跨平台开发Ionic4 第二节

1. 登录页面需要准备数据双向绑定：
2. [(ngModel)]="mobile"
3. [(ngModel)]="password"
4. 登录按钮按下的一瞬间，考虑到用户体验，需要有提示loading
5. 翻阅api：<https://ionicframework.com/docs/components/#loading>
6. 基本先照抄
7. 需要考虑到代码复用，弹框提示组件:Toast和加载组件：loading以及我们的登录注销等等，秀需要进行组件化封装，方便后期维护;
8. 先创建一个provider=>rest
9. Rest.ts准备工作如下：

import { Observable } from 'rxjs/Observable';

import { Http,Response} from '@angular/http';

import 'rxjs/add/operator/map';

import 'rxjs/add/operator/catch';

1. app.module.ts中需要做好准备工作：

(1):

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

(2):

imports: [

...

HttpModule,//全局需要导入Http

...],

1. 需要一堆URL：

//feed

private apiUrlFeeds = 'https://imoocqa.gugujiankong.com/api/feeds/get';

//account

private apiUrlRegister = 'https://imoocqa.gugujiankong.com/api/account/register';

private apiUrlLogin = 'https://imoocqa.gugujiankong.com/api/account/login';

private apiUrlUserInfo = 'https://imoocqa.gugujiankong.com/api/account/userinfo';

private apiUrlUpdateNickName = 'https://imoocqa.gugujiankong.com/api/account/updatenickname';

private apiGetUserQuestionList = "https://imoocqa.gugujiankong.com/api/account/getuserquestionlist";

//question

private apiUrlQuestionSave = 'https://imoocqa.gugujiankong.com/api/question/save';

private apiUrlQuestionList = 'https://imoocqa.gugujiankong.com/api/question/list?index=1&number=10';

private apiUrlGetQuestion = "https://imoocqa.gugujiankong.com/api/question/get";

private apiUrlGetQuestionWithUser = "https://imoocqa.gugujiankong.com/api/question/getwithuser";

private apiUrlAnswer = "https://imoocqa.gugujiankong.com/api/question/answer";

private apiUrlSaveFavourite = "https://imoocqa.gugujiankong.com/api/question/savefavourite";

//notification

private apiUrlUserNotifications = "https://imoocqa.gugujiankong.com/api/account/usernotifications";

1. 在app.mudule.ts中引入
2. import { HttpModule} from '@angular/http';
3. 在rest服务中进行定义

private getUrlReturn(url: string): Observable<string[]> {

return this.http.get(url)

.map(this.extractDate)

.catch(this.handleError);

}

//处理接口返回的数据,处理成json格式

private extractDate(res:Response){

let body =res.json();

return JSON.parse(body)|| {};

}

//处理请求中的错误，考虑了各种情况的错误处理并在console.log中显示error

private handleError(error:Response | any){

let errMsg:string;

if(error instanceof Response){

const body =error.json()||'';

const err = body.error || JSON.stringify(body);

errMsg = `${error.status} - ${error.statusText || ''} ${err}`;

}else{

errMsg = error.message?error.message:error.toString()

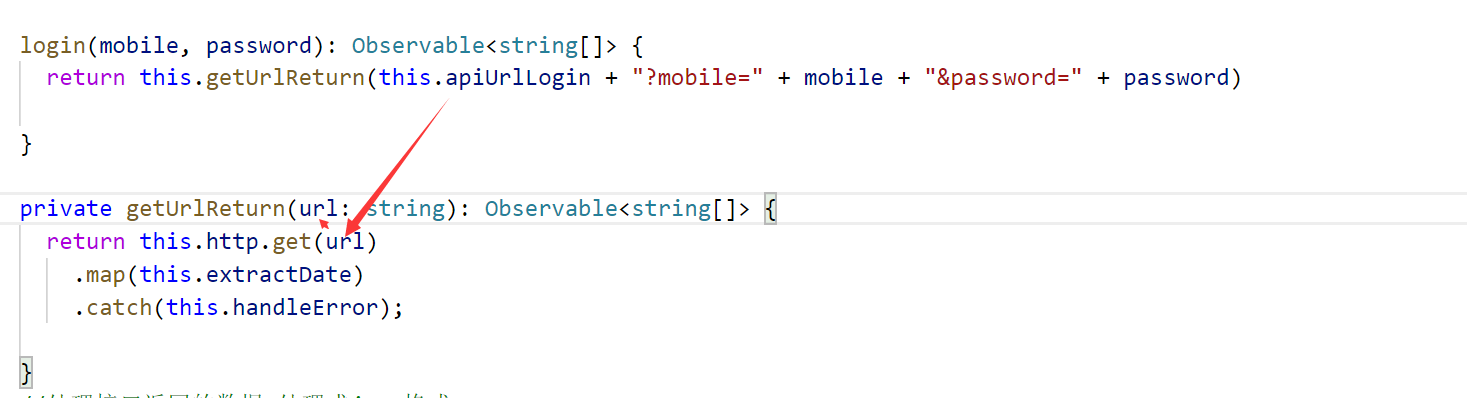
}

console.error(errMsg);

return Observable.throw(errMsg);

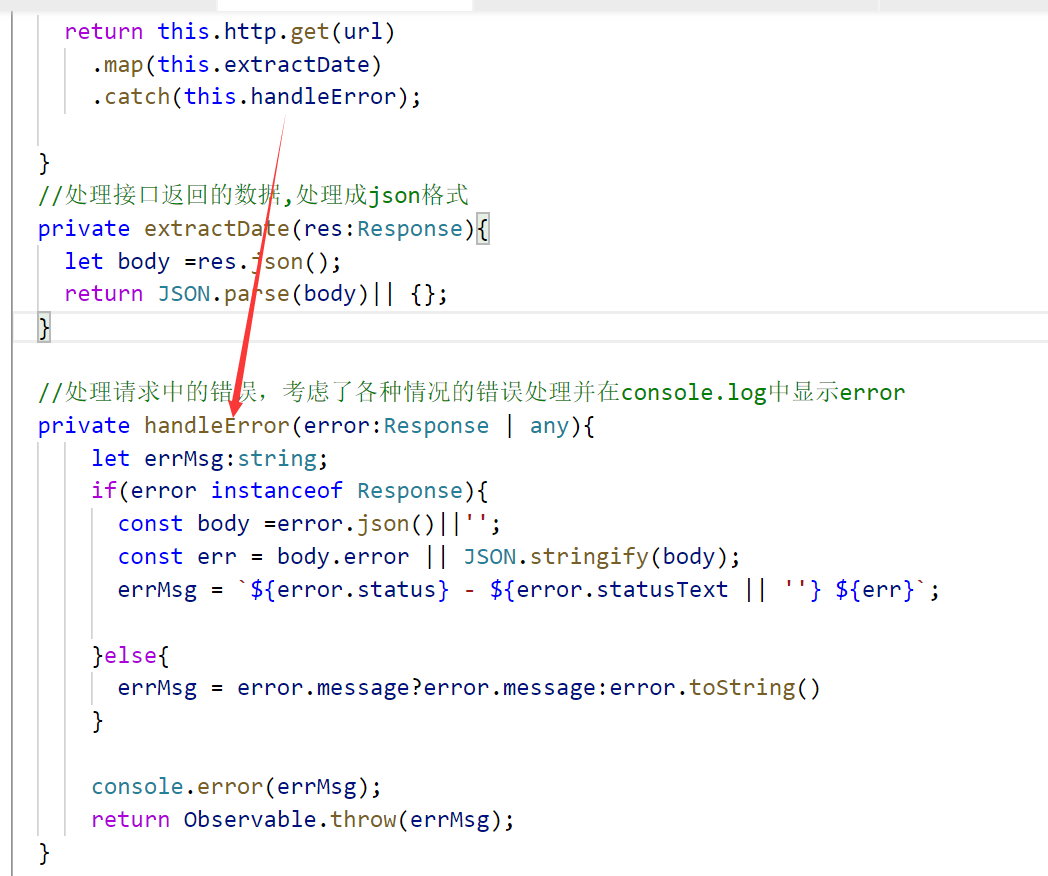
}

1. 解释如图：



1. 

extractDate函数则是封装了数据处理，不管是什么数据，一定是至少是对象类型

1. 

handleError这个函数是专门进行数据错误处理

1. 回归到morepage
2. loading组件只需要一直在读取数据之前转圈即可，所以去除我们的定时器
3. 数据提交需要借助strong服务，ionic已经有现成的了：

App.mudule.ts中需要引入：

import { IonicStorageModule } from '@ionic/storage';

imports: [

....,

IonicStorageModule.forRoot()//全局定义Storage模块

],

1. 在当前组件ts文件中：

(1):import { Storage } from '@ionic/storage';

(2):constructor(...,private storage: Storage,...)

1. 登录ts逻辑代码如下:

gologin() {

const loader = this.loadingCtrl.create({

content: "登录中..."

});

loader.present();

this.rest.login(this.mobile,this.password).subscribe(f=>{

if(f["Status"]=="OK"){

//处理登录成功的页面跳转

//你也可以存储接口返回的 token（用户是否真实 app=>安全协议）

this.storage.set('UserId', f["UserId"]);

loader.dismiss();

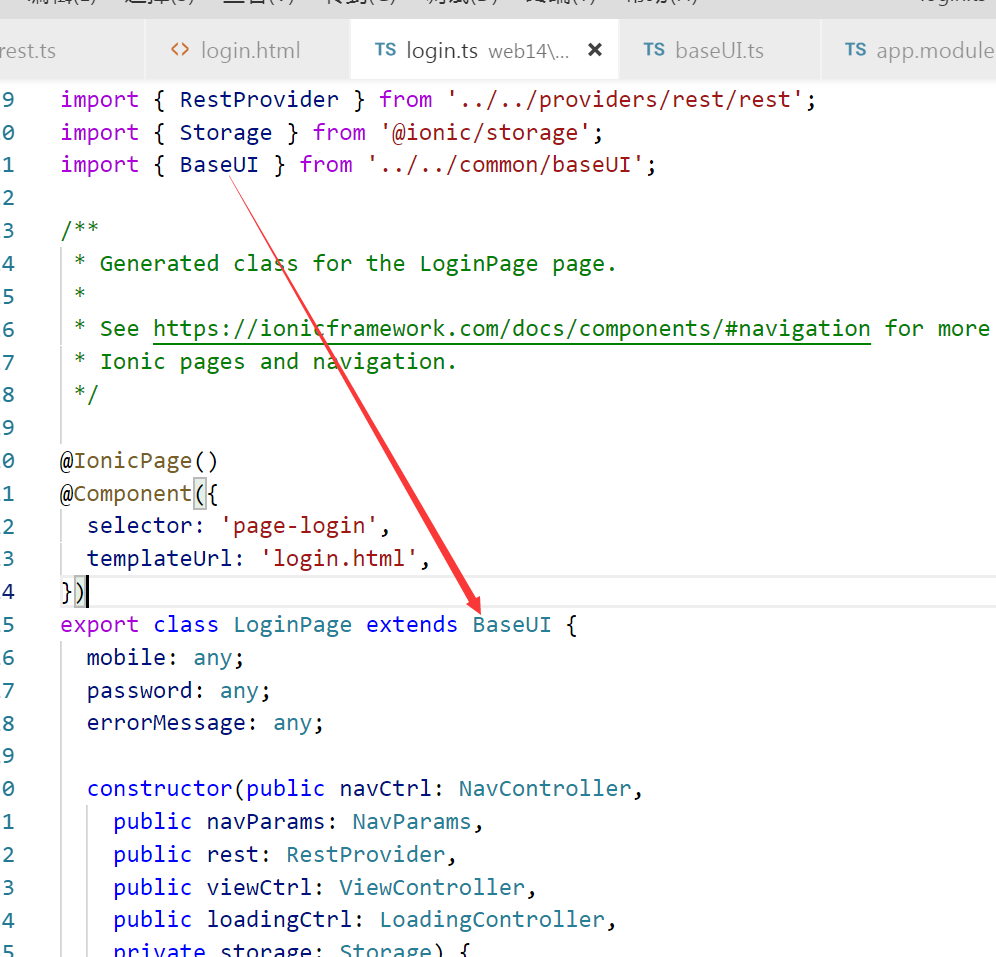
this.dismiss();

}

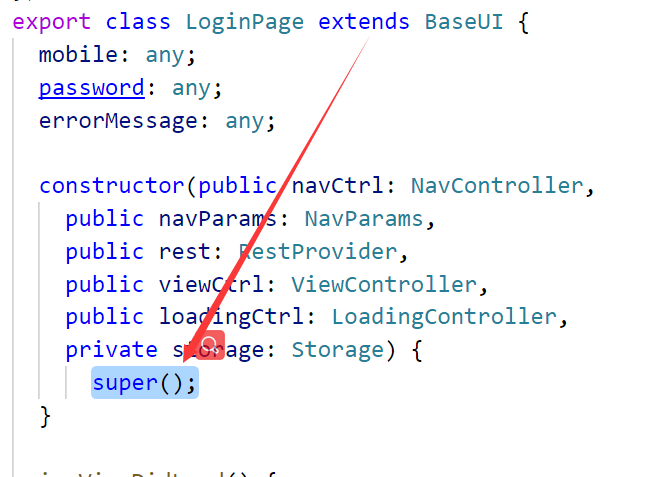
})

}

1. 考虑到我们的loading等组件以后需要代码复用，所以进行封装,在src根目录下新建文件夹common，在里面新建一个baseUI.ts组件，以便日后复用
2. 组件ts中如下引入



1. 解决对象组件引用的报错方法，如图：



1. 引入ToastController