组件，它是所有其它应用视图的宿主。只有根模块需要设置 bootstrap 属性中。

一个最简单的根模块:

app/app.module.ts 文件：

```
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 { }
```

接下来我们通过引导根模块来启动应用，开发过程通常在 main.ts 文件中来引导 AppModule ，代码如下：

app/main.ts 文件：

```
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { AppModule } from './app.module';
platformBrowserDynamic().bootstrapModule(AppModule);
```

组件(Components)

组件是一个模板的控制类用于处理应用和逻辑页面的视图部分。

组件是构成 Angular 应用的基础和核心，可用于整个应用程序中。

组件知道如何渲染自己及配置依赖注入。

组件通过一些由属性和方法组成的 API 与视图交互。

创建 Angular 组件的方法有三步：

- 从 @angular/core 中引入 Component 修饰器
- 建立一个普通的类，并用 @Component 修饰它
- 在 @Component 中，设置 selector **自定义标签**，以及 template **模板**

模板(Templates)

Angular模板的默认语言就是HTML。

我们可以通过使用模板来定义组件的视图来告诉 Angular 如何显示组件。以下是一个简单是实例：

```
<div>
网站地址 : {{site}}
</div>
```

在Angular中，默认使用的是双大括号作为插值语法，大括号中间的值通常是一个组件属性的变量名。

元数据(Metadata)

元数据告诉 Angular 如何处理一个类。

考虑以下情况我们有一个组件叫作 Component ，它是一个类，直到我们告诉 Angular 这是一个组件为止。

你可以把元数据附加到这个类上来告诉 Angular Component 是一个组件。

在 TypeScript 中，我们用 装饰器 (decorator) 来附加元数据。

实例

```
@Component({
   selector : 'mylist',
   template : '<h2>菜鸟教程</h2>'
   directives : [ComponentDetails]
})
export class ListComponent{...}
```

@Component 装饰器能接受一个配置对象，并把紧随其后的类标记成了组件类。

Angular 会基于这些信息创建和展示组件及其视图。

@Component 中的配置项说明：

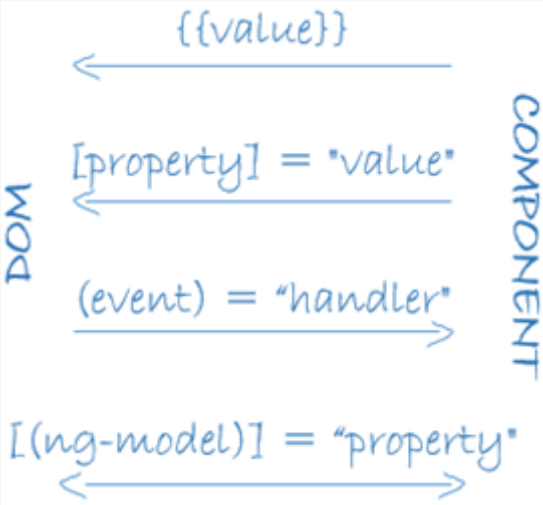
- **selector** - 一个 css 选择器，它告诉 Angular 在 父级 HTML 中寻找一个 <mylist> 标签，然后创建该组件，并插入此标签中。
- **templateUrl** - 组件 HTML 模板的地址。
- **directives** - 一个数组，包含 此 模板需要依赖的组件或指令。
- **providers** - 一个数组，包含组件所依赖的服务所需要的依赖注入提供者。

数据绑定(Data binding)

数据绑定为应用程序提供了一种简单而一致的方法来显示数据以及数据交互，它是管理应用程序里面数值的一种机制。

通过这种机制，可以从HTML里面取值和赋值，使得数据的读写，数据的持久化操作变得更加简单快捷。

如图所示，数据绑定的语法有四种形式。每种形式都有一个方向——从 DOM 来、到 DOM 去、双向，就像图中的箭头所示意的。



- **插值** : 在 HTML 标签中显示组件值。

```
<h3>
{{title}}
<img src="{{ImageUrl}}">
</h3>
```

- **属性绑定**: 把元素的属性设置为组件中属性的值。

```
<img [src]="userImageUrl">
```

- **事件绑定**: 在组件方法名被点击时触发。

```
<button (click)="onSave()">保存</button>
```

- **双向绑**: 使用Angular里的NgModel指令可以更便捷的进行双向绑定。

```
<input [value]="currentUser.firstName"
       (input)="currentUser.firstName=$event.target.value" >
```

指令（Directives）

Angular模板是动态的 。当 Angular 渲染它们时，它会根据指令对 DOM 进行修改。

指令是一个带有"指令元数据"的类。在 TypeScript 中，要通过 @Directive 装饰器把元数据附加到类上。

在Angular中包含以下三种类型的指令：

- 属性指令：以元素的属性形式来使用的指令。

- 结构指令：用来改变DOM树的结构
- 组件：作为指令的一个重要子类，组件本质上可以看作是一个带有模板的指令。

```
<li *ngFor="let site of sites"></li>
<site-detail *ngIf="selectedSite"></site-detail>
```

*ngFor 告诉 Angular 为 sites 列表中的每个项生成一个 <li> 标签。

*ngIf 表示只有在选择的项存在时，才会包含 SiteDetail 组件。

服务(Services)

Angular2中的服务是封装了某一特定功能，并且可以通过注入的方式供他人使用的独立模块。

服务分为很多种，包括：值、函数，以及应用所需的特性。

例如，多个组件中出现了重复代码时，把重复代码提取到服务中实现代码复用。

以下是几种常见的服务：

- 日志服务
- 数据服务
- 消息总线
- 税款计算器
- 应用程序配置

以下实例是一个日志服务，用于把日志记录到浏览器的控制台：

```
export class Logger {
  log(msg: any)   { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any)  { console.warn(msg); }
}
```

依赖注入

> *控制反转（Inversion of Control，缩写为IoC），是面向对象编程中的一种设计原则，可以用来减低计算机代码之间的耦合度。其中最常见的方式叫做依赖注入（Dependency Injection，简称DI），还有一种方式叫"依赖查找"（Dependency Lookup）。*
>
> *通过控制反转，对象在被创建的时候，由一个调控系统内所有对象的外界实体，将其所依赖的对象的引用传递给它。也可以说，依赖被注入到对象中。*
>
> *在传统的开发模式中，调用者负责管理所有对象的依赖，循环依赖一直是梦魇，而在依赖注入模式中，这个管理权交给了注入器(Injector)，它在软件运行时负责依赖对象的替换，而不是在编译时。这种控制反转，运行注入的特点即是依赖注入的精华所在。*

Angular 能通过查看构造函数的参数类型，来得知组件需要哪些服务。 例如， SiteListComponent 组件的构造函数需要一个 SiteService:

```
constructor(private service: HeroService) { }
```

当 Angular 创建组件时，会首先为组件所需的服务找一个注入器（ Injector ）。

注入器是一个维护服务实例的容器，存放着以前创建的实例。

如果容器中还没有所请求的服务实例，注入器就会创建一个服务实例，并且添加到容器中，然后把这个服务返回给 Angular 。

当所有的服务都被解析完并返回时， Angular 会以这些服务为参数去调用组件的构造函数。 这就是依赖注入 。

← Angular 2 TypeScript 环境配置 Angular 2 数据显示 →