张众磊 微信:teduzzl 邮箱:bjzhangzl@tedu.cn

推荐链接:

https://github.com/vuejs/awesome-vue （vue的开源项目集合）

https://angular.io/ (ng官网)

https://angular.cn/ (ng中文官网)

https://www.runoob.com/angularjs/angularjs-intro.html (angularjs1.\* 版本使用 )

https://github.com/PatrickJS/awesome-angular (基于ng的开源项目集合)

http://jsonplaceholder.typicode.com/ (fake api)

https://www.creative-tim.com/bootstrap-themes/angular-dashboard (dashboard效果)

Material:https://material.angular.cn

NgZorro:https://ng.ant.design/docs/introduce/zh

https://ionicframework.com/docs/v3/ (ionic官方手册)

https://ionicons.com/ (ionic官方支持的icon图标网站)

开发建议:

解决问题的思路:

准备数据

显示

做好交互

一、安排

1、angular(ng)以及生态圈(ionic)

目标:

中大型 超大型SPA

webApp、hybridApp

2、react(reactjs、reactNative)

目标：

使用react语法实现nativeApp

3、python以及生态圈(django、数据库)

目标:

使用Django实现后端API（Django是一个全栈类框架，做前端也可以做后端）

二、回顾Vue

生态圈(技术栈):

vue核心库+vueRouter(路由和导航)+vuex(集中式管理数据) + axios+elementUI/mintui/mui/vux+weex(实现nativeAPP)

1、vue核心库

①vue基础语法

<any> {{expression}} </any>

(interplocation插值表达式、mustache胡子语法)

v-for="(tmp,index) in myList" :key="index"

v-bind 属性绑定 ：

v-on:click @click

v-if/v-elif/v-else

v-model="" 双向数据绑定 (数据--》视图/视图--》数据)

②常见属性

Vue.component("",{

data(){return {count:1}},

methods:{},

computed:{}, //计算属性 有依赖缓存的

watch

props //接收父组件传递给子组件的数据

components //局部组件

filters //定义过滤器(过滤器本质是一个有参数有返回值的方法，目标实现将数据执行筛选、过滤、格式化)

directives //自定义指令

。。。

})

③组件

全局

局部

<my-component></my-component>

2、vueRouter

前端路由的本质：建立起一种路由地址和组件之间映射关系

vue.use()

2.1 基础用法

指定容器 <router-view></router-view>

创建组件

路由词典

const myRoutes = [

{path:"/",component:Index},

{path:"/login",component:Login},

{path:"/detail",component:Detail},

{path:"\*",component:NotFound}

]

new Vue({

router:new VueRouter({

routes:myRoutes

})

})

2.2 路由和传参

this.$router.push()

<router-link to=""></router-link>

传参:

配置接收方的路由地址:

/detail ---> /detail/**:**id

发送

this.$router.push('/detail/10')

接收

this.$route.params.id

2.3 嵌套路由

比如一个spa的A组件，需要动态的嵌套B组件、C组件

①给A组件指定一个容器

<router-view></router-view>

②给A组件的路由对象指定一个属性children

{

path:'/a',

component:A,

children:[

{path:'/b',component:B},

...

]

},

2.4 路由守卫

守卫

https://router.vuejs.org/zh/guide/advanced/navigation-guards.html#%E5%85%A8%E5%B1%80%E5%89%8D%E7%BD%AE%E5%AE%88%E5%8D%AB

2.5 路由懒加载

const Foo = ()=>import('./Foo.vue')

const router = new VueRouter({

routes: [

{ path: '/foo', component: Foo } //用此路径的时候，才调用Foo函数引入组件使用

]

})

https://router.vuejs.org/zh/guide/advanced/lazy-loading.html#%E6%8A%8A%E7%BB%84%E4%BB%B6%E6%8C%89%E7%BB%84%E5%88%86%E5%9D%97

3、vuex

设计初衷:解决vue项目变大之后 页面间数据通信

state :初始化/定义数据

getters:在state数据基础之上 派生新数据

mutations:修改数据

actions:异步操作

modules:模块

三、**设计原则和设计模式**

常见的设计原则(记住):

①单一责任原则(SRP)

single responsibility principle

一个文件或者一个模块、一个方法只做一件事

②YANGNI

You aren't gonna need it

不要写不需要的代码

③KISS

Keep it simple and stupid

保证代码足够的简洁

④DRY

Don't Repeat Yourself

不要重复自己

⑤OCP

Open Closed Principle

开闭原则（对于扩展是开放，对于修改是关闭）

⑥高内聚 低耦合

耦合:不同模块之间的关联 （解耦：降低不同模块间的耦合度）

设计模式:（了解:单例模式\装饰器模式\观察者模式）

①创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

②结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

③行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

四、**Angular**（版本:6.0）

ng生态圈:

ng自带:forms http router ..

第三方:ionic AngularMaterial ...

1、概述

框架：是一套优秀的代码集合，需要学习框架的理念，再按照框架的规则和语法要求来使用框架 来完成快速开发

库：专注于一个点

what?

angular(ng)是一个是09年 由Google所推出的js框架，采用模块化的开发方式，在各个模块中来封装组件、服务、指令、管道等来实现业务功能

where?

中大型、超大型SPA(单页面应用 single page application)

强调与vue的开发理念的不同:

vue整个应用全是组件

angular整个应用，可以分两层，先有模块，再有组件

why?

①angular可以面对更复杂的开发需求

②采用**ts**(TypeScript 微软所推出：是es6的超集) ts是一种强类型约束的语言

③angular有着非常丰富的类

④强大的cli工具

how?

(1)搭建环境

npm install -g @angular/cli

ng new my-app

cd my-app

ng serve --open

具体搭建细节:

①确认node版本

如果node版本高于8，控制面板--》程序--》卸载(卸载你当前安装的node高版本程序)；双击打开e盘/software/node-v8.11.1-x64.msi程序，或者去ftp中下载node，安装指定的node程序

②将forStu.zip放到 C:\xampp\htdocs\codes,解压缩到当前文件夹

③将C:\xampp\htdocs\codes\forStu\ng\Angular.rar 复制粘贴到C:\xampp\htdocs\codes\ng目录

④在使用vscode中的内置终端时，如果发现powershell有问题，可以这样

windows+r -->cmd

cd C:\xampp\htdocs\codes\ng\myngapp

启动流程:

启动开发服务器: **npm start** （localhost:4200）

src/main.ts-->启动的主程序

src/app/app.module.ts-->指定要启动的根模块

src/app/app.component.ts-->指定要启动的根组件

(2)开发方式

采用组件来封装视图

采用管道来封装筛选功能

采用服务来实现方法或者数据的共享

采用指令来增强html功能

。。。

2、基础语法

2.1 **组件的创建和使用**

回顾:

Vue

SFC(single file component)： \*\*.vue

全局组件:

Vue.component('login',{template:""})

局部:

Vue.component('login',{components:{

'register':{

template:``

}

}})

<login></login>

组件:可被反复使用的，带有特定功能的视图

angular组件的创建：

在ng的工程中，借助于cli工具

**ng g component demo01**

g：generate生成

demo01：需要创建的组件名

组件的调用:

<app-demo01></app-demo01>

2.2 双花括号

interpolation/mustache/双花括号

<any>{{expression}}</any>

2.3 常用指令 (重点)

(1)循环指令

vue:

<any v-for="tmp in myList"></any>

<any v-for="(tmp,index) in myList"></any>

angular:

<any \*ngFor="let tmp of myList"></any>

<any \*ngFor="let tmp of myList;let myIndex=index"></any>

练习:(:- 15:47)

①完成组件的创建和使用

ng g component demo02-lianxi

到app.component.html中调用

②在demo02-lianxi的组件类中 准备一个对象数组(5个对象) stuList

每个学生对象包含以下属性:name/score

③在demo02-lianxi的html将数据渲染

ul/li 列表项显示学生的姓名

(2)选择指令

<div \*ngIf="expression"></div>

expression表达式执行的结果为真，就会显示，为假就会从dom移除

//实现多重分支判断

<div [ngSwitch]="answer">

<p \*ngSwitchCase="'a'">正确答案是A</p>

<p \*ngSwitchCase="'b'">正确答案是B</p>

<p \*ngSwitchCase="'c'">正确答案是C</p>

<p \*ngSwitchCase="'d'">正确答案是D</p>

<p \*ngSwitchDefault>请输入正确的答案</p>

</div>

score = 70

<div [ngSwitch]="true">

<p \*ngSwitchCase="score>90">A</p>

<p \*ngSwitchCase="score>=80 && score<90">B</p>

<p \*ngSwitchCase="score>=60&&score<80">C</p>

<p \*ngSwitchDefault>D</p>

</div>

(3)事件绑定、属性绑定、双向绑定

事件绑定:

<any (event)="handleEvent**(**myIdex**)**"></any> (a为实参)

触发了event事件 就会执行handleEvent这个方法（注意事项：事件处理函数携带括号）

通过绑定的事件，函数传递参数：

handleEvent(index){

对i做操作...

this.myList.splice(index,0) //删除下标为i的元素

}

属性绑定:

<any [myProp]="expression"></any>

将expression表达式执行的结果 给当前的标签的myProp属性去用

双向数据绑定:

方向1: 将数据绑定到视图

方向2: 将视图中用户操作的结果 绑定到数据

vue: <input v-model="myName"/>

angular:

准备工作:

①找到app.module.ts

②引入表单模块

import {FormsModule} from '@angular/forms'

③调用表单模块

@NgModule({

imports:[

FormsModule

]

})

<表单元素 [(ngModel)]="变量"></表单元素>

|  |
| --- |
| \*\*\*.ts中：  export class Demo04Component implements OnInit {  // 动态添加数据  myCount=''  myList1=[] //储存的是add之后保存的数据  constructor() { }  ngOnInit() {  }  // input输入值，动态添加到myList数组中  addDemo(){  this.myList1.push(this.myCount)  }  delDemo(index){  console.log(index)  this.myList1.splice(index,1);  }  } |
| \*\*\*.html中：  <!-- 动态添加 -->  <input type="text" [(ngModel)]='myCount'>  <button (click)='addDemo()'>add</button>  <ul>  <li \*ngFor='let tmp of myList1;let myIndex=index'>  <span>{{tmp}}</span>  <button (click)='delDemo(myIndex)'>delete</button>  </li>  </ul> |

**day02**

**2.4 管道(Pipe)**

vue中的过滤器 在ng中叫做管道

回顾:

<any>{{expression|myFilter}}</any>

<any>{{expression|myFilter1 | myFilter2}}</any>

<any>{{expression|myFilter(a,b)}}</any>

管道的本质：

就是一个有参数有返回值的方法

管道的定位:

将在类中的定义的数据 经过筛选、过滤、格式化更好的显示在页面

Angular中管道:

<any>{{expression | myPipe}}</any>

<any>{{expression | myPipe:arg1:arg2}}</any>

<any>{{expression | myPipe1 | myPipe2}}</any>

内置管道:

currency货币

number:'a.b-c'控制数字中小数点前后的位数(小数点前最少a位；小数点后最少b位，最多c位)

slice截取

percent百分比

json (json序列化)

uppercase/lowercase(转换为大写/小写)

自定义管道:

0-->女 1-->男

创建:

**ng g pipe sex**

配置:

找到src/app/sex.pipe.ts

在transform方法中，使用value和args，根据管道本身的定位处理，最后将处理后的结果进行返回

|  |
| --- |
| /\*\*  \* 功能：将数字0/1转换为性别女/男  \* @param value 是竖杠前，表达式执行的结构  \* @param args 是第一个参数  \*/  transform(value: any, args?: any): any {  console.log(value,args)  if(value==0){  return '女'  }else{  return '男'  }  } |

调用:

<any>{{expression | sex}}</any>

**2.5 装饰器和元数据**(了解)

装饰器decorator:

用来告诉angular当前类什么类型

@Component 组件

@Pipe 管道

@NgModule 模块

@Injectable ... 服务(注射)

元数据 metaData:

用来告诉angular如何来处理当前的类

@Component{

selector:'demo01'

}

**3、组件间通信**

3.1 props down (父-->子)

回顾:vue

①发送

<my-header myTitle="购物车"></my-header>

②接收

Vue.component('my-header',{

//①允许接收myTitle属性传来的值

//②将值存在myTitle的变量

props:['myTitle'] //this.myTitle

})

angular: (父-->子 Input)

①发送(父组件)

<my-header myTitle="购物车"></my-header>

传递的参数为变量时：

<my-header myTitle="{{myList}}"></my-header>

<my-header [myTitle]="tmp"></my-header>

②接收(子组件)

import { Input } from '@angular/core'

export class Demo07Component implements OnInit {

// 接收myTitle属性所传来的值，将值存在myTitle变量中

@Input() myTitle=''

constructor() { }

ngOnInit() {

console.log(this.myTitle) //使用this.myTitle

}

}

3.2 events up （子-->父 Output EventEmitter $event）

①绑定(父组件)

handleEvent(msg){

console.log(msg)

}

父组件html中放入子组件：

<my-header (myEvent)="handleEvent($event)"></my-header>

$event：angular中所规定的特殊的变量，存储的就是事件触发时传递的数据(实参)

②触发(子组件)

import { Output,EventEmitter } from '@angular/core';

export class Demo08HeaderComponent implements OnInit {

// 准备可以触发myEvent事件的事件触发器

@Output() myEvent=new EventEmitter();

constructor() { }

ngOnInit() { }

// 点击按钮传递false，通过mytEvent事件传递给父组件

handleClick(){

this.myEvent.emit(false)

}

}

子组件html中：

<button (click)='handleClick()'>注销登录</button>

**4、服务**

服务在angular的定位:

将不同的组件之间要共同用到的数据、方法封装在服务类中，以方便复用

比较常见的服务：

基于http请求，基于log管理的，基于心跳服务的。。。。

4.1 服务的封装

**ng g service log**

在服务类的文件中(log.service.ts)，实现要封装的方法

export class LogService {

isDev=true;

constructor() { }

// 在服务类中，封装自己需要的数据和方法

printSth(msg){

if(this.isDev){

console.log(msg)

}

}

}

4.2 服务的调用(组件中)

①引入

import { LogService } from '../log.service'

②实例化——将一个类变成一个对象(实例)的过程

方法一：myService=new LogService();

方法二(推荐)：constructor(private myService : LogService){ }

③调用

this.myService. printSth('abc') //调用服务类中的数据和方法

* 封装的好处：

①封装是为了复用，提高代码的复用率

②提高代码的开发效率

③提高代码质量

**day03**

**5、网络通信**

回顾:

①ajax ②$.get/post ③axios ④promise ⑤async/await ⑥fetch

angular:

基于rxjs（由微软所推出的一种处理异步请求的处理方案）这种方案来处理网络通信的

①准备

app.module.ts中：

import { HttpClientModule } from '@angular/common/http'

@NgModule({

imports:[HttpClientModule]

})

②发起网络请求(组件中)

import { HttpClient } from '@angular/common/http'

实例化：

constructor(private myHttp:HttpClient){}

this.myHttp.get/post( url ).subscribe((result : any )=>{ }) // result : any——result为任意类型(不然会有红杠)

**6、路由**

路由的本质:建立起一种 路由地址和组件(视图)之间的映射关系

**ng g** component/service/pipe/**module** \*\* **--routing**

6.1 基础用法

选择的方案：单独的创建一个自己的模块(负责路由设置)

①创建一个自定义的模块(路由)

ng g module my-module --routing

②配置路由词典信息

指定一个容器(路由出口：组件显示的地方):

<router-outlet></router-outlet>

创建组件

ng g componet demo13-login

配置路由

my-module-routing.module.ts中：

@NgModule中：forChild --改为-->forRoot

添加路径：

const routes:Routes = [

{path:"",component:\*\*}, //默认显示的组件http://localhost:4200

{path:'check/:lid',component:Demo15DetailComponent} //需要接收参数的组件

{path:'\*\*',component:NotFoundComponent} //没有找到路径显示的组件

]

③根模块中调用自定义的模块

app.module.ts

import { MyModuleRoutingModule } from './my-module/my-module-routing.module'

@NgModule({

imports:[MyModuleRoutingModule]

})

* 注意事项:

①设置路由信息时，路由地址无需以 / 开头，写上/，是会报错的

②如何需要设置路由地址为空要访问谁，指定path为""(空字符串)

路由匹配规则:

①路由地址为空 path:""

②指定某一个确定的路由地址 path:"login"

③异常处理 path:"\*\*"

④顺序，应该将先设置为空的，再设置具体路由，最后设置异常情况

**6.2 跳转和传参**

跳转(2中方式):

①编程式导航(编程式，需要跳转的组件中)

import { Router } from '@angular/router'

constructor(private myRouter:Router){ }

this.myRouter.navigateByUrl('register')

②属性来指定跳转

<any routerLink="/detail"></any>

传参:

①配置接收方的路由地址

detail --> detail/:id

②发送

this.myRouter.navigateByUrl("detail/10")

this.myRouter.navigateByUrl("detail/"+this.id) //id为变量时

<any routerLink="/detail/10"></any>

<any routerLink="/detail/{{id}}"></any>

<any [routerLink]=**"**'/detail/'+id**"**></any>

③接收(需要接收参数的组件中)

//vue: this.$route.params.id

import { ActivatedRoute } from '@angular/router'

constrcutor(private myRoute:ActivatedRoute){}

this.myRoute.params.subscribe((result)=>{

//result.id

})

练习:(:- 17:40)

①准备两个组件 demo15-index demo15-detail

②各自指定一个可以访问的路由地址

index: demo15-index

check: demo15-detail

③demo15-index

第一步：发请求http://localhost:8080/product/list

将请求到的data保存，

第二步：显示ul，li显示商品的title

第三步: li支持以routerLink的 跳转到路由地址为check的页面，同时将当前商品的lid作为参数发送

<li \*ngFor="let tmp of \*\*" routerLink="/check/{{tmp.lid}}">

{{tmp.title}}

</li>

④demo15-detail

接收参数，显示在h1

ActivatedRoute

|  |
| --- |
| demo15-index.js中  import { HttpClient } from '@angular/common/http'  export class Demo15IndexComponent implements OnInit {  myList=[]  constructor(private myHttp:HttpClient) { }  ngOnInit() {  var url='http://127.0.0.1:8080/product/list'  this.myHttp.get(url).subscribe((result:any)=>{  console.log(result.data)  this.myList=result.data;  })  }  } |
| demo15-index.html中：  <ul>  <li \*ngFor='let tmp of myList' routerLink='/check/{{tmp.lid}}'>{{tmp.lid}}：{{tmp.title}}</li>  </ul> |
| demo15-detail.js中  import { ActivatedRoute } from '@angular/router'  export class Demo15DetailComponent implements OnInit {  myId=''  constructor(private myRoute:ActivatedRoute) { }  ngOnInit() {  this.myRoute.params.subscribe((result)=>{  console.log(result.lid)  this.myId=result.lid  })  }  } |
| demo15-detail.html中  <p>拿到了demo15-index的参数：{{myId}}</p> |

**day04**

**6.3 路由嵌套**

概念:在一个spa，某一个路由地址对应的组件，组件内部，又需要动态的加载其它的组件（一个路由对象中又嵌套了其它的路由对象）

ng的路由嵌套的套路和vue一模一样

比如email，需要嵌套inbox和outbox

①给email指定一个容器(路由出口)

<router-outlet></router-outlet>

②配置路由

{

path:'email',

component:EmailComponent,

children:[

{path:'inbox',component:\*\*\*},

{path:'outbox',component:\*\*\*},

。。。。

]

}

**6.4 路由守卫**

路由守卫是用来指定一个路由对象是否允许被访问。

在angular中，路由守卫都是通过服务来实现的

服务的使用流程:

①创建一个服务类

**ng g service guard**

②在服务类中 实现执行判断的方法

guard.service.ts中：

import { CanActivate } from '@angular/router'

export class GuardService implements CanActivate{

// 准备此服务类中实现一个守卫的功能

canActivate(){

// 做一些计算和判断，返回true允许访问，返回false不准访问

return false;

}

}

③调用服务

my-module-routing.module.js中

import { GuardService } from '../guard.service' //引入路由守卫服务

const routes: Routes = [

{ path:'admin',

component:Demo17AdminComponent,

canActivate:[ GuardService ]

},

]

* 总结:

①数据

组件间内部通信:

props down(Input)

events up(Output,EventEmitter,$event)

网络通信：

HttpClient(HttpClientModule,subscribe)

路由传参:

ActivatedRoute

②显示

{{ }}

各种指令 \*ngFor/\*ngIf/\*ngSwitchCase..

管道 （筛选、过滤、格式化）

③交互

()

[(ngModel)]

[]

④路由

路由模块 可以做导航和传参

**五、Ionic的使用**

Ionic是基于angular的ui组件库

(css-->bootstrap，vue-->mintui/mui)

推荐一些angular建议的ui组件库:

Material:https://material.angular.cn

NgZorro:https://ng.ant.design/docs/introduce/zh

Ionic:定位于移动端的ui

移动端的3种常见的开发形式:

①WebApp: 运行在手机浏览器，采用web前端技术来实现的网页;跨平台

②NativeApp: 运行在手机OS(Operating System)， 采用android中的java/kotlin,采用iOS中的 oc/swift,调用Google/Apple所提供的SDK(software development kit)中的控件来完成app的开发；性能

③HybridApp: 运行在手机OS(Operating System)

phonegap/AppCan/Dcloud/369Cloud/Flutter/....

1、ionic概述

what?

ionic是一个ui组件库（在后续的升级版本中，也在支持以cdn的方式，支持vue、react、js，但是这个新版本手册不够友好，还有些问题）

ionic = angular+ionicModule+cordova

why?

ionic有着非常多的组件支持

ionic有着强大的cli支持

ionic基于angular，有着非常多的受众

。。。

when?

使用ionic实现webApp，也可以实现hybridApp

how?

①复制

C:\xampp\htdocs\codes\forStu\ng\ionic模板\myProject.zip 粘贴到

C:\xampp\htdocs\codes\ionic

②解压缩到当前的文件夹

③使用vscode打开了目录C:\xampp\htdocs\codes\ionic\myProject

④启动开发服务器(package.json)

**npm start**

localhost:8100

⑤确认ionic是否可用

windows+r --> cmd -->ionic

如果不可以用:

方案1：npm install -g ionic@3.20.0

方案2：

将C:\xampp\htdocs\codes\forStu\ng\ionic模板\ionic\_cmd.zip解压缩到ionic\_cmd里边

将C:\xampp\htdocs\codes\forStu\ng\ionic模板\ionic\_cmd\node\_modules\.bin目录添加到计算机的环境变量

我的电脑--》右键，查看属性--》高级系统设置--》环境变量--》系统变量中的PATH--》编辑，新建--》确定

环境变量设置完成之后，要关闭cmd窗口，重新打开cmd窗口，输入ionic测试看一下是否可用

2、ionic的基础用法

2.1 创建自己的page

得是在基于ionic所创建的工程中:

**ionic g page demo01**

到根模块中声明:

app.module.ts中：

import { Demo01Page } from '../pages/demo01/demo1'

@NgModule({

declarations:[Demo01Page],

entryComponents:[Demo01Page]

})

2.2 调用page

支持很多中调用的方案:

比如:

app.component.ts中：

import { Demo01Page } from '../pages/demo01/demo01'

rootPage:any = Demo01Page

注意：

当前ionic的cli工具有个小问题：

第一次修改一个page的html，有可能不会及时的更新；试着将开发服务器重启

3、ionic的基础组件

3.1 **button**

ion-button

color 颜色 (src/theme/variables.scss 修改$colors颜色设置 是很灵活的)

small/large 小/大

round 圆角

outline 边框

clear 只保留按钮内的内容

block 块级显示

3.2 **list**

item-start/item-end

普通列表

<ion-list>

<ion-item>

①icon列表

<ion-icon item-start name='wifi'></ion-icon>

item-start 在左侧，在右侧(end)

name='wifi' Wi-Fi图标

②头像列表(联系人)

<ion-avatar>

<img/>

</ion-avatar>

③缩略图列表

<ion-thumbnail item-start text-wrap>

<img/>

<p></p>

<button item-end ion-button clear> //右侧显示购物车图标

<ion-icon name='cart'></ion-icon>

</button>

</ion-thumbnail>

</ion-item>

</ion-list>

**下拉**刷新

①标签要写在ionContent的第一个子元素位置

<ion-refresher></ion-refresher>

②绑定事件

<ion-refresher (ionRefresh)="handleRefresh($event)"></ion-refresher>

③手工的通知ionRefresher可以结束刷新动作的

handleRefresh(myRefresher){

//数据操作

//结束刷新动作

myRefresher.complete()

}

**上拉**加载更多

①标签要写在ionContent的最后一个子元素的位置

②绑定事件

<ion-infinite-scroll (ionInfinite)="loadMore($event)">

<ion-infinite-scroll-content loadingText="">

</ion-infinite-scroll-content>

</ion-infinite-scroll>

③在事件处理函数中 处理数据 并结束加载更多的动画的效果

loadMore(myInfinite){

//计算

myInfinite.complete()

}

|  |
| --- |
| demo03.ts  export class Demo03Page {  myList=[]  constructor(public navCtrl: NavController, public navParams: NavParams) {  }  // 中间的商品  ionViewDidLoad() {  console.log('ionViewDidLoad Demo03Page');  for(var i=0;i<50;i++){  this.myList.push('商品'+i)  }  }  // 下拉事件  handleRefresh(myRefresher){  console.log('准备完成刷新的操作')  setTimeout(()=>{  for(var i=0;i<5;i++){  var num=Math.floor(Math.random()\*100);  this.myList.unshift(num)  }  // 结束刷新的动画效果  myRefresher.complete()  },2000)  }  // 上拉事件  loadMore(myInfinite){  console.log('准备加载更多操作')  setTimeout(()=>{  for(var i=0;i<5;i++){  var num=Math.floor(Math.random()\*100);  this.myList.push(num)  }  // 结束刷新的动画效果  myInfinite.complete()  },2000)  }  } |
| demo03.html中：  <ion-content padding>  <!-- 下拉到一定程度松手后默认会触发ionRefresh事件 -->  <ion-refresher (ionRefresh)='handleRefresh($event)'>  <ion-refresher-content pullingText='下拉刷新' refreshingText='正在刷新'></ion-refresher-content>  </ion-refresher>  <!-- 实现列表，商品列表 -->  <ion-list>  <ion-item \*ngFor='let tmp of myList'>{{tmp}}</ion-item>  </ion-list>  <!—上拉加载更多 -->  <ion-infinite-scroll (ionInfinite)='loadMore($event)'>  <ion-infinite-scroll-content loadingText='正在加载更多...'></ion-infinite-scroll-content>  </ion-infinite-scroll>  </ion-content> |

练习:

①完成页面的创建和调用

demo04-lianxi

（先声明再调用）

②在组件类加载完之后 请求

http://localhost:8080/product/list，

将数据保存，将title显示在ionList的ionItem中

**day05**

3.3 **slides**(轮播图)

基础用法:

<ion-slide**s** \*ngIf='myList.length>0'> // 先对myList做判断，不然下面遍历无myList会报错

<ion-slide \*ngFor='let tmp of myList'></ion-slide>

</ion-slides>

slides常见的属性:

autoplay=2000 自动播放，间隔2s

loop 循环

pager 显示指示器

paginationType="bullets/fraction/progress" 指示器样示

effect='fade/cube/flip/slides' 动画展示效果

speed=2000(用来指定动画持续的时间)

3.4 **ionGrid**(栅格布局)

ionic中自带的栅格系统是基于前端的flexbox来实现，可以帮助非常方便的来实现自定义布局

基础用法:

<ion-grid>

<ng-container \*ngFor='let tmp of myList;let myIndex=index'> //元素不能放2个执行语句，增加一个容器ng-container来执行语句(先循环，放外层)，不影响结构

<ion-row \*ngIf='myIndex%2==0'>

<ion-col>

{{tmp}}

</ion-col>

<ion-col>

{{myList[myIndex+1]}}

</ion-col>

</ion-row>

</ng-container>

</ion-grid>

支持的常见属性:

<ion-row align-items-center></ion-row>

行中设置列内容的纵向对齐：

align-items-top/center/end

<ion-col align-self-center></ion-col>

某一列的纵向对齐:

align-self-top/center/end

<ion-row justify-content-center></ion-row>

行中列的横向对齐处理

justify-content-top/center/end

某一列内容的横向对齐 依然可以使用text-align来设置

指定列宽(总共12份)：col-\*

指定偏移量：offset-\*

将某一个向右推、左拉：push-\* pull-\*

**3.5 ionScroll 支持滚动容器**

ionScroll是一个容器，可以配置为支持纵向滚动

<ion-scroll style="height:200px" scrollY>

</ion-scroll>

横向滚动：

<ion-scroll style='height:200px;white-space: nowrap' scrollX>

<ion-card style='display: inline-block' \*ngFor='let tmp of myList'>

...

</ion-card>

</ion-scroll>

**3.6 card卡片**

卡片只是为了更好的展示信息给用户

<ion-card>

<ion-card-header></ion-card-header>

<ion-card-content>

<ion-card-title></ion-card-title>

内容Lorem....

</ion-card-content>

</ion-card>

**3.7 各种交互窗口**

toast(以通知的形式 告诉用户一个操作的结果)

create({

message:"",

duration:2000,

position:'top/middle/bottom'

})

actionSheet (让用户从多个选项中做选择的)

create({

title:'',

buttons:[

{text:'朋友圈',handler:()=>{},role:'cancel/destructive'}

]

})

loading (LoadingController 加载中的窗口)

create({

content:'正在刷新...',

duration:2000

})

alert(AlertController 警告、确认、输入提示)

套路:

①引入类

LoadingController

②实例化

constructor(private loadingCtrl:LoadingController){ }

③创建窗口

var myLoading = this.loadingCtrl.create()

④显示窗口

myLoading.present()

|  |
| --- |
| \*\*.ts  import { Component } from '@angular/core';  import { ToastController,LoadingController,ActionSheetController,AlertController,IonicPage, NavController, NavParams } from 'ionic-angular';  export class Demo08WindowsPage {  constructor( //实例化  public navCtrl: NavController,  public navParams: NavParams,  private toastCtrl:ToastController,  private loadingCtrl:LoadingController,  private asCtrl:ActionSheetController,  private AlertCtrl:AlertController) {  }  // 警告、确认、输入  showAlert(){  var myAlert=this.AlertCtrl.create({  title:'有风险',  message:'此操作有资金风险',  inputs:[  {type:'text',placeholder:'请输入金额'}  ],  // data是一个数据，获取inputs数组中各输入的值  buttons:[  {text:'确认',handler:(data)=>{  console.log('已支付',data[0])  }},  {text:'取消'},  ]  })  myAlert.present();  }  //从多个选项做选择  showActionSheet(){  var myAS=this.asCtrl.create({  title:'分享到',  buttons:[  {text:'微信好友',handler:()=>{  console.log('微信好友分享成功')  }},  {text:'朋友圈'},  {text:'微博'},  {text:'删除',role:'destructive'}, //role添加样式  {text:'取消',role:'cancel'},  ]  })  myAS.present()  }  // 展示一个操作的结果  showToast(){  var myToast=this.toastCtrl.create({  message:'加入购物车失败',  duration:2000  })  // 显示  myToast.present()  }  // 展示刷新的效果  showLoading(){  // ①创建  var myLoading=this.loadingCtrl.create({  content:'正在刷新',  duration:2000  })  // ②显示  myLoading.present()  }  } |

**3.8 模态框**

模态框显示Demo08WindowsPage页面：

// 引入 实例化 创建 显示

import { Demo08WindowsPage } from '../demo08-windows/demo08-windows';

constructor(private modalCtrl:ModalController) { }

// 显示模态窗口

showModal(){

// 此时的create方法，需要最少一个参数，这个参数是你模态窗内部要加载的一个页面类

var myModal=this.modalCtrl.create(Demo08WindowsPage);

myModal.present();

}

关闭模态框：

import { ViewController,IonicPage, NavController, NavParams } from 'ionic-angular';

constructor(private viewCtrl:ViewController) { }

// 关闭模态窗口(绑定事件)

closeModal(){

this.viewCtrl.dismiss()

}

**3.9 表单**

ionic中表单是靠ionList和ionItem来处理

<ion-list radio-group>

<ion-item>

<ion-label floating>用户名</ion-label>

<ion-input></ion-input>

</ion-item>

<ion-item>

<ion-label stacked>密码</ion-label>

<ion-input type='password'></ion-input>

</ion-item>

<ion-item>

<ion-label>记住密码</ion-label>

<ion-checkbox></ion-checkbox>

</ion-item>

<ion-item>

<ion-label>女</ion-label>

<ion-radio></ion-radio> //list加radio-group列为一个组，才能使用

</ion-item>

<ion-item>

<ion-label>男</ion-label>

<ion-radio></ion-radio>

</ion-item>

<ion-item>

表单元素...

</ion-item>

</ion-list>

**3.10 导航、传参**

ionic已经将angular集成到模板项目了，但是并没有采用angular官方的路由处理，提供了一种更简洁的方案

导航:

方案1：编程式导航

import { Demo12DetailPage } from '../demo12-detail/demo12-detail';

this.navCtrl.push(Demo12DetailPage)

方案2: navPush

import { Demo12DetailPage } from '../demo12-detail/demo12-detail';

detail = Demo12DetailPage

页面html中

<any [navPush]="detail"></any>

传参：

(1)发送

①this.navCtrl.push(Demo12DetailPage,{id:10})

②<any [navPush]="detail" [navParams]="{id:10}"></any>

(2)接收参数

this.navParams.get('id')

练习:

①完成两个页面的创建、声明、调用(默认显示demo13-login)

demo13-login demo13-main

②设置demo13-login

模板:实现一个表单：两个输入框 一个登录按钮

功能：点击登录按钮 以编程式导航，跳转到demo13-main,同时将用户名发送过去

③设置demo13-main

模板: h1：‘欢迎\*\*登录’

功能：接收跳转时传来的参数 保存，显示在h1

|  |
| --- |
| main.ts  import { Demo13LoginPage } from '../demo13-login/demo13-login';  export class Demo13MainPage {  uname=''  pwd=''  myLogin(){  this.navCtrl.push(Demo13LoginPage,{  uname:this.uname,  pwd:this.pwd  })  }  } |
| main.html  <ion-content padding>  <ion-list>  <ion-item>  <ion-input [(ngModel)]='uname'></ion-input>  </ion-item>  <ion-item>  <ion-input type='password' [(ngModel)]='pwd'></ion-input>  </ion-item>  </ion-list>  <button ion-button color='danger' style='width:100%' round (click)='myLogin()'>登录</button>  </ion-content> |
| login.ts  ionViewDidLoad() {  var uname=this.navParams.get('uname')  var pwd=this.navParams.get('pwd')  console.log('用户名：'+uname,'密码：'+pwd)  } |

**3.11 tabs(底部导航栏)**

tabs是在移动端经常用到的组件：工具条、导航条、选项卡

使用原则：给每一个tab指定要显示的页面

<ion-tabs>

<ion-tab tabIcon="" tabTitle="" root=""></ion-tab>

<ion-tab></ion-tab>

</ion-tabs>

tabTitle：文本名字

tabIcon：导航图标

[root]：导航到的组件(属性绑定)

具体:

①创建各个页面（每一个tab在按下时要显示的页面）

②引入对应的页面类，保存在类的内部成员变量

③调用ionTabs，设置每一个tab的root、tabTitle、tabIcon

|  |
| --- |
| \*\*.ts  import { Component } from '@angular/core';  import { IonicPage, NavController, NavParams } from 'ionic-angular';  import { Demo10ModalPage } from '../demo10-modal/demo10-modal';  import { Demo11FormsPage } from '../demo11-forms/demo11-forms';  import { Demo12DetailPage } from '../demo12-detail/demo12-detail';  export class Demo14TabsPage {  tabIndex=Demo10ModalPage  tabCart=Demo11FormsPage  tabPerson=Demo12DetailPage  }  } |
| \*\*.html(只写导航栏，其他页面直接显示在上面)  <!-- 点击不同的tab时，加载不同的页面类 -->  <ion-tabs>  <ion-tab tabTitle='主页' tabIcon='home' [root]='tabIndex'></ion-tab>  <ion-tab tabTitle='购物车' tabIcon='cart' [root]='tabCart'></ion-tab>  <ion-tab tabTitle='个人' tabIcon='person' [root]='tabPerson'></ion-tab>  </ion-tabs> |

项目中：

|  |
| --- |
| 详情页加入购物车：  addToCart(){  var myId = this.navParams.get('id')  var url='http://localhost:8080/cart/add?buyCount=1&lid='+myId;  this.myHttp.get(  url,  {withCredentials:true} //进行session验证  ).subscribe((result:any)=>{  console.log(result)  if(result.code==300){  // 未登录，需要跳转到到登录页面  this.navCtrl.push(LoginPage)  }else if(result.code==200){  //添加成功，显示一个toast  var myToast=this.toastCtrl.create({  message:'加入购物车成功',  duration:2000  })  myToast.present();  }  })  } |