**自我介绍**

我是15年5月份开始做Java工程师的，到现在差不多有四年的时间了，之前就一直在北京工作。现在已经在上一家公司离职了，主要是想要给自己换一个工作环境。我在上一家公司工作了四年的时间。工作期间开发过教育类和金融类的项目。最近在做的就是一个金融类型的项目。

我做的这个金融类的项目是一个P2C的项目，也就是个人对企业的投资。提供的就是个人对于企业的一个投资，为的是帮助中小微型企业。在我们爱投资这个项目平台上，我主要做的是这个项目的权限模块，在做这个权限模块的时候，我们的登陆使用的是基于jwt的token认证机制来进行实现。首先在用户注册时会有一个手机短信验证的功能，在做的时候，还会有选择用户角色，也就是借款人和投资人这两个角色。但是在选择角色的时候，只能选择其中的一个，注册前，选择的角色只能选择其中的一个，注册之前必须阅读《用户注册协议》《网站隐私条款》，在注册的时候，短信验证这个功能我们选择了调用阿里大于的短信验证的这个接口。在注册阿里大于，申请短信接口的时候，我的团队已经准备好了申请短信接口的资料，所以我在申请的时候，直接就去申请了，

Xxx你好，我叫田笑涵，来自河北邢台。有四年的java工作经验，熟练掌握struts2，spring，springmvc，hibernate，mybatis等技术框架。我之前就一直在北京工作。我上家公司是在安投融（北京）网络科技有限公司。我最近的一个项目是一个叫爱投资的金融项目，我主要负责一些权限模块，资金管理模块，借贷模块，还有最新动态这些模块。

上一个项目是爱投资，这个项目主要是一个互联网P2C的金融平台。这个项目在2013年的时候就上线了。我主要负责的就是对这个项目的维护和迭代更新。项目整体采用springBoot、springCloud的微服务架构，数据层使用的是springdataJPA，所有的环境都是基于docker环境，在测试工具方面我们选择使用了postman（postman提供了强大的webAPI & 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。密码加密用的是BCrypt， Spring Security 提供了BCryptPasswordEncoder类,实现Spring的PasswordEncoder接口使用BCrypt强哈希方法来加密密码。

### 2.业务

#### 注册+单点登录

用户的【注册】

注册的流程【用户注册，输入自己的手机号，让后会根据这个手机号去给这个手机发送短信，发送短信我们调用的是阿里云通信的短信接口，根据用户输入的手机号，去后台生成验证码，把验证码存放在redis中。然后把我们的验证码发送到用户的手机上，用户查看输入验证码完成注册。】

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息验证,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

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,然后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,并在这俩端搭建RabbitMQ消息队列进行通信,表呢注册时用到了一个用户表,

技术的介绍

阿里的大于短信微服务接口,是现在一块很不错的短信服务这一块,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,有俩种模式,一种一对一,点对点的模式,另一种呢是一对多的一种模式,一个人发送,有多个人消费

具体的细节

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，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hset的这个方式存的值,给了一个key, find 值,并设置了失效时间

<3>然后就是利用JSM来把们的验证码发送sms服务器，这里我们的jmsTemplate这个对象来操作RabbitMQ发生短信,在这里呢,我们把要发送的信息放到MapMessage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 MapMessage 这对象返回，这样呢,完成了我们用jsm技术来发送到sms的一个操作，接下来呢是我们的sms用jsm的RabbitMQ来接收我们刚刚发出的 消息.在sms里呢 ，我们通过 @JmsListener 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 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.

**短信注册遇到的问题：**【当时遇到的问题：恶意发送短信。

就是恶意攻击者采用自动化工具，调用“动态短信获取”接口进行动态短信发送，原因主要是攻击者可以自动对接口进行大量调用。当时我们采取的措施是：采用图片验证码可有效防止工具自动化调用，意思就是当用户进行“获取动态短信” 操作前，弹出图片验证码，要求用户输入验证码后，服务器端再发送动态短信到用户手机上，该方法可有效解决短信轰炸问题。

图片验证码需要去后台进行校验，每次随机生成的图片验证码都是单次有效。

这样虽然解决了一些在注册页面上的恶意发送短信的攻击，但是如果是一些懂java技术的人员的话，就可以使用各种抓包工具，抓到发送短信的请求，然后就可以通过postman等一些插件，直接跨过注册页面去发送请求。所以我们就在后台设置了一个单次ip请求的次数限制。在服务器端限制单个 IP 在有效的时间内的请求次数，一旦用户请求次数(包括失败请求次数)超出我们设定的次数，则暂停对该 IP 一段时间内的请求。若情节特别严重，可以将 IP 加入黑名单，禁止该 IP 的访问请求。这样就可以限制一个 IP 地址的大量请求，避免攻击者通过同一个 IP 对大量用户进行攻击，增加了攻击难度，保障了业务的正常开展。

】

**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，第二部分称为 playload，第三部分称为 signature。**

**• 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**

**前一段时间在微服务中做用户服务的时候使用到了jwt+redis的登录方案构思了一下登录流程，也不知道合不合理。。仅供参考**

**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。**

**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>jjwt</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  
3. 解析jwt文件的工具类   
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MD5Util的工具类的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)**



#### 个人身份信息认证

当用户注册成功登录后，会跳转到个人中心页面完成用户的基本信息认证。用户在登录成功后，会给用户提示让用户进行实名认证，如果用户没有进行实名认证的话，当用户想要申请借款的时候，他就必须经过一个实名认证的过程，点击申请借款，当用户是第一次申请借款时，就会进入一个让用户实名认证的界面，用户输入自己的姓名和身份证号，点击认证的时候，会进入后台进行实名认证。在这里呢我们主要是调用了阿里云的实名认证接口，对用户输入的身份证进行验证。如果用户输入的身份证号正确的话，就会返回一个json字符串到前台。提示用户认证完成。

 当我们用户身份认证通过后，就可以去绑定银行卡，进行投资和借款等操作，绑定银行卡呢，我们也是调用了阿里云的银行卡认证接口，用户填写自己的姓名，身份证号，银行卡号，需要绑定的银行卡所属银行，和手机号（手机号需要填写银行预留的手机号）。这时候我们为了验证这个银行卡是否是这个用户的，所以会给这个银行卡预留的手机号发送一条验证码，用户输入验证码后，点击绑定银行卡，把数据传输到后台，后台会根据前台传输的数据，调用阿里的接口进行验证。如果所填写的信息正确的话，就会返回一个json字符串，返回前台提示用户银行卡绑定成功。完成所有的认证之后，就可以去进行充值，投资，借款等操作。

#### 投资

投资模块主要分前台和后台，前台投资展示投资的标的（biao ,di），用户在登录和没有登录的时候都是可见的。由于标的（biao ,di）页面数据比较固定，为了提高网站的并发量，这一块我们采用的是freemarker生成静态页面，像已投标的人数，这个则是通过js,异步去加载最新的数据。

我先给您说一下，我们标的页面的生成逻辑（biao ,di），由于我们这个项目是分布式的架构，所以在做这块的时候也遇到了分布式的事物问题，当借款用户在我们平台发起借款后，提交相应的信息,我们的审核人员审核通过后，这个时候借款系统需要通知资金系统，扣除这次借款的服务费，而且要通知投资系统来生成前台的投资标的（biao ,di）页面，因为我们系统是使用dubbo进行远程通信的，通过dubbo当然可以实现，但是如果后期借款的通知系统增加。

，或者是相关系统发生变化，那么借款系统就要修改调用的配置，这样借款系统的可维护性和可扩展性就比较差了，所以我们这一块使用的MQ进行通信，我们使用的是activemq的发布订阅模式，当借款审核通过后，借款系统就直接将消息发送发到mq中，然后其他的系统只要订阅了这个了这个主题，就能获取到发送的消息，然后进行消费，完成相关的业务逻辑。这个时候就有可能出现分布式的事物问题了，当审核通过后要修改数据库借款数据的状态，还要发送借款的信息到mq, 但是这个时候有可能会出现，当更新数据库成功但是发送mq消息失败，这样的话其他系统就收不到这个信息了，就会导致数据的不一致，我们的解决方案是将更新数据库和发送消息放到到一个本地事物中，而且是先更新数据库再发送mq消息，这样只要有一个失败，数据库的数据就会进行回滚，从而保证了数据的一致性，还有就是本地的事物执行成功后，有可能消息还没有被消息的消费者消费mq就挂掉了，因为activemq的发布订阅模式默认是不进行持久化的，所以mq挂掉后消息就丢失了，这样又会导致数据的不一致，针对这个问题我们是将消息进行持久化，这样mq挂掉，也没有关系，当mq启动后，消息就可以重新被发送，还有一种是比较坑的，一开始我们也没有注意，就是消息被消费者拿到了，但是还没有处理完业务逻辑，消息的消费者就挂掉了，由于mq的消息签收机制（ack）默认是进行自动签收的，也就是只要消息被消费者接收，不管有没有处理完，mq就会将持久化的信息进行删除，这样也有可能数据不一致，这块我们使用了mq的手动签收机制，在程序都执行完成，我们在最后调用了message.acknowledge(); 进行手动的签收，从而保证业务被处理完成后，mq才会删除消息。

有了投资标的页面后，**投资人**就可以对这些标进行投标了，但是用户在没有登录的时候只能看到标的一些基本信息（像标的名称，介绍，已投资人数，结束时间，以及收益率，借款人的信息等），

登录以后如果没有实名认证和绑定银行卡，则需要实名认证和绑定银行卡完成后，进行充值 然后就可以完成投资了。

投资完成后，则账户里面的资金就会冻结，当在规定的时候满标后，就可以进行放款，放款后冻结的金额会变成投资金额。

如果在规定的时间内没有完成借款则会流标，流标后用户的钱就会退回账户，变成可用金额。

　理财这么久了

　　还在傻傻的用以下公式计算收益?

　　利息=【1+(期限/年)×年化收益率】×本金

　　虽然没有错，但你还是要了解平台的还款方式才行，毕竟它会直接影响到你的收益，[**八点金融**](http://www.8dbank.com/)分析目前互联网金融理财的还款方式最常见的有以下几种：

　　一次性还本付息--最简单直白

　　一次性还本付息是最容易理解的，即在借款周期开始时，出借本金，周期结束时，回收本金和利息。例如标的是1万元，约定期限1个月，年化利率是12%，投资人投资1万元，1个月后，得到本金1万+利息100元，共计10100元。总的来说一次性还本付息的模式对投资人来说是最简单明了的。

　　先息后本--猫腻小，最靠谱

　　先息后本从字面上理解就是先还利息后还本金，其实应该加上“按月”两个字，因为所谓先息的意思是指按月先息，即按月还息，到期还本。举上次例子来说，标的为1万，周期是3个月，年化利率是12%，投资人投资1万元，每个月收到100元的利息，连续收2个月，第三个月到期后收100元利息+1万本金。

　　等额本息--投资前要仔细计算

[**八点金融**](http://www.8dbank.com/)解析等额本息是指，将借款本金和利息总额之和等月拆分，借款人每月偿还相同数额的本息部分。这种计算本息还款的方式中，借款人每月还款额中的本金比重逐月递增、利息比重逐月递减。

　　等额本息还款公式为：每月还款额=[贷款本金×月利率×(1+月利率)^还款月数]÷[(1+月利率)^还款月数-1]

　　等额本金--收益最小

　　等额本金与等额本息类似，不同的是等额本金只是将借款本金平均到每个月，利息也是需要付清上次还款日和此次还款日之间的利息。

　　这种还款方式适用于高收入的人群，因为还款前期的每月还款金额比较大，但是越到最后，其负担会越来越轻，而且其总利息也是比等额本息较少。而对于投资者，该种还款方式的收益比较于等额本息来说还是更小的。

[**八方金融**](http://www.8dbank.com/)指出不同还款方式的差别

　　differences

　　主要体现在以下两个方面：

　　1、周期

　　超短期(低于45天)的项目一般都是一次性还本付息，中短期的项目(6个月以内)一般都是先息后本的方式，长期和超长期的项目一般会是等额本息。

　　2、收益

　　比如都是标的1万，周期1年，12%年化收益，4种还款方式最后算出来的总收益是不同的，一次性还本付息与先息后本的收益都是1200元，等额本息是661.85元，等额本金是650元。

　　除了要考虑的标的的长短、还款方式，最重要的还是要考虑平台的综合实力，平台如果实力强大的，投长一点的借款标也是安全的。

#### Shiro

权限管理：  
权限管理其实就是做认证和授权，用户去访问系统，系统要验证用户身份的合法性，通过验证后，用户就可以访问系统的资源，但系统也会对资源进行控制，用户只有具有资源的访问权限才能访问。我们这里是用shiro这个安全框架，shiro也是现在比较流行的做权限管理的框架，而且实现起来也不会太难，可以有效提高我们的开发效率，降低开发成本。Shiro的核心组件有subject、securityManager、realm（领域）【authenticator（认证器）、authorizer（授权器）】这三个，subject是主体，可以是用户也可以是程序，主体要访问系统，系统需要对主体进行认证、授权；securityManager是安全管理器，主体进行认证和授权都是通过securityManager进行；realm领域，相当于数据源，通过realm存取认证、授权相关数据【cache Manager：缓存管理器，主要对session和授权数据进行缓存，比如将授权数据通过cacheManager进行缓存管理，和ehcache整合对缓存数据进行管理】，我们的存储授权和认证逻辑都是写在realm里的。Spring也支持跟shiro的整合，我们在pom.xml文件中加入shiro-core、shiro-web和与spring整合的shiro-spring【与任务调度quartz整合的shiro-quartz等】，或者是直接加shiro-all也可以,shiro的认证流程是先构建securityManager环境，然后subject主体调用login()提交form表单去做认证，会传一个token，这里有个要注意的点，提交form表单的时候用户名是username，密码是password，当然这个可以修改，但是我认为没有那个必要，提交之后通过securityManager.login()调用authenticator【ModularRealmAuthenticator】执行认证，authenticator最终其实还是通过realm来实现的，前面也说了，realm里放我们自定义的逻辑，我们要自定义一个realm，建一个类继承AuthorizingRealm，重写doGetAuthenticationInfo方法来完成认证，这里的主要逻辑是先通过token来获取身份信息【token.getPrincipal()】，拿它跟数据库里取出的信息作对比，shiro也给我们提供了的方法，用SimpleAuthenticationInfo(userCode, password, this.gteName)就可以实现，要传的参数第一个是用户的身份信息，第二个用户输入的密码，查询不到会返回null,查询到返回认证信息AuthenticationInfo,这里所说的密码都是通过加密的【加密需要配凭证匹配器】，我们使用的MD5加密。通过认证后可以实现登录，因为我们的平台有经理和员工两种角色，经理和员工的权限肯定有区别，这就要重写管理授权的doGetAuthorizationInfo方法了，通过principals获取主角色，查询数据库主角色的权限，把权限添加到SimpleAuthorizationInfo里，然后通过注解和标签的方式实现授权【在方法上用@RequiresPermissions("item:query")注解的方式，页面上<shiro:hasPermission name="item:update"></shiro:hasPermission>标签的方式】，对于已经登录的用户第一次去数据库查权限，第二次获取权限是从缓存里，这样做是考虑到性能和数据库压力的问题，缓存shiro也已经帮我们做好了，但默认是关闭的，我们缓存用的是Memcache来做，做缓存的话就还有一个清除缓存的问题，正常退出和非正常退出的清除缓存shiro帮我们做了，但我们还需要手动实现权限修改后调用realm的clearCache方法清除缓存（clearCache是自己写），这个其实是满足客户的需求，这基本就是我们的权限管理。