知识点回顾

-1.注意事项：

- 每节课用到的html、css在kaikeba-student -> template中
- 后台数据库导出文件在kaikeba -> data -> xx.sql

0.文档：泛型、接口

- TypeScript：中文文档
- Angular：中文文档

1.路由配置 RouterModule

- 配置：const routes: Routes = [{path:'foo',component:FooComponent}]
- 路由模块 RouterModule.forRoot(routes)

2.模板驱动表单 FormsModule

- 数据模型 LoginUser
- 双向数据绑定 [(ngModel)]="model.phone"
- 校验 maxlength="10"
- 校验状态 #phone="ngModel", phone.errors.maxlength
    - phone.invalid/valid/dirty/pristine/touched/untouched

3.Http请求

- HttpClient模块: 导入到BrowserModule后面
- 依赖注入：使用服务时候只管使用，不管实例化

  constructor(private http: HttpClient)
- 发送请求

  post: this.http.post(url，body，config)

  get: this.http.get(url, config)

  put: this.http.put(url，body，config)

  del: this.http.del(url, config)
- 响应处理: 得到一个Observable

  this.http.post(url，body，config).subscribe(next,error)

4.Express中的Session

- 模块: express-session
- 配置:
    - 放在cookie配置下面
    - app.use(session({...}))
- 使用
    - 赋值req.session.xx = xx;

    - 获取req.session.xx
    - 删除delete req.session.xx
- 将session信息存入数据库：express-mysql-session

用户注册、登录

重构工作1：模块化

1. 提取app路由至独立路由模块AppRoutingModule
    - 创建AppRoutingModule
    - 将全局路由在其中配置
    - 在AppModule中引入该模块
2. 提取登录、注册到用户模块UserModule
    - 创建UserModule和UserRoutingModule：ng g m user --routing
    - 在AppModule中引入该模块
    - 将之前在AppModule中声明的登录注册组件移至UserModule
    - 将之前在AppModule中导入的FormsModule移至UserModule
    - 创建登录注册组件的父组件UserComponent: ng g c user/user
    - 将之前在AppComponent中声明的导航菜单移至UserComponent，并添加一个
    - 在UserRoutingModule配置路由嵌套关系{path:'user',component:UserComponent,children:[...]}
3. 将UserModule中用到的组件、服务、类等等都放在user文件夹中

用户注册：

1.编写模板，添加图形验证码和短信验证码两个输入项；编写模型，创建RegisterUser

2.手机号查重验证：

- 创建异步校验指令 phone-validator，在UserModule模块中声明，

```
// Directive: 这是一个指令
@Directive({
  selector: '[appPhoneValidator]',// 选择器
  providers: [ // 将当前类加入到NG_ASYNC_VALIDATORS中等待ng校验时调用
    {provide: NG_ASYNC_VALIDATORS, useExisting: PhoneValidatorDirective, multi: true}
  ]
})
export class PhoneValidatorDirective {

  constructor(private us: UserService) {}

  // 该校验器需要实现一个validate()方法
  validate(control: AbstractControl): Promise<ValidationErrors | null> |
Observable<ValidationErrors | null> {
    // 得到数据结构是Observable<Result<string>>,
    // 但是当前函数需要Observable<ValidationErrors>

    // 使用map操作符进行数据转换
```

```
    // rxjs编程
    return this.us.verifyPhone(control.value).pipe(
      map((r: Result<string>) => {
        // null说明校验通过
        return r.success ? null : {verifyPhone: true};
      }),
      catchError(e => of({verifyPhone: true}))
    );
  }
```

- 在模板中声明该指令

```
<input name="phone" [(ngModel)]="model.phone" appPhoneValidator>
```

3.验证图形验证码

- 安装trek-captcha: npm i -S trek-captcha

```
const captcha = require('trek-captcha')
router.get('/code-img', async (req, res) => {
    try {
        // token:是数字字母表示形式
        // buffer:是图片数据
        const {token, buffer} = await captcha({size: 4});
        console.log(token);
        // session中存储该token在将来验证时使用
        req.session.codeImg = token;
        // 将图片数据返回给前端
        res.json({
            success: true,
            data: buffer.toString('base64')
        })
    } catch (error) {
        // ...
        console.log(error);
    }
})
```

作业：

1. 复习巩固
2. 预习：rxjs使用、ng中路由及传参、ng中如何自定义组件及参数传递
3. 尝试实现图形验证码校验功能