# 云计算 1607C 翟帅卿

# JAVA工程师

1. 项目介绍(按照实训二做的项目写)

|  |
| --- |
| 梦次元  梦次元是一个以二次元动漫ACG以及相关资源交流分享为主的动漫平台,为用户提供动画新番,漫画,音乐,游戏等资源下载和分享等功能,同样也是二次元交友平台,大家在这里可以欢畅淋漓的结交到与自己志趣相同的朋友;  项目采用前后端分离的技术架构，前端采用vue.js构建，服务端采用Spring Cloud微服务架构，系统分为用户层、CDN、负载均衡、前端UI、微服务层、数据层、接口层及DevOps等部分组成 |

1. 功能模块介绍(按照自己得实际项目写)

|  |
| --- |
| 1.图片服务器  本项目采用fastDFS分布式系统作为图片服务器。  FastDFS是用c语言编写的一款开源的分布式文件系统，适合小文件的存储。  FastDFS包括 Tracker server和Storageserver。客户端请求Tracker server进行文件上传、下载，通过Tracker  server调度向Storage server完成文件上传和下载。  使用FastDSF官方提供的Java API实现。  图片服务使用Nginx作为代理服务器，对Storage上部署的Nginx完成负载均衡请求。  2.图片上传流程  1、管理员进入教学管理前端，点击上传图片  2、图片上传至文件系统服务，文件系统请求fastDFS上传文件  3、文件系统将文件入库，存储到文件系统服务数据库中。  4、文件系统服务向前端返回文件上传结果，如果成功则包括文件的Url路径。  5、课程管理前端请求课程管理进行保存课程图片信息到课程数据库。  6、课程管理服务将课程图片保存在课程数据库。  3.认证授权:  本项目采用 Spring security + Oauth2完成用户认证及用户授权。认证授权流程如下：  1、用户请求认证服务完成身份认证。  2、认证服务下发用户身份令牌和JWT令牌，拥有身份令牌表示身份合法，Jwt令牌用于完成授权  3、用户携带jwt令牌请求资源服务。  4、网关校验用户身份令牌的合法，不合法表示用户没有登录，如果合法则放行继续访问。  5、资源服务获取jwt令牌，根据jwt令牌完成授权。  4.CMS页面管理:  CMS （Content Management System）即内容管理系统，本项目对CMS系统的定位是对各各网站（子站点）页  面的管理，本项目的CMS系统不去管理每个子网站的全部资源，比如：图片、CSS、html页面等，主要管理由于运  营需要而经常变动的页面，从而满足根据运营需要快速开发、上线的需求。  功能包括：  1、站点管理，站点就是本项目各各子网站，站点信息包括：站点名称、站点域名、端口、服务器物理路径等。  2、模板管理，由于要对页面进行静态化，使用freemarker引擎技术，所以需要定义模板。  3、页面管理，包括：页面添加、页面修改、页面删除等操作。  4、页面预览，对页面静态化，在浏览器预览页面静态化内容。  5、页面发布，将页面静态化后发布到所属站点服务器。  5:媒体资源管理模块:  1、上传媒资文件  前端/客户端请求媒资系统上传文件。  文件上传成功将文件存储到媒资服务器，将文件信息存储到数据库。  2、使用媒资  课程管理请求媒资系统查询媒资信息，将课程计划与媒资信息对应、存储  3、视频播放  用户进入学习中心请求学习服务学习在线播放视频。  学习服务校验用户资格通过后请求媒资系统获取视频地址。   1. 文件上传模块:   1.通常视频文件都比较大，所以对于媒资系统上传文件的需求要满足大文件的上传要求。http协议本身对上传文件大  小没有限制，但是客户的网络环境质量、电脑硬件环境等参差不齐，如果一个大文件快上传完了网断了，电断了没  有上传完成，需要客户重新上传，这是致命的，所以对于大文件上传的要求最基本的是断点续传  2.断点续传指的是在下载或上传时，将下载或上传任务（一个文件或一个压缩包）人为的划分为几个  部分，每一个部分采用一个线程进行上传或下载，如果碰到网络故障，可以从已经上传或下载的部分开始继续上传  下载未完成的部分，而没有必要从头开始上传下载，断点续传可以提高节省操作时间，提高用户体验性。   1. 视频处理   原始视频通常需要经过编码处理，生成m3u8和ts文件方可基于HLS协议播放视频。通常用户上传原始视频，系统  自动处理成标准格式，系统对用户上传的视频自动编码、转换，最终生成m3u8文件和ts文件，处理流程如下：  1、用户上传视频成功  2、系统对上传成功的视频自动开始编码处理  3、用户查看视频处理结果，没有处理成功的视频用户可在管理界面再次触发处理  4、视频处理完成将视频地址及处理结果保存到数据库  8.媒体资源服务端:  服务端需要实现如下功能：  1、上传前检查上传环境  检查文件是否上传，已上传则直接返回  检查文件上传路径是否存在，不存在则创建  2、分块检查  检查分块文件是否上传，已上传则返回true  未上传则检查上传路径是否存在，不存在则创建。  3、分块上传  将分块文件上传到指定的路径。  4、合并分块  将所有分块文件合并为一个文件。  在数据库记录文件信息。 |
|  |

1. 开发中遇到的技术难点以及解决办法

|  |
| --- |
| 难点1:在处理订单时要用到定时任务，当时采用的是Spring Task来完成，由于一个订单服务会部署多个，多个订单服务同时去处理任务会造成任务被重复处理的情况，如何解决任务的重复处理?  解决方法1：  采用乐观锁解决，在任务表中设置一个version字段记录版本号，取出任务记录同时拿到任务的版本号，执行前对  任务进行锁定，具体的做法是执行update根据当前版本号将版本号加1，update成功表示锁定任务成功，即可开始  执行任务  难点2：如何保证消息不会被重复消费  解决方法2：比如你拿个数据要写库，你先根据主键查一下，如果这数据都有了，你就别插入了，update 一下好吧。比如你是写 Redis，那没问题了，反正每次都是 set，天然幂等性。比如你不是上面两个场景，那做的稍微复杂一点，你需要让生产者发送每条数据的时候，里面加一个全局唯一的 id，类似订单 id 之类的东西，然后你这里消费到了之后，先根据这个 id 去比如 Redis 里查一下，之前消费过吗？如果没有消费过，你就处理，然后这个 id 写 Redis。如果消费过了，那你就别处理了，保证别重复处理相同的消息即可。比如基于数据库的唯一键来保证重复数据不会重复插入多条。因为有唯一键约束了，重复数据插入只会报错，不会导致数据库中出现脏数据。  难点3:如何保证消息的可靠性传输？或者说，如何处理消息丢失的问题？  **R**abbitMQ 弄丢了数据  就是 RabbitMQ 自己弄丢了数据，这个你必须开启 RabbitMQ 的持久化，就是消息写入之后会持久化到磁盘，哪怕是 RabbitMQ 自己挂了，恢复之后会自动读取之前存储的数据，一般数据不会丢。除非极其罕见的是，RabbitMQ 还没持久化，自己就挂了，可能导致少量数据丢失，但是这个概率较小。  设置持久化有两个步骤：   * 创建 queue 的时候将其设置为持久化 这样就可以保证 RabbitMQ 持久化 queue 的元数据，但是它是不会持久化 queue 里的数据的。 * 第二个是发送消息的时候将消息的 deliveryMode 设置为 2 就是将消息设置为持久化的，此时 RabbitMQ 就会将消息持久化到磁盘上去。   必须要同时设置这两个持久化才行，RabbitMQ 哪怕是挂了，再次重启，也会从磁盘上重启恢复 queue，恢复这个 queue 里的数据。  注意，哪怕是你给 RabbitMQ 开启了持久化机制，也有一种可能，就是这个消息写到了 RabbitMQ 中，但是还没来得及持久化到磁盘上，结果不巧，此时 RabbitMQ 挂了，就会导致内存里的一点点数据丢失。  所以，持久化可以跟生产者那边的 confirm 机制配合起来，只有消息被持久化到磁盘之后，才会通知生产者 ack 了，所以哪怕是在持久化到磁盘之前，RabbitMQ 挂了，数据丢了，生产者收不到 ack，你也是可以自己重发的。 消费端弄丢了数据 RabbitMQ 如果丢失了数据，主要是因为你消费的时候，刚消费到，还没处理，结果进程挂了，比如重启了，那么就尴尬了，RabbitMQ 认为你都消费了，这数据就丢了。  这个时候得用 RabbitMQ 提供的 ack 机制，简单来说，就是你必须关闭 RabbitMQ 的自动 ack，可以通过一个 api 来调用就行，然后每次你自己代码里确保处理完的时候，再在程序里 ack 一把。这样的话，如果你还没处理完，不就没有 ack了？那 RabbitMQ 就认为你还没处理完，这个时候 RabbitMQ 会把这个消费分配给别的 consumer 去处理，消息是不会丢的。  难点3:如何保证消息的顺序性？ RabbitMQ 拆分多个 queue，每个 queue 一个 consumer，就是多一些 queue 而已，确实是麻烦点；或者就一个 queue 但是对应一个 consumer，然后这个 consumer 内部用内存队列做排队，然后分发给底层不同的 worker 来处理。  难点4:如何解决消息队列的延时以及过期失效问题？消息队列满了以后该怎么处理？有几百万消息持续积压几小时，说说怎么解决？  解决方案: 大量消息在 mq 里积压了几个小时了还没解决 几千万条数据在 MQ 里积压了七八个小时，从下午 4 点多，积压到了晚上 11 点多。这个是我们真实遇到过的一个场景，确实是线上故障了，这个时候要不然就是修复 consumer 的问题，让它恢复消费速度，然后傻傻的等待几个小时消费完毕。这个肯定不能在面试的时候说吧。  一个消费者一秒是 1000 条，一秒 3 个消费者是 3000 条，一分钟就是 18 万条。所以如果你积压了几百万到上千万的数据，即使消费者恢复了，也需要大概 1 小时的时间才能恢复过来。  一般这个时候，只能临时紧急扩容了，具体操作步骤和思路如下：   * 先修复 consumer 的问题，确保其恢复消费速度，然后将现有 consumer 都停掉。 * 新建一个 topic，partition 是原来的 10 倍，临时建立好原先 10 倍的 queue 数量。 * 然后写一个临时的分发数据的 consumer 程序，这个程序部署上去消费积压的数据，消费之后不做耗时的处理，直接均匀轮询写入临时建立好的 10 倍数量的 queue。 * 接着临时征用 10 倍的机器来部署 consumer，每一批 consumer 消费一个临时 queue 的数据。这种做法相当于是临时将 queue 资源和 consumer 资源扩大 10 倍，以正常的 10 倍速度来消费数据。 * 等快速消费完积压数据之后，得恢复原先部署的架构，重新用原先的 consumer 机器来消费消息。  mq 中的消息过期失效了 假设你用的是 RabbitMQ，RabbtiMQ 是可以设置过期时间的，也就是 TTL。如果消息在 queue 中积压超过一定的时间就会被 RabbitMQ 给清理掉，这个数据就没了。那这就是第二个坑了。这就不是说数据会大量积压在 mq 里，而是大量的数据会直接搞丢。  这个情况下，就不是说要增加 consumer 消费积压的消息，因为实际上没啥积压，而是丢了大量的消息。我们可以采取一个方案，就是批量重导，这个我们之前线上也有类似的场景干过。就是大量积压的时候，我们当时就直接丢弃数据了，然后等过了高峰期以后，比如大家一起喝咖啡熬夜到晚上12点以后，用户都睡觉了。这个时候我们就开始写程序，将丢失的那批数据，写个临时程序，一点一点的查出来，然后重新灌入 mq 里面去，把白天丢的数据给他补回来。也只能是这样了。  假设 1 万个订单积压在 mq 里面，没有处理，其中 1000 个订单都丢了，你只能手动写程序把那 1000 个订单给查出来，手动发到 mq 里去再补一次。 mq 都快写满了 如果消息积压在 mq 里，你很长时间都没有处理掉，此时导致 mq 都快写满了，咋办？这个还有别的办法吗？没有，谁让你第一个方案执行的太慢了，你临时写程序，接入数据来消费，消费一个丢弃一个，都不要了，快速消费掉所有的消息。然后走第二个方案，到了晚上再补数据吧。  。。。。。。。。。。。 |

1. 项目技术亮点分析

|  |
| --- |
| (1)项目框架：项目采用得**spring boot+springcloud**微服务框架，好处是......  (2)**redis技术**：项目中点赞模块使用到redis了，具体是这样使用的.......  (3)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. 断点传续方案   通常视频文件都比较大，所以对于媒资系统上传文件的需求要满足大文件的上传要求。http协议本身对上传文件大  小没有限制，但是客户的网络环境质量、电脑硬件环境等参差不齐，如果一个大文件快上传完了网断了，电断了没  有上传完成，需要客户重新上传，这是致命的，所以对于大文件上传的要求最基本的是断点续传。  什么是断点续传：  引用百度百科：断点续传指的是在下载或上传时，将下载或上传任务（一个文件或一个压缩包）人为的划分为几个  部分，每一个部分采用一个线程进行上传或下载，如果碰到网络故障，可以从已经上传或下载的部分开始继续上传  下载未完成的部分，而没有必要从头开始上传下载，断点续传可以提高节省操作时间，提高用户体验性。  (5)两阶段提交协议(2PC)  为解决分布式系统的数据一致性问题出现了两阶段提交协议（2 Phase Commitment Protocol），两阶段提交由  协调者和参与者组成，共经过两个阶段和三个操作，部分关系数据库如Oracle、MySQL支持两阶段提交协议，本节  讲解关系数据库两阶段提交协议  1）第一阶段：准备阶段（prepare）  协调者通知参与者准备提交订单，参与者开始投票。  协调者完成准备工作向协调者回应Yes。  2）第二阶段：提交(commit)/回滚(rollback)阶段  协调者根据参与者的投票结果发起最终的提交指令。  如果有参与者没有准备好则发起回滚指令。 |
|  |

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 |  |