|  |
| --- |
| Why  Angular2 |
|  | Angular1.x显然非常成功，那么，为什么要剧烈地转向Angular2？ |
|  |  |
|  | 性能的限制 |
|  | AngularJS当初是提供给设计人员用来快速构建HTML表单的一个内部工具。随着时间的推移，各种特性 被加入进去以适应不同场景下的应用开发。然而由于最初的架构限制（比如绑定和模板机制），性能的 提升已经非常困难了。 |
|  |  |
|  | 快速变化的WEB |
|  |  |
|  | 在语言方面，ECMAScript6的标准已经完成，这意味着浏览器将很快支持例如模块、类、lambda表达式、 generator等新的特性，而这些特性将显著地改变JavaScript的开发体验。 |
|  |  |
|  | 在开发模式方面，Web组件也将很快实现。然而现有的框架，包括Angular1.x对WEB组件的支持都不够好。 |
|  |  |
|  | 移动化 |
|  |  |
|  | 想想5年前......现在的计算模式已经发生了显著地变化，到处都是手机和平板。Angular1.x没有针对移动 应用特别优化，并且缺少一些关键的特性，比如：缓存预编译的视图、触控支持等。 |
|  |  |
|  | 简单易用 |
|  |  |
|  | 说实话，Angular1.x太复杂了，学习曲线太陡峭了，这让人望而生畏。Angular团队希望在Angular2中将复杂性 封装地更好一些，让暴露出来的概念和开发接口更简单。 |
|  |  |
|  | ES6工具链 |
|  | 要让Angular2应用跑起来不是件轻松的事，因为它用了太多还不被当前主流浏览器支持 的技术。所以，我们需要一个工具链：  toolchain |
|  |  |
|  | toolchain |
|  |  |
|  | Angular2是面向未来的科技，要求浏览器支持ES6+，我们现在要尝试的话，需要加一些 垫片来抹平当前浏览器与ES6的差异： |
|  |  |
|  | systemjs - 通用模块加载器，支持AMD、CommonJS、ES6等各种格式的JS模块加载 |
|  | es6-module-loader - ES6模块加载器，systemjs会自动加载这个模块 |
|  | traceur - ES6转码器，将ES6代码转换为当前浏览器支持的ES5代码。systemjs会自动加载 这个模块。 |

初识Angular2

写一个Angular2的Hello World应用相当简单，分三步走：

**1. 引入Angular2预定义类型**

1. import {Component,View,bootstrap} from "angular2/angular2";

*import*是ES6的关键字，用来从模块中引入类型定义。在这里，我们从angular2模块库中引入了三个类型： Component类、View类和bootstrap函数。

**2. 实现一个Angular2组件**

实现一个Angular2组件也很简单，定义一个类，然后给这个类添加*注解*：

1. @Component({selector:"ez-app"})
2. @View({template:"<h1>Hello,Angular2</h1>"})
3. class EzApp{}

*class*也是ES6的关键字，用来定义一个类。*@Component*和*@View*都是给类*EzApp*附加的元信息， 被称为*注解/Annotation*。

*@Component*最重要的作用是通过*selector*属性（值为CSS选择符），指定这个组件渲染到哪个DOM对象上。 *@View*最重要的作用是通过*template*属性，指定渲染的模板。

**3. 渲染组件到DOM**

将组件渲染到DOM上，需要使用*自举/bootstrap*函数：

1. bootstrap(EzApp);

这个函数的作用就是通知Angular2框架将*EzApp*组件渲染到DOM树上。

简单吗？我知道你一定还有疑问，别着急，我们慢慢把缺失的知识点补上！

把@Component的selector属性改为"ez-app"，还应该改哪里可以获得和示例同样的结果？

注解/Annotation

你一定好奇*@Component*和*@View*到底是怎么回事。看起来像其他语言（比如python） 的*装饰器*，是这样吗？

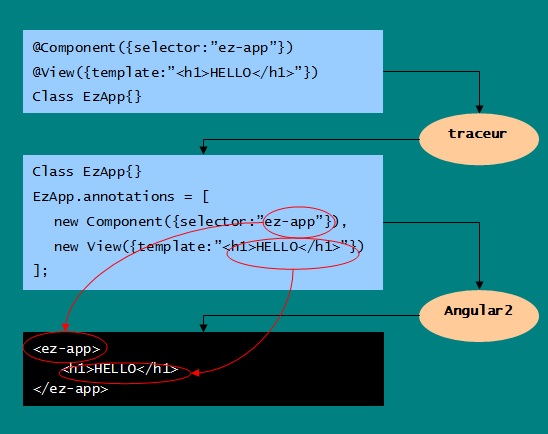
ES6规范里没有装饰器。这其实利用了*traceur*的一个实验特性：*注解*。给一个类 加注解，等同于设置这个类的*annotations*属性：

1. //注解写法
2. @Component({selector:"ez-app"})
3. class EzApp{...}

等同于:

1. class EzApp{...}
2. EzApp.annotations = [new Component({selector:"ez-app"})];

很显然，*注解*可以看做编译器（traceur）层面的*语法糖*，但和python的*装饰器*不同， *注解*在编译时仅仅被放在*annotation*里，编译器并不进行解释展开 - 这个解释的工作是 Angular2完成的：



据称，*注解*的功能就是Angular2团队向traceur团队提出的，这不是traceur的默认选项， 因此你看到，我们配置systemjs在使用traceur模块时*打开注解*：

1. System.config({
2. map:{traceur:"lib/traceur"},
3. traceurOptions: {annotations: true}
4. });

修改示例代码中的EzApp组件，不用注解实现同样的功能。

### 小结

如果你了解一点Angular1.x的bootstrap，可能隐约会感受到Angular2中bootstrap的一些 变化 - 我指的并非代码形式上的变化。

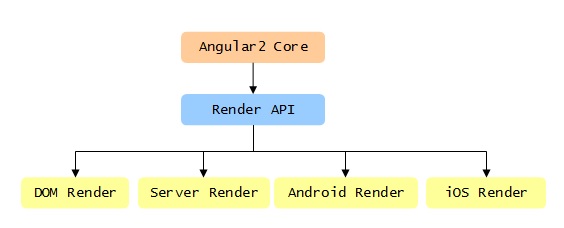
**以组件为核心**

在Angular1.x中，bootstrap是围绕DOM元素展开的，无论你使用ng-app还是手动执行bootstrap() 函数，自举过程是建立在DOM之上的。

而在Angular2中，bootstrap是围绕组件开始的，你定义一个组件，然后启动它。如果没有一个组件， 你甚至都没有办法使用Angular2！

**支持多种渲染引擎**

以组件而非DOM为核心，意味着Angular2在内核隔离了对DOM的依赖 - DOM仅仅作为一种可选的渲染引擎存在：



上面的图中，DOM Render已经实现，Server Render正在测试，iOS Render和Android Render 是可预料的特性，虽然我们看不到时间表。

这有点像React了。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>hello,angular2</title>

<!--模块加载器-->

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<!--Angular2模块库-->

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script>

//设置模块加载规则

System.baseURL = document.baseURI;

System.config({

map:{traceur:"lib/traceur"},

traceurOptions: {annotations: true}

});

</script>

</head>

<body>

<!--组件渲染锚点-->

<my-app></my-app>

<!--定义一个ES6脚本元素-->

<script type="module">

//从模块库引入三个类型定义

import {Component,View,bootstrap} from "angular2/angular2";

//组件定义

@Component({selector:"my-app"})

@View({template:"<h1>Angular2 - 以组件为基石</h1>"})

class EzApp{}

//渲染组件

bootstrap(EzApp);

</script>

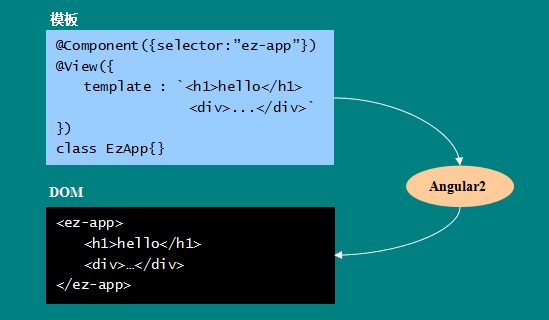
</body>

</html>

最简单的模板

组件的*View注解*用来声明组件的*外观*，它最重要的属性就是*template* - 模板。 Angular2的模板是*兼容*HTML语法的，这意味着你可以使用任何标准的HTML标签编写 组件模板。

所以，在最简单的情况下，一个Angular2组件的模板由*标准的HTML元素*构成，看起来就是 一段HTML码流。Angular2将*原封不同*地渲染这段模板：



有两种方法为组件指定渲染模板：

**1. 内联模板**

可以使用组件的*View注解*中的*template*属性直接指定*内联模板*：

1. @View({
2. template : `<h1>hello</h1>
3. <div>...</div>`
4. })

在ES6中，使用一对*`*符号就可以定义多行字符串，这使得编写内联的模板轻松多了。

**2. 外部模板**

也可以将模板写入一个单独的文件：

1. <!--ezcomp-tpl.html-->
2. <h1>hello</h1>
3. <div>...</div>

然后在定义组件时，使用*templateUrl*引用*外部模板*：

1. @View({
2. templateUrl : "ezcomp-tpl.html"
3. })

在示例的模板中，增加一个输入文本框和一个按钮！

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template - standard HTML</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

@Component({selector : "ez-app"})

@View({

template : `

<h1>Hello,Angular2</h1>

<p>

使用ES6开发Angular2应用的一个好处就是，可以不用拼接模板字符串了。

</p>

<ul>

<li>在模板中可以使用任何标准的HTML元素</li>

<li>如果模板定义在单独的文件里，可以使用templateUrl引入</li>

</ul>

`

})

class EzApp{}

bootstrap(EzApp);

</script>

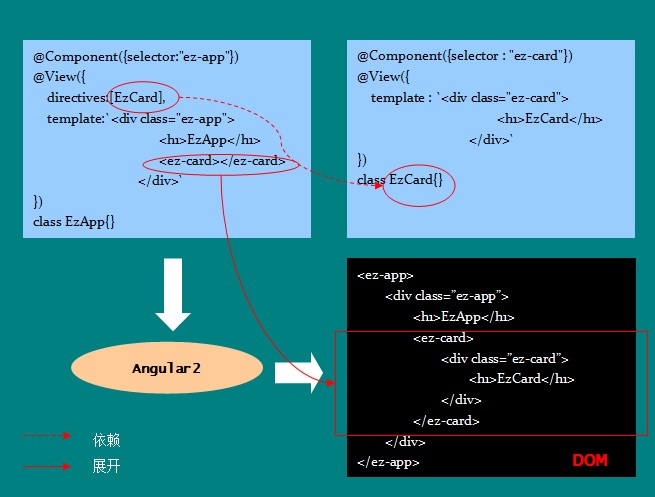
</body>

</html>

directives - 使用组件

在Angular2中，一个组件的模板内除了可以使用标准的HTML元素，也可以使用自定义的组件！

这是相当重要的特性，意味着Angular2将*无偏差*地对待标准的HTML元素和你自己定义的组件。这样， 你可以建立自己的*领域建模语言*了，这使得渲染模板和视图模型的对齐更加容易，也使得模板的语义性 更强：



**声明要在模板中使用的组件**

不过，在使用自定义组件之前，*必需*在组件的ViewAnnotation中通过*directives*属性声明这个组件：

1. @View({
2. directives : [EzComp],
3. template : "<ez-comp></ez-comp>"
4. })

你应该注意到了，*directives*属性的值是一个*数组*，这意味着，你需要在这里声明*所有*你需要在模板 中使用的自定义组件。

修改示例代码：  
1. 增加一个EzLogo组件  
2. 在EzCard组件的模板中使用这个组件

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template - component </title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

directives:[EzCard],

template:`

<div class="ez-app">

<h1>EzApp</h1>

<ez-card></ez-card>

</div>`

})

class EzApp{}

@Component({selector : "ez-card"})

@View({

template : `

<div class="ez-card">

<h1>EzCard</h1>

</div>`

})

class EzCard{}

bootstrap(EzApp);

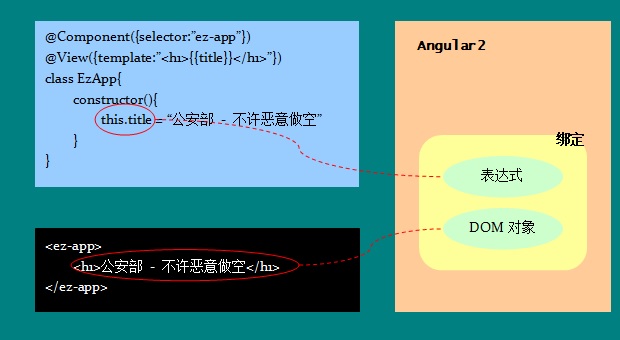
</script>

</body>

</html>

### {{model}} - 文本插值

在模板中使用可以{{表达式}}的方式绑定组件模型中的表达式，当表达式变化时， Angular2将自动更新对应的DOM对象：



上图的示例中，模板声明了h1的内容将绑定到组件实例的title变量。Angular2 框架将实时检测title的变化，并在其变化时自动更新DOM树中h1的内容。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template - bind model</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

template:`

<div>

<h1>{{title}}</h1>

<div>

<span>{{date}}</span> 来源：<span>{{source}}</span>

</div>

<p>{{content}}</p>

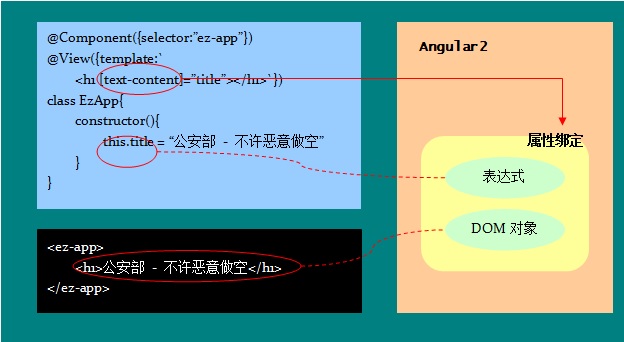
</div>

`

})

### [property] - 绑定属性

在模板中，也可以使用一对中括号将HTML元素或组件的属性绑定到组件模型的某个表达式， 当表达式的值变化时，对应的DOM对象将自动得到更新：



等价的，你也可以使用bind-前缀进行属性绑定：

1. @View({template:`<h1 bind-text-content="title"></h1>`})

很容易理解，通过属性，应用相关的数据流入组件，并影响组件的外观与行为。

需要注意的是，属性的值总是被当做调用者模型中的表达式进行绑定，当表达式变化时，被 调用的组件自动得到更新。如果希望将属性绑定到一个常量字符串，别忘了给字符串加引号，或者， 去掉中括号：

1. //错误，Angular2将找不到表达式 Hello,Angular2
2. @View({template:`<h1 [text-content]="Hello,Angular2"></h1>`})
3. //正确，Angular2识别出常量字符串表达式 'Hello,Angular2'
4. @View({template:`<h1 [text-content]="'Hello,Angular2'"></h1>`})
5. //正确，Angular2识别出常量字符串作为属性textContent的值
6. @View({template:`<h1 text-content="Hello,Angular2"></h1>`})

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template - bind propery</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {bind,Component,View,bootstrap} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

template:`<h1 [style.color]="color">Hello,Angular2</h1>`

})

class EzApp{

constructor(){

this.color = "red";

}

}

bootstrap(EzApp);

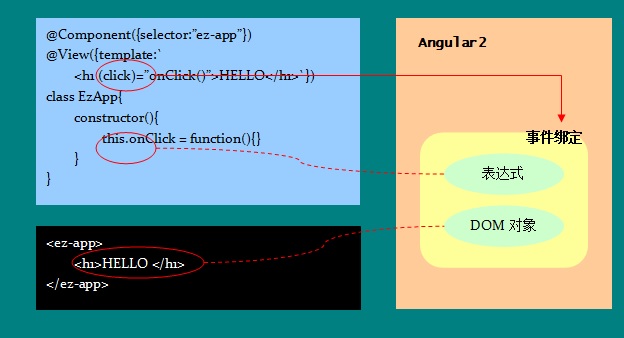
</script>

</body>

</html>

### (event) - 监听事件

在模板中为元素添加事件监听很简单，使用一对小括号包裹事件名称，并绑定 到表达式即可：



上面的代码实例为DOM对象h1的click事件添加监听函数onClick()。

另一种等效的书写方法是在事件名称前加on-前缀：

1. @View({template : `<h1 on-click="onClick()">HELLO</h1>`})

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template - bind propery</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

template:`

<h1>Your turn! <b>{{sb}}</b></h1>

<button (click)="roulette()">ROULETTE</button>

`

})

class EzApp{

constructor(){

this.names = ["Jason","Mary","Linda","Lincoln","Albert","Jimmy"];

this.roulette();

}

//轮盘赌

roulette(){

var idx = parseInt(Math.random()\*this.names.length);

this.sb = this.names[idx];

}

}

bootstrap(EzApp);

</script>

</body>

</html>

### #var - 局部变量

有时模板中的不同元素间可能需要互相调用，Angular2提供一种简单的语法将元素 映射为局部变量：添加一个以#或var-开始的属性，后续的部分表示变量名， 这个变量对应元素的实例。

在下面的代码示例中，我们为元素h1定义了一个局部变量v\_h1，这个变量指向 该元素对应的DOM对象，你可以在模板中的其他地方调用其方法和属性：

1. @View({
2. template : `
3. <h1 #v\_h1>hello</h1>
4. <button (click) = "#v\_h1.textContent = 'HELLO'">test</button>
5. `
6. })

如果在一个组件元素上定义局部变量，那么其对应的对象为组件的实例：

1. @View({
2. directives:[EzCalc],
3. template : "<ez-calc #c></ez-calc>"
4. })

在上面的示例中，模板内的局部变量c指向EzCalc的实例

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template - local var</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

template:`

<h1>

<button>变色</button>

I choose

<b #v\_who>WHO?</b>

</h1>

<button (click)="v\_who.textContent = 'Jason'">Jason</button>

<button (click)="v\_who.textContent = 'Mary'">Mary</button>

<button (click)="v\_who.textContent = 'Linda'">Linda</button>

<button (click)="v\_who.textContent = 'Lincoln'">Lincoln</button>

<button (click)="v\_who.textContent = 'Jimmy'">Jimmy</button>

<button (click)="v\_who.textContent = 'Albert'">Albert</button>

`

})

class EzApp{}

bootstrap(EzApp);

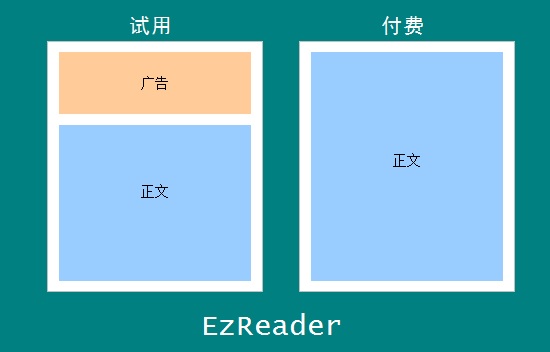
</script>

</body>

</html>

使用条件逻辑

有时我们需要模板的一部分内容在*满足一定条件*时才显示，比如右边示例中的*EzReader*组件， 对于试用用户，它将在正文之上额外显示一个广告：



这是指令*NgIf*发挥作用的场景，它评估属性*ngIf*的值是否为真，来决定是否渲染 *template*元素的内容：

1. @View({
2. template : `<!--根据变量trial的值决定是否显示广告图片-->
3. <template [ng-if]="trial==true">
4. <img src="ad.jpg">
5. </template>
6. <!--以下是正文-->
7. <pre>...

` })

Angular2同时提供了两种*语法糖*，让*NgIf*写起来更简单，下面的两种书写方法和上面 的正式语法是等效的：

1. //使用template attribute
2. <img src="ad.jpg" template="ng-if tiral==true">
3. //使用\*前缀
4. <img src="ad.jpg" \*ng-if="tiral==true">

看起来，显然*\*ng-if*的书写方法更加有人情味儿，不过无论采用哪种书写方法，都将转换 成上面的正式写法，再进行编译。

需要指出的是，*NgIf*是Angular2预置的*指令/Directive*，所以在使用之前，需要：

1. 从angular2库中*引入*NgIf类型定义
2. 在组件的ViewAnnotation中通过属性*directives*声明对该指令的引用

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Interpolation</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

//引入NgIf类型

import {Component,View,bootstrap,NgIf} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

directives:[EzReader],

template:`

<ez-reader [trial]="true"></ez-reader>

`

})

class EzApp{}

@Component({

selector : "ez-reader",

properties:["trial"]

})

@View({

directives:[NgIf],

template : `

<img [src]="banner" \*ng-if="trial==true">

<pre>{{content}}</pre>`

})

class EzReader{

constructor(){

var self = this;

this.\_trial = true;

this.banner = "img/banner.jpg";

this.content = `

“没关系，我已经没有放射性了。”

`;

}

}

bootstrap(EzApp);

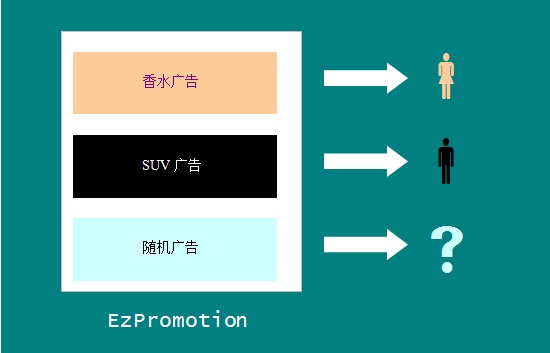
</script>

</body>

</html>

使用分支逻辑

如果组件的模板需要根据某个表达式的不同取值展示不同的片段，可以使用*NgSwitch*系列指令 来动态切分模板。比如右边示例中的广告组件*EzPromotion*，需要根据来访者*性别*的不同推送 不同的广告：



*NgSwitch*包含一组指令，用来构造包含多分支的模板：

**NgSwitch**

*NgSwitch*指令可以应用在任何HTML元素上，它评估元素的*ngSwitch*属性值，并根据这个值 决定应用哪些*template*的内容（可以同时显示多个分支）：

1. <ANY [ng-switch]="expression">...</ANY>

**NgSwitchWhen**

*NgSwitchWhen*指令必须应用在*NgSwitch*指令的子*template*元素上，它通过属性*ngSwitchWhen*指定一个表达式， 如果该表达式与父节点的*NgSwitch*指令指定的表达式值一致，那么显示这个*template*的内容：

1. <ANY [ng-switch]="...">
2. <!--与变量比较-->
3. <template [ng-switch-when]="variable">...</template>
4. <!--与常量比较-->
5. <template ng-switch-when="constant">...</template>
6. </ANY>

**NgSwitchDefault**

*NgSwitchDefault*指令必须应用在*NgSwitch*指令的子*template*元素上，当没有*NgSwitchWhen*指令匹配 时，*NgSwitch*将显示这个*template*的内容:

1. <ANY [ng-switch]="...">
2. <template ng-switch-default>...</template>
3. </ANY>

需要注意的是，*NgSwitch*系列指令都是Angualr2的预置指令，在模板中使用之前，需要

1. 从Angular2库中引入NgSwitch系列指令
2. 通过ViewAnnotation的*directives*属性进行声明

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Interpolation</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

//引入NgSwitch类型

import {Component,View,bootstrap,NgSwitch,NgSwitchWhen,NgSwitchDefault} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

directives:[EzPromotion],

template:`

<ez-promotion gender="Male"></ez-promotion>

`

})

class EzApp{}

@Component({

selector : "ez-promotion",

properties:["gender"]

})

@View({

directives:[NgSwitch,NgSwitchWhen,NgSwitchDefault],

template : `

<div [ng-switch]="gender">

<template ng-switch-when="Male">

<img src="img/male-ad.jpg">

</template>

<template ng-switch-when="Female">

<img src="img/female-ad.jpg">

</template>

<template ng-switch-default>

<h1>Learn Something, NOW!</h1>

</template>

</div>

`

})

class EzPromotion{}

bootstrap(EzApp);

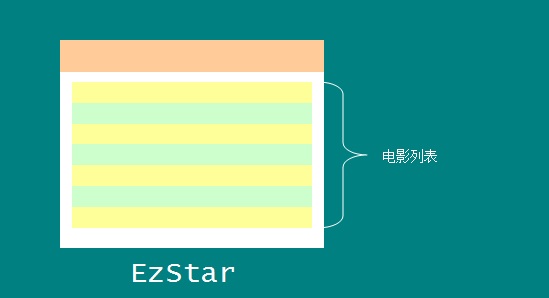
</script>

</body>

</html>

### NgFor- 循环逻辑

如果希望利用一组可遍历的数据动态构造模板，那么应当使用NgFor指令。 例如右边示例中的EzStar组件，用来展示演员的作品列表：



**迭代**

NgFor指令应用在template元素上，对ngForOf属性指定的数据集中的每一项 实例化一个template的内容：

1. <template ng-for [ng-for-of]="items" >
2. <li>----------</li>
3. </template>

如果items数据集有3条记录，那么会生成3个li对象，就像这样：

1. <li>----------</li>
2. <li>----------</li>
3. <li>----------</li>

不过这没多大用。

**使用数据项**

好在我们还可以为数据集的每一项声明一个局部变量，以便在模板内引用：

1. <template ng-for [ng-for-of]="items" #item>
2. <li>{{item}}</li>
3. </template>

假如items数据集是一个数组：["China","India","Russia"]，那么 现在生成的结果就有点用了：

1. <li>China</li>
2. <li>India</li>
3. <li>Russia</li>

**使用数据项索引**

有时还需要数据项在数据集中的索引，我们也可以为数据集的每一项的索引声明一个 局部变量，以便在模板内引用：

1. <template ng-for [ng-for-of]="items" #item #i="index">
2. <li>[{{i+1}}] {{item}}</li>
3. </template>

现在生成的结果更规矩了：

1. <li>[1] China</li>
2. <li>[2] India</li>
3. <li>[3] Russia</li>

**语法糖**

与NgIf类似，Angular2也为NgFor提供了两种语法糖：

1. //使用template attribute
2. <ANY template="ng-for #item of items;#i=index">...</ANY>
3. //使用\*前缀
4. <ANY \*ng-for="#item of items;#i=index">...</ANY>

毫无疑问，应当尽量使用\*ng-for的简便写法，这可以提高模板的可读性。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>NgFor</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

//引入NgSwitch类型

import {Component,View,bootstrap,NgFor} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

directives:[EzStar],

template:`

<ez-star></ez-star>

`

})

class EzApp{}

@Component({

selector : "ez-star"

})

@View({

directives:[NgFor],

template : `

<div>

<h2>{{actor}} - Films</h2>

<ul>

<li \*ng-for="#film of films">{{film.name}} / {{film.year}}</li>

</ul>

</div>

`

})

class EzStar{

constructor(){

this.actor = "Jason Statham";

this.films = [

{name: "Mechanic: Rescurrection" , year : "2016"},

{name: "Spy" , year : "2015"},

{name: "Furious" , year : "2015"}

];

}

}

bootstrap(EzApp);

</script>

</body>

</html>

### styles - 设置模板样式

组件既然处于UI层，就应当好看些，好看是构造良好用户体验的一部分。Angular2的 组件模板基于HTML，那么显然，我们需要通过样式表/CSS来调整组件的外观。

和模板类似，我们有两种方法为组件设置CSS样式：

**1. 内联样式**

可以使用组件View注解的styles属性来设置内联样式：

1. @View({
2. styles:[`
3. h1{background:#4dba6c;color:#fff}
4. `]
5. })

**2. 外部样式**

也可以把样式定义在单独的文件中：

1. /\*ez-greeting.css\*/
2. h1{background:#4dba6c;color:#fff}

然后使用View注解的styleUrls属性来引入外部样式：

1. @View({
2. styleUrls:["ez-greeting.css"]
3. })

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template- styles</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

@Component({selector : "ez-app"})

@View({

styles:[`

div.ez-greeting{font-family:Courier;background:#ede7f6;box-shadow:0 2px 5px 0;}

h1{background:#4dba6c;color:#fff}

div.content{padding:10px;}

`],

template : `

<div class="ez-greeting">

<h1>Hello,Angular2</h1>

<div class="content">

<p>

使用ES6开发Angular2应用的一个好处就是，可以不用拼接样式字符串了。

</p>

<ul>

<li>在样式中可以使用任何标准的CSS语法</li>

<li>如果样式定义在单独的文件里，可以使用styleUrls引入</li>

</ul>

</div>

</div>

`

})

class EzApp{}

bootstrap(EzApp);

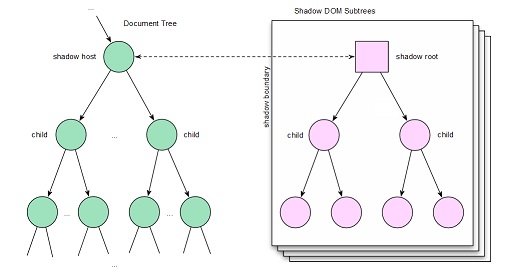
</script>

</body>

</html>

### ShadowDom - 封装私有样式

Angular2采用ShadowDom作为组件的渲染基础，这意味着组件被渲染到独立的 Shadow Tree上，这很好，可以实现DOM对象和样式的良好封装：



但问题是，除了Chrome之外的大多数的浏览器目前还不支持ShadowDom，因此，Angular2 提供了三种将模板渲染到DOM的策略。

以下面的模板为例，我们看三种策略下的渲染结果差异：

1. @View{
2. template:"<h1>hello</h1>",
3. styles:["h1{color:red}"]
4. }

**全局仿真策略/EmulatedUnscopedShadowDomStrategy**

采用这个策略时，Angular2将模板直接插入DOM树，并将模板的样式原封不动地插入head头部。这意味着 不同组件之间的样式可能冲突 : 在右边的示例代码中，你可以清楚的看到，EzApp组件的h1样式污染了其他的 h1元素，所有的h1都变成红色了。

示例代码渲染后的DOM如下：



这个策略不需要浏览器原生支持ShadowDom，是当前版本（alpha.28）的默认策略。

**作用域仿真策略/EmulatedScopedShadowDomStrategy**

采用这个策略时，Angular2将模板直接插入DOM树，对模板的样式重新定义CSS选择符后 插入head头部。由于样式进行了重命名，所以不同组件之间的样式不会冲突。

示例代码在这个策略下的渲染结果是：



这个策略也不需要浏览器原生支持ShadowDom。

**原生策略/NativeShadowDomStrategy**

采用这个策略时，Angular2将在组件的宿主DOM对象上建立一个ShadowDom树，这颗树与主DOM树是隔离 的，所以，这是实现Web组件的理想方案：



如果浏览器原生支持ShadowDom，那么应当使用这个策略。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template - shadowdom strategy</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<h1>我是H1，我在组件外</h1>

<ez-app></ez-app>

<script type="module">

import {bind,Component,View,bootstrap} from "angular2/angular2";

@Component({selector:"ez-app"})

@View({

template:"<h1>我是H1，我在组件内</h1>",

styles:["h1{color:red}"]

})

class EzApp{}

bootstrap(EzApp);

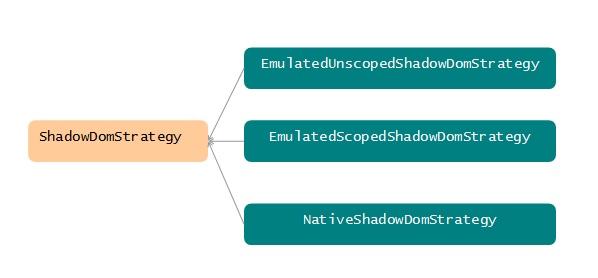
</script>

</body>

</html>

### 设置ShadowDom策略

在Angular2中，ShadowDom的三种策略对应于三个类，这三个类继承自ShadowDomStrategy：



Angular2的内核引用的是父类ShadowDomStrategy，我们只需要从三个继承类中选择 一个与之绑定就可以实现不同策略的选择。下面的示例意味着选择NativeShadowDomStrategy：

1. bind(ShadowDomStrategy).toFactory(function(doc){
2. return new NativeShadowDomStrategy(doc.head);
3. },[DOCUMENT\_TOKEN])

上面的表达式用来提交给Angular2的注入器/DI，可以理解为：如果注入器需要实例化一个ShadowDomStrategy 实例，应当以DOCUMENT\_TOKEN为参数，调用一个匿名的工厂函数，而这个工厂函数将返回一个 NativeShadowDomStrategy类的实例。

ES6支持lambda表达式，因此我们可以写的简单一些：

1. bind(ShadowDomStrategy).toFactory(doc => new NativeShadowDomStrategy(doc.head),[DOCUMENT\_TOKEN])

**补丁包**

在Angular2的alpha.28版本的官方发行包中，没有包含默认ShadowDom策略之外的另两种策略实现模块，因此 我们单独打了一个包render.dev.js。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>template - scoped&shadowdom strategy</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/render.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<h1>我是H1，我在组件外</h1>

<ez-app></ez-app>

<script type="module">

import {bind,Component,View,bootstrap} from "angular2/angular2";

import {ShadowDomStrategy} from 'angular2/src/render/dom/shadow\_dom/shadow\_dom\_strategy';

import {DOCUMENT\_TOKEN} from 'angular2/src/render/dom/dom\_renderer';

import {EmulatedUnscopedShadowDomStrategy} from 'angular2/src/render/dom/shadow\_dom/emulated\_unscoped\_shadow\_dom\_strategy';

import {EmulatedScopedShadowDomStrategy} from 'angular2/src/render/dom/shadow\_dom/emulated\_scoped\_shadow\_dom\_strategy';

import {NativeShadowDomStrategy} from 'angular2/src/render/dom/shadow\_dom/native\_shadow\_dom\_strategy';

@Component({selector:"ez-app"})

@View({

template:"<h1>我是H1，我在组件内</h1>",

styles:["h1{color:red}"]

})

class EzApp{}

var injectables = [

bind(ShadowDomStrategy)

.toFactory((doc) => new EmulatedScopedShadowDomStrategy(doc.head), [DOCUMENT\_TOKEN])

];

bootstrap(EzApp,injectables);

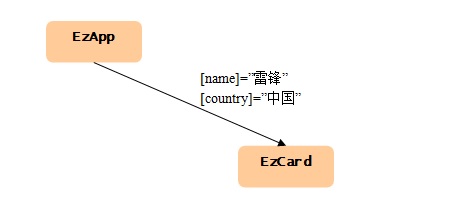
</script>

</body>

</html>

### 属性声明 - 暴露成员变量

属性是组件暴露给外部世界的调用接口，调用者通过设置不同的属性值来定制 组件的行为与外观：



在Angular2中为组件增加属性接口非常简单，只需要在Component注解的 properties属性中声明组件的成员变量就可以了：

1. //EzCard
2. @Component({
3. properties:["name","country"]
4. })

上面的代码将组件的成员变量name和country暴露为同名属性，这意味着在EzApp 的模板中，可以直接使用中括号语法来设置EzCard对象的属性：

1. //EzApp
2. @View({
3. directives : [EzCard],
4. template : "<ez-card [name]="'雷锋'" [country]="'中国'"></ez-card>"
5. })

**提醒**：如果要在模板中使用自定义的指令（组件是一种指令），必须在View注解的directives 属性中提前声明！

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Property</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

//根组件 - EzApp

@Component({selector:"ez-app"})

@View({

directives:[EzCard],

template:`

<div class="ez-app">

<h1>EzApp</h1>

<ez-card></ez-card>

</div>`

})

class EzApp{}

//具有属性接口的组件 - EzCard

@Component({

selector:"ez-card",

properties:["name","country"]

})

@View({

template : `<div class='ez-card'>

My name is <b>{{name}}</b>,

I am from <b>{{country}}</b>.</div>`

})

class EzCard{

constructor(){

this.name = "Mike";

this.country = "Sweden";

}

}

//渲染组件

bootstrap(EzApp);

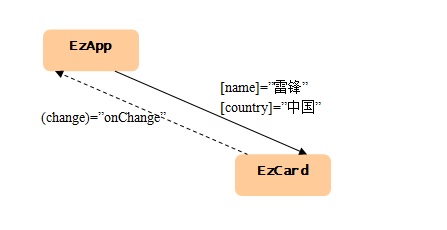
</script>

</body>

</html>

### 事件声明 - 暴露事件源

与属性相反，事件从组件的内部流出，用来通知外部世界发生了一些事情：



在Angular2中为组件增加事件接口也非常简单：定义一个事件源/EventEmitter， 然后通过Component注解的events接口包括出来：

1. //EzCard
2. @Component({
3. events:["change"]
4. })
5. class EzCard{
6. constructor(){
7. this.change = new EventEmitter();
8. }
9. }

上面的代码将组件EzCard的事件源change暴露为同名事件，这意味着在调用者 EzApp组件的模板中，可以直接使用小括号语法挂接事件监听函数：

1. //EzApp
2. @View({
3. template : "<ez-card (change)="onChange()"></ez-card>"
4. })

每次EzCard触发change事件时，EzApp的onChange()方法都将被调用。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>hello,angular2</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap,EventEmitter} from "angular2/angular2";

//根组件 - EzApp

@Component({selector:"ez-app"})

@View({

directives:[EzCard],

template:`

<div class="ez-app">

<h1>EzApp</h1>

<ez-card (change)="onChange($event)"></ez-card>

<pre>{{evtStr}}</pre>

</div>`

})

class EzApp{

constructor(){

this.evtStr

}

onChange(evt){

console.log("sth. occured");

this.evtStr = JSON.stringify(evt,null,"\t");

}

}

//具有事件接口的组件 - EzCard

@Component({

selector:"ez-card",

events:["change"]

})

@View({

template : `<div class='ez-card'>

My name is <b>{{name}}</b>,

I am from <b>{{country}}</b>.</div>`

})

class EzCard{

constructor(){

this.name = "Mike";

this.country = "Sweden";

this.change = new EventEmitter();

//模拟触发事件

setTimeout(()=>this.change.next({

src:"EzCard",

desc:"模拟事件"

}),1000);

}

}

//渲染组件

bootstrap(EzApp);

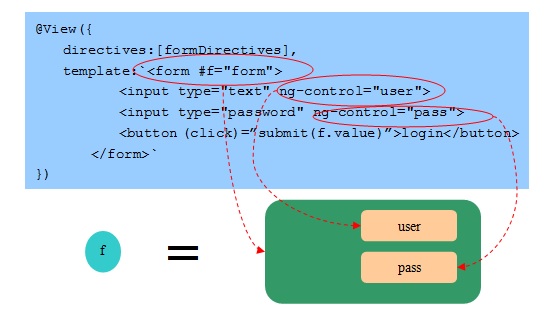
</script>

</body>

</html>

### NgForm - 表单指令

NgForm指令为表单元素/form建立一个控件组对象，作为控件的容器； 而NgControlName指令为则为宿主input元素建立一个控件对象，并将该控件加入到NgForm 指令建立的控件组中：



**局部变量**

通过使用#符号，我们创建了一个引用控件组对象（注意，不是form元素！）的局部变量f。 这个变量最大的作用是：它的value属性是一个简单的JSON对象，键对应于input元素的 ng-control属性，值对应于input元素的值：



**声明指令依赖**

NgForm指令和NgControlName指令都包含在预定义的数组变量formDirectives中，所以我们在 组件注解的directives属性中直接声明formDirectives就可以在模板中直接使用这些指令了：

1. //angular2/ts/src/forms/directives.ts
2. export const formDirectives = CONST\_EXPR([
3. NgControlName,
4. NgControlGroup,
6. NgFormControl,
7. NgModel,
8. NgFormModel,
9. NgForm,
11. NgSelectOption,
12. DefaultValueAccessor,
13. CheckboxControlValueAccessor,
14. SelectControlValueAccessor,
16. NgRequiredValidator
17. ]);

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>NgForm</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap,NgIf} from "angular2/angular2";

//引入form指令集

import {formDirectives} from "angular2/forms";

//EzApp组件

@Component({selector:"ez-app"})

@View({

directives:[formDirectives,NgIf],

template:`

<form #f="form" (submit)="search(f.value)">

<select>

<option value="web">网页</option>

<option value="news">新闻</option>

<option value="image">图片</option>

</select>

<input type="text" ng-control="kw">

<button type="submit">搜索</button>

</form>

<!--给个简单的反馈-->

<h1 \*ng-if="kw!=''">正在搜索 {{kw}} ...</h1>

`,

styles:[`form{background:#90a4ae;padding:5px;}`]

})

class EzApp{

constructor(){

this.kw = "";

}

search(val){

this.kw = val.kw;

//假装在搜索，2秒钟返回

setTimeout(()=>this.kw="",2000);

}

}

bootstrap(EzApp);

</script>

</body>

</html>

### NgControlName - 命名控件指令

如前所述，NgControlName指令必须作为NgForm或NgFormModel的后代使用， 因为这个指令需要将创建的控件对象添加到祖先（NgForm或NgFormModel）所创建 的控件组中。

NgControlName指令的选择符是[ng-control]，这意味着你必须在一个HTML元素上 定义ng-control属性，这个指令才会起作用。

**属性：ngControl**

NgControlName指令为宿主的DOM对象创建一个控件对象，并将这个对象以ngControl属性 指定的名称绑定到DOM对象上：

1. <form #f="form">
2. <input type="text" ng-control="user">
3. <input type="password" ng-control="pass">
4. </form>

在上面的代码中，将创建两个Control对象，名称分别为user和pass。

**属性/方法：ngModel**

除了使用控件组获得输入值，NgControlName指令可以通过ngModel实现模型 与表单的双向绑定：

1. <form>
2. <input type="text" ng-control="user" [(ng-model)]="data.user">
3. <input type="password" ng-control="pass" [(ng-model)]="data.pass">
4. </form>`

ngModel即是NgControlName指令的属性，也是它的事件，所以下面 的两种写法是等价的：

1. <input type="text" ng-control="user" [(ng-model)]="data.user">
2. //等价于
3. <input type="text" ng-control="user" [ng-model]="data.user" (ng-model)="data.user">

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>NgControlName</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap,NgIf} from "angular2/angular2";

import {formDirectives} from "angular2/forms";

@Component({selector:"ez-app"})

@View({

directives:[NgIf,formDirectives],

template:`

<form>

<ul>

<li>姓名：<input type="text" ng-control="name"></li>

<li>姓别：

<select ng-control="gender" [(ng-model)]="data.gender">

<option value="Male">男</optoin>

<option value="Female">女</optoin>

</select>

</li>

<li>地址：<input type="text" ng-control="address" [ng-model]="data.address"></li>

<li>电话：<input type="text" ng-control="telephone" (ng-model)="data.telephone"></li>

<li>已婚：<input type="checkbox" ng-control="marriage" [(ng-model)]="data.marriage"></li>

</ul>

</form>

<pre>{{decode(data)}}</pre>

`,

styles:[`

form{background:#e1f5fe;}

ul{list-style:none;padding:10px;margin:0px;}

li{line-height:30px;}

`]

})

class EzApp{

constructor(){

this.data = {

name : "whoami"

};

}

decode(val){

return JSON.stringify(val,null,"\t");

}

}

bootstrap(EzApp);

</script>

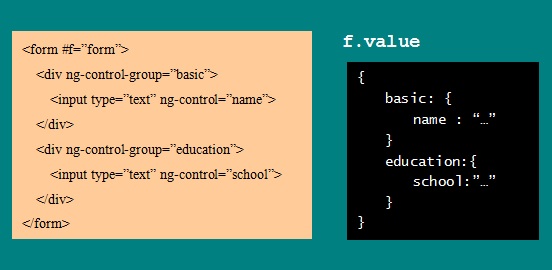
</body>

</html>

### NgCongrolGroup - 命名控件组

NgControlGroup指令的选择符是[ng-control-group]，如果模板中的某个元素具有这个属性， Angular2框架将自动创建一个控件组对象，并将这个对象以指定的名称与DOM对象绑定。

控件组可以嵌套，方便我们在语义上区分不同性质的输入：



和NgControlName指令一样，NgControlGroup指令也必须作为NgForm或NgFormModel的 后代使用，因为这个指令需要将创建的控件组对象添加到祖先（NgForm或NgFormModel）所创建 的控件组中。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>NgControlGroup</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap,NgIf} from "angular2/angular2";

import {formDirectives} from "angular2/forms";

@Component({selector:"ez-app"})

@View({

directives:[NgIf,formDirectives],

template:`

<form #f="form">

<div>基本信息</div>

<!--声明控件组-->

<ul ng-control-group="basic">

<li>姓名：<input type="text" ng-control="name"></li>

<li>地址：<input type="text" ng-control="address"></li>

<li>电话：<input type="text" ng-control="telephone"></li>

</ul>

<div>专业技能</div>

<!--声明控件组-->

<ul ng-control-group="expertise">

<li>英语：<input type="checkbox" ng-control="english"></li>

<li>科技：<input type="checkbox" ng-control="tech"></li>

<li>运动：<input type="checkbox" ng-control="sport"></li>

</ul>

</form>

<!--调试：实时转储模型的值-->

<pre>{{decode(f.value)}}</pre>

`,

styles:[`

div{padding:5px;background:#b3e5fc;color:red;}

form{background:#e1f5fe;}

ul{list-style:none;padding:5px;margin:0px;}

li{line-height:30px;}

`]

})

class EzApp{

decode(val){

return JSON.stringify(val,null,"\t");

}

}

bootstrap(EzApp);

</script>

</body>

</html>

### NgFormControl - 绑定已有控件对象

与NgControlName指令不同，NgFormControl将已有的控件/Control对象绑定到DOM元素 上。当需要对输入的值进行初始化时，可以使用NgFormControl指令。

下面的代码中，使用NgFormControl指令将DOM元素绑定到组件EzComp的成员 变量movie上，我们需要在构造函数中先创建这个Control对象：

1. @View({
2. //将输入元素绑定到已经创建的控件对象上
3. template : `<input type="text" [ng-form-control]="movie">`
4. })
5. class EzComp{
6. constructor(){
7. //创建控件对象
8. this.movie = new Control("Matrix II - Reload");
9. }
10. }

控件/Control是Angular2中对表单输入元素的抽象，我们使用其value属性，就可以获得对应的 输入元素的值。

与NgControlName指令的另一个区别是，NgFormControl不需要NgForm或NgFormModel的祖先。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>NgFor</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

import {Control,formDirectives} from "angular2/forms";

@Component({selector:"ez-app"})

@View({

directives:[formDirectives],

template:`

<div>

<ul>

<!--将输入元素绑定到已经创建的控件对象-->

<li>姓名：<input type="text" [ng-form-control]="name"></li>

<li>地址：<input type="text" [ng-form-control]="address"></li>

<li>电话：<input type="text" [ng-form-control]="telephone"></li>

</ul>

</div>

<!--调试：转储模型信息-->

<pre>{{dump()}}</pre>

`,

styles:[`

form{background:#e1f5fe;}

ul{list-style:none;padding:10px;margin:0px;}

li{line-height:30px;}

`]

})

class EzApp{

constructor(){

//创建控件对象

this.name = new Control("Jason");

this.address = new Control("London U.K.");

this.telephone = new Control("114");

}

dump(){

//读取控件对象的值

var val = {

name : this.name.value,

address : this.address.value,

telephone : this.telephone.value

}

return JSON.stringify(val,null,"\t");

}

}

bootstrap(EzApp);

</script>

</body>

</html>

### NgFormModel - 绑定已有控件组

NgFormModel指令类似于NgControlGroup指令，都是为控件提供容器。但区别在于， NgFormModel指令将已有的控件组绑定到DOM对象上：

1. @View({
2. template : `
3. <!--绑定控件组与控件对象-->
4. <div [ng-form-model]="controls">
5. <input type="text" ng-control="name">
6. <input type="text" ng-control="age">
7. </div>`
8. })
9. class EzComp{
10. constructor(){
11. //创建控件组及控件对象
12. this.controls = new ControlGroup({
13. name :new Control("Jason"),
14. age : new Control("45")
15. });
16. }
17. }

NgFormModel指令可以包含NgControlGroup指令，以便将不同性质的输入分组。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>NgFor</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Inject,Component,View,bootstrap} from "angular2/angular2";

import {Control,ControlGroup,formDirectives} from "angular2/forms";

@Component({selector:"ez-app"})

@View({

directives:[formDirectives],

template:`

<div [ng-form-model]="controls">

<ul>

<li>姓名：<input type="text" ng-control="name"></li>

<li>地址：<input type="text" ng-control="address"></li>

<li>电话：<input type="text" ng-control="telephone"></li>

</ul>

</div>

<pre>{{dump()}}</pre>

`,

styles:[`

div{background:#e1f5fe;}

ul{list-style:none;padding:10px;margin:0px;}

li{line-height:30px;}

`]

})

class EzApp{

constructor(){

this.controls = new ControlGroup({

name : new Control("Jason"),

address : new Control("London U.K."),

telephone : new Control("114")

});

}

dump(){

return JSON.stringify(this.controls.value,null,"\t");

}

}

bootstrap(EzApp);

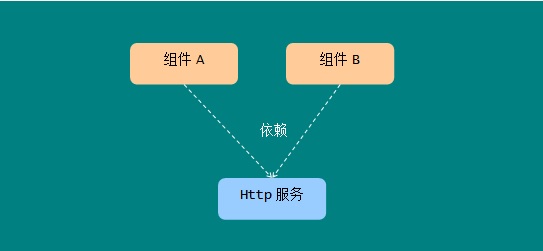
</script>

</body>

</html>

### 服务 - 封装可复用代码

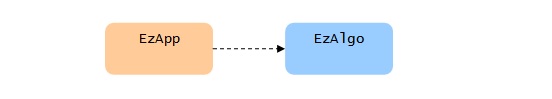
在Angular2中，服务用来封装可复用的功能性代码。比如Http服务，封装了ajax 请求的细节，在不同的组件中，我们只需要调用Http服务的API接口就可以给组件增加 ajax请求的功能了：



Angular2中实现一个服务非常简单直接 ： 定义一个类，然后，它就是服务了：

1. class EzAlgo{
2. add(a,b){return a+b;}
3. sub(a,b){return a-b;}
4. }

上面的代码定义了一个相当弱智的算法服务EzAlgo，它有两个API - add()用来 计算两个数的和,sub()用来计算两个数的差 。在示例中，组件EzApp依赖于这个 服务：



<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Service</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Component,View,bootstrap} from "angular2/angular2";

import {formDirectives} from "angular2/forms";

//定义一个简单的算法服务

class EzAlgo{

add(a,b) { return a+b; }

sub(a,b) { return a-b; }

}

//组件定义

@Component({

selector : "ez-app"

})

@View({

directives:[formDirectives],

template : `

<form>

<input type="text" ng-control="a" [(ng-model)]="a">

+

<input type="text" ng-control="b" [(ng-model)]="b">

=

{{add()}}

</form>`

})

class EzApp{

constructor(){

this.a = 37;

this.b = 128;

//实例化服务对象

this.algo = new EzAlgo();

}

add(){

var a = +this.a,

b = +this.b;

return this.algo.add(a,b);

}

}

bootstrap(EzApp);

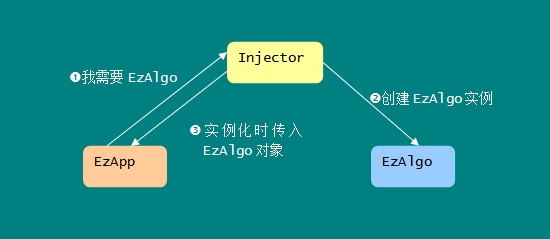
</script>

</body>

</html>

### 注入 - appInjector

在前一节的示例代码中，组件EzAlgo直接在构造函数中实例化了一个EzAlog对象， 这造成了EzApp和EzAlgo的强耦合，我们可以使用Angular2的注入器/Injector进行 解耦：



注入器就像婚姻介绍所，男方在婚介所登记心仪的女性特点，约好见面地点，然后， 坐等发货即可。比如上图：

EzApp组件（男方）使用Component注解的appInjector属性向Angular2框架（婚介所）声明 其依赖于EzAlgo（登记心仪的女性特点），并在其构造函数的参数表中使用Inject注解声明 注入点（约好见面地点），然后，剩下的事儿Angular2（婚介所）就办了：

1. @Component({
2. selector : "ez-app",
3. //声明依赖
4. appInjector : [EzAlgo]
5. })
6. @View(...)
7. class EzApp{
8. //Angular2框架负责注入对象
9. constructor(@Inject(EzAlgo) algo){
10. //已经获得EzAlgo实例了！
11. }
12. }

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>appInjector</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Inject,Component,View,bootstrap} from "angular2/angular2";

import {formDirectives,Control} from "angular2/forms";

//定义一个简单的算法服务类

class EzAlgo{

add(a,b) { return a+b; }

sub(a,b) { return a-b; }

}

//组件定义

@Component({

selector : "ez-app",

appInjector : [EzAlgo]

})

@View({

directives:[formDirectives],

template : `

<form>

<input type="text" ng-control="a" [(ng-model)]="a">

+

<input type="text" ng-control="b" [(ng-model)]="b">

=

{{add()}}

</form>`,

styles:[`

\*{font-size:30px;font-weight:bold;}

input{width:100px;}

`]

})

class EzApp{

//注入参数声明

constructor(@Inject(EzAlgo) algo){

this.a = 37;

this.b = 128;

this.algo = algo;

}

add(){

var a = +this.a,

b = +this.b;

return this.algo.add(a,b);

}

}

bootstrap(EzApp);

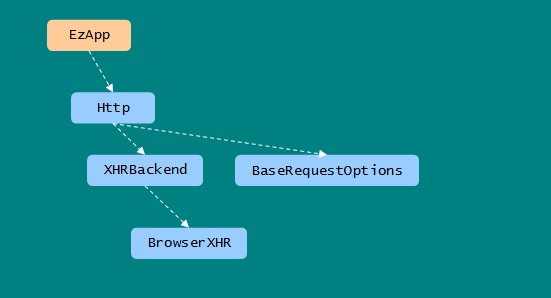
</script>

</body>

</html>

### 注入一个复杂的服务

EzAlgo相当简单，使用new或者使用Injector来获得一个实例看起来差别不大。那如果我们 的EzApp组件要使用Http服务呢？



第一眼看上去，Http服务显然是一个真正有用的服务 - 因为看起来相当的复杂 - Http依赖于XHRBackend和BaseRequestOptions，而XHRBackend又依赖于BrowserXHR。

我们可以有两种方法获得一个Http的实例，以便通过它获得网络访问的功能：

**1. 使用new进行实例化**

如果我们使用传统的new方式创建Http实例，看起来应该是这样：

1. var xhrbe = new XHRBackend(BrowserXHR);
2. var options = new BaseRequestOptions();
3. var http = new Http(xhrbe,options);

这没什么新奇的，就是繁琐一点。

**2. 使用注入器/Injector**

如果使用注入器，我们需要向Angular2框架声明这些层叠的依赖关系：

1. @Component({
2. appInjector : [
3. bind(BrowserXHR).toValue(BrowserXHR),
4. XHRBackend,
5. BaseRequestOptions,
6. Http
7. ]
8. })

bind(BrowserXHR).toValue(BrowserXHR)的意思是，如果需要注入BrowserXHR类型的变量，注入 这个类本身而非其实例。

原理是这样，不过Angular2已经提供了一个定义好的变量httpInjectables，我们直接引用就可以了。

**Observable**

Observable是一种异步编程模式，与Promise不同，Observable等多的是从数据而非行为的角度 来封装异步代码。

Http服务的get()方法返回一个Observable对象，可以把Observable对象看做一个可订阅的 数据流，你通过subscribe()方法订阅以后，每当数据流中有新的数据，你都会得到通知。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>appInjector</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Inject,Component,View,bootstrap,NgFor} from "angular2/angular2";

//引入Http相关预定义类型

import {Http,httpInjectables} from "angular2/http";

//EzApp组件

@Component({

selector:"ez-app",

//注入Http依赖项集合

appInjector:[httpInjectables]

})

@View({

directives:[NgFor],

template:`

<div \*ng-for="#album of band.albums"><img [src]="album.cover"></div>

`,

styles:[`

img{height:200px;width:200px;}

div{float:left;margin:10px;}

`]

})

class EzApp{

//注入Http实例对象

constructor(@Inject(Http) http){

this.band = {};

http.get("api/music.json")//GET请求

.map(rsp=>rsp.json())//将相应转化为JSON格式的数据集

.subscribe(data=>this.band=data[0]);//设置band变量为数据集的第一项

}

}

bootstrap(EzApp);

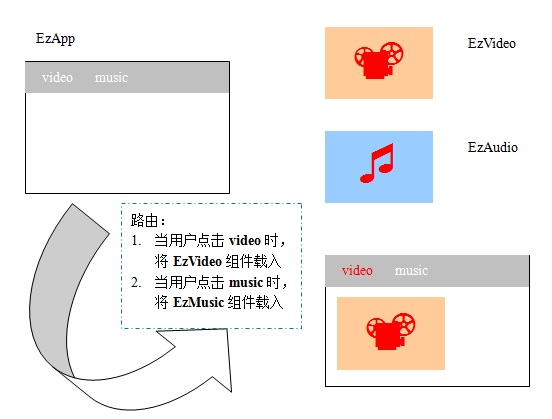
</script>

</body>

</html>

### 路由 - 初体验

一个Web应用通常需要切割为多个不同的组件进行实现，然后根据用户的交互行为（通常 是点击），动态地载入不同的组件，根据用户行为选择组件的过程就是路由：



由于Angular2是面向组件的，所以，Angular2的路由其实就是组件之间的路由。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Router</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<!--注册Angular2路由模块库-->

<script type="text/javascript" src="lib/router.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Inject,Component,View,bootstrap} from "angular2/angular2";

//引入路由相关类型定义

import {LocationStrategy,RouteConfig,RouterOutlet,RouterLink,Router,routerInjectables} from "angular2/router";

//EzApp组件 ： 路由配置与执行

@Component({selector:"ez-app"})

@View({

directives:[RouterOutlet,RouterLink],

template : `

<nav>

<!--声明路由入口-->

<b router-link="video">video</b> |

<b router-link="music">music</b>

</nav>

<main>

<!--声明路由出口-->

<router-outlet></router-outlet>

</main>

`

})

//路由配置

@RouteConfig([

{path:"/video", component:EzVideo,as:"video"},

{path:"/music", component:EzMusic,as:"music"}

])

class EzApp{

constructor(@Inject(LocationStrategy) ls){

ls.pushState = function(){};//simple hack for crash bug.

}

}

//EzVideo组件

@Component({selector:"ez-video"})

@View({

template : `

<h3>视频：轻松一刻</h3>

<embed allowscriptaccess="always" height="482" pluginspage="http://get.adobe.com/cn/flashplayer/" flashvars="list=http%3A%2F%2Fus.sinaimg.cn%2F002TFmqWjx06TNSxqGuz0504010000220k01.m3u8%3FKID%3Dunistore%2Cvideo%26Expires%3D1437020410%26ssig%3DNnoWLSjm2v&amp;fid=1034:882b42973162c7ae7acef23bfaccbe8c&amp;logo=2&amp;uid=1971237631" allowfullscreen="true" width="482" quality="high" src="http://js.t.sinajs.cn/t5/album/static/swf/video/player.swf?v1423545453000454353" type="application/x-shockwave-flash" wmode="transparent">

`

})

class EzVideo{}

//EzMusic组件

@Component({selector:"ez-music"})

@View({

template : `

<h3>音乐：平凡之路</h3>

<embed src="http://player.yinyuetai.com/video/player/2094298/a\_0.swf" quality="high" width="480" height="334" align="middle" allowScriptAccess="sameDomain" allowfullscreen="true" type="application/x-shockwave-flash"></embed> `

})

class EzMusic{}

//注入路由功能依赖项

bootstrap(EzApp,[routerInjectables]);

</script>

</body>

</html>

路由 - 应用步骤

在Angular2中使用路由的功能，有三个步骤：

**1. 配置路由**

为组件注入*Router*对象并通过*config()*方法配置路由：

1. router.config([
2. {path:"/video", component:EzVideo},
3. {path:"/music", component:EzMusic}])

上面的代码中，配置了两条路由：

* 如果用户请求路径为*/video*，那么将在路由出口中激活组件*EzVideo*
* 如果用户请求路径为*/music*，那么将在路由出口中激活组件*EzMusic*

**2. 设置路由出口**

*路由出口*是组件激活的地方，使用*RouterOutlet*指令在组件模板中声明出口：

1. @View({
2. directives:[RouterOutlet],
3. template : `<router-outlet></router-outlet>`
4. })
5. class EzApp{...}

**3. 执行路由**

使用*Router*的*navigate()*方法可以执行指定路径的路由，在示例中，当用户点击 时我们调用这个方法进行路由：

1. @View({
2. template : `
3. <span (click)="router.navigate('/video')">video</span> |
4. <span (click)="router.navigate('/music')">music</span>
5. <router-outlet></router-outlet>`
6. })

我们向*navigate()*方法传入的路径，就是我们通过*config()*方法配置的路径。这样， *Router*就根据这个*路径*，找到匹配的*组件*，在*RouterOutlet*上进行激活。

**在真正开始使用路由的功能之前，我们需要做一些准备工作：**

**1. 引用路由包**

Angular2的路由模块单独打包在*router.dev.js*，因此首先应该引用这个包：

1. <script type="text/javascript" src="lib/router.dev.js"></script>

**2. 引入路由相关的预定义类型**

Angular2的路由模块名为*angular2/router*，我们从这里引入常用类型：

1. import {LocationStrategy,Router,RouterOutlet,routerInjectables} from "angular2/router";

**3. 声明路由相关依赖类型**

在启动组件时，我们需要声明路由相关的依赖类型（即变量：routerInjectables），以便 根注入器可以解析对这些类型的请求：

1. bootstrap(EzApp,[routerInjectables]);

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Router - steps</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<!--引入Angular2路由模块库-->

<script type="text/javascript" src="lib/router.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Inject,Component,View,bootstrap} from "angular2/angular2";

import {LocationStrategy,RouterOutlet,Router,routerInjectables} from "angular2/router";

@Component({selector:"ez-app"})

@View({

directives:[RouterOutlet],

template : `

<nav>

<b (click)="go('/video')">video</b> |

<b (click)="go('/music')">music</b>

</nav>

<main>

<!--声明路由出口-->

<router-outlet></router-outlet>

</main>

`

})

class EzApp{

constructor(@Inject(Router) rt,@Inject(LocationStrategy) ls){

ls.pushState = function(){};

this.router = rt;

//配置路由

this.router.config([

{path:"/video", component:EzVideo},

{path:"/music", component:EzMusic}

]);

}

go(path){

//根据给定的url，选中组件并在outlet中激活

this.router.navigate(path);

}

}

@Component({selector:"ez-video"})

@View({

template : `

<h1>I LOVE THIS VIDEO!</h1>

`

})

class EzVideo{}

@Component({selector:"ez-music"})

@View({

template : `

<h1>THAT'S FANTASTIC MUSIC!</h1>

`

})

class EzMusic{}

bootstrap(EzApp,[routerInjectables]);

</script>

</body>

</html>

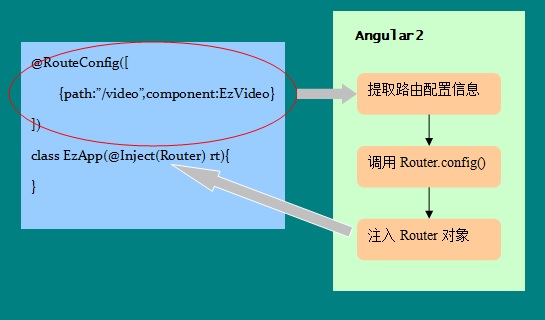
### RouteConfig - 路由配置注解

除了使用Router的config()方法进行路由配置，Angular2还提供了路由注解，允许我们 以注解的方式为组件添加路由：

1. @RouteConfig([
2. {path:"/video", component:EzVideo},
3. {path:"/music", component:EzMusic}
4. ])
5. class EzApp{...}

RouteConfigAnnotation的构造函数参数与Router的config()方法参数一致，都是一个包含 若干配置项的数组。

事实上，它的确只是一个语法糖 ： Angular2在bootstrap一个组件时，会检查组件是否存在 RouteConfig注解，如果存在，则利用这个信息调用Router的config()方法进行路由配置：



不过这样写起来，感觉会流畅一些，声明式的氛围更强烈一些。

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>RouteConfig</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<!--引入Angular2路由模块库-->

<script type="text/javascript" src="lib/router.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Inject,Component,View,bootstrap} from "angular2/angular2";

import {LocationStrategy,RouteConfig,RouterOutlet,Router,routerInjectables} from "angular2/router";

//EzApp组件

@Component({selector:"ez-app"})

@View({

directives:[RouterOutlet],

template : `

<!--声明路由入口-->

<nav>

<b (click)="go('/video')">video</b> |

<b (click)="go('/music')">music</b>

</nav>

<main>

<!--声明路由出口-->

<router-outlet></router-outlet>

</main>

`

})

//路由配置注解

@RouteConfig([

{path:"/video", component:EzVideo},

{path:"/music", component:EzMusic}

])

class EzApp{

//注入路由器对象：Router

constructor(@Inject(Router) rt,@Inject(LocationStrategy) ls){

ls.pushState = function(){};

this.router = rt;

}

go(path){

//根据给定的url，选中组件并在outlet中激活

this.router.navigate(path);

}

}

//EzVideo组件

@Component({selector:"ez-video"})

@View({

template : `

<h1>I LOVE THIS VIDEO!</h1>

`

})

class EzVideo{}

//EzMusic组件

@Component({selector:"ez-music"})

@View({

template : `

<h1>THAT'S FANTASTIC MUSIC!</h1>

`

})

class EzMusic{}

bootstrap(EzApp,[routerInjectables]);

</script>

</body>

</html>

### RouterLink - 路由入口指令

除了使用Router的navigate()方法切换路由，Angular2还提供了一个指令用来 将一个DOM对象增强为路由入口：

1. @View({
2. directives:[RouterOutlet,RouterLink]
3. template : `<nav>
4. <b router-link="video">video</b> |
5. <b router-link="music">music</b>
6. </nav>
7. <router-outlet></router-outlet>`
8. })

RouterLink指令为宿主DOM对象添加click事件监听，在触发时调用Router的 navigate()方法进行路由。

**路由项别名**

需要指出的是，RouterLink并不能直接使用路由项的路径，router-link属性 的值是一个路由项的别名，我们需要在路由配置时通过as属性，为路由项设定别名：

1. @RouteConfig([
2. {path:"/video", component:EzVideo , as:"video"},
3. {path:"/music", component:EzMusic , as:"music"}
4. ])

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>RouterLink</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<!--引入Angular2路由模块库-->

<script type="text/javascript" src="lib/router.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Inject,Component,View,bootstrap} from "angular2/angular2";

import {LocationStrategy,RouteConfig,RouterLink,RouterOutlet,Router,routerInjectables} from "angular2/router";

//EzApp组件

@Component({selector:"ez-app"})

@View({

directives:[RouterOutlet,RouterLink],

template : `

<!--声明路由入口-->

<nav>

<b router-link="video">video</b> |

<b router-link="music">music</b>

</nav>

<main>

<!--声明路由出口-->

<router-outlet></router-outlet>

</main>

`

})

@RouteConfig([

{path:"/video", component:EzVideo,as:"video"},

{path:"/music", component:EzMusic,as:"music"}

])

class EzApp{

constructor(@Inject(Router) rt,@Inject(LocationStrategy) ls){

ls.pushState = function(){};

this.router = rt;

}

}

//EzVideo组件

@Component({selector:"ez-video"})

@View({

template : `

<h1>I LOVE THIS VIDEO!</h1>

`

})

class EzVideo{}

//EzMusic组件

@Component({selector:"ez-music"})

@View({

template : `

<h1>THAT'S FANTASTIC MUSIC!</h1>

`

})

class EzMusic{}

bootstrap(EzApp,[routerInjectables]);

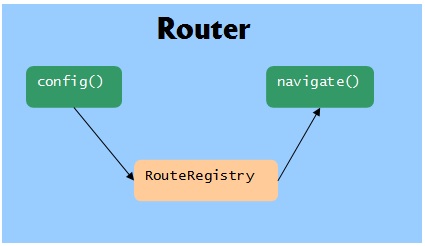
</script>

</body>

</html>

### RouteRegistry - 路由注册表

Router通过config()进行的路由配置，保存在路由注册表中；而Router通过 navigate()执行路由时，利用的就是路由注册表中的信息：



在路由注册表中，保存有三种类型的路由表：

**匹配表/matchers**

匹配表是最基本的路由决策表，它使用正则表达式实现路径向组件的匹配。

1. @RouteConfig([
2. {path:"/video", component:EzVideo},
3. {path:"/music", component:EzMusic}
4. ])

对应生成的匹配表以正则表达式为键值，映射到汇算后的路由项：

1. /^\/video$/g => {path:"/video", ...}
2. /^\/music$/g => {path:"/music", ...}

**重定向表/redirects**

如果在配置时指定了路径重定向，那么将在重定向表中生成路由项。

1. @RouteConfig([
2. {path:"/video", component:EzVideo},
3. {path:"/music", component:EzMusic},
4. {path:"/test", redirectTo:"/video"}
5. ])

对应生成的重定向表以原始路径为键，映射到目标路径：

1. /test => /video

**名称表/names**

如果在配置路由时，通过as属性为路由项设置了别名，那么将为每个别名建立一个 路由项。我们已经知道，指令RouterLink需要使用名称表。

1. @RouteConfig([
2. {path:"/video", component:EzVideo,as:"video"},
3. {path:"/music", component:EzMusic,as:"music"}
4. ])

对应生成的名称表以别名为键值，映射到汇算后的路由项：

1. video => {path:"/video", ...},
2. music => {path:"/music", ...}

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>RouteRegistry</title>

<script type="text/javascript" src="lib/system@0.16.11.js"></script>

<script type="text/javascript" src="lib/angular2.dev.js"></script>

<!--引入Angular2路由模块库-->

<script type="text/javascript" src="lib/router.dev.js"></script>

<script type="text/javascript" src="lib/system.config.js"></script>

</head>

<body>

<ez-app></ez-app>

<script type="module">

import {Inject,Component,View,bootstrap,NgFor} from "angular2/angular2";

import {LocationStrategy,RouteRegistry,RouteConfig,RouterOutlet,RouterLink,Router,routerInjectables} from "angular2/router";

//EzApp组件

@Component({selector:"ez-app"})

@View({

directives:[RouterOutlet,RouterLink,NgFor],

template : `

<nav>

<!--声明路由入口-->

<b router-link="video">video</b> |

<b router-link="music">music</b>

</nav>

<main>

<!--声明路由出口-->

<router-outlet></router-outlet>

</main>

<footer>

<h3>matchers</h3>

<pre \*ng-for="#matcher of dump('matchers')">{{matcher}}</pre>

<h3>redirects</h3>

<pre \*ng-for="#redirect of dump('redirects')">{{redirect}}</pre>

<h3>names</h3>

<pre \*ng-for="#name of dump('names')">{{name}}</pre>

</footer>

`

})

@RouteConfig([

{path:"/video", component:EzVideo,as:"video"},

{path:"/music", component:EzMusic,as:"music"}

])

class EzApp{

//注入路由注册表对象:RouteRegistry

constructor(@Inject(RouteRegistry) rr,@Inject(LocationStrategy) ls){

ls.pushState = function(){};

this.rr = rr;

}

//显示路由注册表内容

dump(type){

var routeRecognizer = this.rr.\_rules.get(EzApp);

if(type === "names"){

var names = [];

routeRecognizer.names.forEach((v,k)=>{

names.push(k+"=>"+JSON.stringify(v))

});

return names;

}else if(type === "matchers"){

var matchers = [];

routeRecognizer.matchers.forEach((v,k)=>{

matchers.push(k+"=>"+JSON.stringify(v))

})

return matchers;

}else if(type === "redirects"){

var redirects = [];

routeRecognizer.redirects.forEach((v,k)=>{

redirects.push(k+"=>"+JSON.stringify(v))

})

return redirects;

}

return [];

}

}

//EzVideo组件

@Component({selector:"ez-video"})

@View({template : `<h1>VIDEO</h1>`})

class EzVideo{}

//EzMusic组件

@Component({selector:"ez-music"})

@View({template : `<h1>MUSIC</h1>`})

class EzMusic{}

bootstrap(EzApp,[routerInjectables]);

</script>

</body>

</html>