**20201309-洪毅凡-前端大作业讲解文章**

首先做为一个小组的组长，我想介绍的整个大作业的大体结构，和本次大作业中感悟到做项目的理解。本次大作业的统一课题为开发一个基于web网页学习平台，所以第一件事就是选取技术框架做为支撑，主流web开发都是采用前后端分离的形式去完成，所有我选取了较为熟悉并且主流使用的前端vue框架，与此同时前端出了需要封装的js还需要页面快速成型的技术支持，比如我们所选用的由饿了么开发团队所带来的element-ui，因为是国内团队所开发的，所以中文文档很友好，并且网上资料齐全，接下来就是后端框架支持，我们的后端编程语言最为熟悉的是java，由于也不需要涉及到高并发，集群部署等问题，所以是选用了单项目的springboot框架，在面对数据层的处理上，我们的想法是尝试新技术，避开了对mybatis的使用，而是选取了在mybatis基础上只做增强而不做减少的mybatis-plus。然后数据库按课题给的要求是使用mysql（其实在我看来是使用monggoDB更为合适，关于这个看法我会在接下来进行解释）。在尝试过之前的开发后，我的开发是项目应该想着如何去做好，而不是如何去做完，无意义且机械的增删改查等重复内容应该是要被避免的，而更深层次的去考虑优化等问题，尝试追寻主流的编程思想。所以首先我第一个介绍的是为避免不安全访问的token验证。

## Token验证

### 我对token的理解和应用

首先我们要了解的是，每次用户向后端发请求必须要进行身份校验，否则如果我知晓了你数据的访问url，只要伪造假请求就可以拿到一大群用户数据，甚至是机密数据，但是用户的请求量是巨大的，而且用户信息是存放在数据库当中的，这么一来用户每次请求都要多出一次对数据库的查询（我们知道频繁访问数据库底层是非常消耗计算资源的，因为每次的查询都要与数据库进行一次连接又断开）所以，既要避免不安全的方位，同时又要避免让服务器进行过大的计算。这样就需要token验证的帮助。当客户端多次向服务端请求数据时，服务端就需要多次从数据库中查询用户名和密码并进行对比，判断用户名和密码是否正确，并作出相应提示。但这样无疑会增加服务器端的运行压力，是否可以有一种方式只需要验证用户就是之前的用户而不需要每次在客户端请求数据时都需要查询数据库判断用户名和密码是否正确。在这种请求下，引入了token来解决服务器端多次访问数据库问题。在介绍token的广泛应用前，我们先了解在token之前是如何解决不安全访问的问题。

### Token的起源

**基于服务器的验证**

　  我们都是知道HTTP协议是无状态的，无状态也就是无法确认发请求用户的身份，比如我根本没注册平台却可以通过伪造假请求得到管理员才能拿到的数据，这种无状态意味着程序需要验证每一次请求，从而辨别客户端的身份。在这之前，程序都是通过在服务端存储的登录信息来辨别请求的。这种方式一般都是通过存储Session来完成。

随着Web，应用程序，已经移动端的兴起，这种验证的方式逐渐暴露出了问题。尤其是在可扩展性方面。

**基于服务器验证方式暴露的一些问题**

1.Seesion：每次认证用户发起请求时，服务器需要去创建一个记录来存储信息。当越来越多的用户发请求时，内存的开销也会不断增加。

2.可扩展性：在服务端的内存中使用Seesion存储登录信息，伴随而来的是可扩展性问题。

3.CORS(跨域资源共享)：当我们需要让数据跨多台移动设备上使用时，跨域资源的共享会是一个让人头疼的问题。在使用Ajax抓取另一个域的资源，就可以会出现禁止请求的情况。

4.CSRF(跨站请求伪造)：用户在访问银行网站时，他们很容易受到跨站请求伪造的攻击，并且能够被利用其访问其他的网站。

在这些问题中，可扩展行是最突出的。因此我们有必要去寻求一种更有行之有效的方法。

**基于Token的验证原理**

基于Token的身份验证是无状态的，我们不将用户信息存在服务器或Session中。

这种概念解决了在服务端存储信息时的许多问题

　　NoSession意味着你的程序可以根据需要去增减机器，而不用去担心用户是否登录。

基于Token的身份验证的过程如下:

1.用户通过用户名和密码发送请求。

2.程序验证。

3.程序返回一个签名的token 给客户端。

4.客户端储存token,并且每次用于每次发送请求。

5.服务端验证token并返回数据。

**每一次请求都需要token。**

token应该在HTTP的头部发送从而保证了Http请求无状态。我们同样通过设置服务器属性Access-Control-Allow-Origin:\* ，让服务器能接受到来自所有域的请求。需要主要的是，在ACAO头部标明(designating)\*时，不得带有像HTTP认证，客户端SSL证书和cookies的证书。

当我们在程序中认证了信息并取得token之后，我们便能通过这个Token做许多的事情。

我们甚至能基于创建一个基于权限的token传给第三方应用程序，这些第三方程序能够获取到我们的数据（当然只有在我们允许的特定的token）

**Tokens的优势**

**无状态、可扩展**

在客户端存储的Tokens是无状态的，并且能够被扩展。基于这种无状态和不存储Session信息，负载负载均衡器能够将用户信息从一个服务传到其他服务器上。

如果我们将已验证的用户的信息保存在Session中，则每次请求都需要用户向已验证的服务器发送验证信息(称为Session亲和性)。用户量大时，可能会造成一些拥堵。

但是不要着急。使用tokens之后这些问题都迎刃而解，因为tokens自己hold住了用户的验证信息。

**安全性**

请求中发送token而不再是发送cookie能够防止CSRF(跨站请求伪造)。即使在客户端使用cookie存储token，cookie也仅仅是一个存储机制而不是用于认证。不将信息存储在Session中，让我们少了对session操作。

token是有时效的，一段时间之后用户需要重新验证。我们也不一定需要等到token自动失效，token有撤回的操作，通过token revocataion可以使一个特定的token或是一组有相同认证的token无效。

**可扩展性（）**

Tokens能够创建与其它程序共享权限的程序。例如，能将一个随便的社交帐号和自己的大号(Fackbook或是Twitter)联系起来。当通过服务登录Twitter(我们将这个过程Buffer)时，我们可以将这些Buffer附到Twitter的数据流上(we are allowing Buffer to post to our Twitter stream)。

使用tokens时，可以提供可选的权限给第三方应用程序。当用户想让另一个应用程序访问它们的数据，我们可以通过建立自己的API，得出特殊权限的tokens。

**多平台跨域**

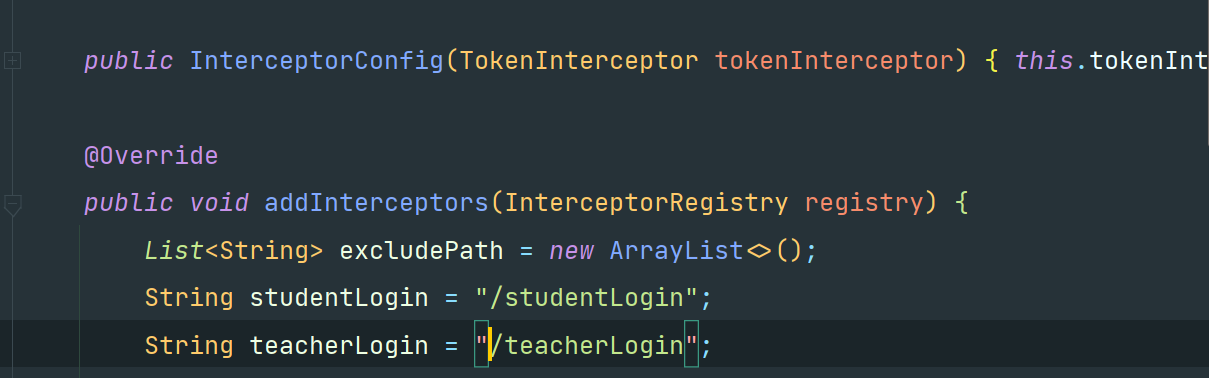
我们提前先来谈论一下CORS(跨域资源共享)，对应用程序和服务进行扩展的时候，需要介入各种各种的设备和应用程序。

Having our API just serve data, we can also make the design choice to serve assets from a CDN. This eliminates the issues that CORS brings up after we set a quick header configuration for our application.

只要用户有一个通过了验证的token，数据和资源就能够在任何域上被请求到。

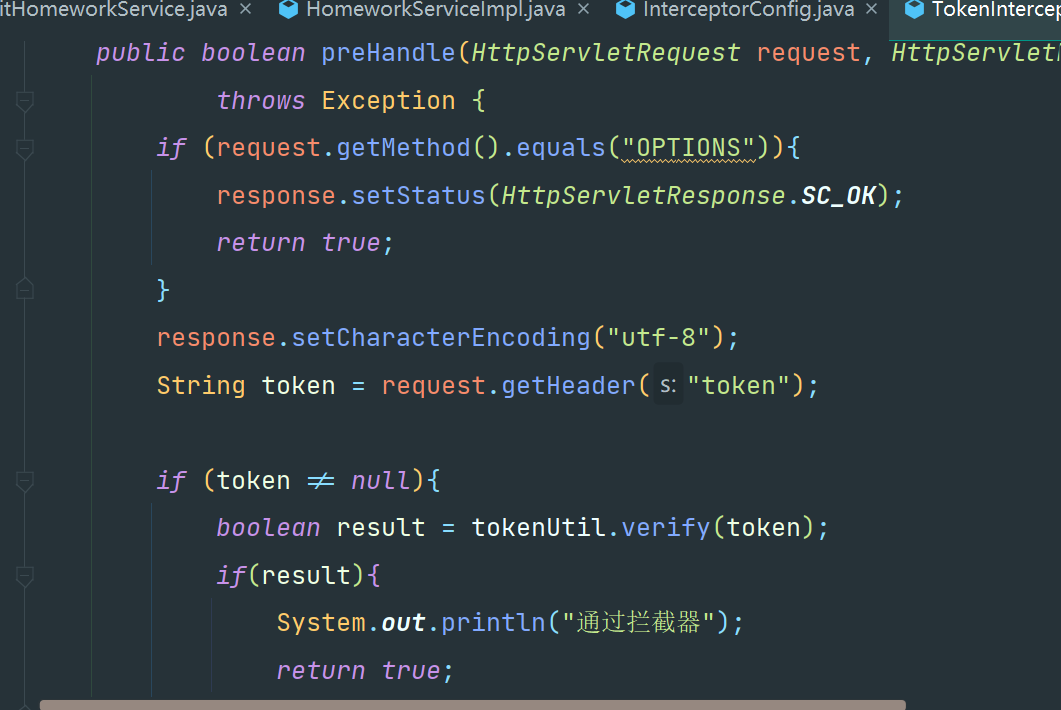
### 我在项目中的应用

首先token要配合拦截器使用，也就是在处理前端所发送的请求之前进行token验证。当然要知道的是比如，登录、注册请求是要绕开拦截器的，因为此时用户还没有token信息，所以要把这些请求路径添加到拦截器白名单。

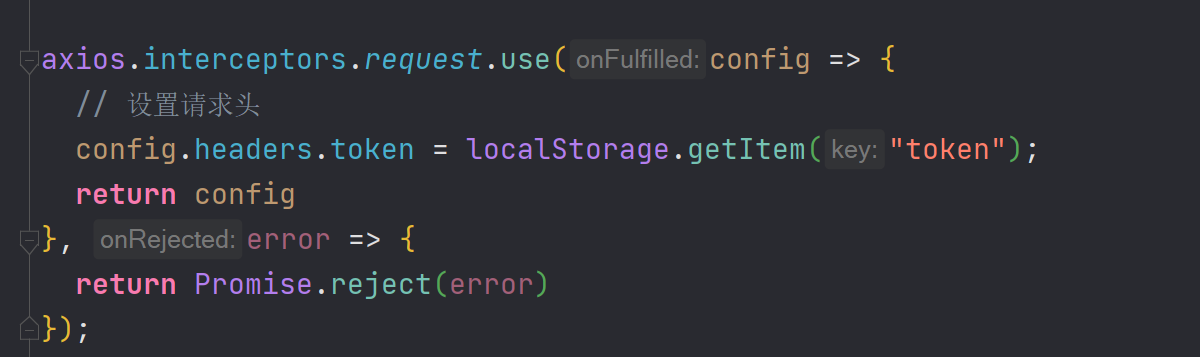


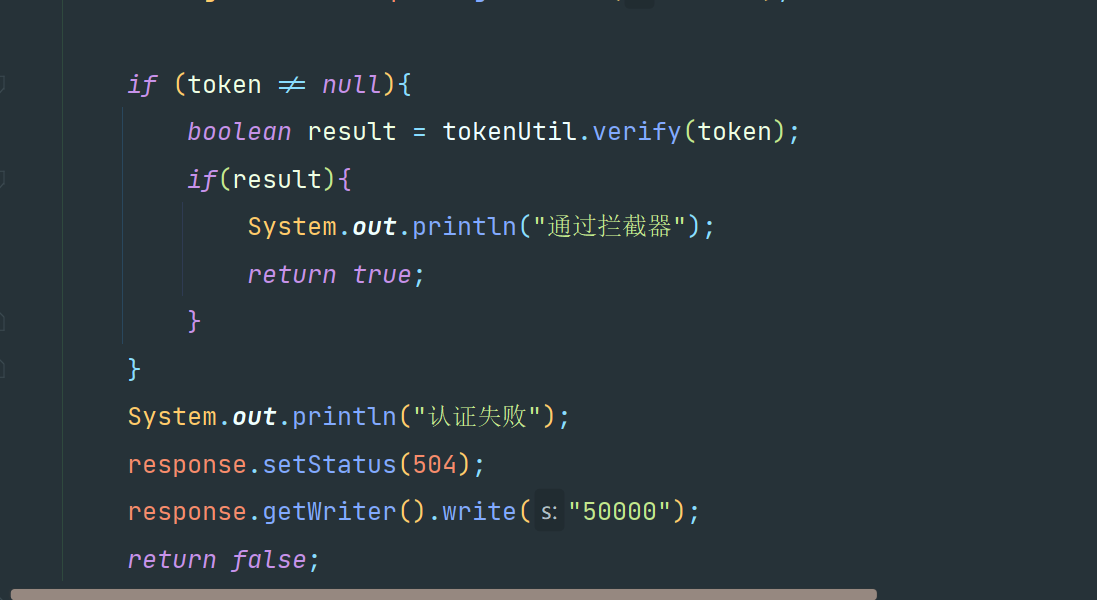


然后是token拦截器的配置，要走过的流程是先获取存在请求头中的token，信息



所以要先在前端的发送请求的axios，中全局配置，让每一次的发送请求，都携带token，当然token需要存在浏览器中，值得注意的一定是每次组件生命周期结束要清空包括token的所有用户信息。

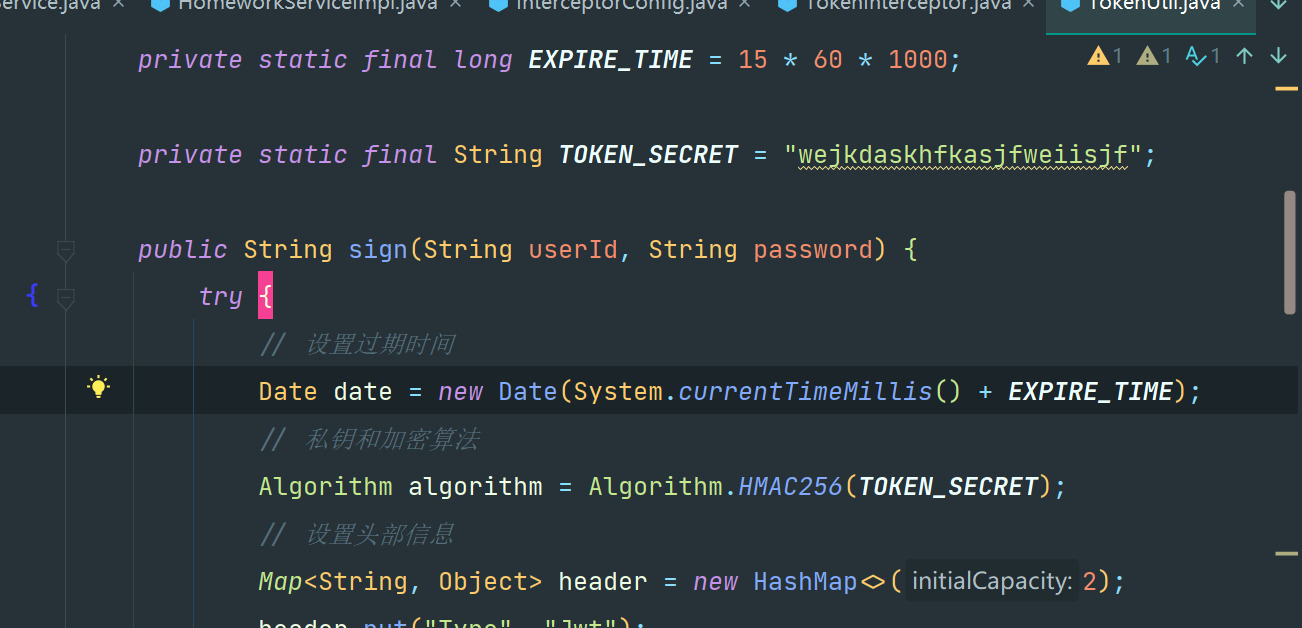




Token拦截器的内容也就是先检查请求头中是否带有token信息，如果有的话就调用验证函数，没有的话就返回给前端并本次请求失败状态码是504。

**如何产生token？**

要知道的是token是一串加密过的字符串，所以涉及到加密首先选用加密算法，和密钥，



我对token的设计是先给出时间设置，然后给一串无序无规则字符串作为密钥，严格来说这个密钥应该定期的去修改，然后获取系统当前的时间戳结合所设置的过期时间并整合用户名，用户密码，密钥调用HMAC256加密生产token。这样我们就得到了token。

**Token的存储，避免计算资源的浪费**

如前面所提到的频繁访问数据库是非常消耗计算资源的方式，所以我们可以考虑把token信息存储在别的地方，比如redis，redis是访问数据非常快（每秒可达几万次），这样降低服务器的负载压力



所以这样自定义好redis的常用工具方法，比如定时的存，按照key去取。

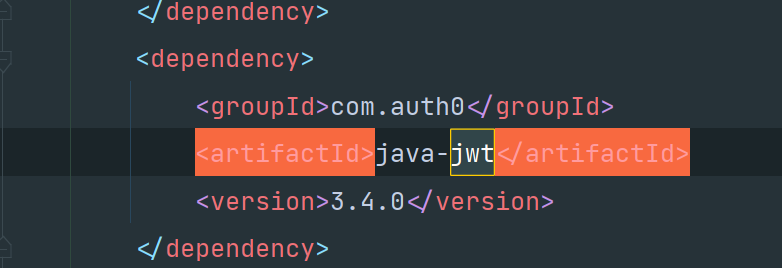


最后要注意的是token要设置过期时间，没当用户发送请求时不断取刷新token在redis中的保存时间。

**关于token的问题总结**

一开始非常难以去理解token的设计思想和实现过程，花了不少的时间，同时token还使用到了各种拦截器，来源于前端和后端，同时为了避免花费大量的时间在数据的查询上还引入了redis，由花了部分时间去学习理解。

**Token的引入和环境配置**



通过pom文件引入一个jwt包。然后安装一个redis即可，redis安装也很方便，整体不需要配置任何环境变量。

## 深入探讨数据库设计问题

数据库设计难点

其实纵观整体来看，要完成的功能难点有如何实现类似于pta的编码功能？如何去为实现老师在线录入题目？如何去实现在线考试，完成做作业，做实验等？

我的主要解决方向是关于题目的设计。首先考虑录题方面，可以知道的是一次考试可能有多道题目，所以要深入思考的第一个问题就是录制题目是一次一提交还是一次性全部提交？毫无疑问，一次性全提交优势更大，首先是如果是一次一提交那么也就是录完了就发请求给后端写入数据库，如果用户回去查看又发一次请求，如果要修改又要发送一次请求。这样频发的请求资源势必会增大服务器的压力。必须要提的是客户的响应时间80%都是在想后端请求数据，而页面的渲染只占据了很小的一部分。所以一次性发送是最优解，但是难度也大大提升，就是如何保存在前端？同时题目的题型包含了很多种，要设计怎样的数据结构去存取是个很严重的问题。一开始我的想法是按照不同的题型去给不同的实体，但是这样前后端穿数据是个大问题，而且把题目显示出来又是个问题。这也就是为什么我说更适用MongoDB这种结构较为松散的数据库。所以作为权衡考虑，既要保证前端能保存数据，又要保证后端能方便接收，所以我把所有的题型成为一种实体，也就是创建一张表，值得一提的是如果一次性对多张表进行修改或是查询也是很消耗时间的，所以这么一看对单表的操作似乎对服务器很友好，但其实也是一种取舍，这样来做的话会让设计变的非常复杂。如我是把本次所有的题型都抽象为三层结构，题目内容，题目选项，题目答案。不难发现这种结构和单选题，多选题，判断题兼容性很好，但是对填空题，问答题之类的适配很差，所以我把填空题的一个个文字部分看作选项，多个答案也写入答案当中。其实这样也算是出于多方考虑的结果。在我看来设计很难达到完美，只能多方评估折中取得较优解。

## 浅析springboot

**Springboot的起源简介**

SpringBoot帮助开发者快速搭建Spring框架；SpringBoot帮助开发者快速启动一个Web[容器](https://cloud.tencent.com/product/tke?from=10680)；SpringBoot继承了原有Spring框架的优秀基因；SpringBoot使得基于Spring的开发过程更加简易。

Change is inevitable, that's the only constant. Become the Future You Imagine (Rob Mee, Pivotal CEO)

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Boot致力于在蓬勃发展的快速应用开发领域（rapid application development）成为领导者。

多年以来，Spring IO平台饱受非议的一点就是大量的XML配置以及复杂的依赖管理。在去年的SpringOne 2GX会议上，Pivotal的CTO Adrian Colyer回应了这些批评，并且特别提到该平台将来的目标之一就是实现免XML配置的开发体验。Boot所实现的功能超出了这个任务的描述，开发人员不仅不再需要编写XML，而且在一些场景中甚至不需要编写繁琐的import语句。

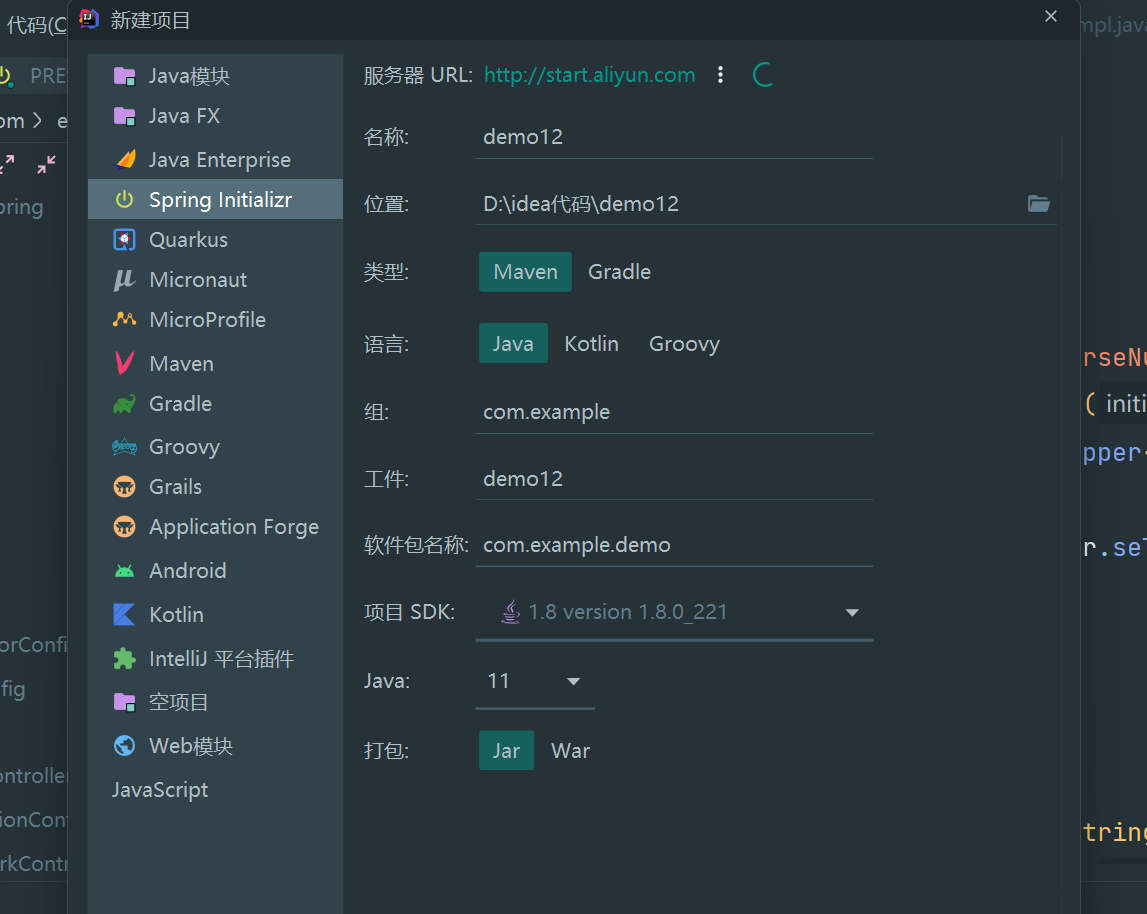
然而，Spring Boot并不是要成为Spring IO平台里面众多“Foundation”层项目的替代者。Spring Boot的目标不在于为已解决的问题域提供新的解决方案，而是为平台带来另一种开发体验，从而简化对这些已有技术的使用。对于已经熟悉Spring生态系统的开发人员来说，Boot是一个很理想的选择，不过对于采用Spring技术的新人来说，Boot提供一种更简洁的方式来使用这些技术。

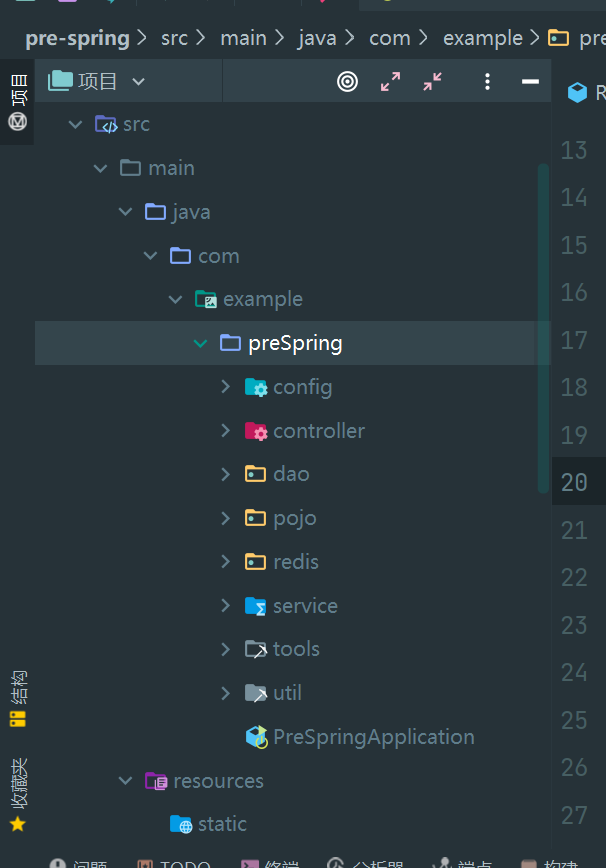
**Springboot的重点和难点**

由于之前已经使用过spring boot，所以我认为重难点是在于去理解spring boot的业务规则，和熟悉里面的项目结构，和配置文件的编写，在我看来spring boot是一个集大成者，基本上手使用是没什么难度，但是想要用到好才是问题所在，比如整mybaitis-plus等各种支持框架，以及灵活的去应用其配置文件带来便利，比如修改端口号，显示本次查询的sql执行，控制文件的上传大小等才是问题。所以会用不难，难的是用的好。

**Springboot 的准备和环境配置**

因为我使用的idea，所以准备项目也就是创建一个spring boot项目即可（记得使用阿里云的，这样拉取的更快），会自动导入基本依赖。由项目后先写一下基本的配置文件，然后根据需要给出整体项目结构，因为是整合了mybatis-plis，所以我的项目结构是如下的。

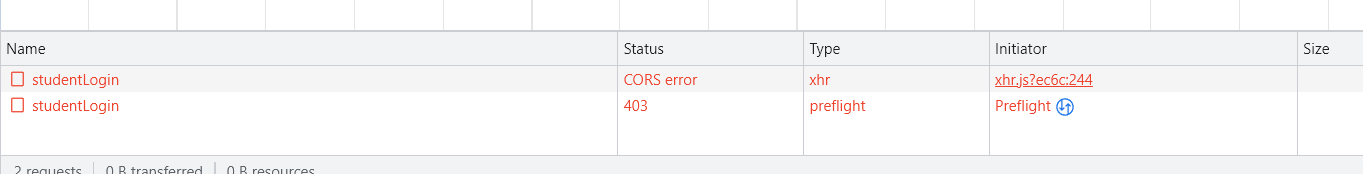




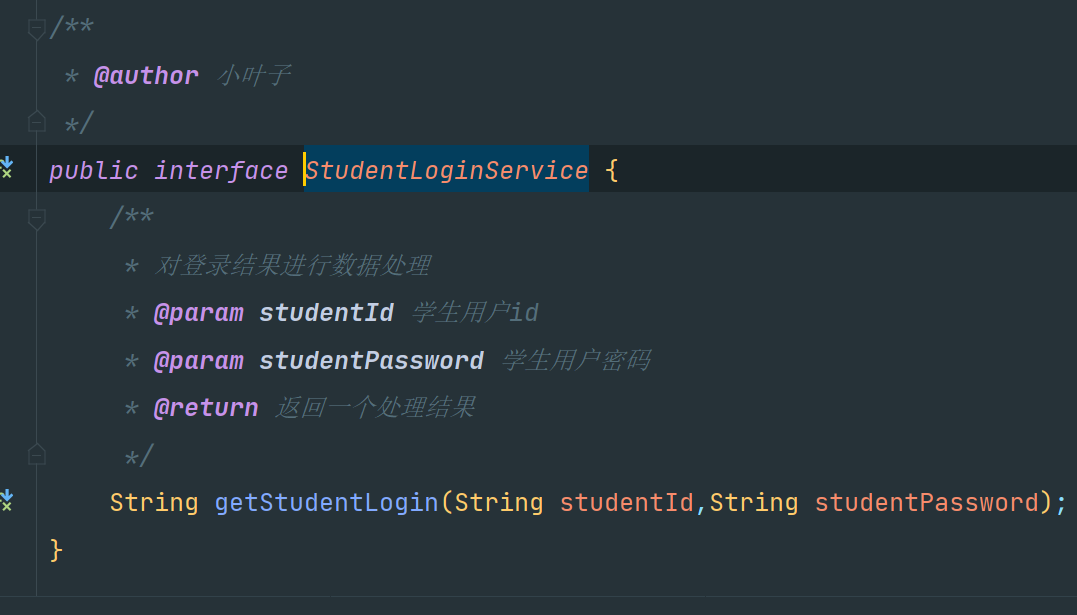
**Springboot的应用**

Springboot通常分为清晰的几层结构，第一层就是和获取前端发送数据的controlller，然后获取数据，如下的去调用相应的服务层接口，要知道的是控制层应该完成的就是接收数据然后转交数据给服务层去处理，而不对数据做任何修改。一开始我遇到了忘记带上@RestController注解，这样会使得前端访问403，还有必须要加上@CrossOrigin注解表示允许跨域请求，否则请求会被拒绝接收。

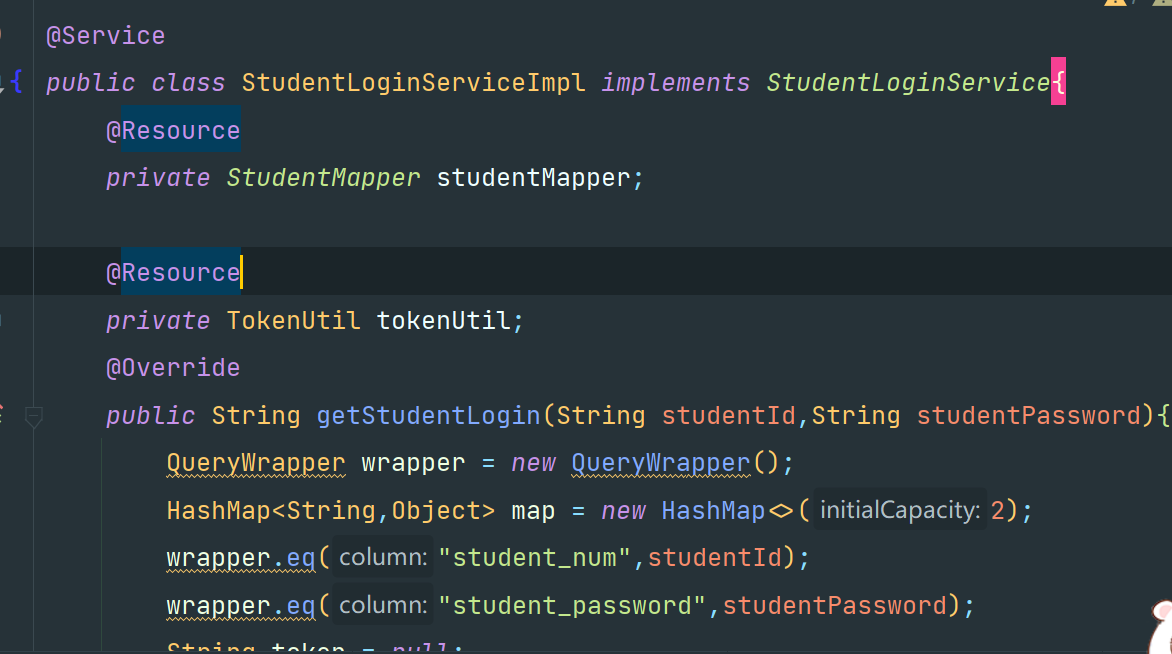


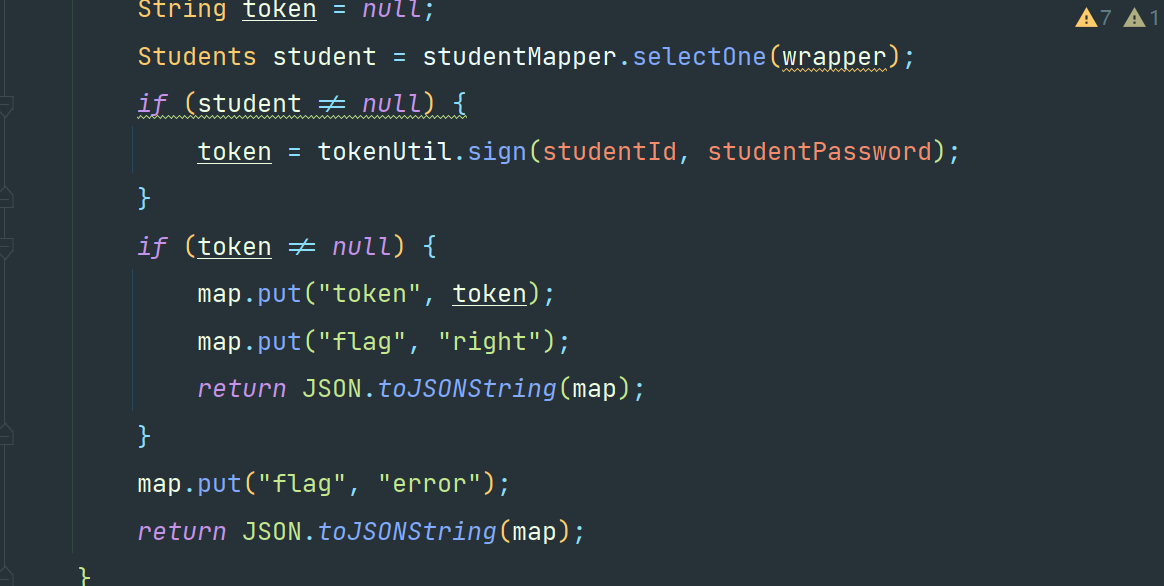


然后完成数据提交到服务层，服务层所提供的是一个接口，里面只是给出方法名称，参数和带上注释表示这个方法的作业和内容，真正的实现是在接口实现类中，必须要说的是因为spring的特性，容器中是允许一个接口有多个实现的，根据实际需要可以自由的选择注入哪个依赖，这也体现出了java的多态语言特性，但是我的应用中并没有体现出这点，因为项目的体量并不是很大的原因。



接着是服务层接口的实现类，要注意的是必须先使用@Service注解把这个依赖注入到spring容器当中，才能被引用，因为控制器引用的是接口而并非接口实现类，但是真正完成数据处理的确实接口，所以要按需引用。我所采用的处理方案是校验完身份，记得带上一个处理结果返回给前台，很多时候是需要到异常处理，因为后端是对用户是不可见的，所以可能出现断点需要告知用户异常。

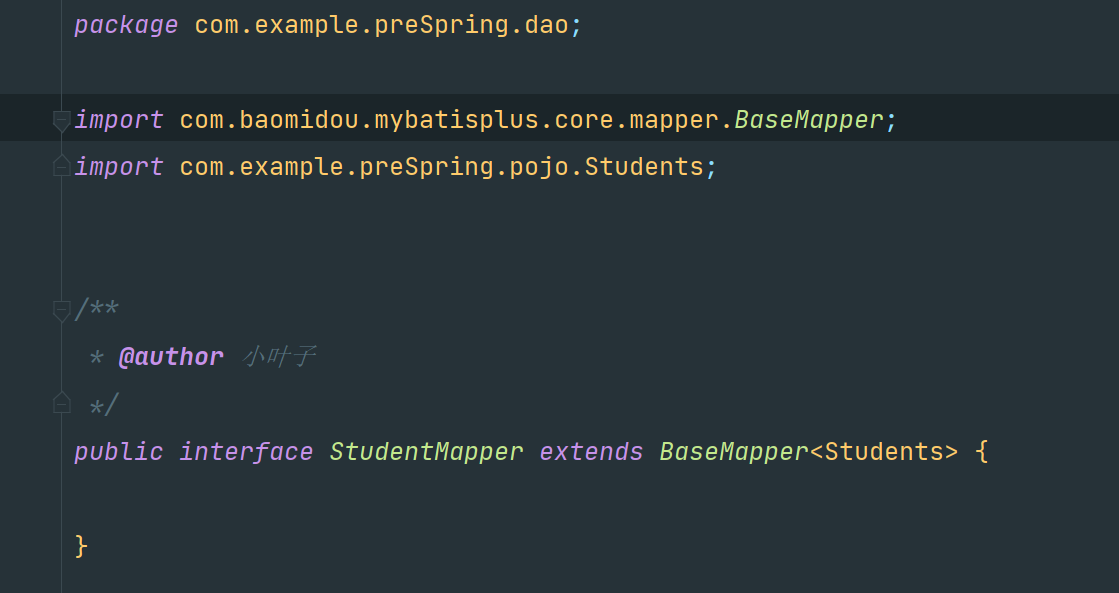


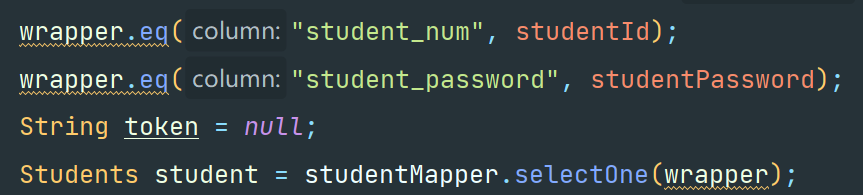


当不仅于此，还有pojo类的支持，也就是实体类，灵活使用springboot注解是会带来很大的便利的，比如@TableName指定该实体映射的数据库表，@Date免去对get，set这种机械的书写，还有可以省去对有参构造函数和无参构造函数的编写。以及我遇到的一个问题，因为数据库中我的表的id是唯一自增的，所以直接插入是会出问题的，需要给予一个注解@TableId去表示其字段是自增的才能进行插入，还要记得是数据库的表中最好有一条预先数据。



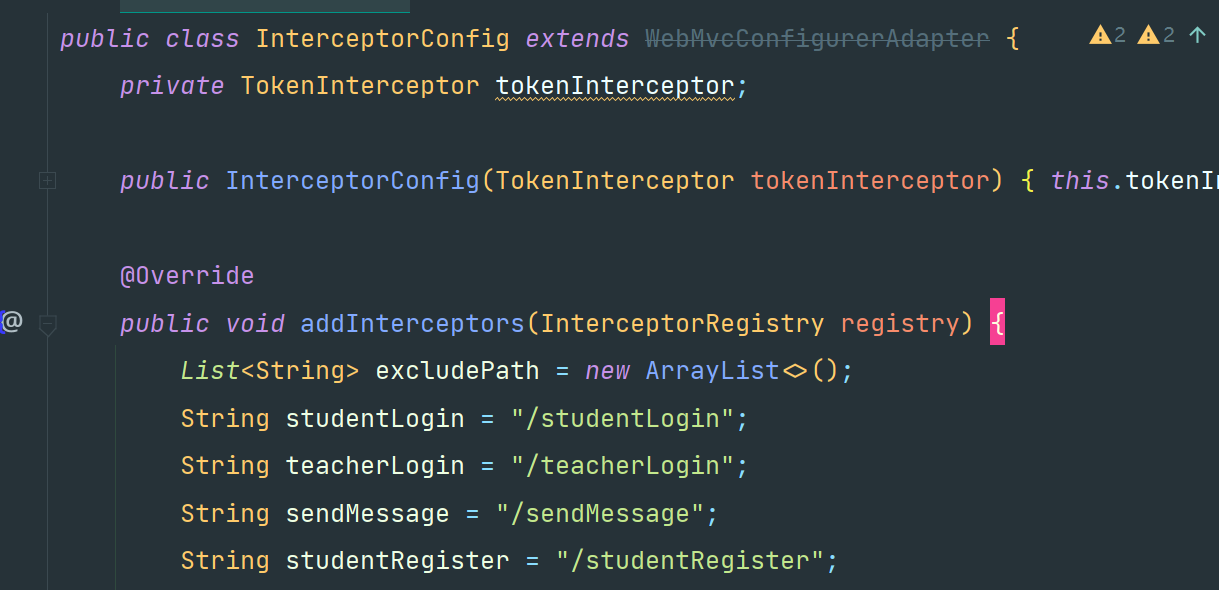
还有dao层，因为使用到的是mybatis-plus，所以我的dao层就是继承于mybatis-plus所提供的基类，并指明所其实体类型。因为mp提供了很多基础的增删改查的手段，可以很多时候避免对sql的编写，但是涉及到很复杂的多表级联查询的话还是需要自己去编写好查询语句的。





还有一些工具类，配置类，一些比如redis的封装后可以放在tools目录下给上@Component注解去给需要的地方调用，和基础的配置类。值得一提的是由于spring容器的特性，应该是通过一些注解把所要供引用的类注入到容器中而不是通过实例化对象来引用，这样会带来一些不必要的错误。





## 心得体会

作为本次项目的主要负责人，我更加深入体会到了一句话，软件的精髓是架构，而并非细节的实现。其实有过一些经验后就会明白真正编写代码应该是只占据项目开发的一下部分，大多数时间应该花在对项目的设计上，这样去减少后期结构的不确定性和反复修改。我的感觉是前后端分离开发需要非常清晰的的架构思想，因为前后端进行的数据交互都是需要一个接口，所有应该从控制器和前端请求接口入手，告诉前端人员我需要你这个功能发送哪些数据，发送哪些格式的数据，发送的url等等基础信息，然后后端人员进行数据处理是经过怎么样的一个流程。而且在开发进行前应该确立本次要使用到的哪些技术，并且严格要求成员使用的版本必须一致，在此基础上要维持代码编写风格的一致性，而且必须必须写注释，不然交接任务时非常难以融并。但由于开发经验的不足，很多时候是边写边改数据结构，这种低效的做法应该尽可能去避免。还有在开发中真的是好的架构才是软件项目的灵魂，因为那些基础的技术学习成本通常并不是很高，写代码的大多时候都是重复且机械的去编写和出错去调试，严格的约束代码的编写顺序和步骤很大程度上降低的出现bug可能，余下的就是重复编写，所以写代码不是项目重点所在。同时在分析一个项目的时刻，严格不仅仅关注于对需求上的功能实现，更应该去深入探讨内在需求，比如安全性的需要，可扩展性的需求，健壮性性的需求，实用性的需求。从这几个方面引申出对发送请求采用token验证的方式，同时减轻服务器的压力又依托于redis缓冲区避免对底层内存的读取访问。在前端用加上各种拦截器和路由守卫去避免不安全访问。在可扩展性，最重要的应该是对项目的解耦，尽可能遵循高内聚低耦合，这样后期的扩展和维护上会大大降低，应用springboot同时有良好的解构思想，比如我把学生的登录和老师登录分离开，每个都有自己的控制层，服务层，等，如果平台需要扩展出新的角色那么只需要继续创建实体，编写响应的服务即可。不仅仅是在后端，前端也应该遵循，合理应用mvvm。在性能问题上，我的思想是尽可能避免前端发送请求的次数，我的想法是一些基础且公开比如用户账号的等数据可以暂存在前端，这样无需频繁重复请求数据，其实还可以通过redis的支持还频繁被请求的数据存在缓存中，再加上能够一次性发送的数据避免多次发送。在健壮性的需求上，我们应该进行大量的异常处理，在可能出现断点的地方去使用try catch环绕。

最后我的想法是开发中应该严格遵循一个开发规范，拓宽自己的知识面而不是重复的练习无意义的增删改查，尽可能多去考虑如何做好而不是做完。