# Angular 路由

# 路由基础

## 什么是路由

在Web 开发中，路由的概念由来已久，简而言之，就是利用URL的唯一性来指定特定的事物，这个事物可以是文件、状态、数据，等等。 服务器端路由早已有之，随着近几年REST理念的流行，为更多的人接受和使用。而浏览器客户端路由，则是随着单页应用 (SPA)的兴起，才被越来越多的前端框架所实现。在Angular中，页面由组件构成，因此URL和页面的对应关系实质上就是URL和组件的对应关系

URL和URI的区别：<http://www.cnblogs.com/wuyun-blog/p/5706703.html>

restFul架构风格： <http://www.ruanyifeng.com/blog/2011/09/restful>

名词： 路由组件（含有路由的组件）。

创建带路由模块的项目请使用： ng new xxx --routing

在Angular路由的工作流程：

1. 当用户在浏览器上输入了URL后，Angular将获取该URL并将其解析生成一个UrlTree实例。
2. 在路由配置中寻找并激活与UrlTree实例匹配的配置项。配置项就是（routes）；
3. 为配置项中指定的组件创建实例
4. 将盖组件渲染于路由组件的模板中<router-outlet>指令所在位置。

具体步骤：

1. Routes路由配置项
2. 设置<router-outlet>占位符
3. 设置触发跳转功能（<a routerLink> / router.navigate() ）

具体对象以及参数信息如下图：



## 后端路由

在Express中路由长这样：

router.get(‘/about’,function(req,res){

res.send(‘About as’);

})

在Ruby中路由长这样：

get ‘/about’, to: ‘pages#about’

class PagesController < actionController :: Base

def ablout

render

end

end

## 路由策略

路由策略决定Angular将使用URL的哪一部分来和路由配置项的path属性进行匹配。Angular提供了PathLocationStrategy和HashLocationStrategy两种策略。以首页为例：

Path（Angular默认）策略时：http://localhost:4200/home

Hash策略时：http://localhost:4200/#/home

Hash策略特点：

1. 浏览器向服务器发送请求时不会带上hash部分的内容。只发<http://localhost:4200>
2. 更改URL的hash部分不会向服务器重新发起请求。这样子在跳转的时候就不会引发页面的刷新和应用的重新加载。

Path策略特点：

1. 使用URL的path部分来进行路由的匹配，会将URL原封不动的发送给服务器。

Path优点：

1. 为服务器渲染提供了可能。

Path条件：

1. 浏览器需要支持HTML5的histroy.pushState() 方法。
2. 需要在服务器端进行配置，将所有的URL重定向到应用首页。实际URL在服务器上并不存在相对应的文件结构。
3. 需要设置一个base路径。Angular将以base路径作为前缀。可以区分不同应用的不同请求。

重点：

1. 当你要把Angular程序部署在对HTML5支持不足环境下时，请使用Hash策略
2. 当传统的集成式部署的时候，请使用Hash策略。

## 路由跳转

对于Angular构建的单页应用而言，页面跳转实质上就是从一个配置项跳转到另一个配置项的行为。

指令跳转：[routerLink]指令，其值是需要跳转的路径。这个指令可以写在任何HTML元素上。触发是click事件。行为：1、拦截click事件，2、根据传递的值跳转。

代码跳转：Router.navigate()方式。

以上两种方式传递的值都是一样的。

## 注意事项：

不要在path参数中添加斜杠，请使用path: 'home'

# 路由参数

Angular组件间传递数据的三种方式，我们今天来介绍第二种路由参数

## Path（使用最多）参数

顾名思义是通过解析URL的path部分来获取参数。

步骤：

1、配置路由配置项：例如path: 'contact/:id'

2、修改我们的指令（代码）的参数："['/contact',1]"

3、占位符不用修改

4、接收数据 ActivatedRoute.snapshot.params['id']

## Query参数

在查询参数中传递数据

步骤：

1、路由配置项不用修改

2、修改我们的指令（代码）的参数："['/contact']" [queryParams]="{id: 1}"

3、占位符不用修改

4、接收数据 ActivatedRoute.snapshot.queryParams['id']

## Data（基本不用）参数

直接在路由配置项中指定参数值。

步骤：

1. 路由配置项：

{path:/content, component: ContentComponent, data:[{isContent:true}]

1. 指令（代码）不用修改
2. 占位符不用修改。
3. ActivatedRoute.data[0][isContent] = true

## 参数订阅和参数快照

参数快照：snapshot 。 ngOnInit只会调用一次。

参数订阅：会以数据流的形式来监控参数的变化。在我们Router中可以不用取消订阅，而在其他的订阅中你需要手动的来取消。

上述两种方式都推荐使用，关键区别在于组件本身会不会调用自己。如果会那么用参数订阅，如果不会用参数快照会简便很多。

# 重定向路由

我们有时候需要对初始化的地址，或者已经不存在的地址进行重定向，这是可以使用重定向路由的语法。

使用方式：

1. 路由配置项中使用redirectTo指向需要重定向的地址。
2. 设置pathMatch可能的值为 full 或者 prefix

当值为full时候需要跟重定向的path完全匹配，当值为 prefix的时候匹配开头。

作用：

1. 可以让我们的首页进来就跳转到你需要展示的页面比如（home）。
2. 使用prefix 的时候可以让我们快速的把整个路径全部都统一处理。

场景：当我们的项目上线之后，用户已经收藏了一堆堆的链接。而我们的项目由于版本更新或者迭代把原先的路径废除了，这个时候就可以快速重定向。

# 路由通配符

解决查找不到或者丢失页面的情况。

步骤：

1、{ path: '\*\*', component: NotFoundComponent }

# 子路由

在前面组件的课程中介绍了，一个组件可以被嵌入到另外一个组件中，从而建立起组件之间的多级嵌套关系。与此类似，Angular也允许一个路由组件被嵌入到另一个路由组件中，建立多级路由嵌套关系。

步骤：

1. 配置路由配置项：例如

{ path: 'contact/:id', component: ContactComponent, children: [

{ path: '', component: SiqiComponent },

{ path: 'derry', component: DerryComponent }

]}

1. 不用修改父路由的指令，而要添加子路由的指令在含有子路由的模板中。
2. 需要添加子路由的占位符，在含有子路由的模板中。ContactComponent中

注意：

子路由的指令需要配置. 例如："['./derry']"

**Matrix**参数：矩阵URL标记法

特点：

1. 可以配置我们路由可选参数。

this.router.navigate(['/xxx', { id: userId, color: 'black' }]);

1. 能够精确定位我们不同层次的路由需要的同名的数据参数。

localhost:4200/contact/name/1 无法区分不同的参数

localhost:4200/contact;id=1/name;id=2

# 辅助路由

Angular允许一个路由组件包含多个Outlet（路由插座）,从而可以在一个路由组件中同时显示多个组件。

主要Outlet（Primary Outlet）有且仅有一个，附属Outlet（Auxiliary Outlet）可以有任意多个，各个附属Outlet通过不同的命名加以区分。每一个Outlet均可以通过路由配置来指定可以显示的组件，这使得Angular可以灵活地对各个组件进行组合，从而满足不同场景需求。

步骤：

1. 路由配置项需要定义要被显示在的插座的名称例如：

{ path: 'chat', component: ChatComponent, outlet: 'aux'},

上面的 aux是自己定义的名称

1. 指令需要修改成如下的方式：

"[{outlets: {aux: ['chat']}}]"

3、 占位符需要新建一个，其名称为 aux。

# 路由守卫

Angular的路由守卫（路由拦截），允许在从一个配置项跳转到另外一个配置项之前执行指定的逻辑，并根据执行的结果来决定是否进行跳转。Angular提供了五类路由拦截：

## CanActivate，激活守卫

功能：处理进入路由的情况

参数：

1、ActivatedRouteSnapshot，表示所要激活的目标配置项（被使用到的路由的信息）。

2、RouterStateSnapshot，表示应用当前所处的路由状态，其包含了当前所需的所有配置项。

通过实现CanActivate接口创建守卫服务。该接口只包含了一个canActivate()方法，最简单的情况，该方法返回true，则表示通过守卫。返回false时，表示不允许，则对守卫目标不允许激活 （不允许访问）。

场景：

1、防止我们的用户去访问一些他根本没有权限访问的内容。

## CanActivateChild，子路由激活守卫

本路由用于控制是否允许激活子路由配置项，其用法与CanActivate守卫相似，大家自行尝试。具体案例参考官网

## CanDeactivate，反激活守卫

功能：处理离开路由的情况

通过实现CanDeactivate接口创建守卫服务，跟CanActivate相比，该方法除了第一个参数为目标配置项对应组件的实例外，其余使用方式与canActivate()方法一样。

场景：

1、用户在输入写非常关键的信息，或者进行保存前的操作的时候，我们要避免不小心退出了当前的页面。

## Resolve，数据预加载守卫

通过实现Resolve<T>泛型接口创建守卫服务。该服务只有一个resolve()方法，用于执行数据预加载逻辑。该方法可以直接将数据返回，在异步情况下也可以通过Observable对象触发。其返回的任何数据（包括false）都将存放于配置项的data参数部分，如果没有预加载到期望的数据，只能通过代码跳转的方式来达到不激活目标配置项的目的。

场景：

1、我们有时候加载表单内容的时候，会出现页面已经出来了，但是数据还没有加载完成。

## CanLoad，模块加载守卫

通过实现CanLoad接口创建守卫服务，当Url匹配到延迟加载的配置项，canload守卫就会对其拦截判断是否能加载。

# 懒加载

Angular应用由一个根模块和任意多个特性模块组成。由于一个大型的项目通常会包含许多的特性模块，如果在首屏加载时便将所有的特性模块加载进来，对于用户体验和服务器负载都会有影响。所以Angular提供了延迟加载（异步加载/懒加载），使得只有在真正需要使用到的时候才加载进来。

使用方式：

1、{ path: 'async', loadChildren: "app/async/async.module#AsyncModule"}

注意：

懒加载的模块，不能放到根模块同级目录下。