# 云计算 1607C 白树浩

# JAVA工程师

1. 项目介绍(按照实训二做的项目写)

|  |
| --- |
| 第一个项目是一个金融平台.我们这个金融平台是分为用户前台和管理后台两部分.主要功能有个人中心，账户，借贷项目，资产详情，后台管理等模块.  因为公司的金融业务涉及面比较广,可能涉及到和很多银行帐户之间的资金往来.一个公司中也会有不同银行的帐户.我们项目开发时一共整合了32家银行的银企直连接口.我负责的是兴业银行和深圳发展银行的部分.  我们在开发的时候有八个人,由人专门负责前台的页面,有人专门写后台,我就是写后台的,前台用的是VUE,有时候也会帮着他们解决下问题。我的工作主要就是根据需求文档，将两家银行所涉及到的业务开发出相应的接口，比如像查询余额的接口，支付接口，支付状态查询接口，代付代扣接口等。  银行的接口大多是采用XML进行数据交互，根据银行下发的银企直连接口文档中的说明，里面有请求和响应的格式，每次请求的数据和响应回来的数据都可以分为两部分。一部分是身份认证。一部分是我调用的接口所需要的数据正文。响应回来的也一样。一部分是身份认证结果，一部分是接口处理结果。拼正文的话很简单，new Element对象，然后用addContent方法把它们嵌套起来。因为每次调用都要做身份认证，所以我为每家银行分别开发了一个工具类。我只需要把接口的正文拼出来，然后传给工具类。工具类里有身份认证的XML拼接，工具类会把XML的声明标签、身份认证、接口需要的数据，这三部分组装到一起做返回。然后通过我们封装的一个工具类将信息发送给银行。我们没有采用httpClient发送请求，因为当时测试过如果用我们封装的发送请求大概是2秒，httpClient大概是5秒。我们封装就是通过JDK原生的封装的，效率高一些，这个我们高级工程师做的，当时看了看。接收响应数据也一样。我也开发一个工具类，将响应回来的数据传给工具类，工具类先取身份认证结果，里面有个标签时code 。如果是0那么就是身份认证成功，那么就会取出来接口调用结果做返回。如果code不是0那么就直接返回错误信息，用log 记录并抛异常。  当发送请求时候也会遇到问题，比如接收不到相应数据，有时候又可能是网络原因。当时根据银行提供的解决方法。如果接收不到响应回来的数据的话，我们会重复发送3次请求，这三次请求的流水号必须一致，这样银行就不会做重复操作了，这个流水号我们是用long类型的系统当前时间拼接上一个随机数组成的。如果银行第一次请求处理完成，但是我们没有受到数据，那么我们会发送第二次请求。因为两次流水号相同，银行再次受到请求就不会作重复操作，而是返回之前的结果。如果三次都没有接收到响应数据，我们会给订单表示为待确认状态，然后开启一个线程，让其15分钟之后调用状态查询接口，根据流水号查询。如果还是没有结果。那么我们会把这个请求标记为异常。只能联系工作人员人工处理了。 |

1. 功能模块介绍(按照自己得实际项目写)

|  |
| --- |
| 1. 开户模块（流程）：新用户通过平台网站发起开户请求，平台系统校验用户身份，同时向管存系统发起开户请求，管存系统接收开户请求，并开户管存个人开户页面，用户在管存开户页面输入姓名，身份证号，银行卡号，预留手机号四要素信息，预留手机号接收短信验证，完成交易密码的设置，并完成开户协议的勾选后，进行提交，管存系统对用户数据进行处理，返回的开户结果，由平台通知用户成功或者失败。 2. 借款申请模块：平台点击我要借款，首先会判断用户是否登陆，没有登陆的话，跳去登陆。我们假设已经登陆，我们会先判断用户是否满足借款条件，有四条：个人用户填写，实名制认证，风控资料分数达到可借款的分数，以及有没有进行视频认证，我们假设这些申请都已经满足，用户就可以进行借款申请，如果当前用户已经有一个借款申请在申请中，则不能再次申请借款，假设不存在借款申请，借款人填写相关信息后将数据传到后台。 3. 满标一审：对于借款主要包括几个重要的对象：bidrequest：借款信息,bid：投资人每次的投标记录Accountflow：账户流水。比如说您想要在我这里借款，我是投资人，当我投资的金额达到您的借款金额时，就进入了满标一审的状态，在满标一审审核时，首先后台要对您的信息再次核对，是否存在当前借款信息，是否处于满标一审状态，如果不满足条件，前台提示用户不满足条件，进行资质审核如果都没有问题，当管理员点击审核通过按钮时，借款对象(bidrequest)的state状态就变成了满标二审状态。同时生成审核历史对象。当管理员点击审核拒绝时，说明该借款进入了满标拒绝的状态，需要退换投资人的钱，因为涉及到了钱的变动，我们在这里有一个账户的流水对象accountflow，这个对象包含当前流水金额，当前流水的状态以及流水的备注信息。用于记录退还投资人的投资的流水，同时该借款人的个人信息中的借款状态移除。借款人可以再次发起借款，投资人也可以再次投资。 4. 当借款进入满标二审审核时，也要对信息进行审核，判断是否有该借款对象以及该标是否是满标二审状态，当后台管理员点击的是审核通过，此时借款人借款成功，那么我们需要修改借款状态为还款中、增加借款人的账户余额、增加借款人账户待还本息、减少借款人可用信用额度、移除借款人借款状态码、支付借款手续费，平台账户的可用余额增加，我们平台当时是对于不足6个月的借款，将会收取借款金额的2%作为服务费用，至于6个月及6个月到12个月的借款，收取的是4%的服务费用，生成平台账户收取手续费流水。对于投资人来说待收本金和待收利息增加，生成成功投资流水，同时我们考虑到后续借款人需要分期还款，所以在成功借款之后生成每一期还款对象(Paymentschedule)和回款对象：还款明细PaymentscheduleDetail：回款明细，还款对象是针对与借款人的，回款对象是针对投资人的，还款对象包含每一期借款人需要还款金额，而回款对象包含投资人每一笔投资生成的本金和利息，这样便于投资人查看每一笔资金的收益情况（生成还款和回款对象）。 5. 当管理员点击审核拒绝时，说明该借款进入了满标拒绝的状态，需要退换投资人的钱，因为涉及到了钱的变动，我们在这里有一个账户的流水对象accountflow，这个对象包含当前流水金额，当前流水的状态以及流水的备注信息。用于记录退还投资人的投资的流水，同时该借款人的个人信息中的借款状态移除。借款人可以再次发起借款，投资人也可以再次投资。 6. 还款模块：借款人根据还款对象来分期，首先判断当前用户账户余额是否大于还款金额，小于则提醒用户账户余额不足，去充值界面充值，假设账户金额大于还款金额，点击还款还款时需要更新几个表的数据，比如投资人账户可用余额增加，代收本金减少，生成账户成功收款流水，若是最后一期，则还款金额应该等于总金额减已经还款的金额，同时还款人的还款状态也要取消等等。 7. 吐槽微服务: 用户可以开启吐槽模式，开始疯狂吐槽。吐槽这个模块相对来说并不是很重要的，即使丢几条吐槽数据也不没有关系，所以我们采用**MongoDB** 非关系型数据库，非关系型数据库的一个特点就是，吞吐量大，可以满足我们这个需求。我们在设置**MongoDB** 的主键的时候是\_id，他不会像**MySQL**一样任何字段都可以设置主键，他的主键必须有下划线。我们在表中还设置了一个父级id这样做是为了让用户可以回复其他用户吐槽的内容，提高用户体验度，也因为这个特性，我们表设计的结构是树形结构。吐槽还有一个点赞的功能，首先，用户必须为登录状态才能进行点赞，在数据初始化的时候我们点赞数就初始化为0，一个用户只能点赞一次，如果我们使用**MySQL**数据库实现点赞效果，我们得先去数据库查点赞数量，然后进行点赞数修改，每次要与数据库进行两次交互。而使用了**MongoDB**数据库后，我们利用**mongoTemplate**对象把**MongoDB**原生命令转为java代码这样我们每次只需要对数据库进行一次交互，降低服务器的压力。用户点赞的时候把用户的id与吐槽的id组成键值对存入**Redis**，当用户进行点击点赞按钮时，先去**Redis**中进行判断用户是否点赞，如果没有，就进行点赞操作，如果已经点赞，则进行取消点赞操作，取消点赞操作成功后把**Redis**中该用户的id进行删除。 8. 短信微服务: 调用第三方短信接口（支付宝，百度），在用户注册时候发送验证码给用户的手机，把第三方生成的验证码与用户的id组合为键值对，保存到**Redis**缓存中，并给这个键值对设定过期时间，当用户输入验证码进行提交操作后，根据用户id进行查询，把**Redis**中保存的验证码与用户的验证码进行比对，如果一样，进行添加用户的操作，同时保存用户手机号，以及把手机号的状态码加入到用户的状态码中，意义为用户已经进行了手机号绑定，用户可以使用手机号或者用户名进行与密码进行登录，也可以使用手机号与验证码进行登录，当然执行地逻辑还是一样的。 |

1. 开发中遇到的技术难点以及解决办法

|  |
| --- |
| 难点1：redis到缓存击穿，缓存雪崩，过期策略以及内存淘汰机制。  解决方法2：比如redis只能存5G数据，可是你写了10G，那会删除5G的数据，怎么删的，还有就是数据已经设置了过期时间，但是时间到了，内存的占用率还是很高，有考虑过原因吗？  Redis采用的是定期删除+惰性删除策略  定时删除，用一个定时器来负责监视key，过期则自动删除。虽然内存及时释放，但是十分消耗CPU资源，在大并发请求下，CPU要将时间应用在处理请求，而不是删除key，因此没有采用这一策略  那定期删除+惰性删除策略是怎么工作的呢  定期删除redis默认每个100ms检查，是否有过期的key,有过期key则删除。需要说明的是，redis不是每个100ms将所有的key检查一次，而是随机抽取进行检查(如果每隔100ms,全部key进行检查，redis岂不是卡死)。因此，如果只采用定期删除策略，会导致很多key到时间没有删除。  于是惰性删除派上用场。也就是说在你获取某个key的时候，redis会检查一下，这个key如果设置了过期时间那么是否过期了？如果过期了此时就会删除。此时也会产生其他的问题,就有了内存淘汰机制。  在redis.conf中配置一行数据maxmemoey-policy volatile-lru/  推荐使用allkeys-lru：当内存不足容纳新写入的数据，在键空间中，移除最近最少使用的key。推荐使用。  就到了缓存穿透和缓存雪崩的问题  在中小型企业中很少一件这类的问题，缓存击穿解决方案（一）利用互斥锁，缓存失效时候，先去获得锁，得到了锁，再去请求数据库。没得锁，则休眠一段时间重试。  （二）采用异步更新策略，无论key是否取到值，都直接返回。Value值中维护一个缓存失效，缓存如果过期，异步起一个线程去读取数据库，更新缓存。需要做缓存预热。  缓存雪崩  给缓存一个失效时间，加上一个随机值，避免集体失效。  使用互斥锁，但是该方案吞吐量明显有所下降。  双缓存。我们两个缓存，缓存A和缓存B。缓存A的失效时间为20分钟缓存B不设置失效时间。  自己做缓存预热操作。然后细分一下几个小点  1从缓存A中读取数据库，有则直接返回。  2A没有数据，直接从B读取数据，直接返回，并且异步启动一个更新线程。  3更新线程同时更新缓存A和缓存B。  难点3：登录状态的问题（jwt） 。  解决方法3：分布式web应用的普及，通过session管理用户登录状态成本越来越高，因此慢慢发展成为token的方式做登录身份验证，然后通过token去取redis中的缓存的用户信息，随着之后的jwt的出现，校验方式更加简单便捷化，无需通过redis缓存，而是直接根据token取出保存的用户信息，以及对token可用性校验，单点登录更为简单。  使用JWT核心代码。引入依赖  会有一个id 主题 签发者 签发时间 签名的算法以及秘钥  Jwt工具类和token验证  难点2：借款模块需要对分布式事务进行解决  解决方法1：借款时就相当于是投资人向借款人转账的一个情况，但是仔细分析里面就有很多的业务，首先一个标肯定是有多个投资人共同投资，那么此时有一个投资人想一个标投资了500元那么在他的充值余额中的五百元就锁定，不能操作，或者是直接先将金额-500元。当次标满标而且每层审核都通过后，此时就进行实时的操作，投资人的账户减钱，借款人的账户加钱，就用到了分布式项目中TCC方案。如果中间出现了审 核不通过的情况就相当于是事务失败回滚进行对投资人的钱进行原样返回。  难点3：安全登录：我们使用Spring Security实现用户的安全登录。编写配置文件spring-security里面。我记得有个http的标签，可以对静态资源放行，并设置登陆成功的跳转页面，默认的登录页面。当用户发送登录请求时，通过配置文件。我们自定义一个认证类，实现UserDetailsService接口。重写loadUserByUserName方法，然后创建角色列表，将在配置文件里面配置的角色放到列表里面，根据用户名去数据库查询是否存在该用户，如果存在将用户的密码与输入的密码进行比对，返回userDetails。这样就完成了安全登录。  JWT token验证：  为了保证接口的安全性，我们使用JWT实现跨域访问，当用户登录成功后，后台生成一个token，就是一个String字符串，这个token包含一些信息，比如当前用户的唯一标识id，用户的手机号，Token的签发时间以及token的过期时间。过期时间我们设置为7200秒。  将这个登陆成功状态信息和token一起返回给前台。前台获取到这个token之后，每次请求接口时，都在请求头携带者token。我们在后台通过拦截器拦截到前台的请求，解析token，如果解析成功，将token重新放到request里面，当调用后台接口是，会首先判断request里面有没有token，如果token存在，可以接做接下来的操作，如果不存在，则向前台响应对应的状态码以及错误提示。  难点4：登录状态的问题（jwt） 。  解决方法3：分布式web应用的普及，通过session管理用户登录状态成本越来越高，因此慢慢发展成为token的方式做登录身份验证，然后通过token去取redis中的缓存的用户信息，随着之后的jwt的出现，校验方式更加简单便捷化，无需通过redis缓存，而是直接根据token取出保存的用户信息，以及对token可用性校验，单点登录更为简单。  使用JWT核心代码。引入依赖  会有一个id 主题 签发者 签发时间 签名的算法以及秘钥  Jwt工具类和token验证 |

1. 项目技术亮点分析

|  |
| --- |
| (1) 项目中我使用了RabiitMQ来实现:解耦、异步、削峰  解耦:如果使用 MQ，A 系统产生一条数据，发送到 MQ 里面去，  哪个系统需要数据自己去 MQ 里面消费。如果新系统需要数据，直接从 MQ 里  消费即可；如果某个系统不需要这条数据了，就取消对 MQ 消息的消费即可。这  样下来，A 系统压根儿不需要去考虑要给谁发送数据，不需要维护这个代码，也  不需要考虑人家是否调用成功、失败超时等情况。  异步:  一般互联网类的企业，对于用户直接的操作，一般要求是每个请求都必须在 200 ms 以内完成，对用户几乎是无感知的。  如果**使用 MQ**，那么 A 系统连续发送 3 条消息到 MQ 队列中，假如耗时 5ms，A 系统从接受一个请求到返回响应给用户，总时长是 3 + 5 = 8ms，对于用户而言，其实感觉上就是点个按钮，8ms 以后就直接返回了，爽！网站做得真好，真快！  削锋:  每天 0:00 到 12:00，A 系统风平浪静，每秒并发请求数量就 50 个。结果每次一到 12:00 ~ 13:00 ，每秒并发请求数量突然会暴增到 5k+ 条。但是系统是直接基于 MySQL 的，大量的请求涌入 MySQL，每秒钟对 MySQL 执行约 5k 条 SQL。  一般的 MySQL，扛到每秒 2k 个请求就差不多了，如果每秒请求到 5k 的话，可能就直接把 MySQL 给打死了，导致系统崩溃，用户也就没法再使用系统了。  但是高峰期一过，到了下午的时候，就成了低峰期，可能也就 1w 的用户同时在网站上操作，每秒中的请求数量可能也就 50 个请求，对整个系统几乎没有任何的压力。  如果使用 MQ，每秒 5k 个请求写入 MQ，A 系统每秒钟最多处理 2k 个请求，因  为 MySQL 每秒钟最多处理 2k 个。A 系统从 MQ 中慢慢拉取请求，每秒钟就拉取  2k 个请求，不要超过自己每秒能处理的最大请求数量就 ok，这样下来，哪怕是  高峰期的时候，A 系统也绝对不会挂掉。而 MQ 每秒钟 5k 个请求进来，就 2k 个  请求出去，结果就导致在中午高峰期（1 个小时），可能有几十万甚至几百万的  请求积压在 MQ 中。  这个短暂的高峰期积压是 ok 的，因为高峰期过了之后，每秒钟就 50 个请求进 MQ，但是 A 系统依然会按照每秒 2k 个请求的速度在处理。所以说，只要高峰期一过，A 系统就会快速将积压的消息给解决掉。 |
|  |

1. 项目答辩中亮点简历审核评定标准

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **考核内容** | **简历样式是否规范** | **项目内容是否完整** | **项目中是否有亮点** | **技术难点解决办法是否高效** | **存在问题** |
| **项目经理 （评定）** |  |  |  |  |  |
| **就业处干事（评定）** |  |  |  |  |  |
| 简历是否合格 | | | | 合格 | 不合格 |

1. 项目答辩成绩评定标准

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **考核内容** | **亮点简历与项目描述是否匹配** | **项目流程是否清晰** | **学生讲解是否流畅** | **项目介绍是否到位** | **技术亮点考核是否合格** | **存在问题** |
| **项目经理 （评定）** |  |  |  |  |  |  |
| **就业处干事（评定）** |  |  |  |  |  |  |
| 项目答辩是否合格 | | | | 合格 | 不合格 |  |