AngularJS

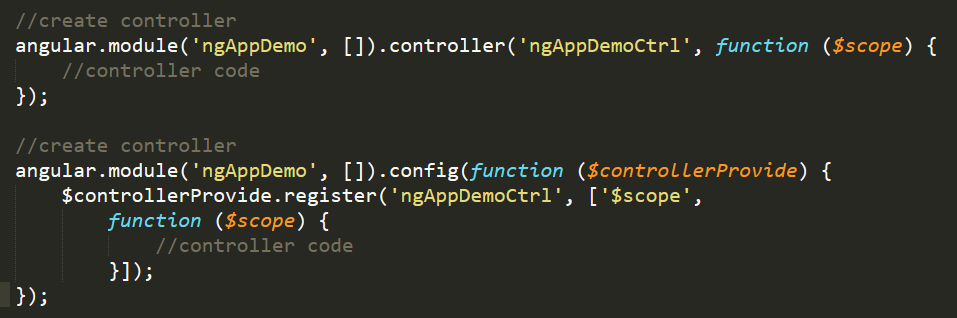
# 依赖注入

创建性依赖：使用new方法创建一个实例对象，而这个新创建的对象，则依赖于new后面的原对象。不利于代码的测试和内部依赖关系的修改，调用也较复杂。

## 依赖注入原理

每个Angular应用都有一个injector注入器来处理依赖对象的创建，注入器实际上是一个负责查找和创建依赖的服务定位器，所有声明的依赖注入对象都是由它来进行处理。此外，当获取injector注入器对象后，还可以调用该对象的get函数来获得任何一个被定义过的服务的实例。

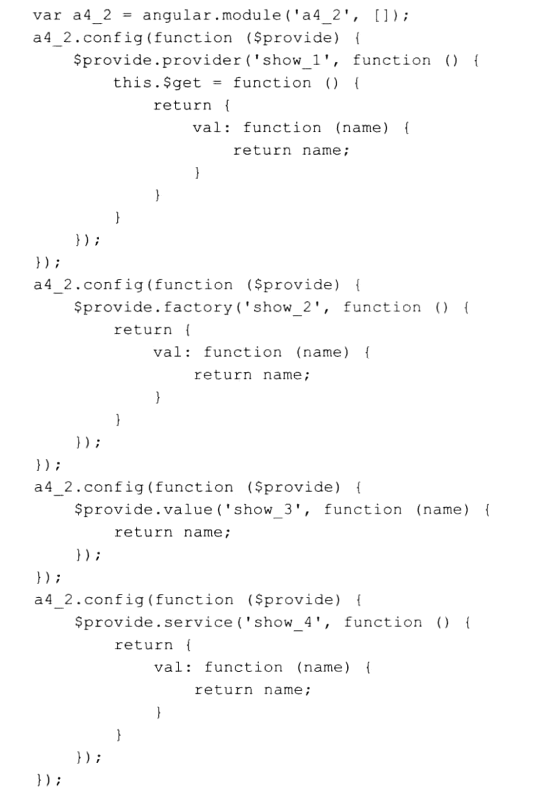
### 控制器实现依赖注入



第一段的本质为第二段代码，在Angular中，可以通过模块中的config函数来声明需要注入的依赖对象，声明的方式是调用provider服务。但在Angular内部，controller控制器并不是由provider服务来创建的，而是通过controllerProvider服务的register方法，完成一个控制器的创建，当控制器创建完成后，再调用injector注入器完成各个依赖对象的注入。

Config:为定义的模版对象注入依赖的各种服务。

Provider服务方法：provider，factory，service，value，这些方法都可以通过服务创建一个自定义的依赖注入对象。（示例如下）

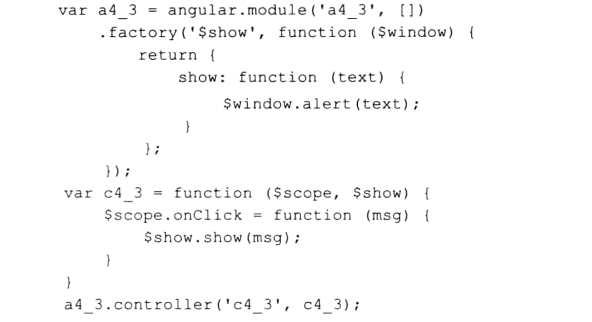


## 依赖注入方式

### 1、推断式注入

猜测式的注入，在没有明确的声明的情况下，Angular会认定参数名称就是依赖注入的函数名，并在内部调用函数对象的toString()方法，获取对应的参数列表。最后调用injectoy将这些参数注入应用的实例中，从而实现依赖注入的过程。

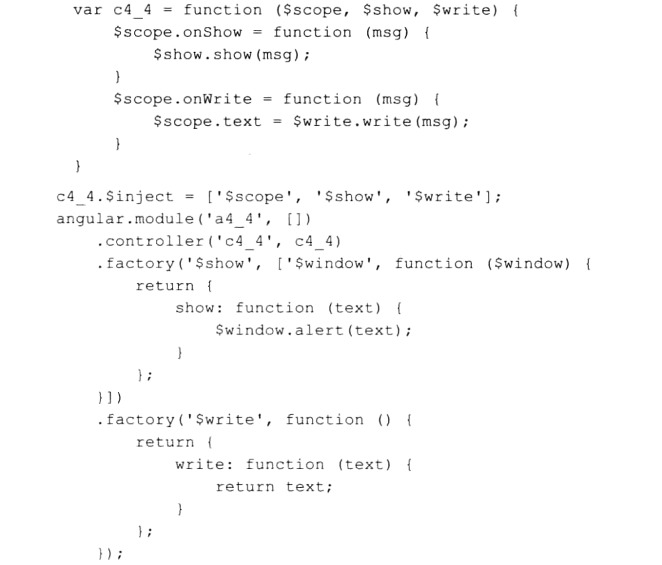
这种注入方式不需要关注注入时参数的先后顺序，Angular会根据依赖的程度自动处理，由于Angular需要根据参数列表分析注入服务，因此这种注入方式不能处理压缩或混淆后的代码，只能处理原始的代码。



### 2、标记式注入

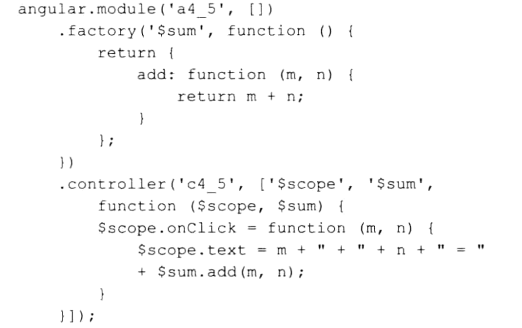
直接调用$injector属性来完成，该属性是一个字符型的数组，其中的各个元素值就是需要注入的各项服务名称，正是由于他是一个数组，导致这种注入方式的顺序非常重要。

正是由于服务名和函数参数名在名称和顺序的一一对应关系，使得服务名与函数提绑定在一起。因此，这种标记式的注入声明可以在压缩或混淆后的代码中执行。



### 3、行内注入\*

在构建一个Angular对象（如控制器对象）时，允许开发人员将一个字符型数组作为对象的参数，而不仅仅是一个函数，在这个数组中，除最后一个必须是函数体外，其余都代表注入对象的服务名，而它们的名称和顺序与最后一个函数的参数是一一对应的。



## $injector API

### Has

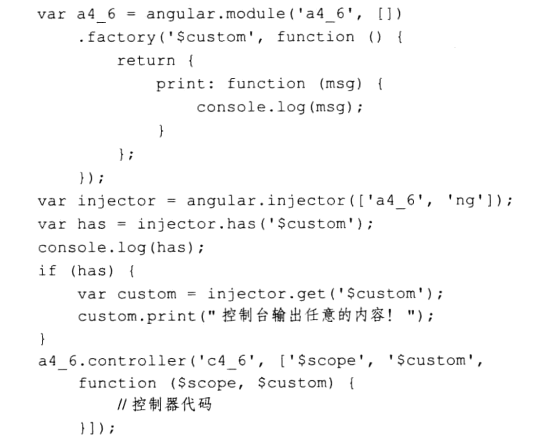
Injector.has(name)

根据传入的名称，从注册的列表中查找对应的服务，返回true、false。

### Get

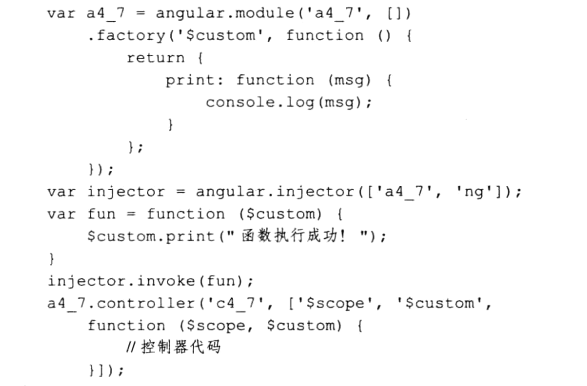
Injector.get(name)

将返回指定名称的服务实例，获取到服务的实例对象后，就可以直接调用服务中的属性和方法。



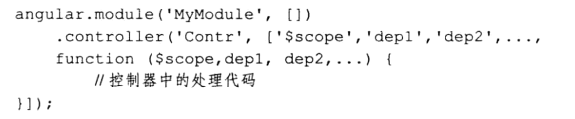
### Invoke

最常用的场景就是执行一个自定义的函数。此外，在执行函数时，还能传递变量给函数自身。调用格式：injector.invoke(fn,[self],[locals])



## 依赖注入的应用场景

### 1、构建控制器



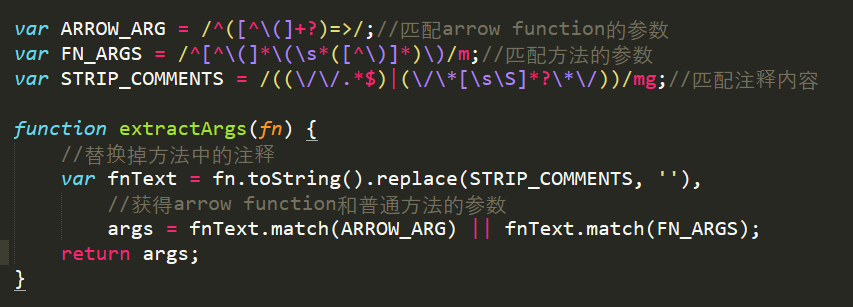
### 2、调用工厂方法



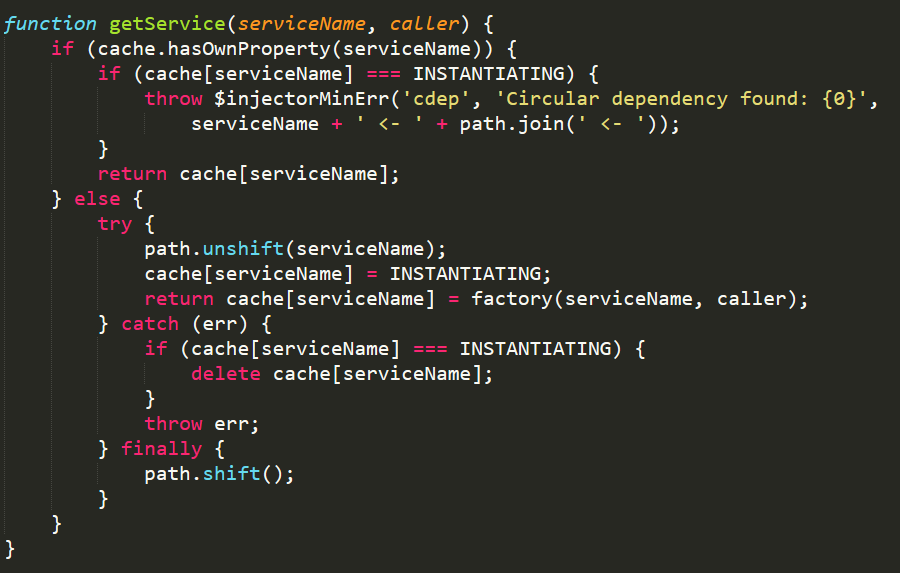
## Angular实现依赖注入

### 1、得到模块的依赖项

Function.prototype.toString();

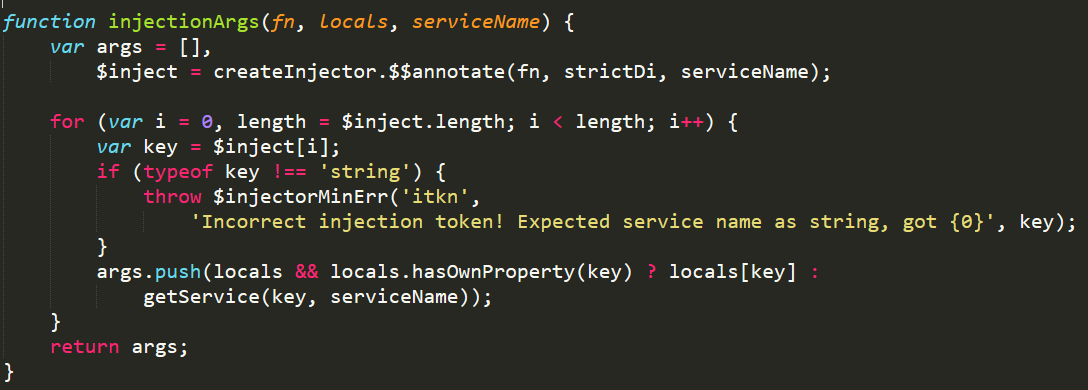


### 2、查找依赖所对应的对象

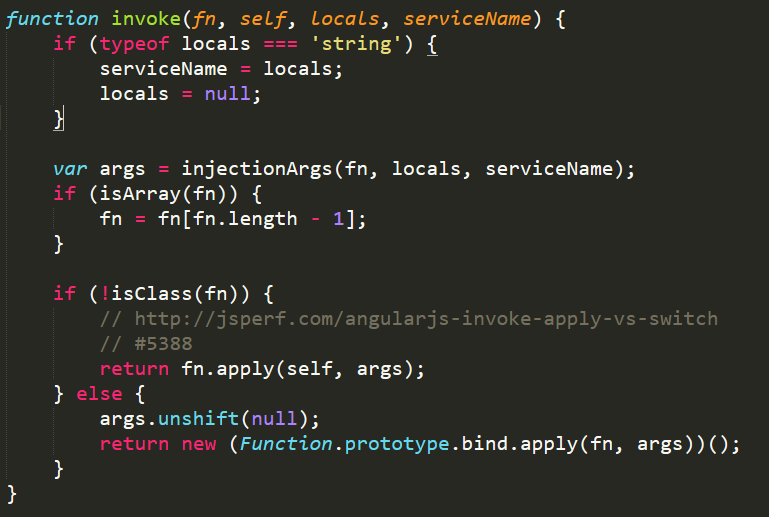


### 3、执行时注入

得到参数



调用



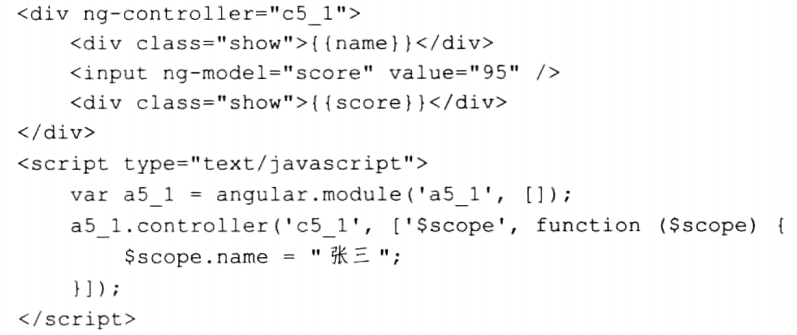
# MVC

优势：提升服务器的性能、减少项目开发时间

缺点：页面渲染缓慢、页面兼容性差，不利于搜索引擎识别

## Model

存储和处理数据，还可以在组件中自定义模版，通过模版实现与View层的通信和数据交互。



### ngRepeater（ng-repeat）

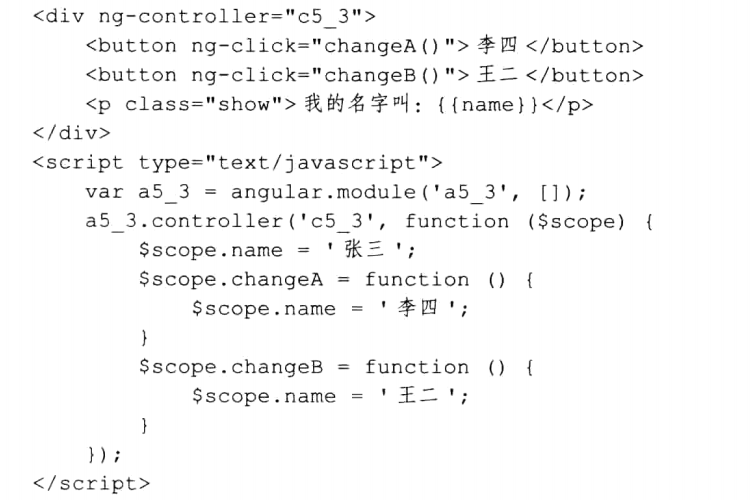


## Controller

本质是一个JavaScript函数，用于衔接页面模版和逻辑代码，并通过添加对象和行为来增强模版中作用于的功能。

### 控制器的属性和方法

当创建一个Angular应用时，除在页面中添加各类视图模版外，通常还需要创建控制器来实现视图模版与逻辑代码间的关联，而实现关联的方法是以scope.$new的形式向控制器中添加模型属性，添加成功后，就可以在视图的模版中通过各种方式进行访问，从而实现数据的关联。



### 控制器方法中的参数

在控制器中添加的方法除了可以重置属性值外，与普通的JavaScript方法一样，也可以定义形参，在调用过程中传递实参，功能强大。

### 控制器中属性和方法的继承（不推荐）



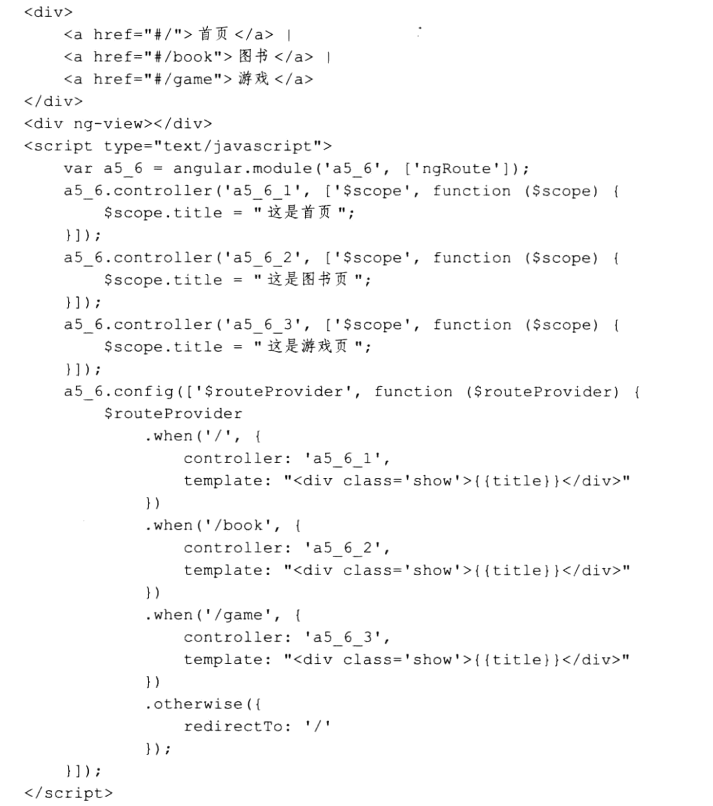
严格来讲，一个控制器只包括一个对应视图模版的业务逻辑，不应将过多的页面DOM操作加入到控制器中，这样容易加大测试操作的复杂度。另外，不属于视图业务逻辑的数据请求和获取，应尽量通过服务调用的形式来实现，而不应在控制器中实现，可以保证控制器的功能单一性。

## View

先经过浏览器加载并渲染后，再根据视图模版和对应控制器模型修改后所包含的DOM元素。它的外形取决于视图模版，内部数据来源于控制器。所以可以通过ng-View指令加载和切换视图模版，并将视图组件通过ng-Controller指令与控制器绑定。

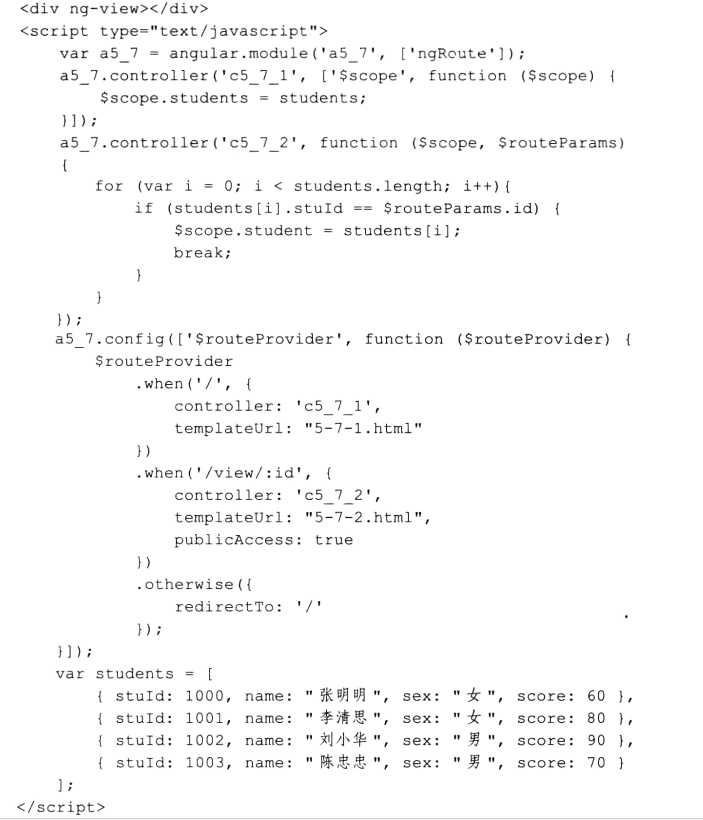
### 模版切换（路由）

在视图模版中使用ng-View指令，并在控制器中引入$routeProvider服务。



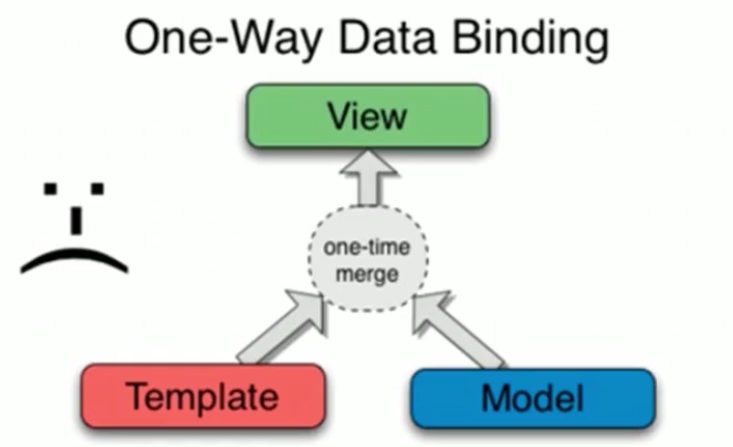
为了实现在一个页面中切换模版的功能，需要在页面中添加一个带有ng-View指令的元素作为装载不通模版的容器，所有切换后的页面都在该容器中展示。

### 切换视图模版时传参数（$routeParams）

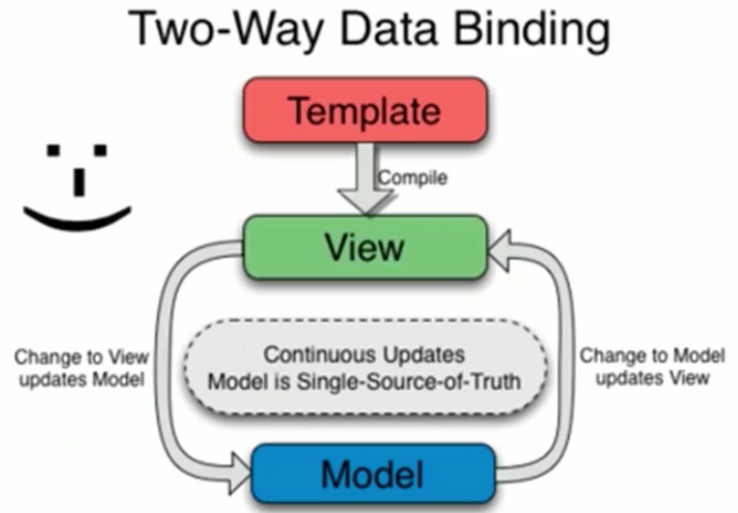


# 双向数据绑定

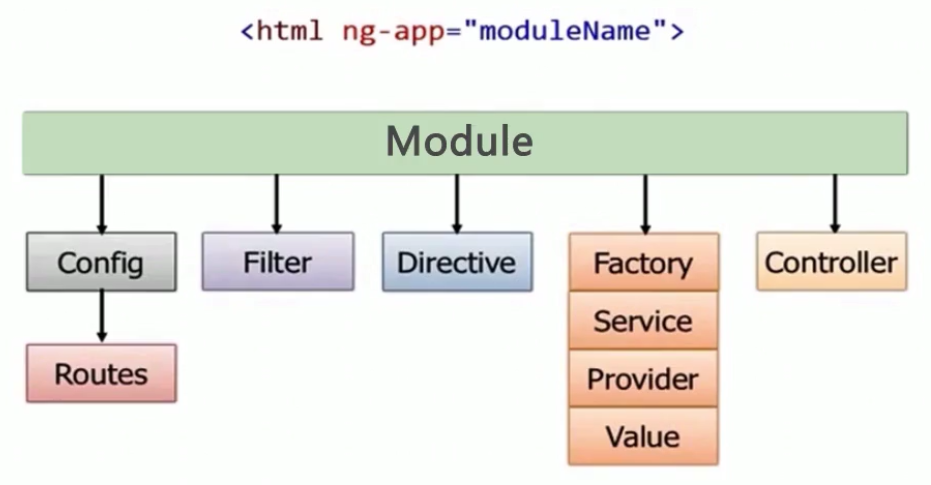
单向：



双向：



# 模块化Module



angular在每次应用启动时，初始化一个Injector实例：



对每一个Angular应用来说，无论是哪个模块，所有的"provider"都是存在相同的providerCache或cache中,所以会导致一个模块管理的问题：module 并没有什么命名空间的作用，当依赖名相同的时候，后面引用的会覆盖前面引用的模块。