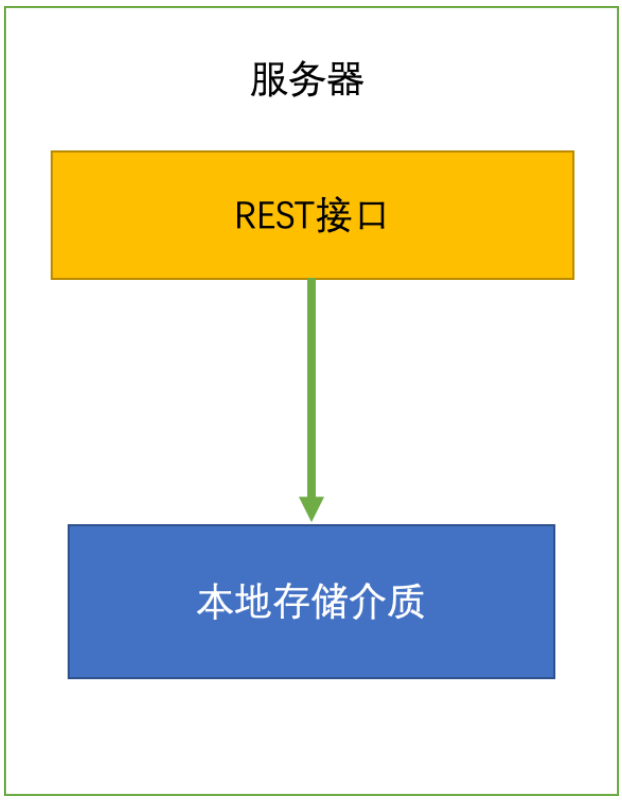
单机版存储架构

流程讲解



单机版的架构非常简单，服务器首先持续监听，通过REST接口判断该对外提供什么服务（调用什么函数返回什么）。具体的逻辑实现交给PUT和GET函数实现。两个函数也很简单，PUT就是把对象写入到磁盘里，GET就是从磁盘里检索。（路径拼接规则当然是我们规定好的）

不足分析

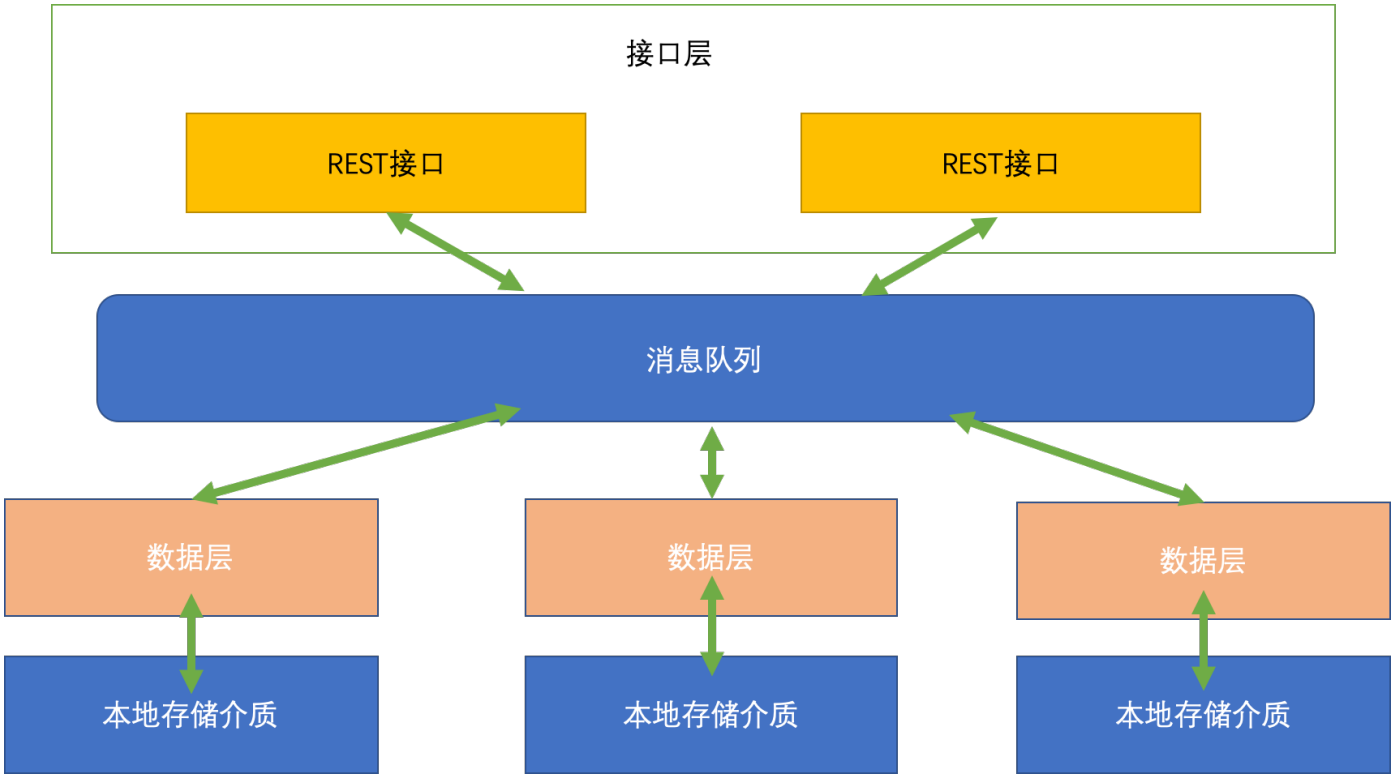
这样有个问题，假如你的网盘服务就这样设计，那这台机器硬盘存满了怎么办？显然我们需要可扩展可插拔的分布式系统。让磁盘快满的时候系统自动将新业务分配到其他空节点，保证业务的顺畅。

分布式系统初建

流程讲解

一开始的单体架构只需要接到任务就执行，因为只有一个地方可以存，而扩展成分布式以后，新来的对象要等待系统给他分配目标节点，才能知道被存在哪，就相当于有了一个分流层，可以理解为负载均衡的思想吧。

这里我们使用一个消息队列来完成这一层的任务



如果大家学过MVC分层设计或者类似的思想就会很容易理解，大家可以把最上面的理解为service业务层，中间类比controller，最下面Dao层。我可能表述不准确但只要你理解就ok。

这样分层的好处就在于，我们可以动态地进行存储节点增删，最上面业务接口完全不需要了解我存放东西的地址在哪，我只需要向消息队列问就可以，消息队列会帮忙统计有哪里可以存，也不需要告诉业务层存在哪了，只在业务层取东西的时候去下面找到即可。

打个比方，你去超市要存包，你只需要给服务员即可，不需要知道他把包具体放在哪个格子了，只要这个服务员记住了到时候能找到给你就行，而就算柜子满了，再加一个柜子也与你无关，服务员会整理对应的信息。这个服务员就充当了消息队列的角色。如果没有这个歌服务员，你需要自己记住柜子的编号去存取，而且柜子满了你也没办法继续存了。

分布式系统中需要设置心跳信号去管理节点，每个节点通过周期性发送心跳信号告诉消息队列自己正在状态正常，意外down掉的节点会被消息队列发现踢出分发列表，新来的业务就不会被发往有问题的地址导致存储失败。

有关心跳信号的实现和周期性管理在代码中都有详细注释。

不足分析

假设我们在写论文的时候，要经过论文1.0、论文1.1、论文1.2一直很多次上传备份，如果老师又要改之前的版本，又希望可以找回旧版本。我们该怎么做？（其实这处的思考有些像产品经理了，不过作为一个优秀的开发人员，普通的需求还是要想到）

版本控制

流程讲解

我们需要在存入数据的时候，添加一些标记，类似在拍照的时候其实每张照片都会有些元数据包括：位置信息，时间，镜头型号，像素，作者等，为什么有些手机P完图还可以恢复原图？因为实际上P完图存的是一个新版本在手机里，原图没有被删除而是做了个标记隐藏起来了。

所以我们在保存的时候就要给对象添加一个版本号，正常来说如果文件名是一样的话，那么我们就要让version自动+1，读取文件的时候如果没有指定版本号我们就取最新版本（这很符合逻辑对吧）（如果你了解MySql的MVCC就会很轻易理解）

我们需要用一个东西来存住数据的版本我们不能使用Go语言的map，原因自己想一想，一定要用数据库，至于数据库的选型，Mysql当然可以，Redis也可以，不过你如果了解ES滚动索引特性，你就会采用ES。

（技术选型这一环节也是面试中可以大谈特谈的一部分，你做项目就是为了表现自己的实力，如果你能清楚地讲出三个数据库的优缺点，选型的过程，而不是说别人项目里都用Redis我就用Redis，相信会在面试官脑海中留下很深的印象）

所以PUT存的逻辑是

1. 根据元信息查询这个对象在系统里最大的版本号maxversion
2. 如果maxversion没查到就是新的对象要存，直接走流程存进去maxversion=1
3. 如果maxversion不等于1说明有旧版本在，那么走流程存进去且该对象的元信息中version=maxversion+1

GET的逻辑也很清晰

1. 首先正常流程查有没有这个对象在，没有直接返回问题
2. 有的话判断有没有指定版本号，制定了就按照版本号拿到地址读取（一般存的时候会用名字+版本号来存）
3. 没有指定版本号就查一下最新版本是多少，走流程把最新的返回出去

不足分析

现在又出现一个问题，如果我不断上传同一个文件到网盘，系统每次都会随机分配一个节点去储存。结果我上传了十次，有十个节点存了一样的文件，这有必要吗？想象一下，如果一万个人都上传了同一部电影，有必要在服务器里存一万个副本吗？显然这是设计上的缺陷，下一节我们来进行解决。

数据校验去重

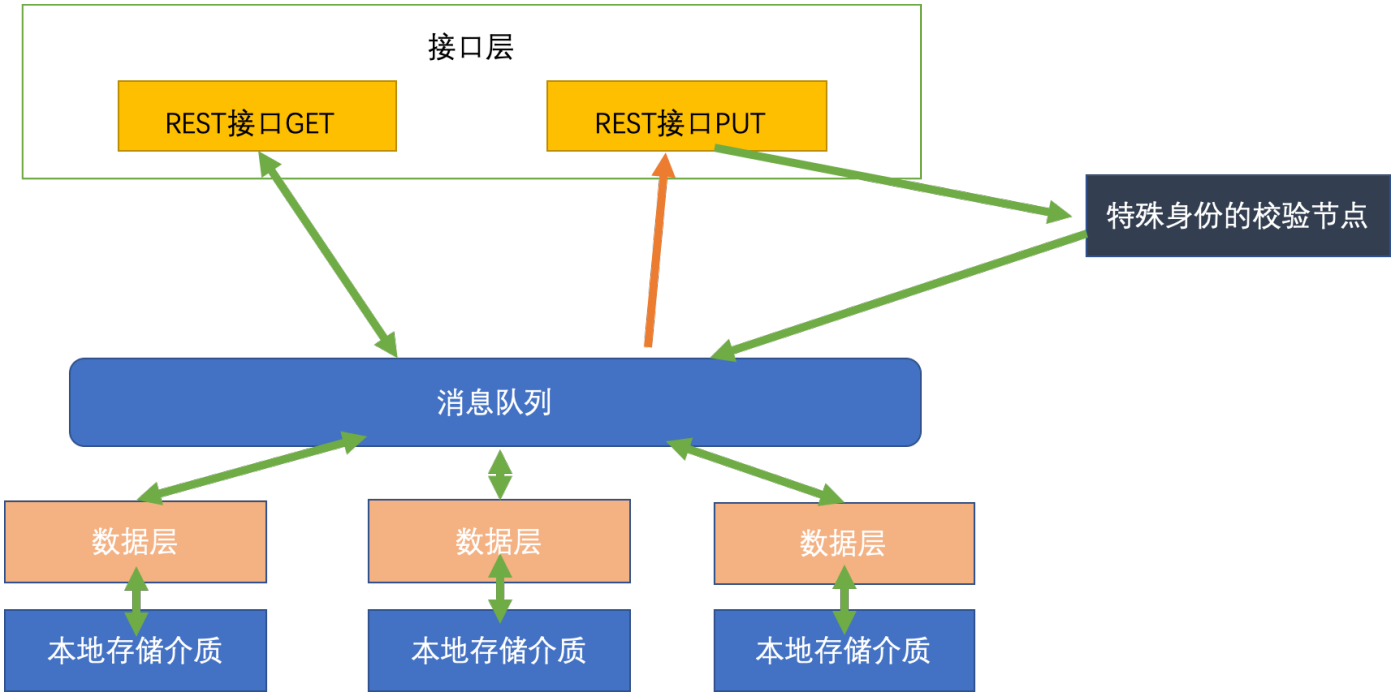
如何判断两个对象是否一样？相信大家能想得出一种方法，哈希值。哈希函数有很多，我们选择SHA-256，世界上从未出现过碰撞，对于我们的项目来说，足够。

那么这个问题就解决了？每次上传时候校验一下哈希值呗，哪有那么简单。

假如你要上传一个200G的蓝光文件，我问你，你怎么校验?

全放在内存里？200G内存全让你占了？

显然我们需要把文件上传到一个其中一个拥有特殊身份的节点里进行校验。我们拿出刚才的一个图来讲解



注意看GET是不受影响的，PUT上传文件的时候我们需要进行这一步，这个节点其实本质上也是个存储服务器，只不过我们让他运行的东西跟别的节点不一样，想要上传，先过他这一关，把东西先给他检查一下，确实系统里没存过，那就走流程存进去。存过了的话就直接问问消息队列原来那份存在哪了就可以。

举个形象的例子，你去图书馆捐书（这个图书馆每本书就存一本），那管理员就会先看看我们有没有这本书，有了的话就不需要了，没有的话，这个管理员会交给其他整理书籍的管理员去存书，第一个管理员就是这个校验节点，后面负责去把书放到书架的管理员就是原来的消息队列，就是这个道理。

不足

通过我们的校验去重，终于让这个系统中每个文件都是独一二的了，但问题又来了，很显然，你独一无二的，万一你这个节点没了，那我文件就彻底找不回来了，没有备份啊，所以我们还需要进行适当的备份。看起来很矛盾，我们费尽心思删除了重复的，我们又希望有重复的副本来做容灾。但其实不矛盾，前面我们是为了高效利用空间，提高系统运行效率，避免重复工作占用资源，后面我们是为了高可用，角度是不一样的，事实上我们也不需通过把一个文件复制完整的几分来实现这个功能。

RS纠删实现数据恢复

流程讲解

为了提高可用性，我们面对大文件和小文件是两种策略，小文件直接做几个副本分散就可以，大文件副本代价太高了，我们采用切片的方式拆分，搭配上RS就删码的思想去实现数据恢复。

关于RS，希望大家去百度一下学习思想，三言两语说不清楚。

你只需要知道RS可以达到，用一些碎片拼凑出完整的整体。我们不再需要完整的副本备份，只需要一些碎片的副本就可以恢复出大文件的整体，这在用户量大的系统中能显著降低存储成本。

不足分析

现在好似一切都进展顺利，我们的系统被设计成分布式、可扩展、高可用、高效率的，但这些都建立在网络良好的前提下，事实上再好的网络也会存在重连现象。

你的迅雷下载99%网断了，重新连接后是从0开始重新下载吗？如果是这样，那原来下一半的东西就成了垃圾，反复几次你会发现磁盘被垃圾占据没有空间了。

所以需要断点续传功能。

断点续传

其实思想很简单，你都不需要自己想，学过TCP没有？TCP怎么进行网络传输的?

切片，一个包一个包传输对吧，断了怎么办？记住传到第几个了，从头部的偏移量开始往后数，第几个断的就从第几个开始传。就是这么简单，但在代码的实现上，确实有一些难度，可能大部分同学看起来有些吃力。

但我还是那句话，面试中重在思想的碰撞，实现的完整度不重要，没人要求你现场跑一遍，展现出你严谨的思考过程，完美解答面试官设下的追问，足以拿下面试。

文档不断完善中，随时更新