1. **自我介绍分析：**

您好，我叫宋晴晴，来自河南，今天来咱们公司应聘Java工程师，我是本科学历，大学专业是计算机科学与技术，毕业之后一直从事Java开发，目前已经有三年的工作经验，在上一家公司是Java工程师，做过的项目涉及到了教育、社交、物流、j金融等，使用的是现在流行的SpringBoot和SpringCloud框架构建的。对于我本人，性格比较稳重，对待工作认真负责，而且喜欢学习一些新的东西，扩充自己的知识，平时喜欢看看博客还有一些有关编程的书籍来提升自己，并且非常热爱这一门职业。对咱们公司呢，也是很早就听说了，非常高兴能够到咱们公司来进行面试，希望能在咱们公司面试成功，能够长期稳定的发展下去。以上就是我的自我介绍，谢谢！您看还有什么要了解的吗？

1. **项目介绍+项目技术亮点分析：**

**伴学优课：**

我最近做的项目是一个教育类型的项目，伴学优课，为学生提供实用的[学习软件](https://www.cr173.com/k/xuexi/" \t "https://www.cr173.com/soft/_blank)，伴学优课是一款以直播和录播为主的在线教APP+精品课程，在直播的授课过程中学生可以和教师进行互动提问里面包含登录注册、课程分类模块、课程明细模块、课程购买、订单支付中心等模块

项目整体采用SpringBoot、SpringCloud的微服务架构，数据层springDataJPA， 所用环境都是基于Docker环境，测试工具用的是Postman（提供功能强大的We b API & HTTP 请求调试），数据库在生产环境中要分片部署(MyCat)通过Eureka实现注册信息集中化管理，微服务之间通过feign相互调用。在微服务架构中通常会有多个服务层调用,然后就用到了熔断器(Hystrix),因微服务一般有不同的网络地址，用到了微服务网关(zull),由于服务数量巨多，为了方便服务配置文件统一管理，实时更新，所以用到了分布式配置中心组件(SpringCloudConfig),最终那我们 把配置文件传到了码云上，为了不重启服务,节省时间,就用到了(SpringCloudBus)。我们还运用了(Jenkins)实现持续集成,实施监控集成中存在的错误，提供详细的日志文件和提醒功能，还能用图表的形式形象地展示项目构建的趋势和稳定性。短信验证方面用的是阿里大于短信接口，使用消息中间件RabbitMQ的消息队列通知短信微服务调用阿里短信接口下发短信，使用Redis存储防止用户多次恶意发送短信。

登录注册模块：

短信验证码

在我们的项目当中很多地方都用到了我们的短信验证的功能

举例说明: 我们在用户注册这块就用到了短验信证,在填写完其他的信息完时,这时候会有一个手机号验证,来验证我们的用户手机号是否可用,和本人是否一致,这样呢我们就会用到一个短信的验证技术,我们用的是一个阿里的大于短息接口。

用到了springBoot这个微服务框架来搭建短信接口

这大体就是我们这个短信微服务的一个简绍

在业务这一块呢

我们先是在注册的一个页面,进行一个短信验证码的发送,调用后台短信这块的一个方法，这个方法在被调用后,在后台生成短信验证码, 通过消息中间件发送我们搭建的sms短信微服务的这个服务器上进行一个短信的发送这个就是我们短信的发送一个业务流程,

为了实现这一块的业务,我们用到了我们搭建的springBoot微服务 sms的短信接口，还有JMS中间件的RabbitMQ的消息队列,服务端的话会用到用户中心这一端,短息微服务的这一端,并  
在注册模块中做了一个短信验证码的验证，当时考虑的是防止用户恶意注册给数据库增加压力。这个项目我当时用的是阿里云提供的短 信接口。 这里还用到了ribbitMQ。这里为什么要用到MQ呢，就因为考虑到如果在并发的情况下多个用户同时注册会给短信接口造成压力。也可能出现一些问 题。所以用到了消息队列给请求排个队。 我可以给您详细说一下这个注册功能具体实现业务吧。

技术的介绍

阿里的大于短信微服务借口,是现在一块很不错的短信服务，对于我们开发这一块来说阿里的大于是很不错的选择，因为对我们这些,中小型的项目来说,不仅完美的支持,还融合了三大运营商的通信能力.符合我们现在项目的一个支持。springBoot呢,是一个轻量级的服务框架,因为我们的阿里的大于也是一个轻量级的服务,相对于spring来说,太多的配置文件,和重量级的配置,比较繁琐,和承重.并不适合我们来做短信服务这个端口,所以呢.我们就用了springBoot.

RabbitMQ这一块呢,是消息中间件,是用来管理消息队列的一个服务,RabbitMQ,有五种模式,

当我们的后台发送短信的方法被调用的时候,首先,我们会判断一下手机号码是否规范,这里,我们会写一个判断手机是否正确的正则工具类,然后调用工具类来判断手机号是否正确这里,如果手机号不正确的话,直接返回到前台,并把错误信息告诉用户，如果手机号正确的话,我们调用发送验证码的方法，发送验证码,

<1>在发送验证码的时候,首先,我们会生成一个6位数的验证码,在生成验证码的时候我们用到的是直接用随机数\*6位来生成验证码,为了避免，验证码会有缺失.这里我们自己定义了一个最大数和最小数，随机生成完毕后，对验证码进行判断，如果验证码小于最小数，n那么我们就让生成的验证码加上定义的最小数来防止验证么丢失。在生成验证码后,我们还要把验证码的存入到redis里,为添加时验证验证码是否正确

<2>redis这块,我们用的是RedisTemplate<String , Object> redisTemplate这个对象来操作redis，redis这里呢,我们用的是hash的这个方式存的值,给了一个key, value值,并设置了失效时间

<3>然后就是利用RabbitMQ来把们的验证码发送sms服务器，这里我们的rabbitMQTemplate这个对象来操作RabbitMQ发送短信,在这里呢,我们把要发送的信息放到Map里,这里的参数,有待发送人的电话号码,还有阿里大于的模板钥匙,还有我们生成的验证码,还有一个是我们要用的模板类型.这样把 Map 这对象返回，这样呢,完成了我们用RabbitMQ技术来发送到sms的一个操作，接下来呢是我们的sms用RabbitMQ来接收我们刚刚发出的 消息.在sms里呢 ，我们通过 @RabbitListener 这个注解来接受发生的消息，在收到消息后我们把消息放到我们的短信服务工具类里,这个短息工具类呢,我们需要配置一些东西的accessKeyId和 accessKeySecret我们放入的数据，有待发送人的电话,使用的短信签名,短信模板,还有就是模板变量的值。设置好这些后,进行发送，并在控制台打印发送状态.这样我们就完成了一个短信发送的操作。

最后当我们用户收到验证码后,并把验证码填写到验证码的框里,进行注册，在后台当我们收到前台发来的注册信息后,我们要从redis里取出我们刚存的验证码和用户刚刚前台填写验证码进行对比和判断,一样则注册成功,不一样注册失败.

这样,我们就完成了一个短信验证注册.

**登录模块**   
说明   
由于微服务大都是分布式的，需要几台服务器部署，当一个用户在其中一台服务器登录后，传统的方式是session保存其登录信息，然后可以使用共享存储共享，比如redis共享，这种方案的缺点在于共享存储需要一定保护机制，因此需要通过安全链接来访问，这时解决方案的实现就通常具有相当高的复杂性了，所以这里使用基于令牌的方式做登录。

JWT简介  
JWT(JSON WEB TOKEN)是一个非常轻巧的规范，这个规范允许我们使用jwt在客户端和服务器之间传递安全可靠的信息。简单理解可以就是一个 Token 值，只是这个 Token 值比较特殊，有两个 . 号分隔。  
  
• 点号第一部分称为 header(头部)，第二部分称为 playload(载荷)，第三部分称为 signature(签名)。  
• 其中 header 和 playload 是两个 base64 编码的 json 字符串，前端直接用 base64 解码即可，这里就是前端的要点之一。  
• signature 是签名信息，JWT 是否有效取决于对签名信息的验证是否通过，当然这是后端的事情了，这是后端的要点之一。  
• 当我们拿到这个 Token 值，我们一般有两种存放方式，一种是 Cookie，则后端后面只要拿 Cookie 这个值即可。另外一种是放在 localStorage，然后前端有请求的时候在拿出来放在请求头，比如这样：  
• 在请求头里加入 Authorization，并加上 Bearer 头（至于为什么要这样格式，可以看 这个

·）：  
fetch('api/user/1', {  
  headers: {  
    'Authorization': 'Bearer ' + token值  
  }  
})  
• 后端拿到 Token 校验是否有效、是否过期等情况。  
  
实操   
  
1.前端服务器收到用户登录请求，传给后台zuul网关。  
  
2.zuul网关把请求分发到用户服务里进行身份验证。  
  
3.后台用户服务验证通过，然后从账号信息抽取出id、nickName、login\_method、login\_time等基本信息（这些信息根据自己的需要定义）组成payload，进而组装一个JWT，把JWT放入redis(因为退出的时候无法使jwt立即作废，所以使用保存在redis中，退出的时候delete掉就可以了，鉴权的时候加一层判断jwt是否在redis里，如果不在则证明jwt已过期作废)，然后包装到json数据返回到前端服务器，这就登录成功了。  
  
4.前端服务器拿到 JWT，进行存储（可以存储在缓存中，也可以存储在数据库中，如果是浏览器，可以存储在 localStorage 中）在后续请求中，在 HTTP 请求头中加上 JWT(前端在每次请求时将JWT放入HTTP Header中的Authorization位,解决XSS和XSRF问题)。  
  
5.登录后，再访问其他微服务的时候，前端会携带jwt访问后台，后台的zuul网关添加拦截器来校验 JWT，验签通过后，进去相应的服务，然后返回相应资源和数据就可以了。  
JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用JWT在用 户和服务器之间传递安全可靠的信息。  
一个JWT实际上就是一个字符串，它由三部分组成，头部、载荷与签名。   
头部（Header） 头部用于描述关于该JWT的最基本的信息，例如其类型以及签名所用的算法等。这也可以 被表示成一个JSON对象。 在头部指明了签名算法是HS256算法。 我们进行BASE64编 码IMG_256http://base64.xpcha.com/，编码后的字符串如下： eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9 小知识：Base64是一种基于64个可打印字符来表示二进制数据的表示方法。由于2 的6次方等于64，所以每6个比特为一个单元，对应某个可打印字符。三个字节有24 个比特，对应于4个Base64单元，即3个字节需要用4个可打印字符来表示。JDK 中 提供了非常方便的 BASE64Encoder 和 BASE64Decoder，用它们可以非常方便的 完成基于 BASE64 的编码和解码  
载荷（playload） 载荷就是存放有效信息的地方。这个名字像是特指飞机上承载的货品，这些有效信息包 含三个部分 {"typ":"JWT","alg":"HS256"}   
（1）标准中注册的声明（建议但不强制使用） iss: jwt签发者 sub: jwt所面向的用户 aud: 接收jwt的一方 exp: jwt的过期时间，这个过期时间必须要大于签发时间 nbf: 定义在什么时间之前，该jwt都是不可用的. iat: jwt的签发时间 jti: jwt的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。  
（2）公共的声明 公共的声明可以添加任何的信息，一般添加用户的相关信息或其他业务需要的必要信息. 但不建议添加敏感信息，因为该部分在客户端可解密.   
（3）私有的声明 私有声明是提供者和消费者所共同定义的声明，一般不建议存放敏感信息，因为base64 是对称解密的，意味着该部分信息可以归类为明文信息。这个指的就是自定义的claim。比如前面那个结构举例中的admin和name都属于自定的 claim。这些claim跟JWT标准规定的claim区别在于：JWT规定的claim，JWT的接收方在 拿到JWT之后，都知道怎么对这些标准的claim进行验证(还不知道是否能够验证)；而 private claims不会验证，除非明确告诉接收方要对这些claim进行验证以及规则才行。 定义一个payload: {"sub":"1234567890","name":"John Doe","admin":true}然后将其进行base64编码，得到Jwt的第二部分。 eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9 签证（signature） jwt的第三部分是一个签证信息，这个签证信息由三部分组成： header (base64后的) payload (base64后的) secret  
这个部分需要base64加密后的header和base64加密后的payload使用.连接组成的字符 串，然后通过header中声明的加密方式进行加盐secret组合加密，然后就构成了jwt的第 三部分。 TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ 将这三部分用.连接成一个完整的字符串,构成了最终的jwt:   
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7Hg Q   
注意：secret是保存在服务器端的，jwt的签发生成也是在服务器端的，secret就是用 来进行jwt的签发和jwt的验证，所以，它就是你服务端的私钥，在任何场景都不应该流 露出去。一旦客户端得知这个secret, 那就意味着客户端是可以自我签发jwt了。

再做注册的时候，我们根据公司提供的API文档，找到对应的发送短信的接口写入到我们的controller当中，在我们的service层中生成验证码，这个验证码使用random随机数生成，当时我们控制这个随机数为六位数这个范围，也就是大于等于100000小于等于999999，得到验证码后，将验证码存到我们的redis当中，并且设置这个验证码的过期时间，因为这个验证码并不会一直都有用，用户使用一次之后，就没有什么太大的用户了，处理好这些后，我们将用户的手机号和验证码放到一个map集合当中，然后将整个集合发送到rabbitMQ指定的队列上等待被消费，这里用rabbitTemplate这个模板调用converAndSend这个方法便可以直接塞入到指定的队列当中，我们在调用短信接口阿里大于的时候，因为我们这个是一个微服务的项目，所以我们专门写了一个工程，这个工程就是专门为了获取rabbitMQ队列上的数据调用阿里大于给我们客户发送短信用的，在这里我们写了一个消息监听类，引入@RabbitListener和@Component以及@RabbitHandler便可消费我们rabbitMQ上的消息，我们往rabbmtMQ传的时候用的MAP集合装的，获取的时候当然也就用Map集合来进行获取，我们在使用阿里云发送短信时，他就提供给我们一个短信工具类，这个短信工具来就可以自动的帮我们发送短信了，我们的短信模板，短信签名，手机号在阿里云官网就可以直接弄好了，他就可一直接给我们指定的手机发送短信，然后我们的短信监听类在使用阿里云提供的的短信工具类这些模板和签名的时候，直接使用@Autowired注入就好了，然后调用工具类里的发送短信方法，便可成功发送短信，我们用户在进行注册的时候，输入验证码后，去拿着我们验证码和我们redis缓存当中的验证码做对比，如果验证码为空和输入有误，则提示用户验证码有问题，如果正常的话则对我们的用户密码进行加密，这里加密的时候采用的是Bcrypt加密算法，我们当时也考虑过用MD5加密，但是后来发现这个东西能够被破解，而Bcrypt这个东西里面有时间戳，时间戳这个东西是几乎不可能被破解的，Bcrypt这个加密算法是Spring Security安全框架提供的，所以我们只需要引入Spring Security这个依赖便可以了，但是我们们引入这个安全框架之后，所有的地址就得被安全框架所控制了，而我们只是用单纯的加密算法，所用这个时候我们建立一个安全配置类，里面引入@Configuration 和@EnableWebSecurity这两个注解，便可配置所有的地址匿名访问，在启动类里面bean注入BCryptPasswordEncdor就可以加密了，我们再进行登陆效验密码的时候，BCryptPasswordEncdor给我们提供了两个方法，encode和matches方法，用encode方法进行加密，matches方法进行用户前台传入的密码和我们数据库加密后的密码进行比对，我们在登录的时候，选择了基于JWT的Token认证机制，客户端使用用户名和密码进行请求登陆，服务端收到请求，去验证用户名和密码，验证成功后，服务端会发送一个Token，再把这个Token发送给客户端，客户端收到Token后把他存储起来，比如经他放到cookie当中，客户端每次向服务器请求资源的时候，都会带着这个Token串，服务端收到请求后，会拿着前台传过来的Token串和我们服务端的Token串进行对比验证，成功就返回给客户请求的数据，

创建token方式首先导入JJWT依赖，设置签发时间，设置密钥，每次的token都是不一样的，因为载荷当中存在时间.

后端一般要做的校验：   
o 签名是否正确  
o token 是否过期（根据 exp 字段）  
o 接收方（aud）是否是我  
  
代码  
1.首先引入依赖  
<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jwt</artifactId>

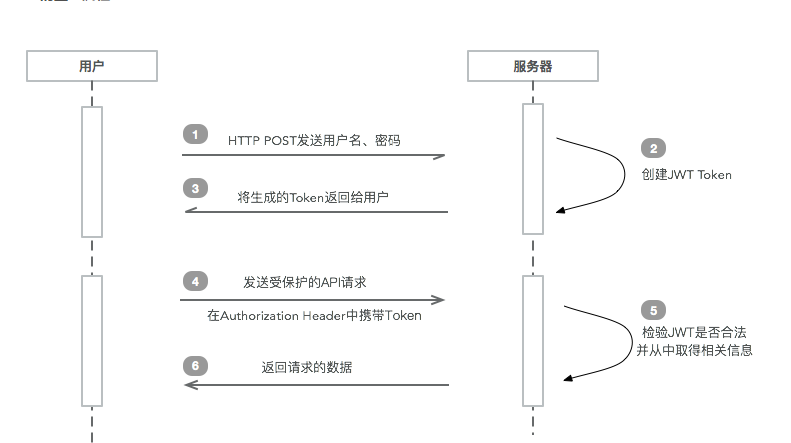
<version>0.6.0</version>

</dependency>  
2. 然后就是生成jwt的工具类，传入一个claims即可，解析jwt文件的工具 类  
3. 创建拦截器类以及配置拦截器类，通过拦截器验证token。  
4. 用户进行登录的时候，去访问jwt的接口，登陆成功以后通过jwt会给用户返回一个token的值，我们主要是把用户的信息保存在redis中，我们会把userid查出来通过jwt进行封装返回，当通过前台业务把jwt返还回来之后把userid再取出来并且将它放入Threadlocal中，它是一个线程绑定的存储空间，相对安全。注册密码使用BCrypt的方法将用户密码传入进去加密，我们登陆首先会从数据库当中获取信息，获取到的结果与加密之后的密码进行匹配，匹配成功就可以完成登录。

优点  
带来的好处  
1.简洁，可以通过URL, POST 参数或者在 HTTP header 发送，因为数据量小，传输速度快。  
  
2.自包含，负载中包含了所有用户所需要的信息，避免了多次查询数据库，服务端也不需要存储 session 信息，做到了服务端无状态。  
  
3.JWT方式将用户状态分散到了客户端中，可以明显减轻服务端的内存压力。除了用户id之外，还可以存储其他的和用户相关的信息，例如该用户是否是管理员、用户所在的分组等。  
  
4.JWT能轻松的实现单点登录，因为用户的状态已经被传送到了客户端。  
  
5.支持移动设备，支持跨程序调用，Cookie 是不允许垮域访问的，而 Token 则不存在这个问题。  
  
6.因为有签名，所以JWT可以防止被篡改  
  
性能问题   
如果将验证信息保存在数据库中，后端每次都需要根据token查出用户id，这就增加了数据库的查询和存储开销。  
Session方式存储用户id的最大弊病在于Session是存储在服务器端的，所以需要占用大量服务器内存，对于较大型应用而言可能还要保存许多的状态，一般还需借助nosql和缓存机制来实现session的存储，如果是分布式应用还需session共享。  
  
兼容问题  
在移动端app里，或者是前后端分离的架构中，用户访问的是前端的web server(如 node.js)，前端的渲染，ajax请求都是由web server完成的，这里就跟传统的不一样了，用户不是直接访问后台应用服务器的，这时候用cookie+session就比较麻烦，问题在于开发繁琐、安全性和客户体验差、有些前端技术不支持cookie(如微信小程序)

首先我们的项目是基于springboot+springcloud实现的 微服务开发，在注册模块中做了一个短信验证码的验证，当时考虑的是防止用户恶意注册给数据库增加压力。这个项目我当时用的是阿里云提供的短信接口。 这里还用到了ribbitMQ。这里为什么要用到MQ呢，就因为考虑到如果在并发的情况下多个用户同时注册会给短信接口造成压力。也可能出现一些问 题。所以用到了消息队列给请求排个队。 我可以给您详细说一下这个注册功能具体实现业务吧。 当用户输入完用户信息手机号之后。点击获取验证码。请求后台获取验证码的方法。首先先要生成一个随机数。当时我是用的apache提供的lang3包的方法。获取了一个6位的随机数。然后把这个6位随机数存到了redis里面一份。并设置过期时间1个小时。然后把这个验证码放到map集合里面放到消息队列里面一份。然后发送短信的接口一直在监听着MQ的消息队列 当接收到发送者发送的消息会请求发送短信方法。给用户发送短信验证码。然后当用户点击注册的时候。会请求到后台注册的方法。首先判断redis里面存储的验证码是否与用户输入的验证码相同。如果相同返回注册成功。如果失败返回验证码错误信息。从而完成这个注册业务。

jwt来验证用户信息.我们在登录成功之后生成token存储到cookie中,校验票据即可访问其他界面.



     1. 用户导航到登录页，输入用户名和密码，进行登录

     2. 服务器对登录用户进行认证，如果认证通过，根据用户的信息和JWT的生成规则生成JWT Token

     3. 服务器将该Token字符串返回

     4. 客户端得到Token信息，将Token存储在localStorage、sessionStorage或cookie等存储形式中。

     5. 当用户请求服务器API时，在请求的Header中加入 Authorization：Token。

     6. 服务端对此Token进行校验，如果合法就解析其中内容，根据其拥有的权限和自己的业务逻辑给出响应结果，如果不通过，返回HTTP 401。

     7. 用户进入系统，获得请求资源

在我们做项目的时候碰到一个问题,我们的token是要设置过期时间的,如果一直是该用户登录,token过期后总需要让该用户重新验证身份信息,会导致用户体验度不高,所以我们在redis中设计了一个表里面存入了用户的id和ip,以及我们在生成token的时候自定义了一个属性来存储用户的ip,这样用户在登录的时候自动存入id以及ip,在token过期后可以判断该用户的ip是否和存入的ip相同,无需登录验证直接实现票据的刷新.

这样也解决了一个问题,因为我们的token是无状态的,我们的服务器是没有办法获取用户信息的,就是如果两个相同终端登录同一个账号,一个token还没过期,有生成另一个token,会导致用户信息的安全问题,在我们的token中存储了ip后,在生成token前可以去redis设计的表中去查询该id是否存在,如果不存在,则添加该用户的id和当前ip.如果存在,则修改该用户的ip为当前ip.生成的token中也存在有我们用户的ip,在第二个客户端登录了之后,第一个客户端的token就是错误的,就可以保证我们用户的信息安全.

1. 注册登录模块：在注册模块，在用户点击发送验证码通过rabbitmq向消息微服务发送一个map类的消息，这个消息包含手机号、验证码等信息（验证码在用户模块已经提前保存在redis中key为sms+手机号，value为验证码），这样消息微服务会对该队列进行监听，接收到消息后调用阿里大于短信接口发送短信，完成注册业务。为了防止有恶意用户进行对接口的调用，前台可以采用把发送验证码的按钮倒计时置灰，后台采用的是布隆过滤器来实现对恶意用户的拦截，如果用户第一次获取验证码，则将手机号经过多个Hash算法为一个整数key，映射到该全为0的bit数组，把key对应的位置的值0变为1。如果过滤器过滤到该手机号对应的位置值为1，则说明该手机号已经获取过验证码（恶意用户）。其中我们用到了redis来统计手机号注册次数并判断执行锁定一小时或者永久冻结等相关操作。

用户进行登录的时候，去访问jwt的接口，登陆成功以后通过jwt会给用户返回一个token的值，我们主要是把用户的信息保存在Threadlocal中，我们会把userid查出来通过jwt进行封装返回，当通过前台业务把jwt返还回来之后把userid再取出来并且将它放入Threadlocal中，它是一个线程绑定的存储空间，相对安全。注册密码使用MD5Util的工具类的encrypt的方法将用户密码传入进去加密，我们登陆首先会从数据库当中获取信息，获取到的结果与加密之后的密码进行匹配，匹配成功就可以完成登录。

搜索模块：

在做这个搜索模块的时候 我们用到的是ES搜索引擎 ：（Elasticsearch是一个实时的分布式搜索和分析引擎。是当前流行的企业级搜索引擎。设计用 于云计算中，能够达到实时搜索，稳定，可靠，快速，安装使用方便

）

我们也想到了用solr 但是通过对比 发现ES更符合我们这个项目的需求

因为ES它在实时搜索的时候效率是高于solr的 Solr更适合传统搜索应用在搜索大的数据的情况下 ES的效率会很高 solr需要配域而ES不需要 相对来说比较简单

我们在做搜索的时候主要是根据用户的需求进行课程搜索，我们是使用docker搭建一个ES的公共服务

一开始我们是用postman调用restapi新建索引和文档 在新建文档的时候 id是由系统自动生成的。在修改文档的时候是根据id来进行修改的 如果id不存在 它就会创建新的文档 删除文档的话也是根据id来进行删除的 通过这个我们发现 如果都是通过rest请求的方式使用ES未免太过麻烦，而且也不够人性化。然后就想到了使用图形化界面来实现ES的日常管理，最常用的就是Head插件 在安装好Head插件后 我们发现了一个问题 就是它在启动之后 点击连接按钮没有反应 按F12发现了一个错误 “No 'Access-Control-Allow-Origin' header is present on the requested resource” 这个错是由于ES默认不允许跨域调用，而es-head是属于前端工程，所以报错。

然后我就修改了ES的配置 让它允许跨域调用

我们在做测试的时候 发现它的分词不太智能 默认的中文分词是将每个字看成一个词 这样是不符合要求的 所以我们就用了IK分词器来解决这个问题 IK 分词器有两种算法 其中ik\_smart为最少切分，ik\_max为最细粒度划分

我们还可以自定义词库

然后 我们又要考虑一个问题 因为我们查询的数据是来自不同的数据库或者表 这样的话就很难以一个比较快的速度去查询数据 这个时候我们就想到了用这个logstash进行同步数据 同步数据库与索引库 （solr是用的定时器 而ES用的是logstash）logstash是一款轻量级的日志搜集处理框架， 咱们在公司里边用这个ES也是分两种 一种是增量索引更新一种是全量索引更新 当我们安装好logstash 需要在logstash的bin文件夹下创建一个.config结尾的文件，在里面配置mysql的链接还有查询课程表的sql语句、es的链接，最主要还是需要配置一个定时任务。这个定时任务就是精确到年月日时分。在这儿我配的是每1小时同步一次。 可以让它定时的去读我的数据库里边的东西，读完之后再把他所读东西搜集起来指定给ES 这样的话就相当于隔1小时ES就被同步一次

ES的核心搜索引擎是通过倒排索引实现的，说倒排索引前我先说一下正排索引，它是通过关键词去文档中直接查找相应的单词，而倒排索引它是先把文档中的内容切分成词条，再经过词干还原、不区分大小写等方式转变成词项存储在倒排索引的数据结构中。倒排索引的数据结构包括由单词组成的词典、单词索引以及关键词出现过的文档编号。当用户在文本框中输入关键词后，es搜索引擎会先去词典中进行查找，找到后会根据单词索引展示文档内容。

支付宝支付整体流程=

当用户点击下单的时候,我们就需要去调用支付接口,所以我们把这个支付接口写在了controller里,专门进行支付宝支付,  
由支付宝去调用,我们就需要传很多参数,参数分为公共参数和请求参数,在公共参数进行传值的时候,我们都用到了Alipayconfig,这个类里提供了APPID,公钥私钥,回调地址等等,我们去调用支付宝封装的一个HTTP柯林特的一个请求,我们使用这个可以直接吧参数往里放,请求参数呢订单号,金额,商品描述,这些传完了之后,支付宝会制动吊起一个支付页面,然后用户就可以进行扫描支付了,当用户付款成功后,(异步请求)在第一次请求回调的时候,支付成功请求回调,第一次会发送两次请求,  
同步请求和异步请求,异步请求不会和同步请求一起回来,不知道他什么时候回来,  同步请求针对于用户,所以我们给用户提供一个页面,来告诉用户付款成功了,为了提供我们平台的安全性,需要我们在做一步验签的工作,我们拿到支付接口对应的demo,验签的时候我们需要一个才可特唯一的一个方法,这个方法支付宝已经给我么封装好了整个验签的五步,验签的方法可以自动完成,我们只需要传支付宝给我们提供的参数,进行验签,验签成功了之后跳转返回success,失败了返回file(失败),成功了我们给用户展示支付成功页面,失败了给用户展示失败的页面,支付成功和验签成功了之后,就证明整套流程已经成功了,我们需要改变订单状态,然后添加一条支付日志,   同步验签也就是为了同步到我们的用户,异步的话是支付宝不定时的给我们发送刚刚用户的订单信息,    开发当中,我们用的是沙箱环境,所以我接不到异步请求,然后我再项目测试当中吧支付功能进一步完善,我们刚开始的时候测试不了异步,支付宝规定异步要找的是公网地址,我们当时服务器还有点问题,所以采用了自己的环境进行开发的,当我们要上线的时候,需要测试,那个时候服务器已经搭建好了,然后我们进行完善,进行与支付宝的异步请求,异步请求就是在整个支付宝里面,支付宝里有异步同步回调(异步回调是4 10 20  ......24小时),超过24小时就不再发送了,24小时之内已经收到了一同消息,就没有写对应的回调

需求:当用户点击提交订单的时候,会跳转到支付页面,在支付页面生成支付宝的二维码,并显示订单号和金额,用户用手机扫描二维码,完成支付功能.当用户支付成功以后跳转到支付成功页面,支付失败的话跳转到错误页面.

订单支付的方式的有很多种,用的比较广泛的有微信和支付宝,我们前期主要用的是支付宝支付.

那么使用支付宝支付呢,我们在开发阶段主要使用的支付宝的沙箱环境,这是支付宝给我们开发人员对外的一种模拟测试的环境,功能

方面和支付宝大致没区别.

在支付宝沙箱后台会有一个应用,这里边有几个配置信息,需要我们注意,因为在项目中我们会用到这几个配置的信息,一个APPID,一个是支付宝网关,这个网关地址中会含有dev三个字母,这说明我们是在沙箱环境中,还有RSA2的加密方式,这里的公钥可以通过支付宝提供的RSA签名工具生成一对公钥秘钥,这里生成的是商户的公钥和秘钥,将商户的公钥放到沙箱后台商用应用公钥中,支付宝会自动匹配出一个支付宝公钥与之对应,另外我们还需要配置一些信息,如支付网关,return-url和notify-url.

return-url:指的是支付宝完成支付之后要跳转的一个页面,所以我们提供的这个url主要用于显示支付结果

notify-url:指的是支付完成后支付宝还要做一个异步的通知,这个接口要求我们处理结束会必须有结果返回,不然支付宝会在25小时内调用8次.调用时间为：4m,10m,10m,1h,2h,6h,15h。如果7次中有一次调用成功了将不会再调用了，如果一直没有调用成功，支付宝会将这条信息状态标记为异常。

我们在这个路径中会进行订单状态的修改,因为这个方法有可能会出现多次,所以我们也要进行去重的处理.具体使用时,用户点击结算订单,我们获取到订单号,支付金额,订单名称,描述信息,提交到我们的后台方法中,在后台直接使用支付宝的SDK就可以完成支付.

具体实现:

因为我们是分布式项目.所以给支付宝支付创建一个服务端,在服务中会有一个方法用于生成alipay的form表单提交,在生成的这个form表单中,有一个script标签会自动提交生成的这个from表单,我们进行了一个小处理,就是将这个script标签去除掉,通过用户在页面上点击支付按钮,在触发form表单提交,这里用到了String类的replace方法,同时我们还需要将订单号和金额返回给前台,订单号我们是使用一种雪花算法生成的,雪花算法是用于分布式高并发下唯一标识的算法,它是由,时间戳,工作机器ID和序列号和一个不用的第一位组成的64位的序列号,它的最大并发量可以在同一秒中自增生成26万个不冲突的可排序的id.在之后这里还使用了一个map集合,将需要返回的数据,全放到这个map集合中,将装好数据的map集合返回给前台.当用户点击订单提交的时候,跳转到支付页面,支付页面会进行初始化,调用后台生成form表单的这个方法,将map中的数据显示在页面中,这里会有一个小问题,就是我们生成的表单是一个字符串,这是由于我们前台使用angularjs,出于安全机制,它会认为你这个数据是一个被注入的不安全数据,就会直接进行展示,所以我们需要用到angularjs的一个sce服务,在js中加入一个filter服务,将这个form表单进行一个过滤处理,这时候就可以正常生成from表单,点击支付按钮以后就会跳到支付宝的支付页面进行一个扫码,就可以完成支付了.