route

2018年11月19日　　17:18

0. 路由基本概述

名称	简介
Routes	路由配置，保存着哪个URL对应展示哪个组件，以及在哪个RouterOutlet 中展示组件。
RouterOutlet	在Html中标记路由内容呈现位置的占位符指令。
Router	负责在运行时执行路由的对象，可以通过调用其navigate()和navigateByUrl ()方法来导航到一个指定的路由。
RouterLink	在Html中声明路由导航用的指令。
ActivatedRoute	当前激活的路由对象，保存着当前路由的信息，如路由地址，路由参数等。

1. 生成带路由模板的命令：
ng new router --routing
这样会在app目录下会多生成一个app-routing.module.ts
如果用ng new router创建的目录没有app-routing.module.ts文件时，在app.module.ts来声明，类似于下图：

```
import {ProductService} from './shared/product.service';

// 声明路由配置
const routeConfig: Routes = [
  {path: '', component: HomeComponent},
  {path: 'product/:productId', component: ProductDatailComponent},
];
```

而在app-routing.module.ts里定义路由如下：

```
const routes: Routes = [
  {path: 'home', component: HomeComponent},
  {path: 'product', component: ProductComponent},
```

路由的配置，Rouyes是个数组，数组内中每个成员都有两个元素组成：路由的path和对应的组件。

2. 路由基本配置：
路由在module.ts中的配置如上图所示，接下来就是在html中的配置了，首先是在主页面即app.component.html里做一个链接配置，如下所示：
<a [routerLink]="['/home']">主页
<a [routerLink]="['/product']">商品详情
Note：使用routerLink时注意参数是个数组，方便后期携带参数到后台调用，下面会提到

3. 在路由中传递数据：
(一)基本方式：
路由传递参数的三种方式：
在查询参数中传递数据
/product?id=1&name=2　　=>　　ActivatedRoute.queryParams[id]
在路由路径中传递数据
{path:/product/:id}　=> /product/1　　=>　　ActivatedRoute.params[id]
在路由配置中传递参数
{path:/product, component: ProductComponent, data:[{isProd:true}]}
=>　　ActivatedRoute.data[0][isProd]

(二)实际代码：
a.依然是在module.ts进行配置：
通过路由来携带参数，路由分为两段，一段是路由，一段是携带的变量
{path: 'product/:id', component: ProductComponent},
b.在后台product.component.ts去接收这个参数

```
//内部构造一个函数主要是声明routeInfo是ActivatedRout类型的
constructor(private routeInfo: ActivatedRoute) { }

//从查询参数中获取传过来的参数（方法一）
this.productId = this.routeInfo.snapshot.queryParams["id"];

//从路由路径中获取传过来的参数(方法二)
this.productId = this.routeInfo.snapshot.params["id"];

//上述两个方式会导致两个物品之间相互查看时profuctId只会被创建一次，从而会使商品信息无法更新
```

```
// 上面的snapshot是参数快照，只能初始化一次，所以用下面方法，参数订阅
// snapshot快照 subscribe订阅，后面用=>定义了一个隐函数，这样可以保证不断更新id值
this.routeInfo.params.subscribe((params: Params) => this.productId = params['id']);
```

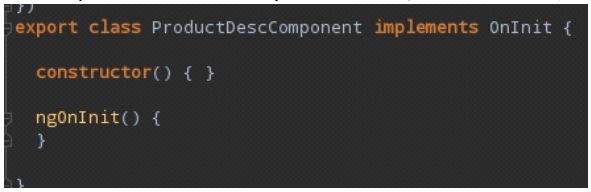
c.对应的html则需要这样更改：
传参方式1
[routerLink]="['/product']" [queryParams]="{id: 1}"就携带一个id=1的参数，然后在后台product.component.ts去接这个参数
对应的url：http://localhost:4200/product?id=1
```
<a [routerLink]="['/product']" [queryParams]="{id: 1}">商品详情</a>
```

传参方式2
routerLink是数组类型，第二个元素为携带的变量
对应的url：http://localhost:4200/product/1
```
<a [routerLink]="['/product', 1]">商品详情</a>
```
None：
这两种的url是不同的，但是对应的path是一样的
对于参数快照和参数订阅的区别在于参数订阅应用于两个商品之间的相互跳转，可以做到加载不同的商品信息，而参数快照只是跳转了，没做跳转之后的商品信息加载，即展示的商品信息依然还是上一个商品的信息

1. 事件绑定：
a.依然是在module.ts进行配置：
```
{path: 'product/:id', component: ProductComponent},
```
b.在component.ts中定义如下：
```
constructor(private router: Router) { }
  toProductDetails() {
    // this.router.navigate后面和html中的routerLink后跟的参数形式是一样的，都是数组类型
    this.router.navigate(['/product', 2]);
  }
```
c.对应的html则需要这样定义：
```
<!--(click)="toProductDetails()"事件绑定，即点击按钮后需要app.component.ts的toProductDetails的处理-->
<input type="button" value="商品详情" (click)="toProductDetails()">
```

2. 重定向路由：
重定向路由比较简单，在app-routing.module.ts中定义：
```
// redirectTo 重定向路由，将''（根路由）重定向到home的路由上，pathMatch: 'full'代表加载home组件的所有信息
{path: '', redirectTo: 'home', pathMatch: 'full'},
```
对应的html则需要这样定义：
```
<a [routerLink]="['/']">主页</a>
```
Note：必须要用['/']开头，导航到根路由，即app-routing.module.ts配置的路由'',如果用./则只是定义到上层路由的根目录，而非整个工程的根目录。

3. 子路由：
a. 生成新的组件，定义其详情页
```
ng g c productdesc
```
然后会product-desc.component.ts在中自动生成组件类ProductDescComponent


```
export class ProductDescComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

b. 在module.ts进行配置：
```
// 通过children来规定他的子路由，其内部和路由格式一样 [{路径，组件}]
  {path: 'product/:id', component: ProductComponent, children:[
      {path: '', component: ProductDescComponent},
      {path: 'seller/:id', component: SellerInfoComponent}
    ]},
```

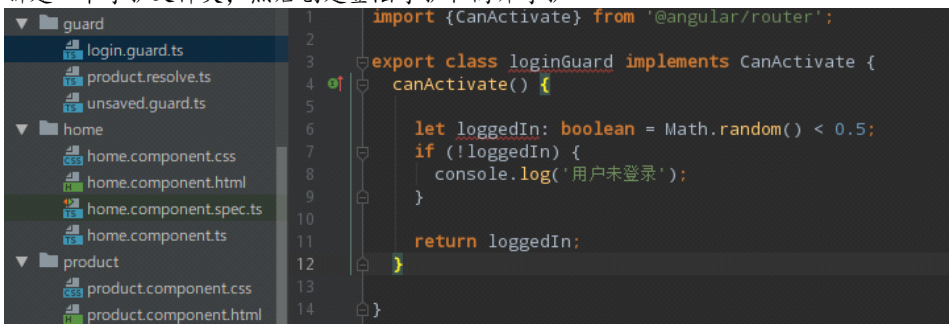
c. 在product.component.html中定义子路由：
```
  <a [routerLink]="['./']">商品描述</a>
  <a [routerLink]="['./seller', 9]">销售员信息</a>
```
Note：['./']是指路由到当前路由下，严格区别['/']与['./'],['/']是路由到根目录下

7. 辅助路由：
   a. 在module.ts进行配置：
      // 通过outlet: 'aux'定义这个组件只显示在名字为aux的插座上，从而作为辅助路由
      {path: 'chat', component: ChatComponent, outlet: 'aux'},
   b. 在html中：
<!--[{outlets: {aux: 'chat'}}]通过这个来指定辅助路由，并与后台对应应用的组件联系-->
<a [routerLink]="[{outlets: {aux: 'chat'}}]">开始聊天

<!--如果要让这个辅助路由和某个主路由绑定，即打开chat就跳转到home界面的话需要加primary来指定主路由是哪个-->
<a [routerLink]="[{outlets: {primary: 'home', aux: 'chat'}}]">开始聊天

<!--{aux: null}当aux为null时，就关闭了那个页面-->
<a [routerLink]="[{outlets: {aux: null}}]">结束聊天

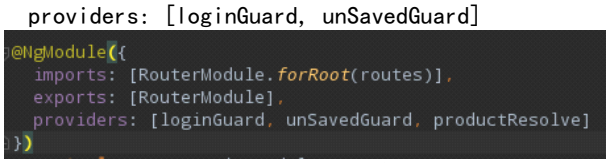
8. 路由守卫：
   a. 新建一个守护文件夹，然后创建登陆守护和离开守护



登陆守护：在canActivate(){…}中写具体判断逻辑，最后返回一个boolean类型给module.ts让canDeactivate判断是否为true，从而判断是否允许登录。

离开守护：声明CanDeactivate时需要一个泛型，这个泛型就是你要离开时的界面的组件，在<...>定义，从而防止离开该界面

   b. 在module.ts配置：
      //添加进入守护，访问者具备一定条件(登陆)后才能访问product,用canActivate属性是进入路由的守护，canActivate是数组，如果要访问，需要依次通过数组内各个组件的所有要求
      // 定义canActivate后需要实例化loginGuard，则需要在@NGModule内实例化即添加providers: [loginGuard]
      // 添加离开守护canDeactivate这个属性是离开界面时的守护，离开某一界面时会产生阻止，原理与canActivate相似
      // 定义canDeactivate后也需要实例化unSavedGuard，在@NGModule内实例化即添加providers:[loginGuard, unSavedGuard]
      {path: 'product/:id', component: ProductComponent, children:[
        {path: '', component: ProductDescComponent},
        {path: 'seller/:id', component: SellerInfoComponent}
      ], canActivate: [loginGuard], canDeactivate: [unSavedGuard]},
      之后实例化：
      providers: [loginGuard, unSavedGuard]



html没什么改变，只是访问对应的路由会有路由守护

9. resolve守护
   resolve守卫的作用就是写一个类实现Resolve，然后实现相应的方法，在该方法中，提前把数据请求等工作做好，然后在路由跳转（如跳转到A组件）的时候把这些数据传过去，这样A组件中要展示的数据就会及时展示出来。但是没有resolve守卫这种机制的话，就只能在A组件显示再从服务器请求数据，则会导致页面展示不及时的问题。
   a. 在要展示的数据(product)组件的component.ts中定义一个需要展示信息的类，方便之后携带信息

```
-routing.module.ts × product.resolve.ts × product.component.ts × lo
  }

}

export class Product {

  constructor(public id: number, public name: string) {

  }
}
```

b. 创建一个路由守卫实现一个接口Resolve，这个接口接受一个泛型，这个泛型(product)是该守卫要解析出来的数据的结构和类型。

c. 实现一个resolve方法，这个方法可以获取路由携带的参数，获取到参数后才可以进入视图。

```
outing.module.ts × product.resolve.ts × product.component.ts × seller-info.component.ts × seller-info.component.html × app.component.html ×
import {ActivatedRouteSnapshot, Resolve, Router, RouterStateSnapshot} from '@angular/router';
import {Product} from '../product/product.component';
import {Injectable} from '@angular/core';
import {Observable} from 'rxjs';

@Injectable()
// 只有加这个装饰器router才可以被注入进来
// 实现一个接口是Resolve，而他的类型是Product，即Product是你需要返回的数据的类
// 当然Product类需要在product.component.ts里面定义一个Product类，才能在这里引用，这个product泛型类需要自己定义一下引入
export class productResolve implements Resolve<Product> {

  constructor(private router: Router) {

  }

  // ActivatedRouteSnapshot: 就是一个大的router的类,类似于之前的this.routeInfo.params
  // 这里的ActivatedRouteSnapshot可以直接获取路由里面的参数，因此接下来就可以直接获取参数值
  resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<Product> | Promise<Product> | Product {

    let productId: number = route.params['id']; // Route.params直接获取参数
    if (productId == 1) {
      return new Product( id: 1,  name: 'iPhone7'); // 正确的话，返回需要的数据

    } else {
      this.router.navigate( commands: ['/home']); // 错误的话导航到主页
      return undefined;

    }
  }

}
```

d. module.ts配置：

// resolve是一个对象，参数的名字（product）就是你想传进去的参数的名字，后期在product.component.ts调用的，product后面跟着的是你定义的resolve的组件，然后在providers声明就好了

```
{path: 'product/:id', component: ProductComponent, children: [
    {path: '', component: ProductDescComponent},
    {path: 'seller/:id', component: SellerInfoComponent}
  ], resolve: {
    product: productResolve
  }
},
```

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule],
  providers: [loginGuard, unSavedGuard, productResolve]
})
```

e. component.ts接收参数：

用data来接收在module.ts中定义的product，从而作为参数返回到html中，product名字要一致才可以

```
constructor(private routeInfo: ActivatedRoute) { }

ngOnInit() {

  this.routeInfo.data.subscribe( next: (data: { product: Product }) => {
    this.productId = data.product.id;
    this.productName = data.product.name;
  });
}

}
```

f. 在product.component.html接收就行

```
<p>
  商品id是：{{productId}}
```

```
    </p>
    <p>
      商品name是：{{productName}}
    </p>
```

Note：关于路由守卫的其他例子链接： https://www.cnblogs.com/chzlh/p/7718773.html