第二节：Angular2路由（Router）的使用

概要：

路由也可以叫做分发器，路由简单的理解就是从一个页面跳转到另一个页面。

在传统的web系统中，每一个页面都是从服务器获得，每一次页面跳转，就会往服务器上发送一个http的请求，服务器上的某一段程序响应这个请求，处理并返回一个html/htm页面。所以在传统的web系统中，路由的定义都是在服务器端。

在单页应用中，页面的跳转不经过服务器，如果由纯JavaScript实现的话，当用户点击一个按钮或链接，打开一个页面时，实际上由某个js程序根据请求的类型，将某一个div的html页面加载到页面上的某一个区域中显示。当这个单页应用的页面比较多、业务比较复杂时，页面之间的跳转、数据传递、状态保存等都是一件非常麻烦的事情。

*帮助理解：硬件路由器的功能是管理硬件设备连接（电脑、手机、pad、智能电视等），angular2 的路由就是管理功能模块及组件的连接*

本节内容：

1. 基本概念
2. 最基本的路由设置
3. 从模块文件中拆分出独立的路由文件
4. route接口定义
5. 模块延时加载（Lazy loading）
6. 路由守卫（Guard）
7. 路由连接routerLink与参数传递获取
8. 教程源码
9. 练习

1. 基本概念
   1. base href

Index.html中配置：

<base href="/lims">

如果是IIS搭建的服务器，此处就是配置IIS中设置的应用程序别名,如上即是别名为lims，大部分时间我们是不用设置的。

1. 载入点

在app.component.html放入：

<router-outlet></router-outlet>

我更愿意把它理解成插座,本身outlet也就是插座的意思，一个插座（载入点）可以连接各种各样的电器（组件）。

1. 最基本的路由设置

在myng2app中 增加“首页”组件home,利用命令窗口或VSC的终端窗口执行：

ng g c home

*备注：ng g c home 是 ng generate component home 的简写*

*后面还可以跟参数 --inline-template --inline-style，代表模板及CSS文件均创建在TS文件中，同样可以简写为：ng g c home --it --is*

1. 完成上面的配置后，在app.component.html中放入插座

<router-outlet></router-outlet>

1. 在app.module.ts中注册路由

import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';

const routes: Routes = [

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'home', component: HomeComponent }

];

@NgModule({

declarations: [……],

imports: [

……

RouterModule.forRoot(routes, { useHash: true })

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

1. 保存，切换到浏览器，不用刷新它将自动刷新，

将看到浏览器显示

app works! 
home works! 

1. 从模块文件中拆分出独立的路由文件

**为什么要拆分？**

我们在实际业务当中，往往是随着开发的进程我们的路由在不间断的更新，如个在module中管理将会非常混乱，下面我们来拆分它

将路由封装成模块，下面就简单的介绍做法

1. 新建app-routing.module.ts
2. 在其中加入如下代码

import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';

const routes: Routes = [

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'home', component: HomeComponent }

];

export const routing = RouterModule.forRoot(routes, { useHash: true });

1. 在app.module.ts中导入routing模块

在头部引入（类似于C#中的using）

import { routing } from './app-routing.module';

在@NgModule中的import导入

imports: […… routing ]

app.module.ts更改后的代码如下：

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

import { HomeComponent } from './home/home.component';

import { routing } from './app-routing.module';

@NgModule({

declarations: [

AppComponent,

HomeComponent

],

imports: [

BrowserModule,

FormsModule,

HttpModule,

routing

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

回到浏览器同样可以看到正确结果，但细心的同鞋会发现在app.module.ts和app-routing.module.ts中均引入了HomeComponent ，让我们有代码洁癖的人很是不爽是吧，所以我们再次分离，将routingf封装成一个真正的module（如果引用的组件不多，还是推荐app-routing.module.ts只保留route数据)

这里推荐安装VSC的 扩展 Angular2 Typescript Snippets,

安装完成后可通过在新建的ts文件中ng2-routing来加入代码片段

更改app-routing.module.ts如下:

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';

const routes: Routes = [

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'home', component: HomeComponent }

];

@NgModule({

imports: [RouterModule.forRoot(routes,{ useHash: true })],

exports: [RouterModule],

})

export class AppRoutingModule { }

export const routedComponents = [HomeComponent];

更改app.module.ts如下:

import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { FormsModule } from '@angular/forms';

import { HttpModule } from '@angular/http';

import { AppComponent } from './app.component';

import { AppRoutingModule,routedComponents } from './app-routing.module';

@NgModule({

declarations: [

AppComponent,

routedComponents

],

imports: [

BrowserModule,

FormsModule,

HttpModule,

AppRoutingModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

其中的 app-routing.module.ts中 useHash: true 配置为true后将在域名后自动增加#号，切换到浏览器将看到URL变为：http://localhost:4200/#/home,app.module.ts中也没有HomeComponent 组件的引用，这样一来所有路由相关的已彻底分离出去。

1. route接口定义

interface Route {

path : string

pathMatch : string

matcher : UrlMatcher

component : Type<any>

redirectTo : string

outlet : string

canActivate : any[]

canActivateChild : any[]

canDeactivate : any[]

canLoad : any[]

data : Data

resolve : ResolveData

children : Routes

loadChildren : LoadChildren

}

部分重要的属性将在下面章节内容中介绍

*“*

*其中已can开头的我们称为路由守卫（在实际业务中非常重要）将在后面章节介绍。*

*loadChildren 将在模块延时加载中介绍*

*”*

可通过官网 <https://angular.io/docs/ts/latest/api/router/index/Route-interface.html> 查看

1. 模块延时加载（lazy loading）

利用loadChildren 可对模块延时加载，模块延时加载能给用户带来更好的体验，但也不是所有模块都应延时加载，也要根据功能模块分析是否应该使用延时加载。

**如何利用loadChildren延时加载？**

举例说明：

在myng2app中 增加“关于”模块about,利用命令窗口或VSC的终端窗口执行：

ng g m about --routing

完成上面命令后，将自动创建about模块，现在我们要对about模块设计为延时加载，按下面步骤

1. 改造app-routing.module.ts，增加路由配置:

{

path: 'about',

loadChildren: 'app/about/about.module#AboutModule'

}

1. 在app.component.html中增加导航如下：

<h1>

{{title}}

</h1>

<header>

<nav>

<a routerLink='/home' >home</a>

<a routerLink='/about' >about</a>

</nav>

</header>

<router-outlet></router-outlet>

1. 进入about模块打开about-routing.module.ts增加模块内的路由配置

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { AboutComponent } from './about.component';

const routes: Routes = [{

path:"",

component:AboutComponent

}];

@NgModule({

declarations:[AboutComponent],

imports: [RouterModule.forChild(routes)],

exports: [RouterModule],

providers: []

})

export class AboutRoutingModule { }

1. 在about.module.ts中导入路由

import { NgModule } from '@angular/core';

import { CommonModule } from '@angular/common';

import { AboutRoutingModule } from './about-routing.module';

@NgModule({

imports: [

CommonModule,

AboutRoutingModule

],

declarations: []

})

export class AboutModule { }

此时完成了路由的延迟加载，下面我们来验证它

先打开首页跟踪其JS加载情况如下：

e 
C O local host:420()/#/home 
app works! 
home about 
home works! 
Elements 
Console Sources Network 
View: 
C) Preserve 
Hide data URL 
Regex 
Filter 
100 ms 
Name 
inline.bundle.js 
vendor.bundle.j 
main.bundle.js 
ng-validate.js 
backend.js 
200 ms 
300 ms 

再切换到about模块跟踪其JS加载情况如下：

e 
C O localhost:4200/#/about 
app works! 
home about 
about works! 
Elements 
Console Sources Network 
View: 
C) Preserve log 
Hide data URLs 
Regex 
Filter 
5000 ms 
Name 
inline.bundle.js 
vendor.bundle.js 
main.bundle.js 
ng-validate.js 
backend.js 
4. chunk.js 
10000 ms I 
15000 ms 
2000 

从图中可以看出，在进入了about模块后多加载了一个4.chunk.js（名字在不同环境下可能不同），这就是about模块WEBPACK打包的JS文件，也就是说在配置了形如 loadChildren: 'app/about/about.module#AboutModule' 时WEBPACK将拆分JS打包，用户在第一次进入模块时才会加载拆分的js包。

通过前面章节我们已建立了一个主模块路由，和一个子模块路由，可以发现两个路由文件基本一致，但也有明显的区别，主路由采用的是 RouterModule.forRoot(routes)，而子路由采用的是 .forChild(routes)

我们再建立一个非延时加载的模块todo，来看两个模块的配置区别

新增todo模块，并在app.component.html中增加todo的导航

ng g m todo --routing

在app-routing.module.ts 中引用

import { TodoModule } from './todo/todo.module';

在imports 节中导入TodoModule,并增加路由配置，

完成后的内容如下：

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from './home/home.component';

import { TodoModule } from './todo/todo.module';

const routes: Routes = [

{ path: '', redirectTo: 'home', pathMatch: 'full' },

{ path: 'home', component: HomeComponent },

{

path: 'about',

loadChildren: 'app/about/about.module#AboutModule'

},

{

path: 'todo',

redirectTo: "todo"

}

];

@NgModule({

imports: [TodoModule, RouterModule.forRoot(routes, { useHash: true })],

exports: [RouterModule],

})

export class AppRoutingModule { }

export const routedComponents =[HomeComponent]

打开新建立的todo-routing.module.ts ，仿照about-routing.module.ts增加路由配置，

此时打开浏览器预览，发现报错了，除了延时加载和非延时加载的区别，其他一样但报错，这是由于

非延时加载模块中直接将todo请求redirectTo扔给了todo模块，但todo模块并没有对todo配置路由，

现在我们把todo-routing.module.ts中的path配上todo如下：

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { TodoComponent } from './todo.component'

const routes: Routes = [{

path:"todo",

component:TodoComponent

}];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule],

providers: []

})

export class TodoRoutingModule { }

切换到浏览器中查看一切正常，所以非延时加载与延时加载需要注意子模块中的路由设定。

1. 路由守卫(Guard)

Guard其实是一系列接口，只要你实现了它的方法，配置了这些Guard，Angular会根据这个方法返回的true或false来判断是否激活这个路由。

它包括几种类型：

* CanActivate  
  这种类型的Guard用来控制是否允许进入当前的路径。
* CanActivateChild  
  这种类型的Guard用来控制是否允许进入当前路径的所有子路径。
* CanDeactivate  
  用来控制是否能离开当前页面进入别的路径
* CanLoad  
  用于控制一个异步加载的子模块是否允许被加载。

以 CanActivate 为例，这个接口的定义如下：

interface CanActivate {

[canActivate](https://angular.io/docs/ts/latest/api/router/index/CanActivate-interface.html#canActivate-anchor)(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) : Observable<boolean>|Promise<boolean>|boolean

}

这个接口定义了一个方法，当你实现这个接口，并把它配置到某一个路由上以后，当用户进入这个路由的路径之前，就会调用它里面的 canActivate() 方法，它第一个参数，就是将要激活的路由，第二个参数是路由器当前的状态。它返回一个布尔型的结果，或者是布尔型的 Promise 或 Observable 。

**Resolve**

resolve就是用来在打开一个页面之前先获取数据，而不是进入页面以后再加载。这个接口中的方法，可以返回任意的对象，也可以返回一个 Promise ，或者 Observable

如果在一个路径上同时设置了 CanActivate 和 Resolve ，首先 CanActivate 接口的方法会被执行，当这个路由可以被激活时， Resolve 接口的方法才会被执行。

示例：为todo增加CanActivate 路由守卫

1. 在app下建立以个core文件用于将来存放核心文件,创建守卫服务类

ng g s ./core/appGuard 或 ng g s ./core/app-guard

Angular-CLI 根据驼峰命名来已中横线来连接，如上两命令将都会生成app-guard.service.ts服务，类名将自动采用大驼峰命名为AppGuardService ，

1. 增加守卫服务代码，由于演示就不把守卫和权限验证分开了（为了结构清晰，守卫和权限验证推荐分开文件实现），直接上代码：

import { Injectable } from '@angular/core';

import { CanActivate,ActivatedRouteSnapshot,RouterStateSnapshot } from '@angular/router';

import { Observable } from 'rxjs/Observable'

class UserToken {}

class Permissions {

canActivate(user: UserToken, path: string): boolean {

//验证代码未实现，直接返回false

console.log(path);

alert("无权访问");

return false;

}

}

@Injectable()

class AppGuardService implements CanActivate {

constructor(private permissions: Permissions, private currentUser: UserToken) {}

canActivate(

route: ActivatedRouteSnapshot,

state: RouterStateSnapshot

): Observable<boolean>|Promise<boolean>|boolean {

return this.permissions.canActivate(this.currentUser, state.url);

}

}

1. 在todo-routing.module.ts中增加路由守卫

import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';

import { TodoComponent } from './todo.component';

import { AppGuardService,Permissions,UserToken } from '../core/app-guard.service';

const routes: Routes = [{

path:"todo",

component:TodoComponent,

canActivate:[AppGuardService]

}];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule],

providers: [ AppGuardService,Permissions,UserToken ]

})

export class TodoRoutingModule { }

现在访问todo将会提示无权访问，如果对todo下所有连接都需要验证，这时就可以用CanActivateChild

1. 路由连接routerLink与参数传递获取

**使用routerLink传递参数**

要在链接中使用参数，假设要实现上面的/todo/detail/12链接，我们需要使用下面的方式：

<a [routerLink]="['/todo/detail', item.id]" routerLinkActive="active">

在这里，我们使用[routerLink]的方式进行数据的绑定，绑定的值，就是['/todo/detail', item.id]，他是一个表达式，Angular会把这个列表中的2个数据解析后拼接在一起，生成’/todo/detail/12’的链接。如果我们的连接格式是’/todo/12/detail’，那就是这样：

<a [routerLink]="['/todo', item.id, 'detail']" routerLinkActive="active">

如果，要传递的参数是这种/todo/detail?id=12&type=important方式，也很简单：

<a [routerLink]="['/todo/detail']" [queryParams]="{id: item.id, type: 'important'}">

**使用代码传递参数**

其实在代码中添加参数，跟上面的方法类似。对于/todo/detail/12类型的链接：

onSelect(item: Item) {

this.router.navigate(['/todo/detail', item.id]);

}

那么，要实现这种/todo/detail?id=12&type=important路径的导航，该怎么办呢？那就是：

onSelect(item: Item) {

this.router.navigate(['/todo/detail'], { queryParams: { id: item.id, type: 'important' } });

}

**在组件中取得参数**

constructor(private route: ActivatedRoute, private router: Router) {}

ngOnInit() {

this.route.params

.map(params => params['filter'] || 'None')

.subscribe(res=>{

console.log(res);

});

同样可以取得 queryParams 参数值

this.route.queryParams

.map(params => params['id'] || 'None')

.subscribe(res=>console.log(res));

1. 教程源码

<https://github.com/myLeon/myNg2App/tree/branch01>

1. 练习

利用 angular-cli 创建 ng2app

要求：

1. 创建分析方法模块（analysis），并且延时加载
2. 创建任务模块（task），不延时加载
3. 任务模块需要身份认证(不用真实验证，可直接返回true,false)
4. 分析方法模块中的analysis.component.html中显示一句：我来自analysis.service.ts的方法test
5. 在首页增加首页、任务、所有分析方法、柴油分析方法，汽油分析方法菜单导航
6. 分析方法模块接收一个路由参数（route param）方法分类：category，用于接收汽油、柴油分类

并在方法页中显示其值，为空时显示none，增加一个查询参数（queryParams）关键字key

注：利用依赖注入实现第4点要求