云计算 1607C 王宏洋

# JAVA工程师

1. 项目介绍(按照实训二做的项目写)

|  |
| --- |
| 景云金融PC是一款互联网技金融的网络借贷平台。平台的模式为信贷公司提供平台，由借贷双方自由竞价，撮合成交，资金借出人获取利息收益，并承担风险，资金借入人到期偿还本金，网络信贷公司收取中介服务费；  该项目使用Spring Boot+Spring Cloud搭建微服务框架，运用Spring Cloud里面的组件如Feign，Ribbon实现之间的相互调用，Eureka将各个微服务注册在服务中心下，并实现集中式管理，Zuul实现网关API微服务，并向前端提供统一的调用接口，实现了前后端分离。Hystrix使用令牌桶算法实现服务熔断，服务限流和降级，并使用Mybatis做数据持久层 。Redis缓存首页部分数据，Apache ActiveMQ /阿里大于短信接口。在Docker容器中安装软件，通过docker宿主机的端口和ip进行连接应用软件，实现操作。 |

1. 功能模块介绍(按照自己得实际项目写)

|  |
| --- |
| 1. 个人注册：**：**平台注册是手机号注册，判断手机号是否被注册，如果被注册则提示错误，没有则Apache ActiveMQ使用它的主要目的是为了降低耦合性，和为了缓存才使用的的异步处理场景使用（阿里大于）发送短信验证，填写验证码，和设置自己的平台密码。用户注册需要对用户信息进行初始化，有三个阶段，第一阶段是判断手机号是否存在第一阶段通过才能进行第二阶段，短信验证，还有设置密码，正则判断密码是由数字和字母组成，不是此规范则提示错误消息，第三步则为之策成功页面，那么此时会跳转到个人页面。提醒用户需要开户。注册成功后我们的用户里面的信息首先都要进行初始化用户的信用额度，和初始剩余信用额度。 2. 开户和登录是有区别的用户的jwt登录使用zuul网关进行鉴权。过程是这样的为了保证接口的安全性：为了保证数据库产生脏数据 3. **个人用户开户：**注册的新用户实现借款投资功能，就要实现开户功能。用户需要开户，就要向平台进行申请开户，平台权限向银行请求开户，银行则返回一个表单去填写信息，填写完向平台返回信息，审核通过后用户开户成功。返回信息的时候不会返回交易密码给平台，交易的时候还是用户和银行之间流通金额，所以，平台只是起一个中间商的关系，从中收取利息，根据国家政策的支持，账户的充值只能向银行进行充值，如果向平台进行充值的话，那么会造成一种现象就是非法集资。 4. 首页：向用户展示一些成交的总金额，还有一些用户注册人数，那么这些数据都是一些热数据，在实时进行更新，我们的首页还有一些广告，都是从数据库中读取，这样当网站访问量达到高峰时段，对数据库压力很大，并且影响执行效率，那么我们也需要将这一部分数据缓存起来。这就是用到了redis，根据id进行修改我们的广告数据 5. 借款页面：首先用户想要进行借款，首先要先进行开户和认证模块，开户模块已经完成，那么需要进行认证模块，首先需要先申请一个借款标，平台返回一个页面填写标的，标题，借款的金额，借款利息，借款期限，还款方式（按月分期，按月到期）等额本息还款法，最小投标，标的标题和内容，填写完后提交到后台进行审核，审核成功后进入招标阶段，审核失败后此标改为审核失败状态，用户的此项操作无论成功还是失败都需要记录。按照规定的时间内完成招标达到满标状态，则进入满标一审，规定时间没有完成招标，此标状态改为流标状态。满标之后进入后台进行满标一审，满标一审通过进入满标二审，不过则把状态改为流标状态，把钱退还给投资人，   通过则进入满标二审阶段。通过满标二审则改为还款的状态，不过则改为流标，把钱退还给每个投资人。  （6）投资人对标进行投资不需要进行任何的审核。 |

1. 开发中遇到的技术难点以及解决办法

|  |
| --- |
| 难点1：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2：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3：部署问题  解决方法3：前端是APP 或PC 调用网关，网关分发到每个微服务，那么问题来了。App可以不通过网关访问微服务，怎么解决。此时将微服务的ip设置为内网ip。网关有两个一个是外网的，一个是内网的外网主要是接受外部的消息，内网主要是处理外网传来的消息进行处理。  。。。。。。。。。。。  难点4：SQL优化  解决方法4：一、对于查询较慢的sql根据慢日志定位查询SQL使用explain指令分析sql（explain关键字段：type ：mysql查询数据的方式）  修改sql或者尽量让sql走索引（避免sql出现\*，避免where对字段进行null值的判断，尽量少使用or等等）。  二、使用不同的数据库、 不同的协议版本分别进行缓存。尽量将一些热数据，访问量大的数据放入到缓存中（redis，memcache等）。  三、我们的数据访问层在执行sql的时候，也是要先需要去解析。可以直接把一些大sql，IO消耗比较大的sql放入到存储空间内，预编译执行。  四、存储过程是一段写好的SQL代码，他是存在数据库的目录中，外部程序可以直接调用数据库里面定义好的存储过程 |

1. 项目技术亮点分析

|  |
| --- |
| (1) SpringCloud p2p 项目中运用到，比如借款里面有一些需要用到Common模块里面的接口那么就需要用feign调用。我们技术负责人在技术选型的时候，考虑到后期项目的扩展与升级，我们选择了当前比较流行的SpringCloud微服务框架。这样使我们项目拆分成若干微服务模块，然后使用eureka实现注册信息的集中化管理，若干微服务将自身服务注册到eureka下，然后为服务之间的电泳是通过feign来实现的（其实springclooud微服务之间的调用是基于rest实现的，这个与dubbo的rpc远程服务调用是有区别的）  同时为了方便实现鉴权、访问控制，我们使用zuul做了一个对接前台的网关模块，同时这些微服务的配置信息，我们采用config组件，将配置信息托管在gogs(码云、github)服务下，同时通过springCloud-bus消息总线实现配置信息变更后的消息通知已经完成配置更新操作。同时在系统健壮性方便我们引用如Hystrix技术，实现服务限流、熔断以及服务降级。  （2）SpringCloud服务限流之令牌桶算法  在高并发的应用中，限流往往是一个绕不开的话题，在zuul上限流是个不错的选择，只需要编写一个过滤器就可以了，关键在于如歌实现限流的算法。常见的限流算法有令牌桶算法：对于很多应用场景来说，除了要求能够限制数据的平均传输速率外，还要求允许莫种程度上的突发传输。这时候漏铜算法可能就不合适了，令牌桶算法更为合适，令牌桶算法的原理是系统会以一个恒定的速度往桶里放入令牌，而如果请求需要被处理，则需要先从桶里获取一个令牌，当听力没有令牌可去世则拒绝服务的请求。Google开源工具包Guava提供了限流工具类RateLimter，该类基于令牌桶算法来完成限流，非常易于使用。当我们的预读取的令牌越多时，那么他暂停的时间就越长，下次的预读取的令牌就越少。 |
|  |

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 |  |