

OSS运维 进阶实战手册

云运维工程师从入门到精通

作者：韩笠



- 《OSS运维基础实战手册》姊妹篇
- 阿里云工程师多年从业经验
- 9大经验场景问题排查技巧
- 超详细自动化工具介绍及性能调优





 阿里云 开发者社区



云服务技术大学
云产品干货高频分享



云服务技术课堂
和大牛零距离沟通



阿里云开发者“藏经阁”
海量免费电子书下载

目录

OSS 自动化工具介绍 4

OSS 工具之 OSSBrowser	4
OSS 工具之 ossutil	10
OSS 工具之 ossimport	18
OSS 工具排查案例	25

视频媒体类 41

OSS 图片处理	41
OSS- 视频截帧	58
OSS 直播推流功能	61

经验场景问题分析 70

OSS 网络拥塞分析	70
OSS signature 计算失败分析	77
OSS crc64 出现不一致分析	83
OSS 上传文件异常分析	85
OSS RequestTimeTooSkewed 问题分析	90
OSS 访问文件 404 分析	93
OSS Postobject 问题分析	96
OSS url 编码问题分析	100
OSS 结合访问控制	102

OSS 性能调优 114

OSS python SDK 调优	114
OSS ossutil 工具调优	119
OSS 公网上传调优	131
OSS Java SDK 调优	135

OSS 自动化工具介绍

OSS 工具之 OSSBrowser

浅谈

ossbrowser，图形版的操作工具，有控制台的基本功能，可以理解是 ossutil 工具的图形版，适用于一些非技术人员来操作 oss，但是性能上并不如 ossutil 那么给力。

使用须知

- ossbrowser 支持断点续传，以及一键暂停和一键恢复。
- ossbrowser 最大支持文件大小是 5GB。
- ossbrowser 最大支持的上传 / 下载线程数最多是 10。
- 遇到多文件并发上传 / 下载 时，需要将并发线程数设置为 10，如果并发的文件 size 百兆 /G，请替换成 ossutil 工具。

经典案例

案例一：

驻云工具无法加载 bucket 中 object。



排查

- 如图是一个第一个非 oss 官方的第三方工具，用户可以尝试在客户端做下基本的 ping 测试先看下网络是否通。
- 检查登陆的 AccesskeyID 的权限是否可以将 bucket 下的内容 list 出来。
- 用 ossbrowser 测试下，看同样的 AccesskeyID 登陆后是否也会报错，如果 ossbrowser 可以正常显示，证明策略没有问题，是第三方放工具的问题。可以联系驻云公司看下是否配置上有特殊的地方。

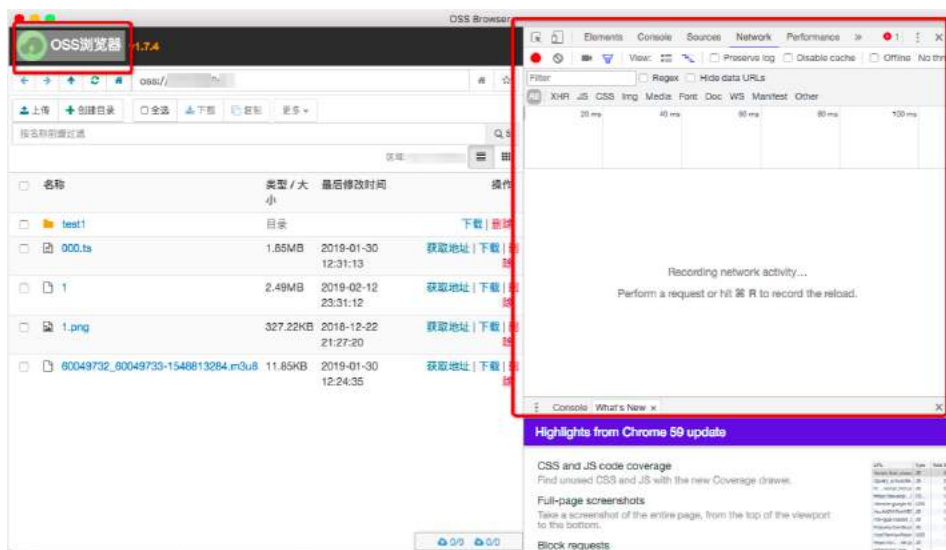
案例二：

- 第一种情况，客户端在国内通过 ossbrowser 上传到国外 OSS，通过公网上传慢，或者网络超时，大量并发，文件 size 比较大。
- 第二种情况，客户端在国外，跨国传输到其他国家的 OSS，比如美国传到香港 OSS 通过公网上传很慢，进度条来回回退，大量并发，文件 size 比较大。
- 国内上传到国内通过公网上传，大量并发，文件 size 比较大。

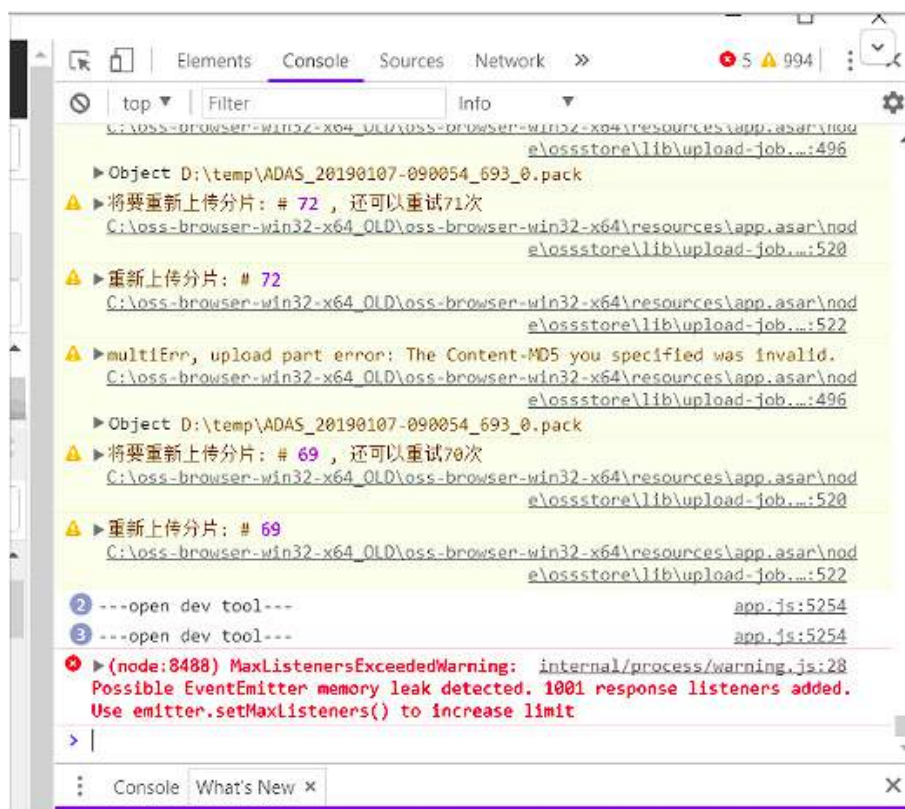
排查

这几种情况统一进行分析排查。

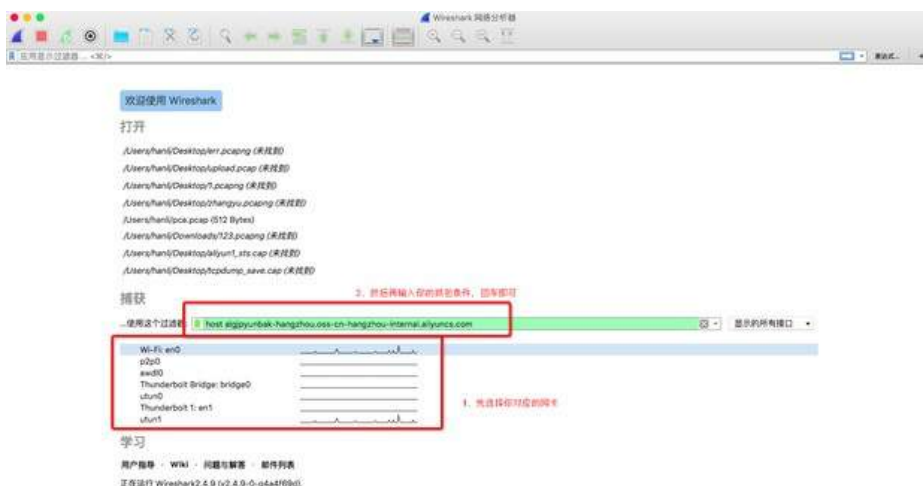
- 1) 首先通过脚本进行网络基础探测，看下客户端的网络延迟、丢包、等指标是否正常；[下载地址](#)
- 2) 当网络延迟不高的情况下、也没有出现明显的丢包，我们要考虑下自己当前设置的 上传 / 下载文件数量是不是很多（超过 10），size 既不是超过了 100M。如果是，请将设置的线程数量提升到最大 10 个，然后点击左上图片 15 下开启 debug 模式，如图。



- 3) 开启 debug 后，将之前的任务先清理掉然后重新上传，看右侧的 debug 是否有明显的错误信息或者断点重传的标记，如图就是在网络丢包超时情况下触发了断点续传，是正常现象，如果断点的出现次数很多，需要关注网络了。



4) 开启网络抓包, 可以用 tcpdump 也可以用 wireshark, 操作如图。



Wireshark - 专家信息 - WLAN (host aptiv.oss-cn-hongkong.aliyuncs.com)

严重	概要	组	协议	计数
Error	Malformed packet (capture error)	Malformed	HTTP	
Warning	Previous segment(s) not captured (common at c...	Sequence	TCP	4
Warning	ACKed segment that wasn't captured (common ...	Sequence	TCP	3
Warning	This frame is a (suspected) out-of-order segment	Sequence	TCP	427
Note	This frame is a (suspected) spurious retransmission	Sequence	TCP	6
Note	This frame is a (suspected) fast retransmission	Sequence	TCP	652
Note	This frame is a (suspected) retransmission	Sequence	TCP	1192
Note	Duplicate ACK (#1)	Sequence	TCP	11239
Chat	Connection establish acknowledge (SYN+ACK): s...	Sequence	TCP	1
Chat	Connection establish request (SYN): server port 80	Sequence	TCP	1
Chat	Connection finish (FIN)	Sequence	TCP	2
Chat	TCP window update	Sequence	TCP	21

抓包结果分析

通过抓包分析发现客户端到服务端的网络重传较多，而且还触发了 TCP 的快速重传，势必导致客户端的发送窗口大小下降；而且大量的丢包重传，拥塞客户端的网络将本来带宽就小得网络出口很容易打满。

总结

当经过前几步排查还是无法定位了的话，只能通过抓包来分析，最后给出的解决方案。

- 客户端替换 ossutil 上传，这样可以自定义分片大小，降低大文件传输时造成的大量重传拥塞。
- 客户端放置一个美国的 OSS 替代香港的 OSS 避免跨国的公网抖动。
- 解决客户端网络丢包的问题；扩容带宽。

案例三：

金融云登陆 OSSbrowser 报错。

Error: Hostname/IP doesn't match certificate's altnames: "Host: xxx.oss-cn-szfinance-a.aliyuncs.com. is not in the cert's altnames:

或者类似其他 endpoint 的证书报错。

排查：

- 1) 先看报错的 endpoint 是哪个 region，是公用云还是专有云、金融云。通过这个报错可以知道是金融云内网。
- 2) 看客户端完整的 OSS browser 登陆配置如何，一般除了公有云、金融云公网以外都是不支持 https 的，那么本次的错误可以看出是由于内网环境，所以并不支持 https。
- 3) 为什么内网（金融云、专有云）不支持 https 呢，因为已经是内网了，也就是访问者肯定都是内部的人才能接入，所以 https 的用户不在也、同时也无法进行证书中心的验证，自然就去掉了 https 的使用。

OSS 工具之 ossutil

浅谈

本章节结合一些实际遇到的案例讲一下各种工具使用。

- 在遇到问题时先确保自己的工具一定是官方的最新迭代版本。
- 使用工具遇到问题时如果遇到 4xx 3xx 2xx 500 等问题，oss 都会返回一个 x-oss-requestID 的 http 头，value 是一串字符，类似 5BEE7AD4C84D-1C447120083C 这个对于排查分析问题非常重要。
- 每个工具基本都带有 log 功能，请使用者务必开启，遇到问题时可以追根溯源。

使用场景

- ossutil 专司于高并发大文件或小文件读写的场景，同时支持大文件内部进行分片时的并发。
- 操作文件时可以支持 include exclude 参数，指定哪些后缀的文件可被操作。
- 只显示当前的层级的目录，可选择性的进行递归。
- 伴随 Linux 系统本身的 crontab 使用支持，强制更新 -f -u 参数。
- 限制 ossutil 的操作速度。

案例一：

ossutil 出现 skip 情况。

```
[root@iZ2Sv4olcc4Z opt]# echo "testlil" > dskydb/test.txt
[root@iZ2Sv4olcc4Z opt]# ./ossutil64 cp -rt -c ~/.ossuti.Lcofig dskydb/test.
txt oss://gres/test.txt
Succeed: Total nun: 1, $ze: 30. OK nun: 1(upload 1 files).
0.372650(s) elapsed I
[root@iZ2Sv4olcc4Z opt] echo "ttest222r" >> dskydb/test.txt
```

```
[root@iz2Sv4olcc4Z opt]. /ossutil64 cp -u -c ~/.ossutilconfig cbkydb/test.
txt oss://gres/test.txt
Succeed: Total num: 1, ize: 38. OK num: 1(skip 1 files), Skip sin 38.
0.252878(s) elapsed
```

排查:

- 1) 使用 -u 强制更新时, 重新对所有的上传文件和原的进行一次比对, 发现美有更改的情况就会跳过, 有发生更改的就会触发上传进行覆, 正常情况。
- 2) 遇到这种情况可以看下本地的 log 哪些文件被跳过了, 将 oss 存储的文件和本地文件做个 MD5 比对确保文件是一致。
- 3) 命令: ./ossutil64 cp -u -r -f aa.test oss://alihua -i -k -e。

案例二:

```
Scanned 213790 objects, Restored 87351 objects, Error 242 objects, Error: Get http://oss-cn-shenzhen.aliyuncs.com/aliyun-ossutil64/aliyun-ossutil64.exe: dial tcp 120.77.167.31:80: connect: cannot assign requested address, Bucket=aliyun-ossutil64, Object=aliyun-ossutil64.exe, Size=11099200, MD5=22f1441720354242905, logPrefix=:
```

排查:

如图用户在操作解冻文件的过程中出现 403, 可能与两个原因有关系。

- 用户使用的子账号操作文件, 权限不够。
- 用户的文件是违禁内容被封禁掉了。

PS: 遇到 403 时, 递归解冻会中断不会继续。这种现象是正常的, 工具在设计之初就是考虑到如果文件遇到 403 的话, 代表没有权限操作该文件, 那么通过该账号下的其他文件也操作不了, 所以就会中断退出。

案例三:

```
[Credentials]
Endpoint=oss-cn-shenzhen.aliyuncs.com
AccessKeyID=AKIAJ2Q911172019
AccessKeySecret=
[root@l19 luqingying]# ./ossutil64 restore oss://... -r -f -c ./oss_mowheng.conf
Error occurs, message: oss: service returned error: StatusCode=400, ErrorCode=OperationNotSupported, ErrorMessage=5A44C3630E4917459DA56386, Bucket=..., Object=jiahaoyy.jpg, See more information in file: o
Scanned 153800 objects. Restored 19911 objects, Error 133 objects.
```

排查:

- 检查用户的命令和官方提供的命令是否完全一致，避免误操作。
- 使用 stat 选项看一下文件的状态是否是已经解冻了，如果以及解冻再次操作，就会出现 400。

案例四:

ossutil 操作 object 出现 “The operation is not valid for the object's state”。

排查:

出现这个问题是因为用户操作的文件是一个归档的文件，不能直接操作，需要先进行解冻 restore 的操作后才能使用。

```
ossuti64 restore oss://bucket/prefix/object -I <accesskey> -k <secretkey> -e
<endpoint>
```

递归解冻。

```
ossuti64 restore oss://bucket/prefix/ -r -I <accesskey> -k <secretkey> -e
<endpoint>
```

案例五:

ossutil 挂载 crontab 中执行报错。

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x40 pc=0x6b4981]

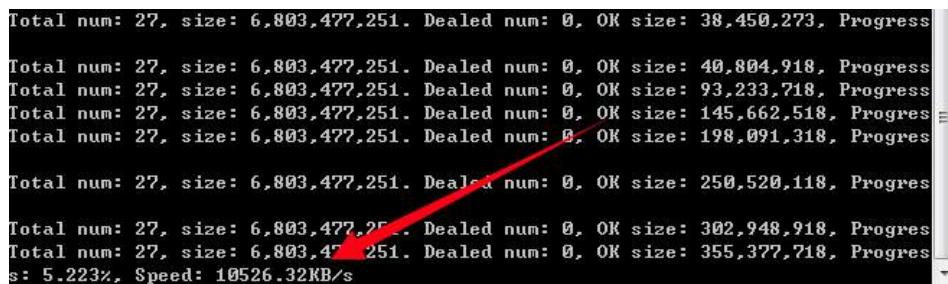
goroutine 1 [running]:
github.com/aliyun/ossutil/lib.DecideConfigFile(0x0, 0x0, 0x7a8017, 0x7)
/Users/fengyu/go/src/github.com/aliyun/ossutil/lib/config_helper.go:57 +0x51
```

排查:

请直接下载 ossutil 的工具最近版本进行处理。这个问题已经解决掉。

案例六:

Windows ossutil 上传 OSS 速率慢不稳定, 客户端是在河北公网, 目标服务端是北京, 存在跨省情况; 首先了解下用户在公网情况下, 能否切换到同 region 的 OSS 上进行访问, 同 region 的 OSS 内网是有阿里环网组成, 如果客户只能在公网使用, 进行下面的排查。



```
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 38,450,273, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 40,804,918, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 93,233,718, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 145,662,518, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 198,091,318, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 250,520,118, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 302,948,918, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 355,377,718, Progress
s: 5.223%, Speed: 10526.32KB/s
```

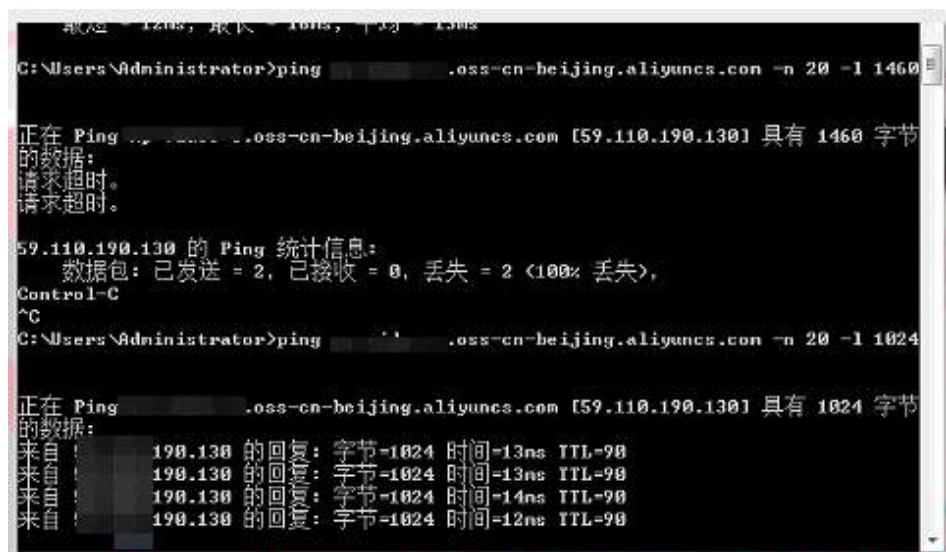
排查:

公网:

- 1) 首先看下用户端 ping OSS 的完整域名的网络 ping 值是否正常, tracert 是否有延迟抖动, 此步骤可以通过脚本来做, 下载脚本后运行, 直接输入完整的 OSS 域名即可; [脚本](#)
- 脚本中是 ping 的是大包 1460, 发现用户端直接网络超时, 此时怀疑是客户的网络有限制或者网络拥塞, 但是 tracert 通的, 再进行下一步排查。

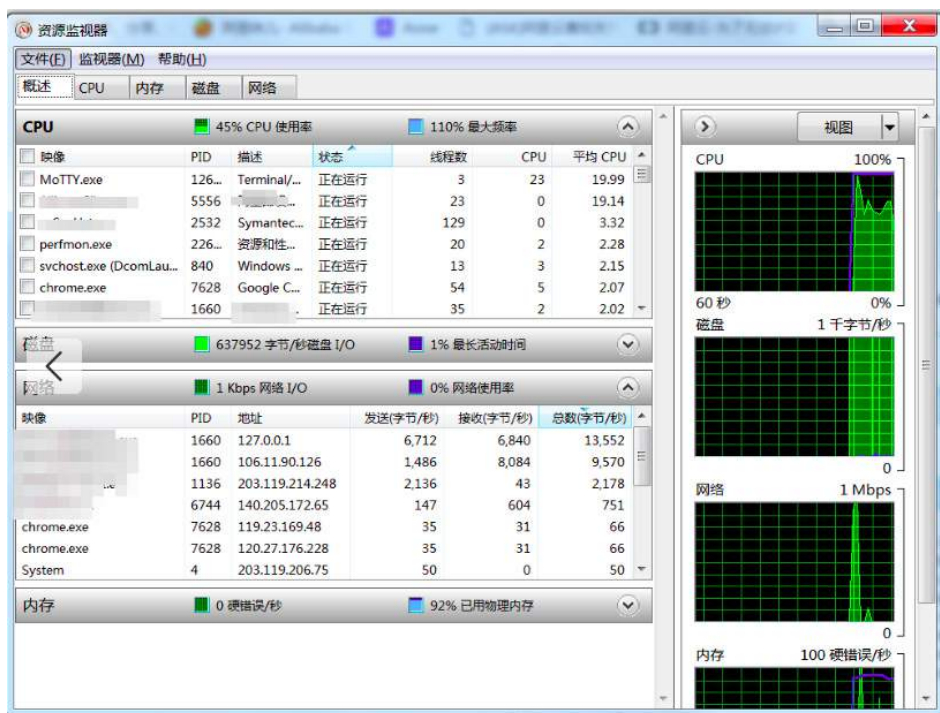


- 尝试降低 ping 包的 len 发现到 1412 ping 可以通过, 说明客户端在网络上做限制, 不允许 1460 大包的传输, 影响了客户端的上行网络吞吐量。



2) 尝试检查本机的网络负载。

- 通过本机资源监控看用户端当时的 CPU 负载和本机内存占用较高，实际登陆到客户端机器上发现系统卡顿，实际资源监控器看到 ossutil 工具线程并没有跑到和设置的 `--job=10 --parallel=10` ($10 \times 10 = 100$) 一样，只是运行了 24 个，说明工具的线程受到了系统 CPU 调度影响，无法将网络吞吐打上去，于是让用户关系了一些占用 CPU 内存较高的应用后，ossutil 线程数终于打到 69。
- 调整前



- 调整后



3) 灵活调整 ossutil 的线程数和分片的并发数。

--bigfile-threshold=52428800 --jobs=10 --parallel=50 --part-size=52428800。

- 遇到大文件数量多，以及单一的大文件时我们要灵活调整 ossutil 的相关参数能够提高我们的 ossutil 的效率。
- 比如这个案例中客户的文件大小平均是 100M 以上，而且客户的出口带宽是共享 200M，也就是客户自己的上线速度理论上也要 20M。
- 针对这种情况，我们可以把分片大小调整为 50M-10M，同时增加 parallel 分片的并发数量，尽可能的打满上行的吞吐。当客户端 CPU 核心比较少的时候不建议分的太小，比如分到 1M 时就会造成 CPU 切片过快，消耗 CPU 计算，影响系统性能。

如果遇到文件数量单一大文件，或者小文件时，可以不用加任何参数，直接上传即可；调整完成后用户的上行出口速率能打到 10M 大 B。

4) 对比其他家的工具。

- 比如七牛的 qfetch 以及 qshell 做了性能对比，上行的传输效率相差并不多，并没有用户反馈了七牛 20M (大 B) 阿里云 4M (大 B)，基本上 qshell 和 qfetch 的效果都是稳定在 10M 左右。

总结下问题以及需要客户下一步解决：

当使用 ossbrowser 上传速度低时可以切换到 ossutil 这个高兴的工具进行测试；需要用户解决下为什么网络出口限制的大包 (1460) 的传输，这个问题不解决很难将网络带宽吞吐提上去。尽量本机不要在负载高的情况下上传一些大文件，如果传输的话可以关闭一些应用卡顿的程序，或者降低下 ossutil 工具的并发线程数量，不然即使设置了 100，但实际受到系统的 CPU 调用影响，不一定能跑到 100。降低线程数，上传的效率也会受到影响。

OSS 工具之 ossimport

合理的迁移配置

目前还是推荐客户按照 VPC 的环境方式进行迁移，在服务体验感上有很大提升。在迁移过程中 OSS SLA 没有承诺迁移速度有明确指标，而且迁移数据和多因素有关（机器配置、数据量、网络、线程、限流等），所以根据实际情况进行问题诊断。

首先在迁移之前，要求使用者要先大致的学习一下我们的配置属性说明，和相关文件的作用，日志存储，异常处理等。https://help.aliyun.com/document_detail/56990.html?spm=a2c4g.11174283.6.1079.sbu1ch

配置迁移文件单机版

迁移前要明确客户端的迁移体量和文件数量。目的是合理的配置 task 和线程数量，以及 ossimport 的工作模型。

- 一般迁移体量小于 30TB 的完全可以采用单机模式进行迁移，单机版可以配置多线程的方式进行迁移，调节 workerTaskThreadNum 参数，需要注意的是如果是高配，物理机的话，数量可以调大，可以参考 平均文件 size * 线程数量，对比 memory 是否够用，同时也要参考 CPU 核心数量是否能抗住这种并发量。
- 当高并发，且文件的 size 较大时，如果配置数量不合理，会出现设备 OOM 的情况，可以通过 /var/log/message 看到，这是要么选择降低并发线程数量，要么是增大机器配置。
- 如果是本地 ECS 或者第三方存储迁移到 OSS，请注意源是否有限流，目前经常会出现源发现有大流量流出后直接给限制住，导致迁移速度无法增长。关键参数 workerMaxThroughput(KB/s)，如果源没有限制，可以释放调大这数。

- 其他 sys.properties 文件中的属性不要随意改动。

配置分布式迁移文件

- 分布式迁移模式的数据体量都是大于 30TB，甚至上 PB 的级别才会用到，迁移模型也比较特殊，是通过 master 带多个 worker 协同工作，类似 nginx 的模型，master 只是用来切割任务和分配任务，对数据量庞大的用户，worker 的数量也要配置好，才能高效迁移。
- 首先通过客户端文件数量，来计算任务数，该项目中客户端总文件数 424421917 个，源头限制 3Gbps，客户的机器数量有 12 台。计划分成 20000 个 task，每个 task 迁移 21221 个文件。每个 worker 机器开 200 线程，并发处理 200 个 task，并发也就是能处理 2400 个，升级 17600 个 task 在队列中。
- 源流限制是 3000Mbps，其实并发 worker 全部也就能跑到 375MB，所以即便是你的线程开的再多，机器配置在高其他的线程也只能阻塞住。

OSS 单机版迁移速度慢优化建议

所有优化都是基于 ECS 的基础上，如果是自己的 PC 优化空间不大。

- 如果是 OSS 迁移到 OSS 可以买 ECS 做迁移，将配置做大点，比 32G 16 核，64G 32 核，16G 8 核 等。将 ECS 的 srcdomain destdomain 都设置为内网 internal 的域名。
- 如果不是 OSS 迁移到 OSS，将配置做大点，比 32G 16 核，64G 32 核，16G 8 核 等。将 ECS 的 destdomain 设置为内网 internal 的域名。
- 将本机的迁移 heap 调大到 ECS 内存的 1/2，如果本机内存使用较高的请调到 1/4。
- 线程数可以开到 workerTaskThreadNum=120。
- 设置 workerMaxThroughput=0。

OSS 分布式版迁移速度慢优化建议

- master 和 worker 机器要使用 ECS 机器进行迁移，master 机器要配置大点，如果想迁移快建议 master 64G 32 核，worker 机器 32G 16 核 或者以上，数量越多越好，客户根据自己的业务情况买，具体情况再分析。将 ECS 的 srcdomain destdomain 都设置为内网 internal 的域名。
- 如果不是 OSS 迁移到 OSS，将配置做大点，比 32G 16 核，64G 32 核，16G 8 核等。将 ECS 的 destdomain 设置为内网 internal 的域名。
- 将本机的迁移 heap 调大到 ECS 内存的 1/2，如果本机内存使用较高的请调到 1/4。
- 配置文件线程数可以开到 workerTaskThreadNum=150。
- 设置 workerMaxThroughput=0。

问题：

ossimport NullPointerException。

排查：

```

2018-10-22 19:56:08 CheckFailed 2014/07/17/10/1405563207248_765640.jpg Detail:get target meta failed. source last modify: Fri Dec 05 13:14:30 CST 2014
2018-10-22 19:56:11 CheckFailed 2014/04/29/13/1399749354935_157126.jpg Detail:get target meta failed. source last modify: Fri Dec 05 12:12:40 CST 2014
2018-10-22 19:56:13 CheckFailed 2014/03/27/21/1395928028232_87328.jpg Detail:get target meta failed. source last modify: Fri Dec 05 12:31:40 CST 2014
2018-10-22 19:56:15 CheckFailed 2014/03/27/21/1395926432302_87295.jpg Detail:get target meta failed. source last modify: Fri Dec 05 12:31:40 CST 2014
2018-10-22 19:56:15 CheckFailed 2014/03/27/13/1401169378900_323524.jpg Detail:get target meta failed. source last modify: Fri Dec 05 13:01:35 CST 2014
Exception in thread "pool-2-thread-56" java.lang.NullPointerException
    at com.aliyun.ossimport.worker.task.ResetTaskUtils.store(ResetTaskUtils.java:30)
    at com.aliyun.ossimport.worker.task.LocalImpl.LocalImportTask.run(LocalImportTask.java:191)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:724)
Exception in thread "pool-2-thread-58" java.lang.NullPointerException
    at com.aliyun.ossimport.worker.task.ResetTaskUtils.store(ResetTaskUtils.java:30)
    at com.aliyun.ossimport.worker.task.LocalImpl.LocalImportTask.run(LocalImportTask.java:191)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:724)
Exception in thread "pool-2-thread-59" java.lang.NullPointerException
    at com.aliyun.ossimport.worker.task.ResetTaskUtils.store(ResetTaskUtils.java:30)
    at com.aliyun.ossimport.worker.task.LocalImpl.LocalImportTask.run(LocalImportTask.java:191)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:724)
Exception in thread "pool-2-thread-57" java.lang.NullPointerException
    at com.aliyun.ossimport.worker.task.ResetTaskUtils.store(ResetTaskUtils.java:30)
    at com.aliyun.ossimport.worker.task.LocalImpl.LocalImportTask.run(LocalImportTask.java:191)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:724)
Exception in thread "pool-2-thread-59" java.lang.NullPointerException
    at com.aliyun.ossimport.worker.task.ResetTaskUtils.store(ResetTaskUtils.java:30)
    at com.aliyun.ossimport.worker.task.LocalImpl.LocalImportTask.run(LocalImportTask.java:191)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:724)
Exception in thread "pool-2-thread-60" java.lang.NullPointerException
    at com.aliyun.ossimport.worker.task.ResetTaskUtils.store(ResetTaskUtils.java:30)
    at com.aliyun.ossimport.worker.task.LocalImpl.LocalImportTask.run(LocalImportTask.java:191)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1145)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:615)
    at java.lang.Thread.run(Thread.java:724)
2018-10-22 19:59:22 CheckFailed 2014/08/28/13/1409205318161_109069.jpg Detail:get target meta failed. source last modify: Fri Dec 05 11:25:19 CST 2014

```

- 空指针的目前存在于低版本的 ossimport 工具中，官网的新版本已经修复，下载新版本后重启任务进程即可解决。
- get target meta failed 是正常报错，ossimport 会自动重试，重试如果还是失败，最后会变成 failed success 任务，可以用 sh console.sh stat 看到，错误原因一般因为上传到 OSS 文件失败，无法拉去 meta 信息进行校验导致，可以不用关心。

问题：

ossimport 上传 oss 超时。

```
[ERROR] [2018-10-18 15:52:25 771] net.bioslink.transfer.util.OSSClientUtil - 上传阿里云失败:
com.aliyun.oss.ClientException: SocketException
    at com.aliyun.oss.common.utils.ExceptionFactory.createSocketException(ExceptionFactory.java:71)
    at com.aliyun.oss.common.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:128)
    at com.aliyun.oss.common.DefaultServiceClient.sendRequestImpl(DefaultServiceClient.java:123)
    at com.aliyun.oss.common.DefaultServiceClient.sendRequest(DefaultServiceClient.java:68)
    at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:84)
    at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:149)
    at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:113)
    at com.aliyun.oss.internal.OSSObjectOperation.writeObjectInternal(OSSObjectOperation.java:907)
    at com.aliyun.oss.internal.OSSObjectOperation.putObject(OSSObjectOperation.java:146)
    at com.aliyun.oss.OSSClient.putObject(OSSClient.java:475)
    at com.aliyun.oss.OSSClient.putObject(OSSClient.java:457)
    at net.bioslink.transfer.util.OSSClientUtil.writeFile(OSSClientUtil.java:120)
    at net.bioslink.transfer.main.Command.uploadFile(Command.java:158)
    at net.bioslink.transfer.main.Command.execute(Command.java:59)
    at net.bioslink.transfer.main.Handler.run(Handler.java:62)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1110)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:603)
    at java.lang.Thread.run(Thread.java:728)
Caused by: org.apache.http.conn.HttpHostConnectException: Connect to oss-cn-shenzhen.aliyuncs.com:80 [oss-cn-shenzhen.aliyuncs.com/157.255.170.199] failed: 连接超时
    at org.apache.http.impl.conn.DefaultHttpClientConnectionOperator.connect(DefaultHttpClientConnectionOperator.java:151)
    at org.apache.http.impl.conn.PoolingHttpClientConnectionManager.connect(PoolingHttpClientConnectionManager.java:353)
    at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:300)
    at org.apache.http.impl.execchain.MainClientExec.execute(MainClientExec.java:236)
    at org.apache.http.impl.execchain.ProtocolExec.execute(ProtocolExec.java:184)
    at org.apache.http.impl.execchain.RedirectExec.execute(RedirectExec.java:110)
    at org.apache.http.impl.client.InternalHttpClient.doExecute(InternalHttpClient.java:184)
    at org.apache.http.impl.client.CloseableHttpClient.execute(CloseableHttpClient.java:82)
    at com.aliyun.oss.common.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:123)
    ... 15 more
```

排查：

通过上传报错即可知道 socket timeout，基本上和网络脱不了关系，延迟过高、丢包、网络不通都会出现这种问题。

最有效的办法是客户端抓包走一波，然后通过抓包分析症结。

问题：

原文件带有特殊符号。

```
[2018-10-17 10:01:50] [pool-1-thread-6] [ERROR] - Oss get meta failed, bucket:prd-export-file key: data_export/saleExportOrderTask/2017/009/taskId:334/订单完成数据-SM广告/订单完成数据-SM广告.21p_exception
[ErrorCode]: NoSuchKey
[RequestId]: 56C693AA5A5C72ABCE4489
[HostId]: null
    at com.aliyun.oss.common.utils.ExceptionFactory.createOssException(ExceptionFactory.java:184)
    at com.aliyun.oss.internal.OSSResponseHandler.handleOSSResponseHandler(OSSResponseHandler.java:56)
```

排查:

出现这种错误, 要检查下原文件是否存在特殊字符。

```
, " ' \ / & <> \ ? :
```

如果是以上带有特殊字符的原文件, 需要经过本地先特殊字符处理掉, 有些特殊字符可能在 Linux 的文件系统中可能无法正常显示, 可能导致 OSS 获取原文件失败, 最好自己将这种特殊的字符处理掉。

问题:

设置增量迁移, 任务一直存在。

排查:

isIncremental=true incrementalModelInterval=86400。

设置了这增量迁移参数后, 如果第一次全量迁移完后, 进程会一直存在, 根据设置的间隔时间发起增量扫描, 属于正常现象。

问题:

LocalScanner.run Exception:java.lang.StringIndexOutOfBoundsException: String index out of range.

```
[2018-09-26 10:33:44] [INFO] try lock ./master/jobs/local_test/.lock succeed
[2018-09-26 10:33:44] [INFO] start job:local_test
[2018-09-26 10:33:44] [INFO] init job:local_test, bucket:, prefix:/[REDACTED]/TRMCentreTemp/A01/2014/, thread:10000
[2018-09-26 10:33:44] [ERROR] LocalScanner.run Exception:java.lang.StringIndexOutOfBoundsException: String index out of range: -1
at java.lang.String.substring(String.java:1931)
at com.aliyun.ossimport.master.task.ImportTaskBuilder.add(ImportTaskBuilder.java:45)
at com.aliyun.ossimport.master.scanner.LocalScanTask.add(LocalScanner.java:72)
at com.aliyun.ossimport.master.scanner.LocalScanTask$SingleThreadVisitor.visitFile(LocalScanner.java:86)
at com.aliyun.ossimport.master.scanner.LocalScanTask$SingleThreadVisitor.visitFile(LocalScanner.java:1)
at java.nio.file.Files.walkFileTree(Files.java:2670)
at java.nio.file.Files.walkFileTree(Files.java:2742)
at com.aliyun.ossimport.master.scanner.LocalScanTask.run(LocalScanner.java:132)
at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
at java.util.concurrent.FutureTask.run(FutureTask.java:266)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
at java.lang.Thread.run(Thread.java:745)
```


排查:

迁移的源文件设置了软链接单机版的暂时不支持，分布式版本的可以支持。

问题:

如何降低 ossimport 的迁移带宽，占用内存？

排查:

1) 如果降低迁移带宽或者内存，将导致整体的 ossimport 迁移速度下降，耽误迁移时间，请做好预期。

- javaHeapMaxSize，默认是 1024 最低的堆栈配置，想要调整迁移占用内存可以调整堆栈大小，不能少于 1024。
- workerMaxThroughput(KB/s)，可以控制它迁移速度，单位是 KB。
- workerTaskThreadNum，迁移线程数，默认 60，可以调整。

通过以上几个参数可以降低迁移的占用带宽和内存消耗。所有配置文件都在 sys.properties 里面；

问题:

迁移过程中，出现 NoSuchKey，但是本地文件存在，或者第三方存储源文件存在，为什么报 404？

```
[2018-12-05 10:33:13] [ERROR] Oss get meta failed, bucket:single key:mpg/
All Japan 3200 Database Complete/Japan Data
base/J100040.mpg Exception:com.aliyun.oss.OSSException: Not Found
[ErrorCode]: NoSuchKey
[RequestId]: 5C0738E8D06C7BFCBDFFBAC1
[HostId]: null
    at com.aliyun.oss.common.utils.ExceptionFactory.
createOSSException(ExceptionFactory.java:104)
    at com.aliyun.oss.internal.OSSErrorHandler.
handle(OSSErrorHandler.java:56)
    at com.aliyun.oss.common.comm.ServiceClient.
```

```

handleResponse(ServiceClient.java:253)
    at com.aliyun.oss.common.comm.ServiceClient.
sendRequestImpl(ServiceClient.java:135)
    at com.aliyun.oss.common.comm.ServiceClient.
sendRequest(ServiceClient.java:69)
    at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:94)
    at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.
java:156)
    at com.aliyun.oss.internal.OSSObjectOperation.
getObjectMetadata(OSSObjectOperation.java:398)
    at com.aliyun.oss.OSSClient.getObjectMetadata(OSSClient.java:582)
    at com.aliyun.oss.OSSClient.getObjectMetadata(OSSClient.java:576)
    at com.aliyun.ossimport.worker.uploader.OssUploader.
getMeta(OssUploader.java:374)
    at com.aliyun.ossimport.worker.task.localimpl.LocalAuditTask.
checkWithRetry(LocalAuditTask.java:92)
    at com.aliyun.ossimport.worker.task.localimpl.LocalImportTask.
audit(LocalImportTask.java:101)
    at com.aliyun.ossimport.worker.task.localimpl.LocalImportTask.
run(LocalImportTask.java:150)
    at java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
    at java.lang.Thread.run(Unknown Source)

```

排查:

ossimport 的工作过程是要先将源文件 list 出来, 然后分成不同的 task 去执行, 当所以任务执行完成后, 再进行统计的文件校验比对, 其方式是通过本地发起一条 http 的 header 请求到 OSS, 如果校验通过表示上传成功, 如果校验失败上传就失败。

通过报错可以看到是 ossimport 执行了一个 http 请求到 OSS 端, 获取了 404 的结果, 这种情况只有两种可能。

- 迁移的数据源是 OSS, 并且 OSS 上没有这个文件。
- 源文件存在, 但是迁移过程中出现了失败任务, 没有迁移成功, 在最后 ossimport 校验时没有请求到目标 bucket 这条 object 所以出现 404。

OSS 工具排查案例

浅谈

本章节结合一些实际遇到的案例讲一下各种工具使用。

- 在遇到问题时先确保自己的工具一定是官方的最新迭代版本。
- 使用工具遇到问题时如果遇到 4xx 3xx 2xx 500 等问题，oss 都会返回一个 x-oss-requestid 的 http 头，value 是一串字符，类似 5BEE7AD4C84D-1C447120083C 这个对于排查分析问题非常重要。
- 每个工具基本都带有 log 功能，请使用者务必开启，遇到问题时可以追根溯源。

一、使用场景

- ossbrowser，图形版的操作工具，有控制台的基本功能，可以理解是 ossutil 工具的图形版，适用于一些非技术人员来操作 oss。
- ossfs，将 oss mount 到本地放使用，利用的是 fuse 用户态的文件系统，然后通过开源的 s3 协议和 oss 做了一个网络映射，不推荐先敏感业务或者高并发使用，个人当作私人仓库标适合。
- ossftp，和开源的 ftp 一样，将 oss 挂在本地后，当作远程的仓库上传下载用。
- ossutil，目前 oss 对外工具性能最好，支持功能较多的高并发读写工具。操作易用，而且可以嵌入到脚本使用。

二、使用遇到问题

ossbrowser

案例：驻云工具无法加载 bucket 中 object

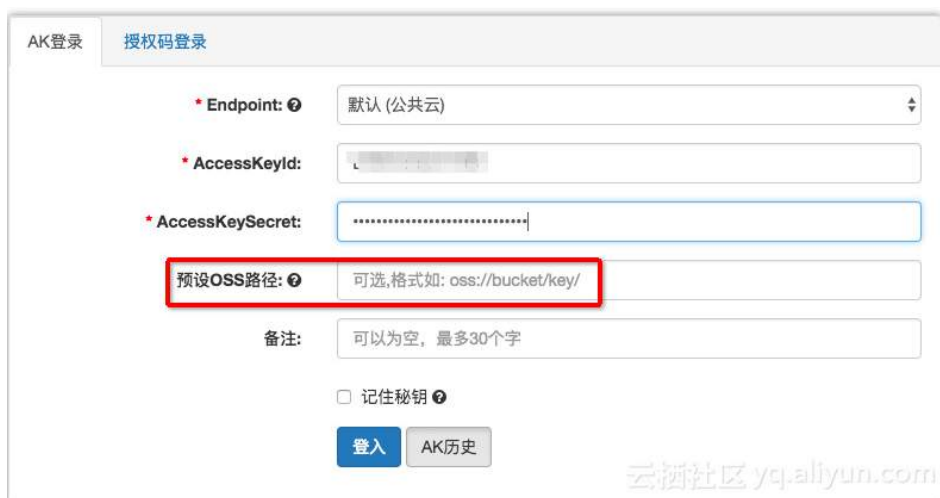


排查：

- 如图是一个第一个非 oss 官方的第三方工具，用户可以尝试在客户端做下基本的 ping 测试先看下网络是否通。
- 检查登陆的 AccesskeyID 的权限是否可以将 bucket 下的内容 list 出来。
- 用 ossbrowser 测试下，看同样的 AccesskeyID 登陆后是否也会报错，如果 ossbrowser 可以正常显示，证明策略没有问题，是第三方放工具的问题。可以联系驻云公司看下是否配置上有特殊的地方。

案例：ossbrowser 使用注意

- 如果登陆的 AK 没有 AliYunOSSFullAccess 权限，截图的位置必须要预设 endpoint。



- 遇到并发文件较多时，建议将任务数调大。



ossfs

ossfs 的报错都会有明显的 message，需要收集到这些 message，根据 message 判断是否直接看出问题，比如 socket 建联失败，或者响应的状态码 4XX 5XX 等，一般 403 是权限问题被 deny，400 是用户的操作方法有误，5xx 一般和网络抖动以及客户端业务有关系，使用前先将 debug-log 功能开启。

如果使用 ossfs 发现性能很差，建议使用 ossutil，因为 ossfs 是将远端的 OSS 挂载到本地磁盘，如果对业务性能敏感性很高的业务，不建议使用 ossfs，而且该工具也不是原子性，存在本地操作成功，但 oss 远端操作失败的风险。

如果发现 ossfs 在 ls 目录文件时很慢，可以增加调优参数 通过 `-omax_stat_`

`cache_size=xxx` 参数增大 stat cache 的 size，这样第一次 ls 会较慢，但是后续的 ls 就快了。

案例：ossfs 偶尔出现断开的情况

```
Nov 8 02:56:28 i2w29emgcy43hdoultcoez kernel: [30273] 0 30273 33017 027 81 0 0 AliyunDun
Nov 8 02:56:28 i2w29emgcy43hdoultcoez kernel: [18398] 0 18398 1301998 317450 955 0 0 java
Nov 8 02:56:28 i2w29emgcy43hdoultcoez kernel: Out of memory: Kill process 3017 (ossfs) score 581 or sacrifice child
Nov 8 02:56:28 i2w29emgcy43hdoultcoez kernel: Killed process 3017 (ossfs) total-vm:8046372kB, anon-rss:478780kB, file-rss:268kB, shmem-rss:9kB
Nov 8 02:56:45 i2w29emgcy43hdoultcoez s3fs[2968]: s3fs.cpp:118: list_bucket(2459): xdrReadMemory returns with error.
Nov 8 02:56:45 i2w29emgcy43hdoultcoez s3fs[2968]: s3fs.cpp:s3fs_readdir(2382): list_bucket returns error(1).
Nov 8 02:56:45 i2w29emgcy43hdoultcoez s3fs[2968]: s3fs.cpp:118: list_bucket(2459): xdrReadMemory returns with error.
Nov 8 02:56:45 i2w29emgcy43hdoultcoez s3fs[2968]: s3fs.cpp:s3fs_readdir(2382): list_bucket returns error(1).
Nov 8 02:56:01 i2w29emgcy43hdoultcoez systemd: Started Session 46140 of user root.
```

排查：

- 出现问题后也不知原因，于是开启了 ossfs 的 debug 日志进行分析加上 `-d -o f2` 参数，ossfs 会把日志写入到系统 `/var/log/message`。
- 日志发现是 ossfs 在 listbucket listobject 申请内存过多，导致触发了系统的 oom 将进行 killer，找到了元凶。
- listobject 是要发起 http 请求到 oss 获取一个 meta 信息，如果客户的文件很多，ls 会消耗系统的大量内存来获取文件的 meta，这是正常情况。

解决方案：

- 通过 `-omax_stat_cache_size=xxx` 参数增大 stat cache 的 size，这样第一次 ls 会较慢，但是后续的 ls 就快了，因为文件的元数据都在本地 cache 中。默认这个值是 1000，大约消耗 4MB 内存，请根据您的机器内存大小调整为合适的值。
- 使用 `ls -f` 命令，这个命令会消除与 OSS 的 n 次 http 请求。
- ossfs 在读写时会占用磁盘写大量的 temp cache，和 nginx 差不多，可能会导致磁盘空间不满，最好能常清理一下。
- 使用 osstuil 替代 ossfs，非线上敏感业务可以用 ossfs，要求可靠性、稳定性的还是用 osstuil。

案例：访问 403 deny

```
root@izbp199:~# cd /usr/local/cloudfs# ossfs /mnt/oss -ourl=http://oss-cn-shenzhen-internal.aliyuncs.com
ossfs: invalid credentials
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>AccessDenied</Code>
  <Message>The bucket you are attempting to access must be addressed using the specified endpoint. Please send all future requests to this endpoint.</Message>
  <RequestId>78C86364671B80AAF</RequestId>
  <HostId>oss-cn-shenzhen-internal.aliyuncs.com</HostId>
  <Bucket>/</Bucket>
  <Endpoint>oss-cn-hangzhou-internal.aliyuncs.com</Endpoint>
</Error>
```

云栖社区 yq.aliyun.com

排查：

类这种有明显报错的很好判断，明显是 endpoint 指定错误。

- bucket 和 endpoint 不匹配。
- bucket UID 和实际的 Accesskey 对应的 UID 不一致。

案例：

cp 出现 input/output error。

```
[root@rheladmin ~]# du -sh *
74G caohh.zip
[root@rheladmin lixiao]# cp caohh.zip /data/ossfs/
cp: writing '/data/ossfs/caohh.zip': Input/output error
mncp: closing '/data/ossfs/caohh.zip': Input/output error
[root@rheladmin ~]#
```

云栖社区 yq.aliyun.com

排查：

- 看下当时的磁盘读写是否出现高负载的情况。io error 都是捕获到这系统磁盘的错误。
- 可以尝试增加分片参数，控制文件读写，ossfs -h 看下 multipart 选项。

使用 rsync 同步出现 input/output error

```
cp: writing '/data/tmp/I_201704/12/request.1491926430217.avro': Input/output error
cp: closing '/data/tmp/I_201704/12/request.1491926430217.avro': Input/output error
```

云栖社区 yq.aliyun.com

- ossfs 与 rsync 同步使用会出现问题，而且用户对一个 141G 的文件进行 cp 操作，对 ossfs 本身的磁盘读写比就是一个挑战。

好定位。

```
> PUT /.fuse_hidden0000000300000002 HTTP/1.1
Host: pgone.oss-cn-hangzhou-internal.aliyuncs.com
Accept: */*
Authorization: OSS LYnddfOj:l9hXsWS4KubHT+b0=
Content-Type: application/octet-stream
Date: Sat, 22 Dec 2018 11:52:22 GMT
User-Agent: aliyun-sdk-http/1.0()/ossfs1.80.5
x-oss-acl: private
x-oss-copy-source: /pgback/tmpn18XKo
x-oss-meta-gid: 0
x-oss-meta-mode: 33152
x-oss-meta-mtime: 1545479542
x-oss-meta-uid: 0
x-oss-metadata-directive: REPLACE
Content-Length: 0
Expect: 100-continue

< HTTP/1.1 100 Continue
< HTTP/1.1 403 Forbidden
< Server: AliyunOSS
< Date: Sat, 22 Dec 2018 11:52:22 GMT
< Content-Type: application/xml
< Content-Length: 331
< Connection: keep-alive
< x-oss-request-id: 5C1E2576D9D458BE30B9D539
< x-oss-server-time: 2
* HTTP error before end of send, stop sending
```

- 通过拿到的 http request、response 信息看到用户有一个 rename 的操作，并且返回的 requestId 信息。
- 结合 ossfs 的源码得知，ossfs 在挂载 oss 到本地成功后，会发起一个类似探针的操作进行 rename 和 chown 的操作，此过程是为了类似验证 OSS 权限和连通性的操作。
- 如果出现类似的 copy 操作被 403 后，就需要您检查下您的 bucket 是否为归档 bucket，如果不是归档继续下面排查。
- 确认您的 Accesskey 对应的权限是否有对应的操作权限。
- 把 ossfs 卸载然后进入挂载的目录看下直接 touch 会不会报错。

如果以上仍不能解决您的问题，请将 requestId 提供到阿里云进行排查。

ossutil

ossutil 目前没有做限速功能, 计划支持中, 目前真多多文件并发上传是通过

`--jobs` (控制多个文件并发) `--parallel` (控制单一文件并发) 参数控制的。

案例:

ossutil 出现 skip 情况。

```
[root@iZ2Sv4olcc4Z opt]# echo "testlil" > dskydb/test.txt
[root@iZ2Sv4olcc4Z opt]# ./ossutil64 cp -rt -c ~/.ossuti.Lcofig dskydb/test.
txt oss://gres/test.txt
Succeed: Total num: 1, $ze: 30. OK nun: 1(upload 1 files).
0.372650(s) elapsed I
[root@iZ2Sv4olcc4Z opt] echo "ttest222r" >> dskydb/test.txt
[root@iZ2Sv4olcc4Z opt]. /ossutil64 cp -u -c ~/.ossutilconfig cbkydb/test.
txt oss://gres/test.txt
Succeed: Total num: 1, size: 38. OK nun: 1(skip 1 files), Skip sin 38.
0.252878(s) elapsed
```

排查:

使用 `-u` 强制更新时, 重新对所有的上传文件和原的进行一次比对, 发现美有更改的情况就会跳过, 有发生更改的就会触发上传进行覆, 正常情况。

遇到这种情况可以看下本地的 log 哪些文件被跳过了, 将 oss 存储的文件和本地文件做个 MD5 比对确保文件是一致。

命令: `./ossutil64 cp -u -r -f aa.test oss://alihua -i -k -e`。

案例: ossutil 文件递归解冻时遇到 403



排查:

如图用户在操作解冻文件的过程中出现 403, 可能与两个原因有关系。

- 用户使用的子账号操作文件, 权限不够。

- 用户的文件是违禁内容被封禁掉了。

PS: 遇到 403 时, 递归解冻会中断不会继续。这种现象是正常的, 工具在设计之初就是考虑到如果文件遇到 403 的话, 代表没有权限操作该文件, 那么通过该账号下的其他文件也操作不了, 所以就会中断退出。

案例: ossutil 文件递归解冻时遇到 400

```
[Credentials]
language=zh
endpoint=oss-cn-shenzhen.aliyuncs.com
accessKey=AKIAJ2...
accessKeySecret=...
[root@119 luqingying]# ./ossutil64 restore oss://... -r -f -c ./oss_mowheng.conf
Error occurs, message: oss: service returned error: StatusCode=400, ErrorCode=OperationNotSupported, ErrorMessage=Id=5A44C3630E49174590A56386, Bucket=oss-cn-shenzhen-aliyuncs.com, Object=jiahaoyy.jpg. See more information in file: o
Scanned 153800 objects. Restored 19911 objects, Error 133 objects.
```

排查:

- 检查用户的命令和官方提供的命令是否完全一致, 避免误操作。
- 使用 stat 选项看一下文件的状态是否是已经解冻了, 如果以及解冻再次操作, 就会出现 400。

案例:

ossutil 操作 object 出现 “The operation is not valid for the object’s state”。

排查:

出现这个问题是因为用户操作的文件是一个归档的文件, 不能直接操作, 需要先进行解冻 restore 的操作后才能使用。

```
ossuti64 restore oss://bucket/prefix/object -I <accesskey> -k <secretkey> -e
<endpoint>
```

递归解冻

```
ossuti64 restore oss://bucket/prefix/ -r -I <accesskey> -k <secretkey> -e
<endpoint>
```

案例:

ossutil 挂载 crontab 中执行报错。

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x40 pc=0x6b4981]

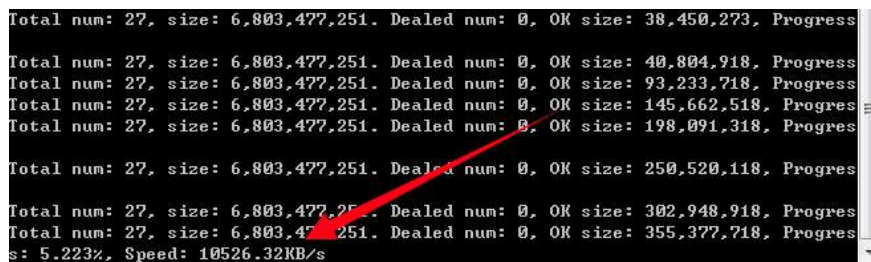
goroutine 1 [running]:
github.com/aliyun/ossutil/lib.DecideConfigFile(0x0, 0x0, 0x7a8017, 0x7)
/Users/fengyu/go/src/github.com/aliyun/ossutil/lib/config_helper.go:57 +0x51
```

排查:

请直接下载 ossutil 的工具最近版本进行处理。这个问题已经解决掉。<http://gosspublic.alicdn.com/ossutil/1.4.2/ossutil64?spm=a2c4g.11186623.2.10.2ca74af8IGi3C8>

案例:

Windows ossutil 上传 OSS 速率慢不稳定客户端是在河北公网, 目标服务端是北京, 存在跨省情况。



```
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 38,450,273, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 40,804,918, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 93,233,718, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 145,662,518, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 198,091,318, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 250,520,118, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 302,948,918, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 355,377,718, Progress
s: 5.223%, Speed: 10526.32KB/s
```

首先了解下用户在公网情况下, 能否切换到同 region 的 OSS 上进行访问, 同 region 的 OSS 内网是有阿里环网组成, 如果客户只能在公网使用, 进行下面的排查。

排查:

公网:

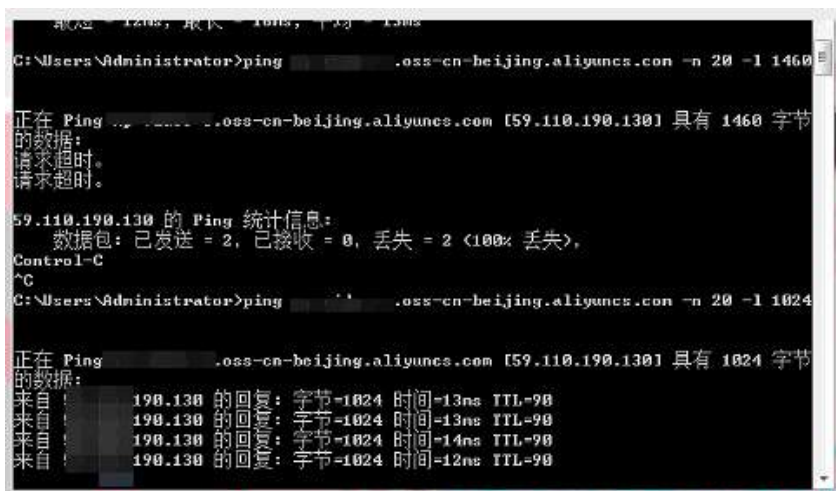
- 1) 首先看下用户端 ping OSS 的完整域名的网络 ping 值是否正常, tracert 是否有延迟抖动, 此步骤可以通过脚本来做, 下载脚本后运行, 直接输入完整的 OSS 域名即可。

脚本:

- 脚本中是 ping 的是大包 1460，发现用户端直接网络超时，此时怀疑是客户的网络有限制或者网络拥塞，但是 tracert 通的，再进行下一步排查。



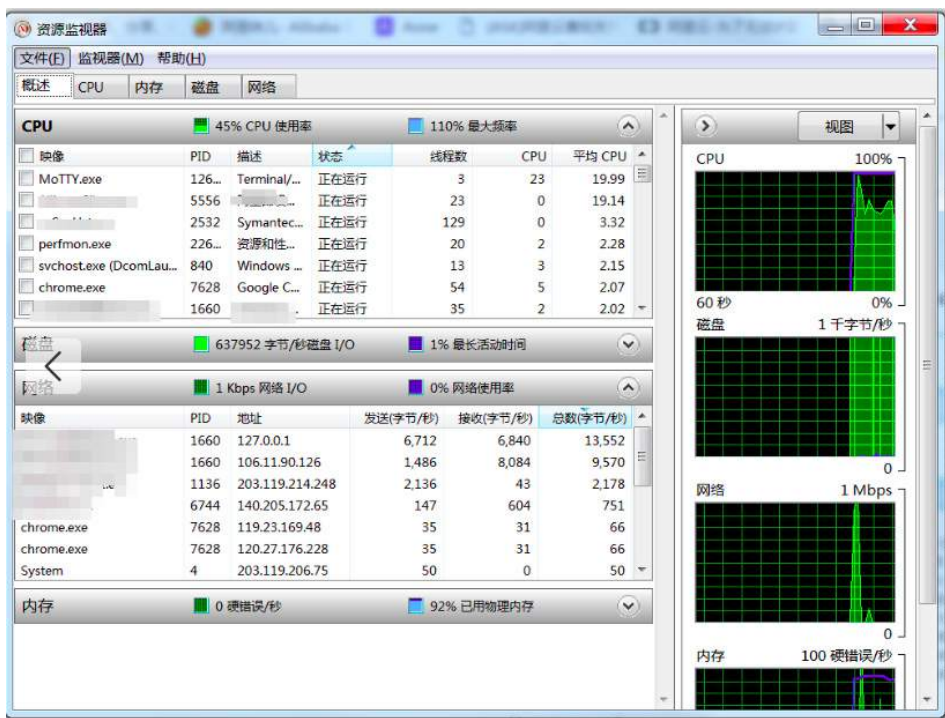
- 尝试降低 ping 包的 len 发现到 1412 ping 可以通过，说明客户端在网络上做限制，不允许 1460 大包的传输，影响了客户端的上行网络吞吐量。



2) 尝试检查本机的网络负载。

- 通过本机资源监控看用户端当时的 CPU 负载和本机内存占用较高，实际登陆到客户端机器上发现系统卡顿，实际资源监控器看到 ossutil 工具线程并没有跑到和设置的 `--job=10 --parallel=10` ($10 \times 10 = 100$) 一样，只是运行了 24 个，说明工具的线程受到了系统 CPU 调度影响，无法将网络吞吐打上去，于是让用户关系了一些占用 CPU 内存较高的应用后，ossutil 线程数终于打到 69。

调整前



调整后



3) 灵活调整 ossutil 的线程数和分片的并发数。

- `--bigfile-threshold=52428800 --jobs=10 --parallel=50 --part-size=52428800`。
- 遇到大文件数量多，以及单一的大文件时我们要灵活调整 ossutil 的相关参数能够提高我们的 ossutil 的效率。
- 比如这个案例中客户的文件大小平均是 100M 以上，而且客户的出口带宽是共享 200M，也就是客户自己的上线速度理论上也要 20M。
- 针对这种情况，我们可以把分片大小调整为 50M-10M，同时增加 parallel 分片的并发数量，尽可能的打满上行的吞吐。当客户端 CPU 核心比较少的时候不建议分的太小，比如分到 1M 时就会造成 CPU 切片过快，消耗 CPU 计算，影响系统性能。
- 如果遇到文件数量单一大文件，或者小文件时，可以不用加任何参数，直接上传即可；调整完成后用户的上行出口速率能打到 10M 大 B。

4) 对比其他家的工具。

- 比如七牛的 qfetch 以及 qshell 做了性能对比，上行的传输效率相差并不多，并没有用户反馈了七牛 20M (大 B) 阿里云 4M (大 B)，基本上 qshell 和 qfetch 的效果都是稳定在 10M 左右。

总结下问题以及需要客户下一步解决

当使用 ossbrowser 上传速度低时可以切换到 ossutil 这个高兴的工具进行测试；需要用户解决下为什么网络出口限制的大包 (1460) 的传输，这个问题不解决很难将网络带宽吞吐提上去。尽量本机不要在负载高的情况下上传一些大文件，如果传输的话可以关闭一些应用卡顿的程序，或者降低下 ossutil 工具的并发线程数量，不然即使设置了 100，但实际受到系统的 CPU 调用影响，不一定能跑到 100。降低线程数，上传的效率也会受到影响。

osscommand

案例：

OSS 存储文件数量和本地的文件数量不符，用 ossutil 就是正确的。

```
root@hangzh:~# if python osscmd config --id=ossadmin --key=ossadmin --host=oss-cn-beijing.aliyuncs.com
Your configuration is saved into /root/.osscredentials
[root@hangzh:~]# python osscmd downloadallobject oss://ossadmin/mnt
get ossadmin/mnt/oss.png to /mnt/ossadmin/oss.png OK
Total being downloaded objects (num: 1), they are downloaded into /mnt
OK num:1, SKIP num:0, FAIL num:0
0.558(s) elapsed
[root@hangzh:~]#
```

云栖社区 yq.aliyun.com

排查：

osscmd 下载，是直接将 oss 云存储的 object 下载下来，不会包含 prefix，有多少 object 就累计总和，不会出现误差。

为什么 oss 存储的比 osscmd 统计的多？

- 因为 oss 上的 prefix 也算是一个 object，oss 上一切都是文件，没有目录的概念，prefix 被认为是 object 计算后，总的数量就会比 osscmd 看到的多。
- 要想判断是否有失败文件，只要关心 fail num 的数量即可，为 0 代标没有失

败的，skip 如果不为 0 说明用户之前有下载过文件，又重复下载一遍，但是文件内容没更新所以被计到 skip 中。

为什么 ossutil 正常？

- 因为 ossutil 下载是将整个目录结构下载下来，统计的方式是和 OSS 一致的，将 prefix 也计算在 object 中，所以和 OSS 云端看到的一致。

视频媒体类

OSS 图片处理

背景

针对 OSS case 的基础排查，以及如果查看图片基础原始参数进行故障定位。

必要信息

- 处理图片的需求请描述清楚
- 图片处理的原图链接
- 图片处理后的链接

查看图片原基础参数

示例: <http://zhangyb.oss-cn-shanghai.aliyuncs.com/1.jpg?x-oss-process=image/info>

```
{
  "AEInfo": {"value": "36 116 32 64 0 164 0 0 17 100 99 144 0 56 0 63"},
  "AELock": {"value": "0"},
  "AEMeteringSegments": {"value": "53 50 52 46 70 50 56 54 84 59 64 65 65 66 65 71"},
  "AFPoint": {"value": "65535"},
  "AutoBracketing": {"value": "0 0"},
  "BatteryInfo": {"value": "2 65 172 168 5 1"},
  "BlackPoint": {"value": "0 0 0 0"},
  "ByteOrder": {"value": "MM"},
  "CPUFirmwareVersion": {"value": "254 244 255 243"},
  "ColorInfo": {"value": "32 131 31 100 31 125 32 156 33 72 32 246 31 51 31 10 0 0"},
  "ColorSpace": {"value": "0"},
  "ColorTemperature": {"value": "0"},
  "ComponentsConfiguration": {"value": "1 2 3 0"},
  "Compression": {"value": "6"},
  "Contrast": {"value": "1"},
  "CustomRendered": {"value": "0"},
  "DSPFirmwareVersion": {"value": "254 244 255 243"},
  "Date": {"value": "7 215 4 8"},
  "DateTime": {"value": "2007:04:08 07:10:22"},
  "DateTimeDigitized": {"value": "2007:04:08 07:10:22"},
  "DateTimeOriginal": {"value": "2007:04:08 07:10:22"},
  "Destination": {"value": "12"},
  "DestinationDST": {"value": "0"},
  "DigitalFilter": {"value": "0"},
  "DriveMode": {"value": "0 0 0 0"},
  "EffectiveV": {"value": "2048"},
  "ExifFlag": {"value": "598"},
  "ExifVersion": {"value": "48 50 50 49"},
  "ExposureBiasValue": {"value": "0/10"},
  "ExposureCompensation": {"value": "50"},
  "ExposureMode": {"value": "1"},
  "ExposureProgram": {"value": "1"},
  "ExposureTime": {"value": "15/1"},
  "FNumber": {"value": "80/10"},
  "FileSize": {"value": "9944007"},
  "FileSource": {"value": "3"},
  "Flash": {"value": "16"},
  "FlashADump": {"value": "0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0"},
  "FlashBDump": {"value": "0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0"},
  "FlashExposureCompensation": {"value": "0"},
  "FlashInfo": {"value": "0 246 63 66 0 0 0 0 48 20 236 0 0 0 0 0 0 0 0 0 0 0 0 0 0 48"},
  "..."
}
```

遇到问题时，如果有明显的参数超过显示等问题，可以先看下原始参数中的信息，是否有超标，类似如下原始参数中的宽、高、像素、等等，（我们已知的 OSS 单边长度不能超过 4X4096，乘积不能低于 4096X4096）。

案例：Picture exceed the maximum allowable rotation range

分析：

出现这种问题基本都是原图的单边长度超过了 4096 的限制。或者四边乘机超过了 406X4096，可以用 OSS 的 ?x-oss-process=image/info 参数获取图片的信息判断是否超过限制。

```
{
  "FileSize": {"value": "3962"},
  "Format": {"value": "jpg"},
  "ImageHeight": {"value": "278"},
  "ImageWidth": {"value": "492"},
  "ResolutionUnit": {"value": "2"},
  "XResolution": {"value": "120/1"},
  "YResolution": {"value": "120/1"}}
```

我们在看下图片的原参数的宽高，可知道已经超过了我们的单边限制 4096X4，所以出现不能旋转的异常，对于这种问题，我们先要想自适应关闭，然后在用 resize 处理图片。(http://zhangyb.oss-cn-shanghai.aliyuncs.com/1.jpg?x-oss-process=image/auto-orient,0/resize,m_lfit,h_2000,w_2000,limit_1/sharpen,100/)

案例：OSS 图片尺寸超限

分析：

先按要求看下基本的报错信息和原始 URL 参数。

```
▼<Error>
  <Code>ZoomSizeInvalid</Code>
  <Message>The longest edge length is : 16384.</Message>
  <RequestId>598BBB975F57002F368E0A17</RequestId>
  <HostId>img1.epetbar.com</HostId>
</Error>
```

结论：

根据报错信息判断是图片超过显示，我们请求一下图片的原始信息看下是否超过限制。通过 info 信息可以看到用户原始图片的 height 超过限制。

案例：开启了 OSS 违规检测，图片被判定违规，但是外部还能访问到？

OSS 没有封禁功能，这个服务是内容识别来做的。属于安全产品。用户调用

内容识别后，发现是违规图片只是被冻结，不在控制台上显示，但不会被删除，正常的保存在 bucket 中。如果要不被别人访问，用户需要手动点击违规并删除，或者批量删除。详细的文档说明：https://help.aliyun.com/document_detail/28423.html?spm=5176.11065259.1996646101.searchclickresult.1de73273NHoYrM

案例：通过 OSS 获取主色调和图片不符

分析：

首先测试一下获取图片主色调的参数，查看原图的主色调 <http://static01.versa-ai.com/video/preview/497dae10-2bb6-4559-8d01-351728a51b29.jpg?x-oss-process=image/average-hue>

主色调计算不是按照屏幕颜色占比来计算的，是按照图片中心的主颜色来定的色调，计算逻辑如下：

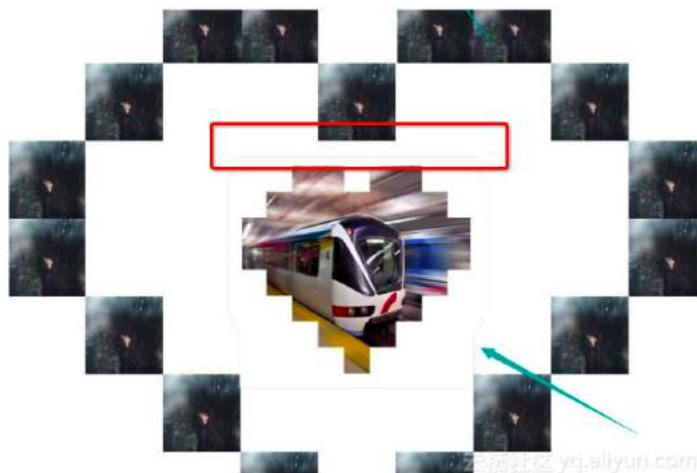
- 计算整个图片的色调的平均值 (avg_hue)。
- 遍历每个像素，计算该像素的色调值与 avg_hue 的色差（即将二者相减后取绝对值），如果该色差大于一个阈值（本文中取 30），则将该像素加入到“醒目像素”的列表。
- 计算整个“醒目像素列表”的颜色均值，得到的结果即为该图片的主色调。

案例：oss 文字水印，一个文字水印 是否可以分两行显示文字，如果不可以，是否可以一个图片 添加多个文字水印

- 文件水印分行显示：不支持。
- 添加多个文字水印：支持实例：http://image-demo.img-cn-hangzhou.aliyuncs.com/example.jpg?x-oss-process=image/resize,w_300,h_300/watermark,type_d3F5LXplbmhlaQ,size_30,text_SGVsbG8g5Zu-54mH-5pyN5YqhIQ,color_FFFFFFFF,shadow_50,t_100,g_se,x_10,y_10/watermark,type_d3F5LXplbmhlaQ,size_30,text_SGVsbG8g5Zu-54mH-

[5pyN5YqhlQ,color_FFFFFFFF,shadow_50,t_100,g_se,x_100,y_100](#)

案例：图片水印合成出现黑线



分析：

- 首先这个黑印不是因为 OSS 造成的，水印的方式是将两张图片重回，如果水印两张图是不通 RGB 图片，覆盖后色差产生的黑线是正常现象不管用任何图片处理都会存在。
- `?x-oss-process=image/average-hue` 是查询图片 RGB 的参数，可以直接在图片后面加上查看到原图的 RGB，两张图完全是不一样的。RGB 的介绍可以参考一下网上的开源说明 <https://yq.aliyun.com/articles/421836?spm=5176.10695662.1996646101.searchclickresult.3bd79460fu5pVr>。

背景图是 {“RGB”: “0x0e0e0e”} 水印是 {“RGB”: “0xffffffff”}，打完水印肯定会复现类似边框的效果。

结论：

可以通过透明度参数 `t_[1-100]` 来调整透明度将边框去掉，`t` 的值不固定按实际效果调整大小。

https://mt.xiuxiu.com/magicpicture/pic/server/blank.jpg?x-oss-process=image/watermark,image_bWFNaWNwaWN0dXJIL3N0YXJQdXp6bGUvcGljX-2hYXJ0XzEucG5nP3gtb3NzLXByb2Nlc3M9aW1hZ2UvcmlvZaXplLG1fZm-l4ZWQsd18xMDgwLGhfMTA4MCxsaW1pdF8w,g_nw,x_0,y_0,t_10

案例：通过 CDN 回原 OSS 图片处理不生小。

分析：

遇到 CDN 回原到 OSS 图片不生小的问题，不管用的是什么效果，请直接固定 OSS 的域名进行测试。利用下面的 URL 进行基础分析。

http://test.oss-cn-beijing.aliyuncs.com/MomClass/ChuX-in/3_2_336_462.jpg@30-30bl

http://test.img-cn-beijing.aliyuncs.com/MomClass/ChuX-in/3_2_336_462.jpg@30-30bl

- img-cn-region.aliyuncs.com 是老版本的 OSS 域名，图片处理的分隔符和图片处理语法和新版的 oss 域名都不一样。
- oss-cn-region.aliyuncs.com 这类域名是 2017 年使用的新域名，不兼容 img 域名的图片处理语法和分隔符“@”，需要自己在 OSS 控制台上手动执行同步，将 img 域名图片处理同步到 OSS。

结论：

上述的老域名的高斯处理效果，如果搬迁到 oss 的域名后，需要按照新的方式来处理，如下。

http://test.oss-cn-beijing.aliyuncs.com/MomClass/ChuX-in/3_2_336_462.jpg?x-oss-process=image/blur,r_3,s_30

案例：为何图片经过 OSS 缩略之后尺寸变大了？

参考：<https://yq.aliyun.com/articles/74634?spm=5176.10695662.1996646101.searchclickresult.672545f3KE153V>

案例：图片处理提示已损坏

This XML file does not appear to have any style information

```
<?xml version="1.0" encoding="UTF-8" ?>
<Error>
  <Code>BadRequest</Code>
  <Message>This image format is not supported.</Message>
  <RequestId>5B8424B7C90E8417DA3334F7</RequestId>
  <HostId>oss-cn-beijing.aliyuncs.com</HostId>
</Error>
```

出现以下集中情况可以参考如下处理过程。

- 用户的源文件在本地可以正常打开，但是上传到到 OSS 就无法进行图片处理，总是反馈 图片损坏 damage。
- 用户的源文件上传前可以正常显示，上传后无法显示。

分析：

- 以上情况出现后，先获取原始的 OSS URL 地址，然后使用 ?x-oss-process=image/info 先查看下原图信息，如果图片是好的是可以查看出源图片的属性信息的，如果查不到，直接报错，说用用户原图就是损坏的。
- 为什么下载到本地可以显示？因为本地的图片查看工具是对图片做了补偿修复的。而 OSS 不对对损坏的图片进行处理，所以在浏览器上无法显示。
- 可以用开源的 imagemagic 工具来验证这个问题。随便进行什么转换，下面是一个 resize 的测试用例，结果发现出现了 error 说明图片是损坏的。

```
convert -resize 1024x768 1123331261_15353307414801n.jpg。
```

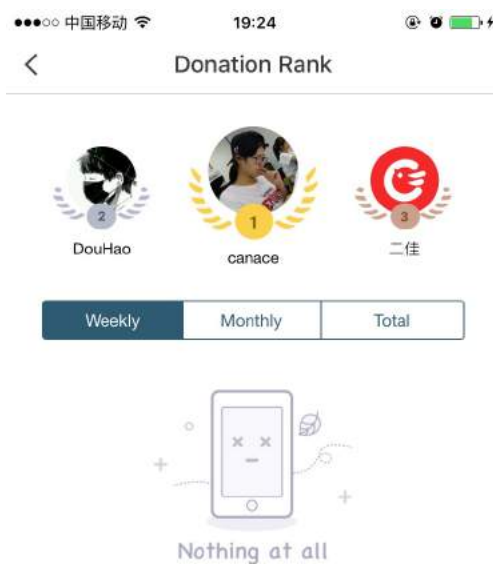
案例：图片处理参数如何加入到 signature

https://oss-de.roe.com.cn/47169215bed7496db34031e6ef1a02d8.jpg?OSSAccessKeyId=LTAIJn79E&Expires=1533650025&Signature=4z1qqhMzE97A2iks7QmYH6l%3D&x-oss-process=image/resize,w_100/quality,q_80

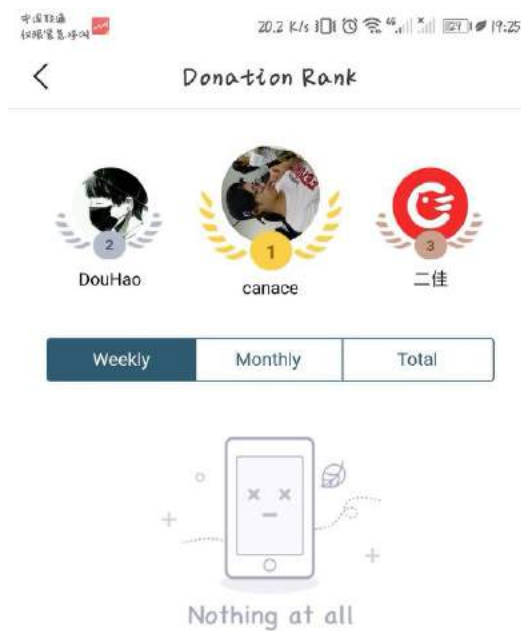
图片处理的参数要放到 signature 一起计算，否则会出现 403 的问题。

案例：存储在 OSS 图片旋转了 90 度

这是直接访问 OSS 的效果。



这通过 CDN 访问的效果。



资源社区 yq.aliyun.com

分析:

- 直接访问 OSS 正常, 说明 OSS 存储是没问题的。
- 但是通过 CDN 访问在浏览器中就出现旋转, 无非就是和浏览器的处理有关, 通过图片处理参数 info 查看原图参数带有 rotation 90, 旋转参数, cs 代码可以针对这种参数过滤调不做旋转。否则会旋转 90 度。

案例：OSS 能否识别请求的自定义 query 参数动态缩放

请求范本。

http://i.1909.com/img01/M00/*.jpg?w=xxx

w=xxx 就是客户端传的动态参数，目前还无法适配这种业务需求。

案例：BadRequest

```
<Error>
<Code>BadRequest</Code>
<Message>This image format is not supported.</Message>
<RequestId>5BA33754CBF4583BA2</RequestId>
<HostId>b.oss-cn-beijing.aliyuncs.com</HostId>
</Error>
```

云栖社区 yq.aliyun.com

分析：

先利用 imagemagic 的 convert 命令看下原图的格式，如果不在图片处理支持的范围是会出现这种情况的。通过 convert 发现原图是 svg，所以返回是预期的。

案例：InvalidArgument

```
<Code>InvalidArgument</Code>
<Message>The value: 0 of parameter: w is invalid.</Message>
<RequestId>5BA21FD8A642F41E6478</RequestId>
<HostId>luo.oss-cn-beijing.aliyuncs.com</HostId>
</Error>
```

云栖社区 yq.aliyun.com

分析：

遇到这种参数错误的先看下原图请求参数 20180899269957.jg@0w_2e_1l_1an.src 类似这种请求参数的，都是历史 img-cn-xx 域名支持的格式。转换成新的 oss-cn-xxx 域名后是不支持 img 域名的请求方式的。而且老域名是不支持 https 访问的。

案例：图片缩略后颜色变亮了



分析：

- 先分析下原图的编码，可以用开源工具获取，如果原图是 RGB 的话，压缩是不会变色的，如果原图是 CMYK 的话，压缩后会产生偏色。
- 目前对 CMYK 的兼容还在支持中，图片色彩空间被挤压产生的色彩变化。

案例：经过 CDN 后图片处理没有效果

https://bianqian.com/2018/11/01/5225a43d630442c9b-beb6684094544d5.jpg?x-oss-process=image/resize,m_lfit,h_50,w_50/format,png

分析：

- 直接看下 CDN 是否开启的去问号的功能，如果开启了忽略 ? 功能就会出现这种问题。

案例：Target bucket does not reside in the same data center as source bucket.

Target bucket does not reside in the same data center as source bucket.

- post 转储时的 bucket 和 header 中的 host 地址不一致，转储要求源 bucket 和目标 bucket 同一个 region，否则返回 400。

案例：调用 OSS 处理超过最大旋转范围

```
<Message>
Picture exceed the maximum allowable rotation range.
</Message>
```

- 使用 imagemagic 工具先看下原图是否自带了 auto-orient 自适应旋转的属性。
- 或者可以测试下 <http://image-demo.oss-cn-hangzhou.aliyuncs.com/f.jpg?x-oss-process=image/auto-orient,0/foramt,jpg> 加个 auto-orient,0 参数 如果可以正常处理就说明原图是支持了这个自适应旋转的。

- 带了自适应旋转的参数后，要求图片的宽高不能超过 4096。

案例：

手机端访问 CDN URL，携带了图片处理，发现访问 CDN 变成空白图片，刷新后又能再次出来，浏览器可以正常显示。



云栖社区 yq.aliyun.com

排查：

既然浏览器都可以访问，只有手机端不行，问题已经可以判断出 OSS 是正常的，不然浏览器肯定不会显示正常。

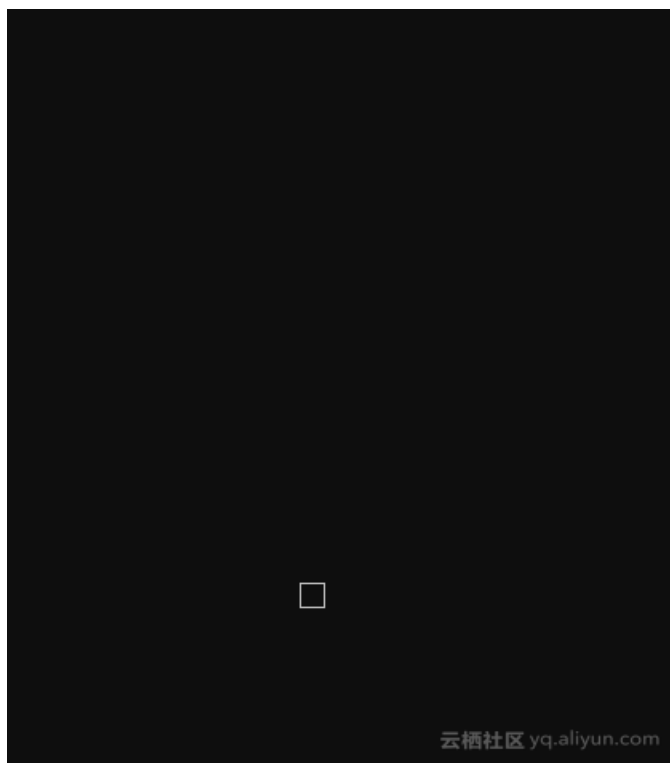
- 1) 出现问题后，先剥离 CDN 直接访问 OSS 看下图片是否能访问，如果 OSS 访问都出现异常，不可能 CDN 会正常，那为什么 CDN 刷新一下就可以显示了？因为 CDN 有缓存，一般这种情况，用户很可能开启了 “参数过滤” 功能，将 ? 后边的参数都忽略掉了，这样 CDN 就会命中客户所有的请求。表面上看就是访问 CDN 正常，访问 OSS 异常。第 1 步排查完后，基本可以定位是哪里的的问题，如果 OSS 一直都能访问，而 CDN 访问异常，说明 CDN 节点网络导致加载失败，或者 CDN 缓存了错误内容，如果是 OSS 始终显示不出来，证明图片本身就存在问题了，再向下排查。
- 2) 浏览器可以访问了，说明图片本身没有坏，也没有冻结，唯一可能就是图片本身的编码不被手机浏览器支持。通过 ffmpeg 可以看到用户的原图是 webp 格式的。

```
Input #0, png_pipe, from 'http://oss-cn-shanghai.aliyuncs.com/201811/198c7cf3ac6a4d259ae810e205e2b4bf.jpg?/x-oss-process=style/fm_webp_720p':
  Duration: N/A, bitrate: N/A
    Stream #0:0: Video: weep, rgb24(pc), 1080x1920 [SAR 5906:5906 DAR 9:16], 25 tbr, 25 tbn, 25 tbc
      2.58 M-V: -0.000 fd= 0 aq= 0KB vq= 0KB sq= 0B f=0/0
```

结论，经过上述排查知道了，图片的格式是 webp，特点是 ios 默认不支持，android 支持，因为 webp 是 chrome 推出的一个新的图片编码方式。ios 需要自己编译 webp 的支持库才可以。最后参考 <https://blog.csdn.net/wsyz768/article/details/78986918>

案例：

存储在 OSS 的图片打不开，原图和经过图片处理后的图片都打不开。



排查:

还是先用 imagemagick 这个开源工具看下图片的编码沟通能否正确被解析出来。

```
[root@edas02 aliyun-oss-php-sdk]# wget https://zhangyb.mobi/test/123.jpg
--2018-11-22 10:55:16-- https://zhangyb.mobi/test/123.jpg
正在解析主机 zhangyb.mobi (zhangyb.mobi)...
已发出 HTTP 请求, 正在等待回应 ... 200 OK
长度: 4141232 (3.9M) [image/jpeg]
正在保存至: "123.jpg"

100%[=====] 4,141,232 12.5MB/s 用时 0.3s

2018-11-22 10:55:16 (12.5 MB/s) - 已保存 "123.jpg" [4141232/4141232])

[root@edas02 aliyun-oss-php-sdk]# identify 123.jpg
identify: Not a JPEG file: starts with 0x00 0x00 `123.jpg' @ error/jpeg.c/
JPEGErrorHandler/316.
[root@edas02 aliyun-oss-php-sdk]#
```

通过 identity 命令可以确认图片本身的编码构成出现问题，并非是存储到 OSS 出现的问题，类似问题都可以用这个工具先来分析下。

案例：

图片处理完后背景色多了一个分割线。



排查：

- 1) 图片并没有所谓的分割线，只是图片 RGB 处理后的色彩构成问题。
- 2) 原图是 RGB 的真彩色 ImageHeight": { "value": "2560" "ImageWidth": { "value": "1440" , 像素被你 裁剪到 m_fill,h_1920,w_1080 , RGB 的像素点位被压缩，发生变化很正常。

3) 可以将绝对质量提高到 100，quality,Q_100，目前只有这个方法。

案例：

客户图片上传到 OSS 访问报错。

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Error>
  <Code>ImageDamage</Code>
  <Message>The image file may be damaged.</Message>
  <RequestId>5D3E6D5C88FF7D2AAF956545</RequestId>
  <HostId>myfcomicaadminmark.oss-cn-hangzhou.aliyuncs.com</HostId>
</Error>
```

排查：

- 先看图片下载到本地后是否可以打开，如果打不开则证明是图片本身的问题，和 OSS 图片处理没关系。
- 实际访问下测试看看是否有报错，如果看到有明显的 requestId 报错，可以拿 requestId 反馈阿里云查询日志。
- 看下原图是否超过了一些限制，限制如下。

OSS- 视频截帧

使用描述

用户对存储在 OSS 的视频文件进行视频截图处理，抽取视频中某个时间点或者某个视频帧作为截图参考。

注意事项

- 当前仅支持对视频编码格式为 H264 的视频文件进行视频截帧。
- OSS 当前没有默认保存视频截帧的操作，视频截帧的图片需手动下载到本地。

截图的主要参数

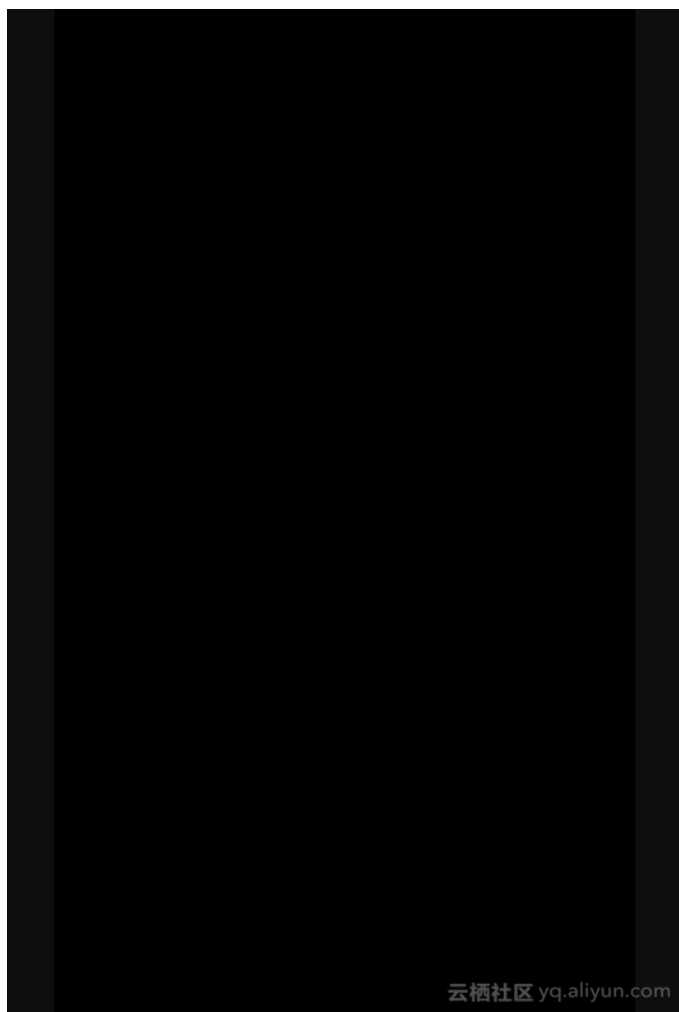
参数	描述	取值范围
t	指定截图时间	[0,视频时长] 单位: ms
w	指定截图宽度, 如果指定为0, 则自动计算。	[0,视频宽度] 单位: 像素 (px)
h	指定截图高度, 如果指定为0, 则自动计算。如果w和h都为0, 则输出为原视频宽高。	[0,视频高度] 单位: 像素 (px)
m	指定截图模式, 不指定则为默认模式, 根据时间精确截图。如果指定为fast, 则截取该时间点之前的最近的一个关键帧。	枚举值: fast
f	指定输出图片的格式。	枚举值: jpg、png
ar	指定是否根据视频信息自动旋转图片。如果指定为auto, 则会在截图生成之后根据视频旋转信息进行自动旋转。	枚举值: auto

常见问题

视频截帧是黑屏

原始调用参数：

http://vods-weimi.com/video/sd/2018/12/f15c7c5a5b7d7220a0941928e6b-9b58a.mp4?x-oss-process=video/snapshot,t_8000,f_jpg,m_fast



问题排查：

排查类似问题，需要大家本地先具备解码的工具，类似 ffplay ffmpeg ffprobe 的解码工具，目的先分析下原视频的关键帧率分布，以及视频帧率是否正常；例如这个视频第一个关键帧本身就是黑的，fast 模式会截取指定时间之前的最近一个关键帧，往后一点就有第二个关键帧了。

http://vods-weimi.com/video/sd/2018/12/f15c7c5a5b7d7220a0941928e6b-9b58a.mp4?x-oss-process=video/snapshot,t_10000,f_jpg,m_fast



OSS 直播推流功能

场景描述

目前有很多直播爱好者使用的是 OSS + RTMP 做的推流直播，其中不乏一些企业级别应用，由于 OSS 作为推流的接收略微有一些复杂，故单独开篇讲一下。其实不建议使用 OSS+RTMP 做直播推流，因为功能性相比专业的阿里云直播产品来说，OSS 的推流适合监管备份等特定场景，客户端对直播推流的延迟要求不是很敏感。如果对直播的拉流推流延迟有高敏感的场景，建议大家使用阿里云视频直播服务，可以做到上下行链路加速，且支持多样化的直播功能适配。

使用基础

简单的直播知识

您需要了解甚至掌握简单的直播知识技巧才能熟练的使用 OSS 直播，不仅是针对 OSS 还涉及到客户端的推流等相关问题，所以需要明白直播的相关基础。

【音视频头介绍】

OSS 的直播功能是建立在 RTMP 直播传输协议的基础上，所以需要指导一些基础的 RTMP 知识：RTMP 的音视频流的封装形式和 FLV 格式相似，流媒体服务器向客户端发送包含 H264 和 AAC 的 RTMP 直播流，需要首先发送这两个 header，没有这些信息播放端是无法解码音视频流的，其中音频 tag 格式如下。

- 1) AVC sequence header
- 2) AAC sequence header

长度	字段	说明
5 bit	audioObjectType	编码结构类型, AAC-LC为2
4 bit	samplingFrequencyIndex	音频采样率索引值, 44100对应值4
4 bit	channelConfiguration	音频输出声道, 2
	ASpecificConfig	该结构包含以下三项
1 bit	frameLengthFlag	标志位, 用于表明IMDCT窗口长度, 0
1 bit	dependsOnCoreCoder	标志位, 表明是否依赖于corecoder, 0
1 bit	extensionFlag	选择了AAC-LC, 这里必须为0

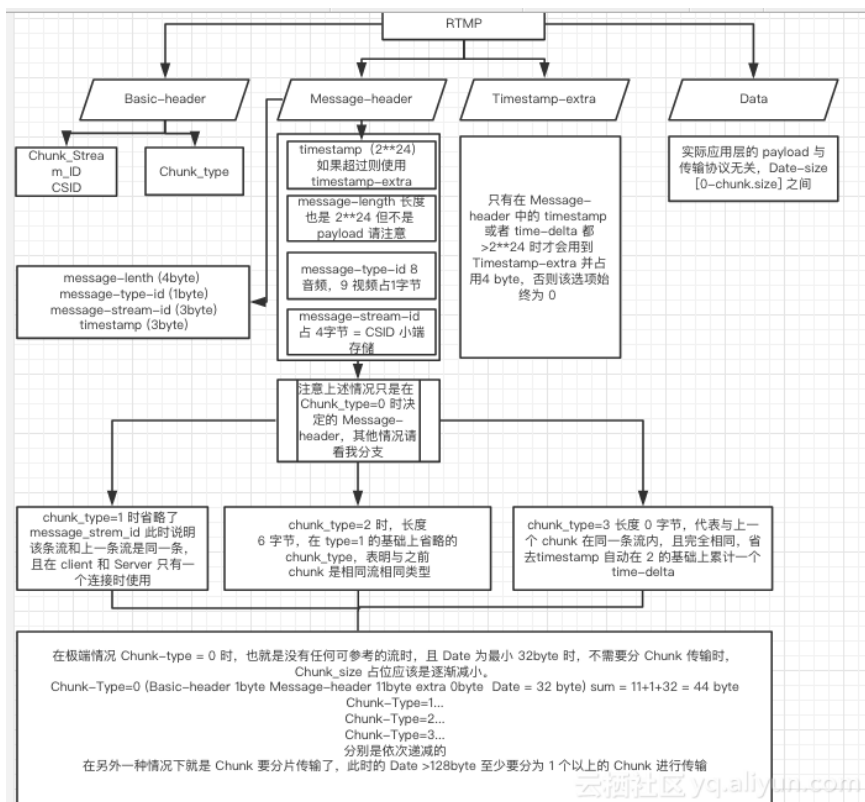
从上面推论出 AAC sequence header 内容的前 2 个字节是 0xAF 0x00, 我们来看一个示例。

3707	13.915023	192.168.107.94	113.240.243.236	RTMP	75 Audio Data
3714	13.956044	113.240.243.236	192.168.107.94	TCP	60 1935 → 53539 [ACK] Seq=
3715	13.956109	192.168.107.94	113.240.243.236	RTMP	428 Audio Data[Audio Data
3716	13.956422	113.240.243.236	192.168.107.94	TCP	60 1935 → 53539 [ACK] Seq=
3718	13.963572	192.168.107.94	113.240.243.236	TCP	190 53539 → 1935 [PSH, ACK]
3719	13.963862	113.240.243.236	192.168.107.94	TCP	60 1935 → 53539 [ACK] Seq=
Internet Protocol Version 4, Src: 192.168.107.94, Dst: 113.240.243.236					
Transmission Control Protocol, Src Port: 53539 (53539), Dst Port: 1935 (1935), Seq: 3372, Ack: 4199, Len: 21					
Real Time Messaging Protocol (Audio Data)					
RTMP Header					
00.. = Format: 0					
..00 0100 = Chunk Stream ID: 4					
Timestamp: 0					
Body size: 9					
Type ID: Audio Data (0x00)					
Stream ID: 1					
RTMP Body					
Control: 0xaf (HE-AAC 44 kHz 16 bit stereo)					
1010 = Format: HE-AAC (10)					
.... 11.. = Sample rate: 44 kHz (3)					
..... 1. = Sample size: 16 bit (1)					
..... 1. = Channels: stereo (1)					
Audio data: 0012100000070000					
0000	74 1f 4a 26 50 a9 40 8d	5c 00 2b 07 08 00 45 00	t.J&P.@. \.+...E.		
0010	00 3d 70 7b 40 00 80 06	f8 5b c0 a8 6b 5e 71 f0	.m{&... [.k^q.		
0020	f3 ec d1 23 07 8f 4f 6f	23 dd 91 45 ca 5e 50 18	...#.0o #..E.^P.		
0030	00 00 98 17 00 00 04 00	00 00 00 00 09 08 01 00		
0040	00 00 af 00 12 10 00 00	07 00 00		

- 3) ADIF: Audio Data Interchange Format 音频数据交换格式。这种格式的特征是可以确定的找到这个音频数据的开始, 不需进行在音频数据流中间开始的解码, 即它的解码必须在明确定义的开始处进行。故这种格式常用在磁盘文件中。
- 4) ADTS: Audio Data Transport Stream 音频数据传输流。这种格式的特征是它是一个有同步字的比特流, 解码可以在这个流中任何位置开始。它的特征类似于 mp3 数据流格式。

【RTMP 内容介绍】

以下是我对 RTMP 总结的一张完整描述图。



对象存储推流架构

看下官网对于 OSS 推流的过程定义

- 1) 只能使用 RTMP 推流的方式，不支持拉流。
- 2) 必须包含视频流，且视频流格式为 H264。
- 3) 音频流是可选的，并且只支持 AAC 格式，其他格式的音频流会被丢弃。
- 4) 转储只支持 HLS 协议。
- 5) 一个 LiveChannel 同时只能有一个客户端向其推流。

RTMP 的推流格式：

- demo: rtmp://your-bucket.oss-cn--hangzhou.aliyuncs.com/live/test-channel。
- live 等同于 RTMP 的 APP 挂载点。
- test-channel 等同于 RTMP 的 stream name。

RTMP URL 推流签名：

- demo:

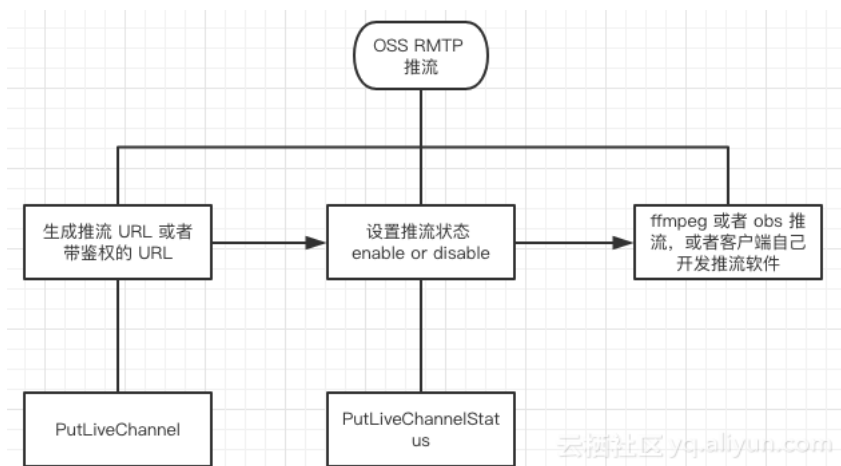
```
rtmp://${bucket}.${host}/
live/${channel}?OSSAccessKeyId=xxx&Expires=yyy&Signature=zzz&${params}
```

- 推流前 LiveChannel 有两种 Status: enabled 和 disabled, 用户可以使用本接口在两种 Status 之间进行切换。处于 disabled 状态时, OSS 会禁止用户向该 LiveChannel 进行推流操作; 如果有用户正在向该 LiveChannel 推流, 那么推流的客户端会被强制断开 (可能会有 10s 左右的延迟)。

对象存储推流的流程汇总图如下

生成推流 URL API

设置推流状态 API



使用 JAVA SDK 生成推流地址

我们现在用 java 的 SDK 演示一下如上的推理过程，在跑 SDK 之前，需要先搭建好一套本地的 eclipse 环境，如下是我用的 eclipse，如果有没搭建请网上搜索一下 eclipse 的搭建方式（之前需要安装 JDK 且配置环境变量）。

环境要求

- Eclipse 版本: Version: Neon.3 Release (4.6.3)。
- JDK 版本: jdk1.8.0_144。
- OSS: 公开读（为了验证推流功能是否正常，我们用公开读的方式快速测试）。
- 我们采用主函数入口的方式，实例化其他类进行调用，这样方便类的拆分，不用都集合在主函数中。
- 主函数 domain，实例化 OSSClient 对象传入到 RtmpTest 类中测试。
- 所有的 jar 都会通过官方的 maven 解决的依赖关系，https://help.aliyun.com/document_detail/32009.html

```
package javasdk;

import java.io.FileNotFoundException;
import java.text.ParseException;
import java.util.HashMap;
import java.util.Map;

import com.aliyun.oss.OSSClient;

public class domain {
    public static void main( String[] args ) throws ParseException,
FileNotFoundException
    {

        System.out.println( "Hello World!" );
        String accessid = "AK";
        String secretkey = "SK";
        String objectpath = "C://Users//hanli.zyb//Desktop//running.png";
        String bucket = "bucket";
        String object = "running";
        String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
```

```

        // OSS + rtmp 推流
        String bucketName = "ali-hangzhou";
        RtmpTest pushoss = new RtmpTest();
        OSSClient ossClient = new OSSClient(endpoint, AK, SK);
        pushoss.testCreateLiveChannel(bucketName,ossClient);

    }
}

package javasdk;

import java.text.ParseException;
import java.util.Date;
import java.util.List;
import junit.framework.Assert;

import org.junit.Ignore;
import org.junit.Test;

import com.aliyun.oss.OSSClient;
import com.aliyun.oss.OSSErrorCode;
import com.aliyun.oss.OSSException;
import com.aliyun.oss.common.utils.DateUtil;
import com.aliyun.oss.model.CannedAccessControlList;
import com.aliyun.oss.model.CreateLiveChannelRequest;
import com.aliyun.oss.model.CreateLiveChannelResult;
import com.aliyun.oss.model.ListLiveChannelsRequest;
import com.aliyun.oss.model.LiveChannel;
import com.aliyun.oss.model.LiveChannelInfo;
import com.aliyun.oss.model.LiveChannelListing;
import com.aliyun.oss.model.LiveChannelStat;
import com.aliyun.oss.model.LiveChannelStatus;
import com.aliyun.oss.model.LiveChannelTarget;
import com.aliyun.oss.model.LiveRecord;
import com.aliyun.oss.model.PushflowStatus;
import com.aliyuncs.DefaultAcsClient;
import com.aliyuncs.profile.DefaultProfile;
import com.aliyuncs.profile.IClientProfile;

public class RtmpTest {
    String bucketName = "bucket";
    final String liveChannel = "stream name";
    @Test
    public void testCreateLiveChannelDefault(String bucketname,OSSClient
    ossClient) {

        try {
            CreateLiveChannelRequest createLiveChannelRequest = new

```

```

CreateLiveChannelRequest(
    bucketName, liveChannel);
    CreateLiveChannelResult createLiveChannelResult = ossClient.
createLiveChannel(createLiveChannelRequest);
    LiveChannelInfo liveChannelInfo = ossClient.
getLiveChannelInfo(bucketName, liveChannel);

    ossClient.deleteLiveChannel(bucketName, liveChannel);
} catch (Exception e) {
    Assert.fail(e.getMessage());
}
}

@Test
public void testCreateLiveChannel(String bucketname,OSSClient ossClient)
{
    final String liveChannel = "normal-create-live-channel";
    final String liveChannelDesc = "my test live channel";

    try {
        LiveChannelTarget target = new LiveChannelTarget("HLS", 100, 99,
"myplaylist.m3u8");
        CreateLiveChannelRequest createLiveChannelRequest = new
CreateLiveChannelRequest(
            bucketName, liveChannel, liveChannelDesc,
LiveChannelStatus.Enabled, target);

        CreateLiveChannelResult createLiveChannelResult = ossClient.
createLiveChannel(createLiveChannelRequest);
        System.out.println(createLiveChannelResult.getPublishUrls());
        /*Assert.assertEquals(createLiveChannelResult.getPublishUrls().
size(), 1);
        Assert.assertTrue(createLiveChannelResult.getPublishUrls().
get(0).startsWith("rtmp://"));
        Assert.assertTrue(createLiveChannelResult.getPublishUrls().
get(0).endsWith("live/" + liveChannel));
        Assert.assertEquals(createLiveChannelResult.getPlayUrls().size(),
1);
        Assert.assertTrue(createLiveChannelResult.getPlayUrls().get(0).
startsWith("http://"));
        Assert.assertTrue(createLiveChannelResult.getPlayUrls().get(0).
endsWith(liveChannel + "/myplaylist.m3u8"));*/

        /* LiveChannelInfo liveChannelInfo = ossClient.
getLiveChannelInfo(bucketName, liveChannel);
        Assert.assertEquals(liveChannelInfo.getDescription(),
liveChannelDesc);
        Assert.assertEquals(liveChannelInfo.getStatus(),

```

```

LiveChannelStatus.Disabled);
        Assert.assertEquals(liveChannelInfo.getTarget().getType(),
"HLS");
        Assert.assertEquals(liveChannelInfo.getTarget().
getFragDuration(), 100);
        Assert.assertEquals(liveChannelInfo.getTarget().getFragCount(),
99);
        Assert.assertEquals(liveChannelInfo.getTarget().
getPlaylistName(), "myplaylist.m3u8");*/

        // ossClient.deleteLiveChannel(bucketName, liveChannel);
    } catch (Exception e) {
        Assert.fail(e.getMessage());
    }
}
}
}

```

其中我们最关注的是 `testCreateLiveChannel` 类，创建了推流地址，其中参数 `LiveChannelStatus.enable` 是要让推流变为可用状态。`running` 主函数，打印出推流地址。`rtmp://hangzhou.oss-cn-hangzhou.aliyuncs.com/live/normal-create-live-channel`。

```

public void testCreateLiveChannel(String bucketname, OSSClient ossClient) {
    final String liveChannel = "normal-create-live-channel";
    final String liveChannelDesc = "my test live channel";

    try {
        LiveChannelTarget target = new LiveChannelTarget("HLS", 100, 99,
"myplaylist.m3u8");
        CreateLiveChannelRequest createLiveChannelRequest = new
CreateLiveChannelRequest(
            bucketName, liveChannel, liveChannelDesc,
LiveChannelStatus.Enabled, target);

        CreateLiveChannelResult createLiveChannelResult = ossClient.
createLiveChannel(createLiveChannelRequest);
        System.out.println(createLiveChannelResult.getPublishUrls());
        /*Assert.assertEquals(createLiveChannelResult.getPublishUrls().
size(), 1);
        Assert.assertTrue(createLiveChannelResult.getPublishUrls().
get(0).startsWith("rtmp://"));
        Assert.assertTrue(createLiveChannelResult.getPublishUrls().
get(0).endsWith("live/" + liveChannel));
        Assert.assertEquals(createLiveChannelResult.getPlayUrls().size(),
1);
    }
}

```



```
        Assert.assertTrue(createLiveChannelResult.getPlayUrls().get(0).
startsWith("http://"));
        Assert.assertTrue(createLiveChannelResult.getPlayUrls().get(0).
endsWith(liveChannel + "/myplaylist.m3u8"));*/

        // ossClient.deleteLiveChannel(bucketName, liveChannel);
    } catch (Exception e) {
        Assert.fail(e.getMessage());
    }
}
```

经验场景问题分析

OSS 网络拥塞分析

基础排查

- ping 工具，目的测试到对端的 IP 链路是否有丢包，RTT (Round-Trip Time) 是否有大的波动。详细命令。

```
ping -c 100 -i 0.01 -s 1024。
```

```
! [1] (https://yqfile.alicdn.com/a573e890357aee4205b86b68a1ce2cb431f8044f.jpeg)
```

- mtr -n 通过 MTR 可以看到每一条的路由是否有丢包。

Keys: Help Display mode Restart statistics Order of fields quit									
Host									
		Packets					Pings		
		Loss%	Snt	Last	Avg	Best	Wrst	StDev	
1.	???								
2.	11.220.36.9	0.0%	4	3.3	202.3	3.0	799.5	398.1	
3.	11.220.36.130	50.0%	4	53.4	48.0	42.6	53.4	7.6	
4.	11.217.38.222	0.0%	4	2.2	2.2	2.1	2.4	0.0	
5.	42.120.253.9	0.0%	4	2.5	2.5	2.5	2.6	0.0	
6.	140.205.26.205	0.0%	4	34.6	34.8	34.5	35.1	0.0	
7.	116.251.118.49	0.0%	4	35.1	35.1	35.1	35.2	0.0	
8.	11.204.180.121	0.0%	4	41.2	41.1	41.0	41.2	0.0	
9.	11.218.196.90	0.0%	4	34.0	34.0	34.0	35.0	0.0	
10.		0.0%	4	36.5	36.5	36.5	36.5	0.0	

- telnet 80 端口是否能通。保证 80 是通的才能下载。

```
Trying 185.94...
Connected to .oss-cn-beijing.aliyuncs.com.
Escape character is '^]'.
^]
HTTP/1.1 400 Bad Request
Server: AliyunOSS
Date: Tue, 30 Oct 2018 04:34:56 GMT
Content-Type: text/html
Content-Length: 267
Connection: close
```

- 提供报错时的 OSS response header 中的 requestID 信息，一般 500 2XX 3XX 4XX 都会有 requestID 返回，504、502、503 这种网络超时的状态没有 requestID。

```
< Server: AliyunOSS
< Date: Wed, 17 Oct 2018 06:33:07 GMT
< Content-Type: application/xml
< Content-Length: 288
< Connection: keep-alive
< x-oss-request-id: 5BC6D7A383B4CE2EB3C3FFDF
```

案例：DNS 解析不稳定导致 curl 延迟

```
[root@k8stest-node-1 ~]# time curl oss-cn-shenzhen.aliyuncs.com
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>AccessDenied</Code>
  <Message>Anonymous access is forbidden for this operation.</Message>
  <RequestId>5BC3F3F5EDCFF727439475A2</RequestId>
  <HostId>oss-cn-shenzhen.aliyuncs.com</HostId>
</Error>

real    0m0.086s
user    0m0.006s
sys     0m0.005s
[root@k8stest-node-1 ~]# time curl oss-cn-shenzhen.aliyuncs.com
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>AccessDenied</Code>
  <Message>Anonymous access is forbidden for this operation.</Message>
  <RequestId>5BC3F41BAD894AECA187FB7</RequestId>
  <HostId>oss-cn-shenzhen.aliyuncs.com</HostId>
</Error>

real    0m10.545s
user    0m0.004s
sys     0m0.007s
[root@k8stest-node-1 ~]# time curl oss-cn-shenzhen.aliyuncs.com
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>AccessDenied</Code>
  <Message>Anonymous access is forbidden for this operation.</Message>
  <RequestId>5BC3F5FC34F3CD42CCF57160</RequestId>
  <HostId>oss-cn-shenzhen.aliyuncs.com</HostId>
</Error>

real    0m10.546s
user    0m0.005s
sys     0m0.007s
[root@k8stest-node-1 ~]#
```

第一次执行

第二次执行

第三次执行

云栖社区 yq.aliyun.com

```

[root@82test-node-1 ~]# for ((i=1;i<10;i++));do curl -o /dev/null -s -v %${time_name}lookup:%${time_total} oss-cn-shenzhen.aliyuncs.com -I -echo -e: done
0.061:0.079
0.061:0.079
0.061:0.079
5.515:5.531
0.061:0.078
0.061:0.079
0.061:0.080
0.061:0.079
0.061:0.079
[root@82test-node-1 ~]# for ((i=1;i<10;i++));do curl -o /dev/null -s -v %${time_name}lookup:%${time_total} oss-cn-beijing.aliyuncs.com -I -echo -e: done
0.061:0.142
0.061:0.139
0.061:0.138
0.061:0.139
0.061:0.139
0.061:0.137
0.061:0.140
0.061:0.143
[root@82test-node-1 ~]# for ((i=1;i<10;i++));do curl -o /dev/null -s -v %${time_name}lookup:%${time_total} oss-cn-beijing.aliyuncs.com -I -echo -e: done
0.061:0.141
0.061:0.137
0.061:0.136
0.061:0.140
0.061:0.145
0.061:0.142
0.061:0.140
0.061:0.148
0.061:0.138

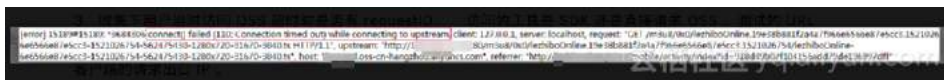
```

云栖社区 yq.aliyun.com

分析:

通过上述信息基本可以判断是 DNS 问题，本次 curl 的时间都不稳定，加上 DNS 解析时间后，发现是卡在了 DNS 解析上，经过沟通发现阿里云的 DNS 223.5.5.5 在广东的节点已经撤销，建议使用运营商的 DNS 或者 114 的 DNS。

案例：本地机房下载 OSS 资源超时



分析:

- 首先理清楚自己访问 OSS 架构是否是直接请求到 OSS 还是中间有 proxy 代理，如果有 proxy 代理的情况下先自己排查 client 到 proxy 的链路情况。
- 自己先找个其他 region 的 bucket 看下是否能否复现问题，以此排除掉是不是 OSS 的问题。
- 最好能够在客户端抓包分析一下，看看网络上卡在哪里导致的连接失败。

案例：下载 socketTimeout

```
2018-10-15 11:12:24,309 ERROR [da6d3f048d0742e7a666b26519eac6e9] [downloadExecutor-13] c.d.d.v.p.FaceIdProviderImpl: 下载异常
java.net.SocketTimeoutException: timeout
at okio.Okio$4.newTimeoutException(Okio.java:230) ~[okio-1.13.0.jar!/:?]
at okio.AsyncTimeout.exit(AsyncTimeout.java:285) ~[okio-1.13.0.jar!/:?]
at okio.AsyncTimeout$2.read(AsyncTimeout.java:241) ~[okio-1.13.0.jar!/:?]
at okio.RealBufferedSource.indexOf(RealBufferedSource.java:345) ~[okio-1.13.0.jar!/:?]
at okio.RealBufferedSource.readUtf8LineStrict(RealBufferedSource.java:217) ~[okio-1.13.0.jar!/:?]
at okio.RealBufferedSource.readUtf8LineStrict(RealBufferedSource.java:211) ~[okio-1.13.0.jar!/:?]
at okhttp3.internal.http1.Http1Codec.readResponseHeaders(Http1Codec.java:187) ~[okhttp-3.9.0.jar!/:?]
at okhttp3.internal.http.CallServerInterceptor.intercept(CallServerInterceptor.java:88) ~[okhttp-3.9.0.jar!/:?]
at okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147) ~[okhttp-3.9.0.jar!/:?]
at okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:121) ~[okhttp-3.9.0.jar!/:?]
at okhttp3.logging.HttpLoggingInterceptor.intercept(HttpLoggingInterceptor.java:212) ~[logging-interceptor-3.9.0.jar!/:?]
at okhttp3.internal.http.RealInterceptorChain.proceed(RealInterceptorChain.java:147) ~[okhttp-3.9.0.jar!/:?]
at okhttp3.internal.connection.ConnectInterceptor.
```

常见于 SDK、API 调用时的报错，客户源可能是在云主机或者 PC 端。通过文章开始所说的信息我们判断是否是必现问题，如果问题必现的话很容易能定位。如果不容易出现只能分层排查。

分析：

- 先看下主机的 socket 资源是否足够分配，通过可以用 netstat 或者 ss 命令来查看本机的 socket 连接数，如果主机 TCP 占用较慢，很容易出现连接数资源不够分配的情况。

```
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags Type State I-Node PID/Program name Path
unix 2 [ ACC ] STREAM LISTENING 17312 2335/AllyunDun /tmp/Aegis-<Guid(5A2C30A2-A87D-490A-9281-6765EDA07CBA)>
unix 2 [ ACC ] SEQPACKET LISTENING 10547 1/systemd /run/udev/control
unix 2 [ ACC ] STREAM LISTENING 10557 1/systemd /run/lvm/lvmstat.socket
unix 2 [ ] DGRAM 418 1/systemd /run/systemd/notify
unix 2 [ ] DGRAM 420 1/systemd /run/systemd/cgroups-agent
unix 2 [ ACC ] STREAM LISTENING 428 1/systemd /run/systemd/journal/stdout
unix 5 [ ] DGRAM 431 1/systemd /run/systemd/journal/socket
unix 7 [ ] DGRAM 433 1/systemd /dev/log
unix 2 [ ACC ] STREAM LISTENING 10175 1/systemd /var/run/dbus/system_bus_socket
unix 2 [ ACC ] STREAM LISTENING 10432 1/systemd /run/systemd/private
unix 2 [ ACC ] STREAM LISTENING 10447 1/systemd /run/lvm/lvmpolld.socket
unix 2 [ ACC ] STREAM LISTENING 17313 2335/AllyunDun /usr/local/Aegis/Aegis-<Guid(5A2C30A2-A87D-490A-9281-6765EDA07CBA)>
unix 2 [ ] DGRAM 10470 1/systemd /run/systemd/shutdown
unix 3 [ ] STREAM CONNECTED 11878 460/dbus-daemon /var/run/dbus/system_bus_socket
unix 3 [ ] DGRAM 10843 364/systemd-udev 508/polkitd
unix 3 [ ] STREAM CONNECTED 12997 335/systemd-journal /run/systemd/journal/stdout
unix 3 [ ] STREAM CONNECTED 11694 335/systemd-journal /run/systemd/journal/stdout
unix 3 [ ] STREAM CONNECTED 15976 2260/sshd
unix 3 [ ] DGRAM 10844 364/systemd-udev 515/irqbalance
unix 3 [ ] STREAM CONNECTED 12487 335/systemd-journal /run/systemd/journal/stdout
unix 3 [ ] STREAM CONNECTED 11504 335/systemd-journal /run/systemd/journal/stdout
unix 3 [ ] STREAM CONNECTED 10185 1/systemd
```

- 看下主机的 ulimit -n 的文件描述符是否够用。
- 如果用户使用的是 SDK，需要确认 OSSClient 在初始化时是否限制了连接

数和超时时间。如果通过前面测试发现网路不好抖动很大，建议把 socket-timeout 的时间放长些。

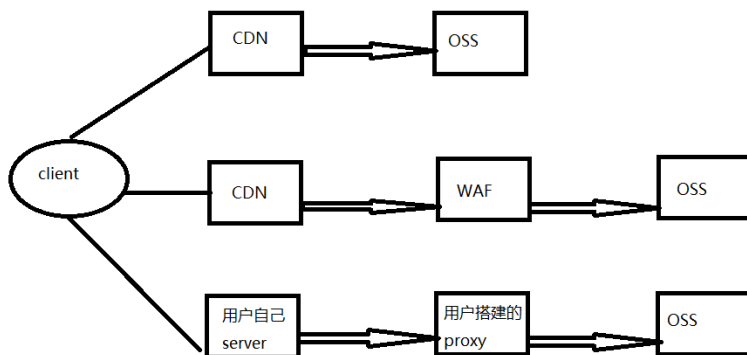
```
// 创建 ClientConfiguration。ClientConfiguration 是 OSSClient 的配置类，可配置代理、
// 连接超时、最大连接数等参数。
ClientConfiguration conf = new ClientConfiguration();

// 设置 OSSClient 允许打开的最大 HTTP 连接数，默认为 1024 个。
conf.setMaxConnections(200);
// 设置 Socket 层传输数据的超时时间，默认为 50000 毫秒。
conf.setSocketTimeout(10000);
// 设置建立连接的超时时间，默认为 50000 毫秒。
conf.setConnectionTimeout(10000);
// 设置从连接池中获取连接的超时时间(单位：毫秒)，默认不超时。
conf.setConnectionRequestTimeout(1000);
// 设置连接空闲超时时间。超时则关闭连接，默认为 60000 毫秒。
conf.setIdleConnectionTime(10000);
// 设置失败请求重试次数，默认为 3 次。
conf.setMaxErrorRetry(5);
// 设置是否支持将自定义域名作为 Endpoint，默认支持。
conf.setSupportCname(true);

// 创建 OSSClient 实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret,
conf);

// 关闭 OSSClient。
ossClient.shutdown();
```

一些特殊的架构场景，比如加了一些 proxy 产品，这种情况经常会遇到瓶颈，需要分开来看，如下图是我们总结一些常用的架构。



第一种架构

- 先确认访问到 CDN 的 URL 是否回到了 OSS，还是直接访问 OSS 超时了。
- 如果是访问 CDN 出现超时，需要确认是某个节点还是大面积节点出现问题。可以通过 17ce 这种批量测试网站检查下。
- 如果是不同的 client 请求到同一个 CDN 节点超时，很可能 CDN 节点故障需要工单升级处理。
- 如果是访问 CDN 正常，但是固定 OSS 源站出现超时，经过不同的客户端测试都能复现证明 OSS 确实出现问题，需要工单升级处理。如果访问 CDN、OSS 都没有超时，很可能是 CDN 回 OSS 超时。这种回源链路超时，基本很难复现，需要升级工单快速跟进处理。

第二种架构

- 还是一样的方法，先确认是访问 CDN、waf、OSS 哪个产品出现的超时。定位好环节后再进行分析。
- 客户端有条件的情况下建议先查下到 WAF 的日志，或者 WAF 的回源日志确认下是否是 WAF 的问题导致超时。PS WAF 对回源 CDN 如果过于频繁会出现被拉黑的情况，目的是为了防攻击，如果出现回源 WAF 超时要升级工单确认下是否触发了防攻击的策略。

第三种架构

- 与之前比较，多了一个 proxy 的转发在用户的业务 server 和 OSS 之间。这种情况先排查 server 到 proxy 之间的链路。
- server-proxy 是否有链路抖动，ping MTR 结果都可以。
- proxy 带宽是否有被打满。
- proxy 是否有 NAT 的转换导致 OSS 建立连接 session 混乱。
- proxy 到 OSS 的链路，可以通过 ping MTR 测试。

案例：通过内网地址 wget 下载慢

```
>  
< HTTP/1.1 200 OK  
< Server: AliyunOSS  
< Date: Tue, 30 Oct 2018 04:46:23 GMT  
< Content-Type: image/jpeg  
< Content-Length: 3962  
< Connection: keep-alive  
< x-oss-request-id: 5BD7E21F99C1BD4D9CBE6836  
< Accept-Ranges: bytes  
< ETag: "E6441F6DAA203912A397107F0515008F"  
< Last-Modified: Fri, 17 Aug 2018 09:46:32 GMT  
< x-oss-object-type: Normal  
< x-oss-hash-crc64ecma: 13078585244275181964  
< x-oss-storage-class: Standard  
< Content-MD5: 5kQfbaogORKjlxB/BRUAjw==  
< x-oss-server-time: 60
```

- 如果 type 类型是 normal / multipart 文件读取数据是多线程的，一般情况下不会慢，如果慢的话，需要提供 requestID 升级阿里云查询下。
- 如果是 append 文件读取速度是单线程的，符合预期。

结论：

append 类型的文件是追加写，wget 下载时，服务端的 read 是单线程，所以速度提不上去。

OSS signature 计算失败分析

背景

出现 signature 一般出现客户端自签名调 API 的操作中, signature 的计算稍微复杂点, 建议最好用 SDK 来替代计算的过程和多样性。如果业务强需求, 先要读懂如果计算 signature。

签名分类

Header 头中携带签名。

https://help.aliyun.com/document_detail/31951.html?spm=a2c4g.11174283.6.1078.40437da2HGkyMH

URL 中携带签名。

https://help.aliyun.com/document_detail/31952.html?spm=a2c4g.11186623.6.1079.7b61734cNAir2U

签名区别

Header	URL
不支持设置 expires	支持设置 expires
常用 method GET、POST、PUT	常用 method GET、PUT
date 时间是 GMT 格式	date 替换成 expires 变成时间戳
signature 不需要 URL encode	signature 需要 URL encode

计算签名 demo

当客户通过 header 或者 URL 中自签名计算 signature 时，经常会遇到计算签名失败“The request signature we calculated does not match the signature you provided”，可以参考以下 demo 演示了如何调用 API 自签名时上传 Object 到 OSS，注意签名和 header 加入的内容。

```
#!/usr/bin/env python
#Author: hanli
#Update: 2018-09-29

from optparse import OptionParser
import urllib, urllib2
import datetime
import base64
import hmac
import sha
import os
import sys
import time

class Main():

    # Initial input parse

    def __init__(self,options):

        self.ak = options.ak
        self.sk = options.sk
        self.ed = options.ed
        self.bk = options.bk
        self.fi = options.fi
        self.oj = options.objects
        self.left = '\033[1;31;40m'
        self.right = '\033[0m'
        self.types = "application/x-www-form-urlencoded"
        self.url = 'http://{0}.{1}/{2}'.format(self.bk,self.ed,self.oj)

    # Check client input parse

    def CheckParse(self):

        if (self.ak and self.sk and self.ed and self.bk and self.oj and self.fi)
        != None:
            if str(self.ak and self.sk and self.ed and self.bk and self.oj and
```

```

self.fi):
    self.PutObject()
else:
    self.ConsoleLog("error","Input parameters cannot be empty")

# GET local GMT time

def GetGMT(self):

    SRM = datetime.datetime.utcnow()
    GMT = SRM.strftime('%a, %d %b %Y %H:%M:%S GMT')

    return GMT

# GET Signature

def GetSignature(self):

    mac = hmac.new("{0}".format(self.sk), "PUT\n\n{0}\n{1}\n/{2}/{3}".
format(self.types,self.GetGMT(),self.bk,self.oj), sha)
    Signature = base64.b64encode(mac.digest())

    return Signature

# PutObject

def PutObject(self):

    try:
        with open(self.fi) as fd:
            files = fd.read()
    except Exception as e:
        self.ConsoleLog("error",e)

    try:
        request = urllib2.Request(self.url, files)
        request.add_header('Host','{0}.{1}'.format(self.bk,self.ed))
        request.add_header('Date','{0}'.format(self.GetGMT()))
        request.add_header('Authorization','OSS {0}:{1}'.format(self.ak,self.
GetSignature()))
        request.get_method = lambda:'PUT'
        response = urllib2.urlopen(request,timeout=10)
        fd.close()
        self.ConsoleLog(response.code,response.headers)
    except Exception,e:
        self.ConsoleLog("error",e)

# output error log

def ConsoleLog(self,level=None,mess=None):

```

```

    if level == "error":
        sys.exit('{0}[ERROR:]{1}{2}'.format(self.left,self.right,mess))
    else:
        sys.exit('\nHTTP/1.1 {0} OK\n{1}'.format(level,mess))

if __name__ == "__main__":

    parser = OptionParser()
    parser.add_option("-i",dest="ak",help="Must fill in Accesskey")
    parser.add_option("-k",dest="sk",help="Must fill in AccessKeySecrety")
    parser.add_option("-e",dest="ed",help="Must fill in endpoint")
    parser.add_option("-b",dest="bk",help="Must fill in bucket")
    parser.add_option("-o",dest="objects",help="File name uploaded to oss")
    parser.add_option("-f",dest="fi",help="Must fill localfile path")

    (options, args) = parser.parse_args()
    handler = Main(options)
    handler.CheckParse()

```

请求头:

```

PUT /yuntest HTTP/1.1
Accept-Encoding: identity
Content-Length: 147
Connection: close
User-Agent: Python-urllib/2.7
Date: Sat, 22 Sep 2018 04:36:52 GMT
Host: yourBucket.oss-cn-shanghai.aliyuncs.com
Content-Type: application/x-www-form-urlencoded
Authorization: OSS B0g3mdt:1NCA4L0P43Ax

```

响应头:

```

HTTP/1.1 200 OK
Server: AliyunOSS
Date: Sat, 22 Sep 2018 04:36:52 GMT
Content-Length: 0
Connection: close
x-oss-request-id: 5BA5C6E4059A3C2F
ETag: "D0CAA153941AAA1CBDA38AF"
x-oss-hash-crc64ecma: 8478734191999037841
Content-MD5: 0MqhU5QbIp3Ujqghy9o4rw==
x-oss-server-time: 15

```

注意：

- Signature 中所有加入计算的参数都要放在 header 中，保持 header 和 Signature 一致。
- PUT 上传时，Signature 计算的 Content-Type 必须是 application/x-www-form-urlencoded。
- 通过 header 方式进行签名认证时无法设置过期时间。目前只有 SDK、URL 签名支持设置过期时间。

使用问题**案例：通过微信小程序请求 OSS 返回签名失败，通过浏览器正常****分析：**

- 只要通过浏览器访问，鉴权通过就证明 OSS 的签名校验是正常的没有问题，可以先排除掉 OSS 端。
- 客户端一定要在微信小程序上部署 HTTP 抓包，对后续分析很重要，抓包中可以看到所有的请求头和请求参数。
- 通过浏览器访问时的 HTTP 抓包。



```

GET /1530903286-0OSSAccessKeyId=...&Expires=1540915205&Signature=XDdcDn8%zi... HTTP/1.1
Accept: */*
C-Seq: 123
Cache-Control: no-cache
Connection: Keep-Alive
Content-Type: application/octet-stream
Host: ...
Range: bytes=0-262143
User-Agent: MicroMessenger Client

HTTP/1.1 403 Forbidden
Server: Tengine
Content-Type: application/xml
Content-Length: 785
Connection: keep-alive
Date: Fri, 12 Oct 2018 09:47:19 GMT
x-oss-request-id: 5BC06DA7BE...
x-oss-server-time: 0
Via: cache2.l2cm10-1[27,403-1280,M], cache27.l2cm10-1[29,0], cache2.cn1453[239,403-1280,M], cache8.cn1453[243,0]
X-Swift-Error: orig response 40X error
Ali-Swift-Global-SaveTime: 1539337639
X-Cache: MISS TCP_MISS dirn:-2:-2
X-Swift-SaveTime: Fri, 12 Oct 2018 09:47:19 GMT
X-Swift-CacheTime: 0
X-Swift-Error: orig response 40X error
Timing-Allow-Origin: *
EagleId: 7488861c15393376395951725e
  
```

- 通过 403 和 200 的抓包反复对比发现，通过小程序发出的 HTTP 请求和浏览器发起的 HTTP 请求的 URL、signature、expires 都一样，唯一的区别就是微信小程序携带了 Content-type，而通过 Chrom 的请求是没有携带 Content-type，怀疑矛头指向了这里。
- 经过代码确认，发现 signature 计算时是没有包含 Content-type 头的，而小程序发起的请求携带的 Content-type，OSS 收到后会按照携带了 Content-type 去计算 signature，所以每次计算都不一样。

遇到类似问题，抓包是最能快速看到问题的。同时也必须要了解下 OSS 请求 header 中携带了 Content-type，那么 signature 计算就要加上 Content-type，保持一致。

案例：通过多个语言版本 OSS SDK 测试，在使用 CDN 结合 OSS 用法时，客户端使用 CDN 域名计算 signature，发起 HEAD 请求，OSS 收到后返回 403。

```

Loaded assembly: Microsoft.GeneratedCode [External]
System.Net.WebException: The remote server returned an error: (403) Forbidden.
   at Aliyun.OSS.Common.Communication.ServiceResponse.EnsureSuccessful() [0x00021] in <d8db38433d62487181c
   at Aliyun.OSS.Common.Handlers.ErrorResponseHandler.ErrorHandle (Aliyun.OSS.Common.Communication.ServiceR
   at Aliyun.OSS.Common.Handlers.ErrorResponseHandler.Handle (Aliyun.OSS.Common.Communication.ServiceRespon
   at Aliyun.OSS.Common.Communication.ServiceClient.HandleResponse (Aliyun.OSS.Common.Communication.Service
   at Aliyun.OSS.Common.Communication.ServiceClient.Send (Aliyun.OSS.Common.Communication.ServiceRequest re
   at Aliyun.OSS.Common.Communication.RetryableServiceClient.SendImpl (Aliyun.OSS.Common.Communication.Serv
   at Aliyun.OSS.Commands.OssCommand.Execute () [0x00014] in <d8db38433d62487181cd2ba5c760defd>;0
   at Aliyun.OSS.Commands.OssCommand`1[T].Execute () [0x00000] in <d8db38433d62487181cd2ba5c760defd>;0
   at Aliyun.OSS.OssClient.GetObjectMetadata (System.String bucketName, System.String key) [0x0001f] in <d8
   at Aliyun.OSS.Samples.Program.Main (System.String[] args) [0x00023] in <774ebbbef5443700aa5fc54070467a5

```

分析：

出现这个问题不区分什么 SDK 都会出现，问题原因是由于客户端发起的 HEAD 请求在通过 CDN 回原到 OSS 时，CDN 回原是用 GET 请求，而 OSS 收到时就用 GET 请求方式去计算签名，得到的结果肯定和客户端计算不一致，可以升级到阿里云 CDN 处理。以上分析只适合上述场景。

问题可以通过 tcpdump 抓包或者 Wireshark 对比一下即可知道。

OSS crc64 出现不一致分析

客户端的循环冗余校验和服务端不一致，可以上传成功

```
[问题反馈]
OSS服务，报这个问题
[RequestId]: 
[ClientChecksum]: -2352902568697244055
[ServerChecksum]: -901841434708813993
可能原因是什么
```

排查

出现类似问题，需要先收集信息，经常发生在 Java sdk 上传的过程中。

- requestID 出现 checksum 不一致的情况一定会伴随有 requestID 信息。
- 上传方式是通过 SDK 还是自己调 API 上传，需要明确 SDK 的版本信息。
- 问题能否稳定复现，能复现最好提供网络抓包便于问题分析。
- tcpdump -i < 出口网卡 > -s0 host -w checksum.pcap

分析：

- 收集到 requestID 后，先升级到阿里云确认是否已经上传成功，一般 checksum 不一致并不会导致上传失败，checksum 是对文件进行循环冗余计算的方法，和 MD5 不一样，如果 MD5 不一致，文件上传一定失败。如果日志记录上传成功，可以校验文件的 MD5 是否一致，如果一致的话请忽略 checksum 不一致的情况。
- 出现 checksum 不一致的情况和网络以及系统主机都有关系，先通过网络抓包分析下，客户端发起 request header 中的 crc64 是否和 requestID 查到

的 crc64，如果一致的话说明用户上传网络传输中被篡改了，如果不一致的话说明客户本地写入到内存后就出现问题。

- 如果用户本地计算写入到内存和服务端的不一致，需要用户在本地代码中进行 debug，在上传请求发起前，先打印出 crc64 记录到本地，看看与 SDK 计算的是否一致，确保文件在上传前不要有其他进程在占用，或者读写。如果本地计算的 crc64 和 SDK 一致 说明不是计算问题，而是写入到内存后出现了问题，说明是客户端的系统导致的异常。

建议：

- 如果出现 checksum 不一致，但是文件 MD5 一致的话客户可以将 crc64 关掉，替换成 MD5 的校验方式，在 header 增加 Content-MD5 头，MD5 值要全部大写，然后在 base64 编码。
- Java SDK 关闭 crc64 的方法。

关闭 crc 的方法如下。

https://help.aliyun.com/document_detail/32010.html?spm=a2c4g.11186623.6.702.5bb84b4eKVgJfk

```
String endpoint = "<yourEndpoint>";
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";
ClientConfiguration conf = new ClientConfiguration();
conf.setCrcCheckEnabled(false);

// 创建 OSSClient 实例。
OSSClient ossClient = new OSSClient(endpoint, accessKeyId, accessKeySecret,
conf);
```

- 使用 https 协议上传，将 endpoint 协议头改为 https 即可，看下更换为 https 协议后问题是否还能复现。

OSS 上传文件异常分析

案例一：

OSSBrowser “no space left on”



分析：

有明显报错信息的先 Google 一下，看看是操作系统错误还是 OSSBrowser 抛出，经过鉴定错误是 Linux 系统底层返回，说明当时系统的 OSSBrowser 所在的目录磁盘满了，开源的错误到处可以看到，自行解决即可。

案例二：

分片上传成功，下载 404。

分析：

如果分片上传成功，肯定不会出现访问 404 的情况，检查下。

- 上传成功后 OSS 有没有返回 requestID，如果没有返回证明是没有上传成功的。
- 客户端的代码是否判断分片上传成功就去下载文件是有问题的，分片上传成功不代表文件都上传完成。
- 分片上传应该在 complete 合并分片之前进行判断 httpstatu==200 & requestID != None 才是真正的上传成功。
- 如果文件已经合并完成，并且返回 requestID，但还是下载 404，需要升级阿里云进行处理。

案例三：

IO error

背景：

客户端上传 OSS 过程中出现 “uploadFile:fail Write error: ssl=0x76d62a40: I/O error during system call, Software caused connection abort ”

分析：

这个是标准的 android 系统的报错，不是 oss 的问题，有很多开源的错误都有描述，建议参考下。

[参考 1](#)

[参考 2](#)

案例四：

背景： Could not resolve host

2018/10/24 14:03:47 hlg_matting_PreF (pid=15693)(error): parseUrl2Image.hpp(186), read_element: image_url_service connect fail:

Could not resolve host: da.oss-cn-hangzhou-internal.aliyuncs.com, url: <https://da.oss-cn-hangzhou-internal.aliyuncs.com/matting/0/images/20181024-135354-f5ee.jpg>

上传 OSS 文件解析失败，DNS 无法解析于域名，此问题需要排查下 DNS 是否工作正常，或者主机上的 DNS 服务是否正常。

案例五：

背景：python SDK 分片上传失败

用户通过 python SDK 的分片上传函数上传到 OSS 失败，碎片管理中出现很对碎片。

<input type="checkbox"/>	文件名	上传 ID	文件碎片
<input type="checkbox"/>	accesslog_2018_09.tar.gz	E609B68C0F0647BF902DF 6032902BBD5	点击统计
<input type="checkbox"/>	accesslog_2018_10.tar.gz	EA9E2EDFBA00450688127 E17D2AED4CA	点击统计

云栖社区 yq.aliyun.com

- 先确认是直接传到 OSS，还是通过其他 proxy 传输到 OSS (类似 CDN)，如果经过 CDN 再上传到 OSS 需要在 OSS 上配置跨域的头，Access-Control-Allow-Origin、Access-Control-Allow-Method、Access-Control-Allow-header，并且将 Etag 暴露出去。
- 客户端上传失败是因为网络超时，还是捕获到异常上传失败，需要详细看下捕获到的 SDK 异常信息分析，如果是网络超时导致上传失败，建议使用断点续传来替代普通上传。断点续传支持分片，并发，已经弱网的兼容。
- 清理掉上传失败的碎片文件重新上传。
- 当以上操作都解决不了你的问题时，需要提供以下信息升级阿里云便于快速定位。
 - 提供 SDK 异常时返回的 requestID，这个属性是 response header 中携

带的记录了完整的 OSS 请求过程。

- 客户端部署 tcpdump，然后重新运行代码上传，保存抓包。

```
tcpdump -i <网卡出口名称> -s0 host <访问 oss 的域名> -w  
faild.pcap
```

案例六：

C# SDK 分片上传报错

```
错误    2019/2/13 10:24:28    ServerApi    0    无    "Failed to stop service.  
System.InvalidOperationException: An unhandled exception was detected --->  
System.IO.IOException: Unable to read data from the transport connection: The  
connection was closed.  
    at Aliyun.OSS.Util.AsyncResult.WaitForCompletion()  
    at Aliyun.OSS.Common.Communication.RetryableServiceClient.  
EndSend(IAsyncResult ar)  
    at Aliyun.OSS.Util.OssUtils.EndOperationHelper[TResult](IServiceClient  
serviceClient, IAsyncResult asyncResult)  
    at SAAS.Common.Aliyun.MultipartUploadSample.UploadPartCallback(IAsyncResult  
ar)  
    at Aliyun.OSS.Common.Communication.ServiceClientImpl.<c__  
DisplayClass5.<BeginSetRequestContent>b__4(IAsyncResult ar)  
    at System.Net.LazyAsyncResult.Complete(IntPtr userToken)  
    at System.Threading.ExecutionContext.RunInternal(ExecutionContext  
executionContext, ContextCallback callback, Object state, Boolean  
preserveSyncCtx)  
    at System.Threading.ExecutionContext.Run(ExecutionContext  
executionContext, ContextCallback callback, Object state, Boolean  
preserveSyncCtx)  
    at System.Threadin..."
```

排查

- 出现类似报错是因为客户设置的分片过小导致，先问题出用的是分片上传 还是断点续传。
- 如果是分片上传，根据文件总大小合理的提升分片 size，比如客户原来设置的是 50M，现在可以提升到 100M 测试。
- 另外请使用断点续传重新替换下分片上传，此案例在分析过程中通过客户端的抓包发现大量的丢包重传，这种情况下是普通的分片可能直接报错，不会再进

行重传了。

No.	Time	Source	Destination	Protocol	Length	Info
1928742	12:39:32.1	0.000193800	0.000001800	TCP	60	61 80 -> 51811 [ACK] Seq=26 Ack=405948128 Win=1587840 Len=0
1928743	12:39:32.1	0.000141800	0.000001800	TCP	60	61 80 -> 51811 [ACK] Seq=26 Ack=405948128 Win=1587840 Len=0
1928744	12:39:32.1	0.000445800	0.000001800	TCP	54	64 51872 -> 80 [RST, ACK] Seq=7921183 Ack=26 Win=0 Len=0
1928745	12:39:32.1	0.000141800	0.000001800	TCP	60	61 [TCP Dup ACK 1908757#1] 80 -> 51872 [ACK] Seq=26 Ack=790881 Win=0 Len=0
1928746	12:39:32.1	0.000054000	0.000001800	TCP	54	64 51878 -> 80 [RST, ACK] Seq=8077600 Ack=26 Win=0 Len=0
1928747	12:39:32.1	0.000054000	0.000001800	TCP	54	64 51881 -> 80 [RST, ACK] Seq=19991851 Ack=26 Win=0 Len=0
1928748	12:39:32.1	0.000071800	0.000001800	TCP	60	61 [TCP Dup ACK 1908185#1] 80 -> 51879 [ACK] Seq=26 Ack=8883353 Win=0 Len=0
1928749	12:39:32.1	0.000020000	0.000001800	TCP	54	64 51879 -> 80 [RST, ACK] Seq=1379558 Ack=26 Win=0 Len=0
1928750	12:39:32.1	0.000027000	0.000001800	TCP	60	61 [TCP Dup ACK 182255#1] 80 -> 51881 [ACK] Seq=26 Ack=19978745 Win=0 Len=0
1928751	12:39:32.1	0.000038000	0.000001800	TCP	54	64 51878 -> 80 [RST, ACK] Seq=7551181 Ack=26 Win=0 Len=0
1928752	12:39:32.1	0.000058000	0.000001800	TCP	60	61 [TCP Dup ACK 1928788#1] 80 -> 51873 [ACK] Seq=26 Ack=13784457 Win=0 Len=0
1928753	12:39:32.1	0.000054000	0.000001800	TCP	54	64 51878 -> 80 [RST, ACK] Seq=1838381 Ack=26 Win=0 Len=0
1928754	12:39:32.1	0.000064000	0.000001800	TCP	54	64 51882 -> 80 [RST, ACK] Seq=1356241 Ack=26 Win=0 Len=0
1928755	12:39:32.1	0.000061800	0.000001800	TCP	54	64 51877 -> 80 [RST, ACK] Seq=9899841 Ack=26 Win=0 Len=0
1928756	12:39:32.1	0.000011800	0.000001800	TCP	60	61 [TCP Dup ACK 1928653#1] 80 -> 51876 [ACK] Seq=26 Ack=18276889 Win=0 Len=0
1928757	12:39:32.1	0.000018000	0.000001800	TCP	60	61 [TCP Dup ACK 1908734#1] 80 -> 51876 [ACK] Seq=26 Ack=1446641 Win=0 Len=0
1928758	12:39:32.1	0.000018000	0.000001800	TCP	60	61 [TCP Dup ACK 1908734#1] 80 -> 51876 [ACK] Seq=26 Ack=1446641 Win=0 Len=0

案例七：

OSS 上传后，但显示长度为 0

排查

1) 首先排查下客户端上传是用什么方式 (SDK、API、工具) 不同的方法可能使用也是不同的。确认好使用 SDK 我们使用客户端的原文件进行上传测试，看问题是否可以复现。

- 可以复现，说明我们自己可以进行排查无需用户配合。
- 我们自己复现不了说明问题是客户个案，肯定不是工具或者产品的问题。

2) 获取当前问题发生的 OSS requestID。

- 如果是工具，可以通过增加 log 的方式来获取，比如 ossutil 有时候会在控制台显示，也可以通过当前目录下的隐藏的操作日志来确认。
- 如果是 SDK，客户可以通过代码返回的结果对象中获取到这个属性，比如 java SDK。

3) 当获取到 requestID 查询到日志后，先看下用户上传的结果是 200 还是异常的，如果是 200 说明上传成功；其次看下用户写入的长度是多大？如果客户写入的就是 0 字节，那么和 MD5 没有任何关系，OSS 除了分片、断点续传方法对第一片的大小有限制 (100KB) 外，其余的普通上传方法均没有对文件最小值做限制，也就是用户可以上传一个空文件。

OSS RequestTimeTooSkewed 问题分析

RequestTimeTooSkewed

- 经常遇到，但是原因比较多，分析难以下手，具体的表象可以看下面的截图，由于客户端（下文称之为 client）发出的请求时间和实际上服务端（下文称之为 oss）收到的时间差大雨 15min 导致（oss Time - client Time > 15min）。

```
021[INFO ] 2018-08-07 17:21:00 080 | [ ] | [schedulerFactory_Worker-20] cn.damai.mz.server.boss
021[ERROR] 2018-08-07 17:21:58 619 | [[traceID:647255405159911424]] | [catalina-exec-15] cn.dama
ion: The difference between the request time and the current time is too large.
[ErrorCode]: RequestTimeTooSkewed
[RequestId]: 5B6964B6338A4801BF748DAF
[HostId]: oss-cn-hangzhou.aliyuncs.com
[ResponseError]:
<?xml version="1.0" encoding="UTF-8"?>
<Error>
  <Code>RequestTimeTooSkewed</Code>
  <Message>The difference between the request time and the current time is too large.</Message>
  <RequestId>5B6964B6338A4801BF748DAF</RequestId>
  <HostId>oss-cn-hangzhou.aliyuncs.com</HostId>
  <MaxAllowedSkewMilliseconds>900000</MaxAllowedSkewMilliseconds>
  <RequestTime>2018-08-07T08:49:41.000Z</RequestTime>
  <ServerTime>2018-08-07T09:21:58.000Z</ServerTime>
</Error>
```

云栖社区 yq.aliyun.com

时间标准

- 先排除掉最简单的问题，确认时间是否为标准的 UTC、GMT、CST 时间，如果时区不是东八区，只要换算成 +8 小时一致即可。有的人可能使用自己的 NTP 时钟同步出现异常，导致 client 和 oss 收到时间相差 15min。

```
hanlideMacBook-Air:~ hanli$ date
2018年10月24日 星期三 00时34分14秒 CST
hanlideMacBook-Air:~ hanli$
```

排查代码

- 如果是用阿里云的 OSS SDK 的话，先检查下 OSS SDK 初始化链接数是不是大，client 的并非请求是否已经超过了 SDK 的设置。

以下用 JAVA SDK 为例子默认的 maxconnect 是 1024。在本机执行 netstat 命令看下 client 程序对应的 TCP 链接数有没有超过 1024。

排查主机问题

- 使用 netstat 命令看下主机的 TCP 链接数 (UDP TCP) 有没有超过 ulimit 的设置。
- 查看主机出口的网络带宽有没有被打满。

是有经过网络代理

- client 是直传到 OSS，还是经过 proxy 传输到 OSS，如果有代理先要排查 client 到 proxy 链路是否有抖动丢包重传，以及 proxy 到 OSS 的链路。
- proxy 如果链接数或者带宽被打满都会造成上传延迟、拥堵，导致 RequestTimeTooSkewed。

排查网络问题

如果使用阿里云的 ECS 建议走内网的 internal 形式的域名操作 OSS 这个是阿里云内部网络，性能很稳定速度也很快，如果走公网的话并不是很可靠。

走公网的情况就需要做测试了。

- 如果 client 到 OSS 是走公网上传下载，发生 RequestTimeTooSkewed 问题时，可以同步 ping -c 50 -i 0.01 -s 1024 通过 ping 可以发现抖动和丢包。

```

/24*339 tes from .219.206: icmp_seq=18 ttl=93 time=2.980 ms
Request timeout for icmp seq 19
1032 bytes from .219.206: icmp_seq=19 ttl=93 time=11.906 ms
1032 bytes from .219.206: icmp_seq=20 ttl=93 time=3.108 ms
1032 bytes from .219.206: icmp_seq=21 ttl=93 time=2.578 ms
1032 bytes from .219.206: icmp_seq=22 ttl=93 time=3.298 ms
1032 bytes from .219.206: icmp_seq=23 ttl=93 time=2.666 ms
1032 bytes from .219.206: icmp_seq=24 ttl=93 time=2.648 ms
1032 bytes from .219.206: icmp_seq=25 ttl=93 time=2.458 ms
1032 bytes from .219.206: icmp_seq=26 ttl=93 time=2.639 ms
1032 bytes from .219.206: icmp_seq=27 ttl=93 time=2.814 ms
1032 bytes from .219.206: icmp_seq=28 ttl=93 time=2.690 ms
1032 bytes from .219.206: icmp_seq=29 ttl=93 time=2.610 ms
1032 bytes from .219.206: icmp_seq=30 ttl=93 time=2.682 ms
1032 bytes from .219.206: icmp_seq=31 ttl=93 time=2.716 ms

```

- traceroute <OSS 域名> 看下每一跳的延迟。
- mtr <OSS 域名> 看下公网链路是否有丢包。
- 当时异常方法都查不到原因只能使用终极办法 tcpdump 、或者 Wireshark 抓包。

tcpdump -i <出口网卡> -s0 host <OSS 域名> -w slow_packet.pcap。

OSS 访问文件 404 分析

什么是 404

404 标准的 http code 状态码，代表用户请求的资源在服务端不存在，404 并不是一个异常状态码？而是一个正常的响应。换句话说 404 已经成为了一个结果，这种响应常见在 client 端下载 OSS 的资源时出现。

做好预防

很多情况如果服务端的历史日志保存时间有限，那么出现问题时也无法查证，所以建议使用 OSS 先把日志功能开启，将 OSS 每天的数据都自动备份到用户指定的目录下，费用低廉。



why 404

- 检查 客户端提供的 objectname 是否正确，确保不存在低级的拼写错误。
- 客户端之前上传是否成功，如果成功 OSS 会返回 requestid，凡事没有 requestID 的并不能保证已经上传到 OSS 成功。

客户端可以根据开启的 log 查询下对应时间点 OSS 出现 404 的情况。

- OSS 是否开启过 生命周期 功能，开启这个功能，触发生命周期管理规则后，会将满足条件的 URL 完全上除掉。

客户端上传成功，但是下载返回 404

首先这种情况是不存在，如果已经上传到 OSS 的文件，除了会冗余写多份，也受到权限的严格限制，匿名非法的访问是不可能直接删除的，除非使用者自己把 bucket 设置为公共读写（高危权限）一般对上传成功有个误区，认为返回 200 就代表文件已经上传成功其实这并不准确。会造成两种情况。

- 收到状态码 200 的情况后立刻取请求 OSS，返回了 404，等一会再访问就 200，怀疑到 OSS 返回 200 和真实的写入成功不同步。
- 上传成功收到 200 状态码，但访问一直都是 404
- 分片上传或者断点续传已经收到了 200 并切有 requestID 的情况下，访问 OSS 仍然返回 404。

其实以上两种情况都是对返回状态的判断不准确导致，可靠的判断方式是，if result.status==200 && result.requestID != None 的情况下才是上传成功。有的 http 请求被 http 劫持的情况下会收到一个伪造的 200 状态码并不是真实的返回。

另外分片上传或者断点续传比叫特殊，是采用将原文件切片的形式上传，每一个分片 multipart 上传成功都会返回 200 和 requestID，这种情况应该以 complete 合并分片后返回的 200 & requestID 为准。

极特殊的情况是客户端使用分片上传时对用的 uploadID 凭证是 A，但是完成合并时用的 uploadID 凭证是 B，OSS 找不到原始的 A 凭证，所以报 404 无法完成上传。

```
FileNotFoundError: [Errno 2] No such file or directory: 'C:\Program Files\Python\Python37\python.exe'
Traceback (most recent call last):
  File "C:\Program Files\Python\Python37\python.exe", line 2, in <module>
    raise exceptions.NoSuchUpload: {'status': 404, 'request-id': '5A17E992C03048E256CC571', 'details': {'HostId': 'oss-cn-beijing-internal.aliyuncs.com', 'Message': 'The specified upload does not exist. The upload ID may be invalid, or the upload may have been canceled.', 'upload-id': '200x331294096cmj440cc4d0', 'RequestId': '5A17E992C03048E256CC571'}}
FileNotFoundError: [Errno 2] No such file or directory: 'C:\Program Files\Python\Python37\python.exe'
root@jumpserver:~# vim ../bin/backup_mysql.py
```

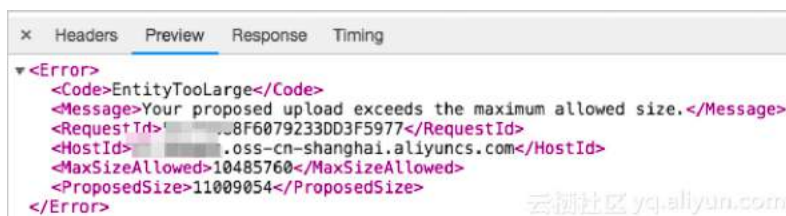
文件之前一直存在，突然访问 404

有以下几个原因：

- 客户端的子账号有权限，把 OSS 的文件删除了。
- 具备客户合法的 AK SK 通过 API 删除了。
- 具备合法账号密码的人在控制台删除了。
- 拥有者设置过生命周期过期被删除了。
- bucket 和其他 bucket 有跨区复制的关系，其他的 bucket 执行了删除操作，同步到当前的 bucket，所以文件被删除。

OSS Postobject 问题分析

EntityTooLarge



出现这种问题基本是客户端的 post 代码中限制的请求的 body 大小，排查这种问题最好是通过抓包，或者构造 post 源码上看下。当你的 bucket 不是 public-read-write 时，需要客户端的 post 代码中提供 post policy，这这属性中，有关于 content-length-range 的设置，不注意的话这个地方基础设置都是 10M，建议检查一下这个地方。



Post policy中必须包含expiration和conditions。

Expiration

Expiration项指定了policy的过期时间，以ISO8601 GMT时间表示。例如"2014-12-01T12:00:00.000Z"指定了Post请求必须发生在2014年12月1日12点之前。

Conditions

Conditions是一个列表，可以用于指定Post请求的表单域的合法值。注意：表单域对应的值在检查policy之后进行扩展，因此，policy中设置的表单域的合法值应当对应于扩展之前的表单域的值。Policy中支持的conditions项见下表：

名称	描述
content-length-range	上传文件的最小和最大允许大小。该condition支持content-length-range匹配方式。

PostObject 没有发生回调

使用 OSS PostObject 的 callback 没有发生回调。但是通过 putobject 用同样的 callback 是有回调发生的，怀疑 PostObject 有问题？



分别测试 PUT 和 POST 效果如下。

POST 代码

PUT 代码

The image displays a Wireshark packet capture of a POST request. The packet list on the left shows a POST request from 192.168.1.100 to 192.168.1.100 on port 80. The packet details on the right show the request body with a Content-Type of application/x-www-form-urlencoded. The packet bytes on the bottom show the raw data of the request.


```

eYJleH8pcnF9ak9UjJogTjIxtAjAtMDEtMDk0NTkiNDQ6MDAwMiiSiImHybnRpdGlvbnN0I0BbllyJ3BzZS0KXz0LXklmd0w0c1yYk5nZSIzIDAaIEEwWdGjHicYwDWMfIdfQ==
--9431149156168
Content-Disposition: form-data; name="Signature"

gncf07AgdeByqa%#@tol8Xxzo#-
--9431149156168
Content-Disposition: form-data; name="callback"

eyJ3YlksYm9jaFYybCIGInhdIHR46Ly80My45Hy4MTYwMTY4L1JldmlzZS5hc2h4IiwY2FsbG9hZ2Y2c2b2R5SiJoieIwiYnVja2V0XCIXJHlidmRrZXRP9LfwiZc216ZVudPSR7c216ZXI9IHR=
--9431149156168
Content-Disposition: form-data; name="file"; filename="1.txt"
Content-Type: text/plain

12345678910
--9431149156168--
HTTP/1.1 200 OK
Server: AliyunOSS
Date: Mon, 12 Feb 2018 06:39:22 GMT
Content-Type: application/json

```

结果:

业务服务器上也如愿的抓到了请求。

[illegible]

OSS url 编码问题分析

案例：经过 url encode 编码访问失败

编码前: <http://oss-cn-hangzhou.aliyuncs.com/fun-punch-hls/ji-test/c133249354654050a66ec4341e61c23f?Expires=1540451197&OSSAccessKeyId=aasn8khCBdFe&Signature=HeYJhDzmjSOfZyQM%3D>

编码后: <http://oss-cn-hangzhou.aliyuncs.com/fun-punch-hls/ji-test/c133249354654050a66ec4341e61c23f?Expires%3d1540451197%26OSSAccessKeyId%3daasn8khCBdFe%26Signature%3dHeYJhDzmjSOfZyQM%3d>

- 在浏览器请求 oss 资源，不需要将整个查询参数进行编码，只需要对计算出来 signature 进行编码即可。

URL签名，必须至少包含signature、Expires和OSSAccessKeyId三个参数。

- Expires** 这个参数的值是一个UNIX时间（自UTC时间1970年1月1号开始的秒数，详见Wikipedia），用于标识该URL的超时时间。如果OSS接收到这个URL请求的时候晚于签名中包含的Expires参数时，则返回请求超时的错误码。例如：当前时间是1141889060，开发者希望创建一个60秒后自动失效的URL，则可以设置Expires时间为1141889120。
- OSSAccessKeyId** 即密钥中的AccessKeyId。
- Signature** 表示签名信息。所有的OSS支持的请求和各种Header参数，在URL中进行签名的算法和在Header中包含签名的算法基本一样。

```
Signature = urlencode(base64(hmac-sha1(AccessKeySecret,
  VERB + "\n"
  + CONTENT-MD5 + "\n"
  + CONTENT-TYPE + "\n"
  + EXPIRES + "\n"
  + CanonicalizedOSSHeaders
  + CanonicalizedResource)))
```

其中，与header中包含签名相比主要区别如下：

- 通过URL包含签名时，之前的Date参数换成Expires参数。
- 不支持同时在URL和Header中包含签名。
- 如果传入的Signature、Expires、OSSAccessKeyId出现不止一次，以第一次为准。
- 请求先验证请求时间是否晚于Expires时间，然后再验证签名。
- 将签名字符串放到url时，注意要对url进行urlencode
- 临时用户URL签名时，需要携带 security-token，格式如下：

```
xpires=1141889120&signature=vjbyPxybdZaNmCa%2ByT272YEAiv4%3D&security-token=SecurityToken
```


案例：URL 特殊字符访问失败

原始 URL

<http://testsanmao.oss-cn-beijing.aliyuncs.com/> 北师大版九上物理第十五章 %20 怎样传递信息——通信技术简介第 2 节《广播和电视》参考教案 + 习题 %20 北师大版九上物理第十五章 %20 怎样传递信息——通信技术简介第 2 节《广播和电视》当堂练习 .doc。

分析：

URL 中的 + 要经过 URL encode，如果不经过编码，+ 会被转义成空格。

OSS 结合访问控制

企业上云的安全威胁排名

- | | |
|--------------------|---------------|
| 1. 数据泄露 | 7. APTs |
| 2. AK/密码泄露/缺少IAM实践 | 8. 数据丢失 |
| 3. API接口安全 | 9. 云技术了解不充分 |
| 4. 系统和应用软件漏洞 | 10. 恶意利用云服务 |
| 5. 账号共享/劫持问题 | 11. 拒绝服务(DoS) |
| 6. 内部攻击(员工/外包/ISV) | 12. 技术模块共享问题 |

part1 – 云账号及其安全

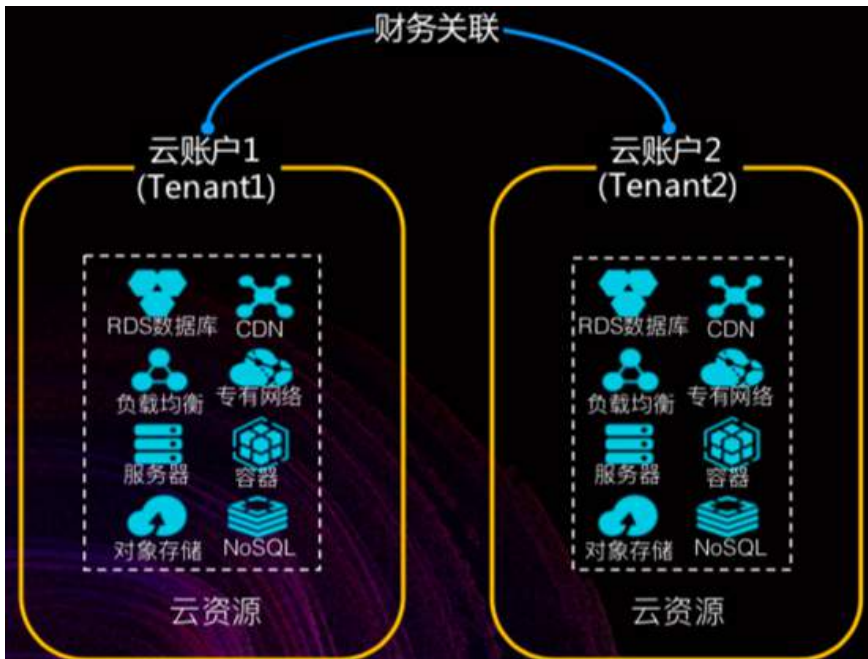
云平台上多租户隔离的基本主体

- 阿里云现在所有云产品彼此之前都是隔离，主账号和 RAM 子账号也都是隔离的，彼此不能互访。
- 同一个主账号 UID 下，不同的云产品默认不能互访，需要在主账号授权的产品授权的权限下才能访问。
- 未经过主账号授权的情况下，其他主账号 uid 是不能访问用户自己的云产品和控制台。



认识云账户

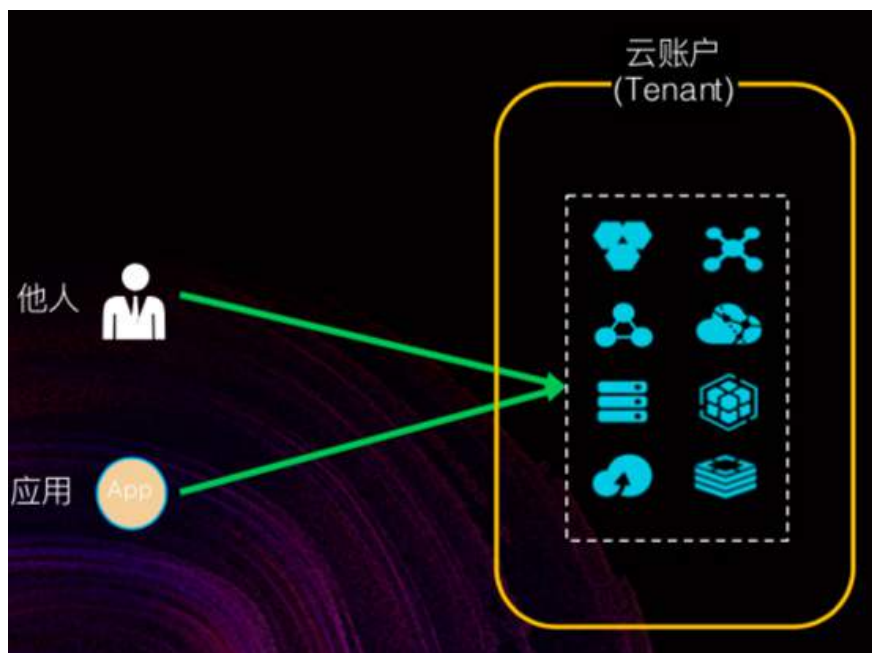
- 统一出票
- 统一出账
- 共享信誉额度



云账户安全

- 云账户安全就是要保护云资源以防止未经授权访问，即便是同个云账号下的不同产品也不能互访，除非云账号自身允许。
- 认识云账号凭证 (Credentials)。
 - 登陆密码验证 (password)
 - mfa 多因素验证
 - api 访问 (ak/sk)

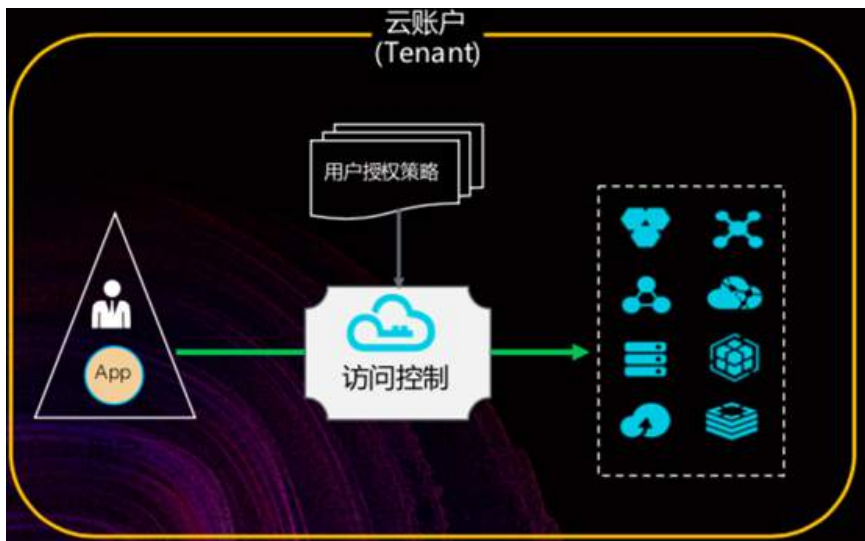
part2 – 用户身份管理与访问控制



谁是 user

1. 用户自己登陆。
2. 用户授权自己的 ram 子账号登陆。
3. 用户授权别人加的 ram 账号登陆。

RAM 核心功能



集中用户管理

- 所有的用户都可以在控制台上统一的可视化界面处理，统一的 api 接入。

谁是应用

- 比如阿里云提供的工具，类似 oss 的 brower。
- 客户的应用程序代码 (app、服务端程序) 通过 sts 或者 云账号的 ak/sk。
- ecs 的 meta 网关信息也可以操作 <https://www.alibabacloud.com/help/zh/doc-detail/54579.htm>。

阿里云 RAM 的特色

ABAC 模型: AttributeBasedAccessControl。这种是我们常用的自定义 policy 需要自己写控制语句 ABACvsACL: 就是我们常用系统策略，权限粒度小，都是一个管理权限或者只读、只写的权限。

	控制维度	适用场景	风控能力
ACL	低 (谁, 操作, 对象)	少 (只能描述自主访问控制模型)	弱 (完全依赖用户密码安全)
ABAC	高 (谁, 操作, 对象, 多维限制条件)	多 (能描述多种访问控制模型)	强 (可以不依赖用户密码安全)

一个实际的授权场景

```
{
  "Version": "1",
  "Statement": [ {
    "Effect": "Allow", "Action": "ecs:StopInstance", "Resource": "acs:ecs:cn-
hangzhou:*:*", "Condition": {
      "StringEquals": { "ecs:tag/env": "production"
    }, "Bool": {
      "acs:MFAPresent": "true" },
      "IpAddress": { "acs:SourceIp": "42.120.88.0/24"
    } }
  ]
}
```

以上策略意思是 针对 42.120.88.0, 允许操作 production 组内的 ecs 实例进行停止操作。为了方便我们给他分开三块看, 这样会比较清晰。

第一块: 固定的外层语法不变, 即使有多条策略也是在者一个 statement 内部, 用“,” 分开。

```
{
  "Version": "1"
  "Statement": [
    " 这里是第二块 "
  ]
}
```

第二块：我们简称三板斧，因为内容是固定的，只不过变化 value 而已。

- 三板斧就是 effect, action, resource, 这三个是一组，包含在一个 {} 内，第二条语句要用 “,” 隔开写在第二个 {} 内。
- effect: 只有 Allow 和 Deny。
- action : 可以写多条时要用 [] 包括，比如 [“acs:ecs:cn-hangzhou:1982222:instance/i-zxxxxesd”, “acs:oss:cn-beijing:1299:bucket/prefix/object”]。
- action : 填写的是你要限制对应的产品的 API 名称，写多个时要用 [] 包括主，比如 [“ecs:CreateInstance”, “ecs:StopInstance”]。
- product: 填写产品名称 slb、ecs 、oss、vpc 等。
- regionID: cn-shanghai、cn-hangzhou 等。
- uid: 云账号 uid。

```
{
  "Session": "1"
  "Statement": [
    {
      "Effect": "Allow / deny",
      "Resource": "acs:product:regionid:uid:*",
      "Action": "apiname"
    },
    {
      " 设置并行的第二条语句 "
    }
  ]
}
```

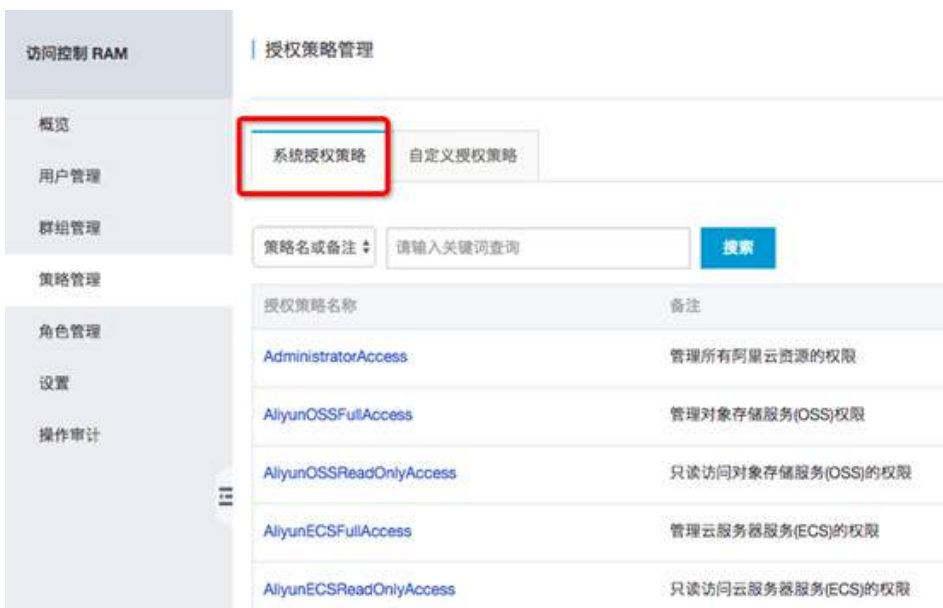
Part 3: 最佳实践

下面我实际操作如果新建 ram 子账号、授权策略。

- 新建账户



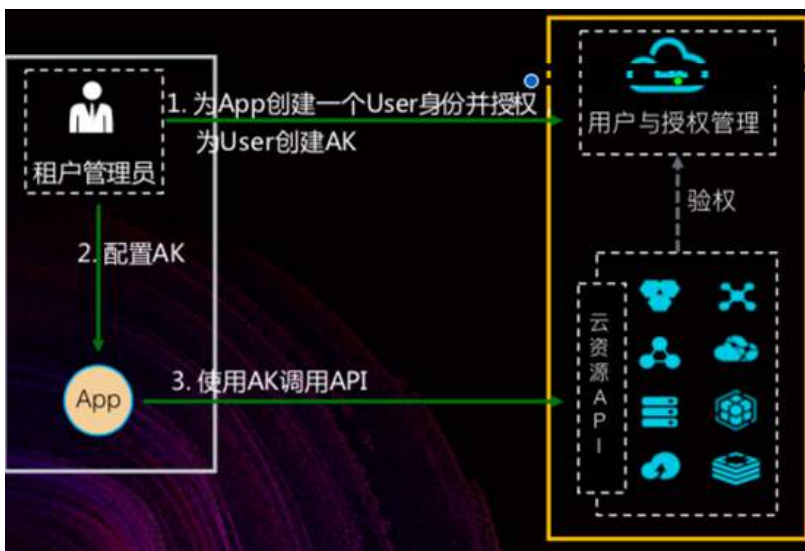
- 系统策略



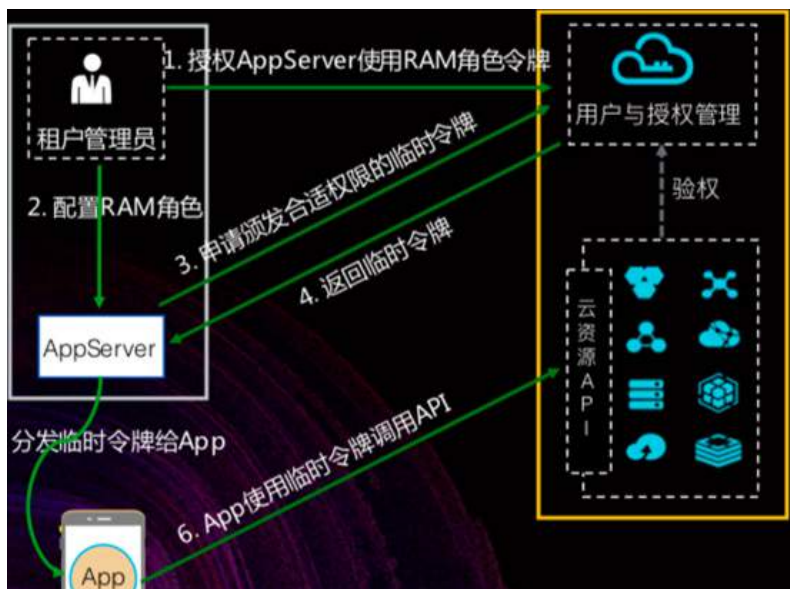
- 自定义策略



- 使用 RAM 子账号 ak sk



App 安全天使 STS



为什么说是安全天使

当前端上的 APP 不可能直接使用客户的 ak sk，风险性极高，一但恶意攻击者那到你 APP 数据包，揭秘出源码中的 ak sk，您的云产品将暴露，任何人都可以操作您子账号下所有授权的产品，即使删除 ak sk 也可能导致服务端的其他业务出现链接异常，由此 sts 应运而生。

临时、最小粒度、可控

- 临时: sts 的令牌有效期是 900-3600 秒，一但过期将失去效力。
- 最小粒度: sts 只能操作角色扮演了策略的对应产品，简单说就是，把用户想授权的各类云产品抽象出各种角色，给每个角色赋予不通的权限，ram 子账号扮演了哪种角色就可以有哪种权限，及时 sts 信息泄露，客户只要删除 sts 角色即可，或者将角色和 ram 子账号解绑，盗取者也没有用了，而且并不影响用户其他使用 ak sk 的服务端业务。
- 可控: 生成 sts 是放在用户自己的服务器上所以安全可用。

sts 创建、代码实践

由于 sts 也要新建 ram 子账号存在与 part3 重复的地方，所以建立 ram 子账号的位置我就不演示了。

- 新建角色，选择用户角色，当前账号，如果选择其他云账号是给其他云账号下的子账号授权访问自己的云产品，要区分概念。这里我们给自己的云账号授权，所以默认。



创建角色

1: 选择角色类型

2: 填写类型信息

3: 配置角色基本信息

4: 创建成功

选择受信云账号，受信云帐号将可以使用此角色来访问您的云资源。

选择云账号

当前云账号

其他云账号

* 受信云账号ID:

可以访问[账户管理](#)->[安全设置](#)获取账号ID。

上一步

下一步

- 给角色创建一条自定义策略或者系统策略都行。

访问控制 RAM

授权策略管理

创建授权策略

STEP 1: 选择授权策略类型

STEP 2: 编辑授权策略内容

STEP 3: 策略描述

* 授权策略名称:

长度为1-128个字符，允许英文字母、数字、短划线“-”

备注:

策略内容:

```
1 {
2   "Version": "1",
3   "Statement": [
4     {
5       "Effect": "allow",
6       "Action": "*",
7       "Resource": "*"
8     }
9   ]
10 }
```

授权策略格式定义

新建授权策略

授权策略名称

策略内容

策略描述

- 给角色绑定我们刚才自定义的 policy。



- 让 ram 有权调用角色，这样 ram 子账号就有了角色对应的产品策略，所以要
让角色和 ram 关联。



- 最后一部利用服务端的代码，填入我们刚才建立 ram、角色时得到的所有信息，就能生成 sts 令牌。
- rolearn 就是我们在创建角色时控制台看到的。
- rolesession 就是角色名称。



```
public class StsServiceSample {
    public static void main(String[] args) {
        String endpoint = "sts.aliyuncs.com";
        String accessKeyId = "<access-key-id>";
        String accessKeySecret = "<access-key-secret>";
        String roleArn = "<role-arn>";
        String roleSessionName = "session-name";
        String policy = "{\n" +
            "    \"Version\": \"1\", \n" +
```

```

        "    \"Statement\": [\n" +
        "        {\n" +
        "            \"Action\": [\n" +
        "                \"oss:*\"\n" +
        "            ], \n" +
        "            \"Resource\": [\n" +
        "                \"acs:oss:*:*:*\" \n" +
        "            ], \n" +
        "            \"Effect\": \"Allow\"\n" +
        "        }\n" +
        "    ]\n" +
        "};";

try {
    // 添加endpoint(直接使用STS endpoint, 前两个参数留空, 无需添加 region ID)
    DefaultProfile.addEndpoint("", "", "Sts", endpoint);
    // 构造 default profile(参数留空, 无需添加 region ID)
    IClientProfile profile = DefaultProfile.getProfile("",
accessKeyId, accessKeySecret);
    // 用 profile 构造 client
    DefaultAcsClient client = new DefaultAcsClient(profile);
    final AssumeRoleRequest request = new AssumeRoleRequest();
    request.setMethod(MethodType.POST);
    request.setRoleArn(roleArn);
    request.setRoleSessionName(roleSessionName);
    request.setPolicy(policy); // Optional
    final AssumeRoleResponse response = client.
getAcsResponse(request);
    System.out.println("Expiration: " + response.getCredentials().
getExpiration());
    System.out.println("Access Key Id: " + response.getCredentials().
getAccessKeyId());
    System.out.println("Access Key Secret: " + response.
getCredentials().getAccessKeySecret());
    System.out.println("Security Token: " + response.
getCredentials().getSecurityToken());
    System.out.println("RequestId: " + response.getRequestId());
} catch (ClientException e) {
    System.out.println("Failed: ");
    System.out.println("Error code: " + e.getErrCode());
    System.out.println("Error message: " + e.getErrMsg());
    System.out.println("RequestId: " + e.getRequestId());
}
}
}
}

```

OSS 性能调优

OSS python SDK 调优

初始化

在初始化时 python 有两个地方可以做调整。

connect_timeout

可以增大客户端在数据读写过程中的超时时间，常用在客户端到 OSS 公网情况下上传大文件时增长时间，防止在公网抖动或者丢包情况下出现传输超时。

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 账号进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint 以杭州为例，其它 Region 请按实际情况填写。
endpoint = 'https://oss-cn-hangzhou.aliyuncs.com'

# 设置连接超时时间为 30 秒。
bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>', connect_timeout=30)
```

防劫持

OSS 服务端默认支持了泛域名的证书，客户可以用 https 协议传输方式出现被劫持的情况。

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 账号进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint 以杭州为例，其它 Region 请根据实际情况填写。将 http 改为 https 就是走 https 加密
endpoint = 'https://oss-cn-hangzhou.aliyuncs.com'

bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>')
```

性能调优

enable_crc

crc 的作用是校验客户端的文件完整性，文件越大对 cpu 计算消耗越高，上传时间成本越高，所以一般都建议用户端，关闭 crc 提高传输效率，使用 Content-Md5 的方式替代 crc64。

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 账号进行 API 访问或日常运维，请登录 RAM 控制台创建 RAM 账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint 以杭州为例，其它 Region 请根据实际情况填写。
endpoint = 'https://oss-cn-hangzhou.aliyuncs.com'

bucket = oss2.Bucket(auth, endpoint, '<yourBucketName>', enable_crc=False)
```

日志开关

OSS 的日志采用 logging 模块采集。通过 ossclient 初始化时 set logging 的开关和存放位置；但是记录日志的同时会影响 SDK 上传性能，一般不建议用户开启，排查问题时采用到；平常客户只要记录 requestID 或者 OSS 返回的 header 头就足够了。

```

# -*- coding: utf-8 -*-

import os
import logging
import oss2
from itertools import islice

# 初始化 AccessKeyId、AccessKeySecret、Endpoint 等信息。
# 请将 <AccessKeyId>、<AccessKeySecret>、<Bucket> 及 <访问域名> 分别替换成对应的具体
# 信息。
access_key_id = '<AccessKeyId>'
access_key_secret = '<AccessKeySecret>'
bucket_name = '<Bucket>'
endpoint = '<访问域名>'

log_file_path = "log.log"
# 开启日志
oss2.set_file_logger(log_file_path, 'oss2', logging.INFO)

# 创建 Bucket 对象。
bucket = oss2.Bucket(oss2.Auth(access_key_id, access_key_secret), endpoint,
bucket_name)

# 遍历文件目录
for b in islice(oss2.ObjectIterator(bucket), 10):
    print(b.key)
# 获取文件元信息
object_meta = bucket.get_object_meta('object')

```

分片上传下载

python SDK 在 分片的基础上增加了断点传输，既可以在分片基础上增加并发，也支持断点记录的功能，在遇到网络问题断开时不需要每次从文件开始重传，从记录断点的位置继续传输。

分片大小建议

100M, 1M 分片

1000M, 10M 分片

10G, 100M 分片

100G, 1G 分片

参数	描述	是否必需	默认值
bucket	存储空间名称。	是	无
key	OSS 文件名称。	是	无
filename	本地文件。OSS 文件将下载到该文件中。	是	无
store	记录本地分片下载结果的文件。下载过程中，断点信息会保存在此文件中，如果下载中断了，再次下载时会根据文件中记录的点继续下载。	否	HOME 目录下建立的 .py-oss-download 目录。
multipart_threshold	文件长度大于该值时，则使用断点续传下载。	否	10MB
part_size	分片大小。	否	自动计算
progress_callback	下载进度回调函数。	否	无
num_threads	并发下载的线程数。	否	1

断点上传

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 账号进行 API 访问或日常运维，请登录 https://ram.console.aliyun.com 创建 RAM 账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint 以杭州为例，其它 Region 请按实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com',
    '<yourBucketName>')
# 当文件长度大于或等于可选参数 multipart_threshold (默认值为 10MB) 时，会使用分片上传。如未使用参数 store 指定目录，则会在 HOME 目录下建立 .py-oss-upload 目录来保存断点信息。
oss2.resumable_upload(bucket, '<yourObjectName>', '<yourLocalFile>')

# 如使用 store 指定了目录，则保存断点信息在指定目录中。如使用 num_threads 设置上传并发数，
# 请将 oss2.defaults.connection_pool_size 设成大于或等于线程数。默认线程数为 1。
oss2.resumable_upload(bucket, '<yourObjectName>', '<yourLocalFile>',
    store=oss2.ResumableStore(root='/tmp'),
    multipart_threshold=100*1024,
    part_size=100*1024,
    num_threads=4)
```

断点下载

```
# -*- coding: utf-8 -*-
import oss2

# 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 账号进行 API 访问或日常运维，请登录 https://ram.console.aliyun.com 创建 RAM 账号。
auth = oss2.Auth('<yourAccessKeyId>', '<yourAccessKeySecret>')
# Endpoint 以杭州为例，其它 Region 请根据实际情况填写。
bucket = oss2.Bucket(auth, 'http://oss-cn-hangzhou.aliyuncs.com',
    '<yourBucketName>')

# 请将 oss2.defaults.connection_pool_size 设成大于或等于线程数，并将 part_size 参数设成大于或等于 oss2.defaults.multiget_part_size。
oss2.resumable_download(bucket, '<yourObjectName>', '<yourLocalFile>',
    store=oss2.ResumableDownloadStore(root='/tmp'),
    multiget_threshold=20*1024*1024,
    part_size=10*1024*1024,
    num_threads=3)
```

ending

如果发现上述方法都试过，但是吞吐率还是上不去，可以用 iftop 命令分析下，并且同时跑多个程序多进程运行将网络带宽打满；如果多进程也打不满，那就要查自己的机器带宽是否限制了上行流量，或者自己机器带宽被打满。

OSS ossutil 工具调优

ossutil 能做什么

功能概述

ossutil 是 go 编写的命令行高效快捷工具，工具执行效率、支持功能、容错功能无疑是运维必备首选，同时在弱网条件下支持探测模式，测试客户端的网络上下行速率。

- ossutil 专司于高并发大文件或小文件读写的场景，同时支持大文件内部进行分片时的并发。
- 操作文件时可以支持 include exclude 参数，指定哪些后缀的文件可被操作。只显示当前的层级的目录，可选择性的进行递归。
- 伴随 Linux 系统本身的 crontab 使用支持，强制更新 -f -u 参数。
- 限制 ossutil 的操作速度。

支持选项

Commands:

- mb cloud_url [options]

创建 Bucket

1. [cloud_url] [options]

列举 Buckets 或者 Objects

2. cloud_url [options]

删除 Bucket 或 Objects

3. cloud_url [options]

显示 bucket 或者 object 的描述信息

set-acl cloud_url [acl] [options]

设置 bucket 或者 objects 的 acl

set-meta cloud_url [meta] [options]

设置已上传的 objects 的元信息

4. src_url dest_url [options]

上传，下载或拷贝 Objects

5. cloud_url [options]

恢复冷冻状态的 Objects 为可读状态

create-symlink cloud_url target_url [options]

创建符号链接

read-symlink cloud_url [options]

读取符号链接文件的描述信息

6. cloud_url [meta] [options]

生成 object 下载链接

7. file_name [options]

探测命令，支持多种功能探测

8. `dir_name [options]`

创建一个目录，在 oss 中目录名字有后缀字符 '/'

9. `bucket_url [local_xml_file] [options]`

设置、查询或者删除 bucket 的 cors 配置

10. `src_bucket_url target_bucket_url [options]`

设置、查询或者删除 bucket 的 log 配置

11. `bucket_url referer [options]`

设置、查询或者删除 bucket 的 referer 配置

12. `oss_object uploadid [options]`

列出没有完成分块上传的 object 的分块信息

13. `bucket_url [options]`

获取 bucket 所有未完成上传的 multipart object 的分块大小以及总和

14. `local_file_name oss_object [options]`

将本地文件内容以 append 上传方式上传到 oss 中的 appendable object 中

15. `object [options]`

将文件内容输出到标准输出

`bucket-tagging bucket_url [tag_parameter] [options]`

设置、查询或者删除 bucket 的 tag 配置

`bucket-encryption bucket_url [options]`

设置、查询或者删除 bucket 的 encryption 配置

`cors-options oss_url [options]`

向 oss 发送 http options 请求，用于 CORS 检测

16. `bucket_url local_xml_file [options]`

设置、查询或者删除 bucket 的 lifecycle 配置

17. `bucket_url local_xml_file [options]`

设置、查询或者删除 bucket 的 website 配置

`bucket-qos bucket_url [local_xml_file] [options]`

设置、查询或者删除 bucket 的 qos 配置

`user-qos [local_file] [options]`

查询用户的 qos 配置

`bucket-versioning bucket_url [versioning_parameter] [options]`

设置、查询 bucket 的 versioning 配置

18. `bucket_url [options]`

获取 bucket 或者指定前缀（目录）所占的存储空间大小

`bucket-policy bucket_url [local_json_file] [options]`

设置、查询或者删除 bucket 的 policy 配置

`request-payment bucket_url [payment_parameter] [options]`

设置、查询 bucket 的访问者付费配置

`object-tagging cloud_url [tag_parameter] [options]`

设置、查询或者删除 object 的 tag 配置

Additional Commands:

help [command]

获取命令的帮助文档

config [options]

创建配置文件用以存储配置项

hash file_url [options]

计算本地文件的 crc64 或 md5

update [options]

更新 ossutil

场景分类

ossutil help。

拷贝单个小文文件

ossutil cp \$localfile oss://\$bucket/ -i \$AccesskeyID -k \$Accesskey-Secretkey -e \$endpoint。

递归拷贝本地目录小文件到 oss

ossutil cp -r \$local_path oss://\$bucket/ -i \$AccesskeyID -k \$Accesskey-Secretkey -e \$endpoint。

拷贝单个大文文件

ossutil cp \$localfile oss://\$bucket/ --part-size=50000000 --big-file-threshold=54857600 --parallel=5 -i \$AccesskeyID -k \$Accesskey-Secretkey -e \$endpoint。

递归拷贝本地目录大文件到 oss

ossutil cp \$localfile oss://bucket/ --part-size=50000000 --big-file-threshold=54857600 --jobs=20 --parallel=5 -i \$AccesskeyID -k \$AccesskeySecretkey\$ -e \$endpoint。

常见问题

oss 上传下载慢

1. 如果能稳定复现可以用 --loglevel 参数 debug 请求过程，将记录的 OSS 返回的 requestID 反馈到阿里云售后排查。
2. 客户可以使用 probe 探测命令生成一个网络探测报告，通过检测结果看下客户端网络是否存在慢请求的问题。
3. 如果客户端是 ECS，并且和 OSS 是同 region，可以使用内网的 endpoint 进行传输，内网传输速率一般可以达到 50M - 100M 左右（自己测试），比如北京的 OSS 内网 endpoint 就是 oss-cn-beijing-internal.aliyuncs.com。
4. 客户端可以在使用上面推荐的大文件分片上传，下载的功能，如果文件数量多，可以同时增加并发参数提高网络吞吐。
5. 如果客户端还是存在慢的问题，可以用 wireshark 或者 tcpdump 抓包分析下找出慢的瓶颈。

ossutil 出现 skip 情况

```
[root@iZ2Sv4olcc4Z opt]# echo "testlil" > dskydb/test.txt
[root@iZ2Sv4olcc4Z opt]# ./ossutil64 cp -rt -c ~/.ossuti.Lcofig dskydb/test.txt oss://gres/test.txt
Succeed: Total nun: 1, $ze: 30. OK nun: 1(upload 1 files).
0.372650(s) elapsed I
[root@iZ2Sv4olcc4Z opt] echo " ttest222r" >> dskydb/test.txt
[root@iZ2Sv4olcc4Z opt]. /ossutil64 cp -u -c ~/.ossutilconfig cbkydb/test.
```



```
txt oss://gres/test.txt
Succeed: Total num: 1, size: 38. OK num: 1(skip 1 files), Skip sin 38.
0.252878(s) elapsed
```

1. 使用 `-u` 强制更新时，重新对所有的上传文件和原的进行一次比对，发现美有更改的情况就会跳过，有发生更改的就会触发上传进行覆，正常情况。
2. 遇到这种情况可以看下本地的 log 哪些文件被跳过了，将 oss 存储的文件和本地文件做个 MD5 比对确保文件是一致。
3. 命令：`./ossutil64 cp -u -r -f aa.test oss://alihua -i $accesskeyID -k $accesskeyIDSecret -e $endpoint`。

ossutil 访问出现 403



如图用户在操作解冻文件的过程中出现 403，可能与两个原因有关系。

1. 用户使用的子账号操作文件，权限不够。
2. 用户的文件是违禁内容被封禁掉了。
3. 遇到 403 时，递归解冻会中断不会继续。这种现象是正常的，工具在设计之初就是考虑到如果文件遇到 403 的话，代表没有权限操作该文件，那么通过该账号下的其他文件也操作不了，所以就会中断退出。

访问 OSS 出现 400



检查用户的命令和官方提供的命令是否完全一致，避免误操作。使用 stat 选项看一下文件的状态是否是已经解冻了，如果以及解冻再次操作，就会出现 400。

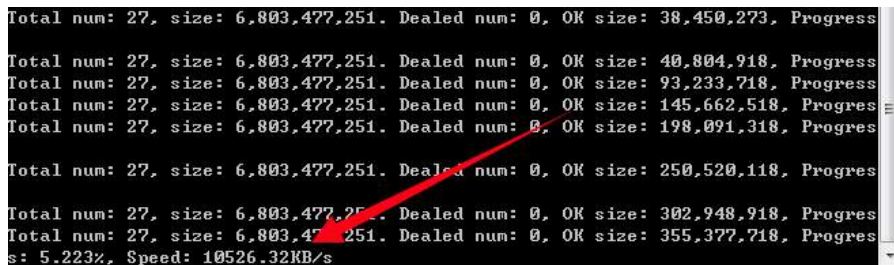
The operation is not valid for the object's state

出现这个问题是因为用户操作的文件是一个归档的文件，不能直接操作，需要先进行解冻 restore 的操作后才能使用。

- ossuti64 restore oss://bucket/prefix/object -l \$accesskey -k \$secretkey -e \$endpoint。
- 递归解冻 ossuti64 restore oss://bucket/prefix/ -r -l \$accesskey -k \$secretkey -e \$endpoint。

上传速率不稳定

Windows ossutil 上传 OSS 速率慢不稳定，客户端是在河北公网，目标服务端是北京，存在跨省情况；首先了解下用户在公网情况下，能否切换到同 region 的 OSS 上进行访问，同 region 的 OSS 内网是有阿里环网组成，如果客户只能在公网使用，进行下面的排查。



```
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 38,450,273, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 40,804,918, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 93,233,718, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 145,662,518, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 198,091,318, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 250,520,118, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 302,948,918, Progress
Total num: 27, size: 6,803,477,251. Dealed num: 0, OK size: 355,377,718, Progress
s: 5.223%, Speed: 10526.32KB/s
```

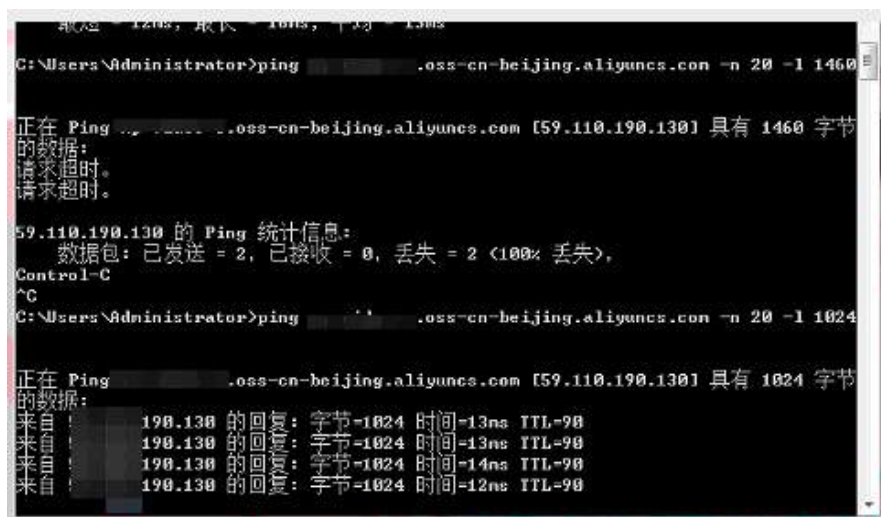
公网情况:

首先看下用户端 ping OSS 的完整域名的网络 ping 值是否正常，tracert 是否有延迟抖动，此步骤可以通过脚本来做，下载脚本后运行，直接输入完整的 OSS 域名即可：[测试脚本](#)

- 脚本中是 ping 的是大包 1460，发现用户端直接网络超时，此时怀疑是客户的网络有限制或者网络拥塞，但是 tracert 通的，再进行下一步排查。



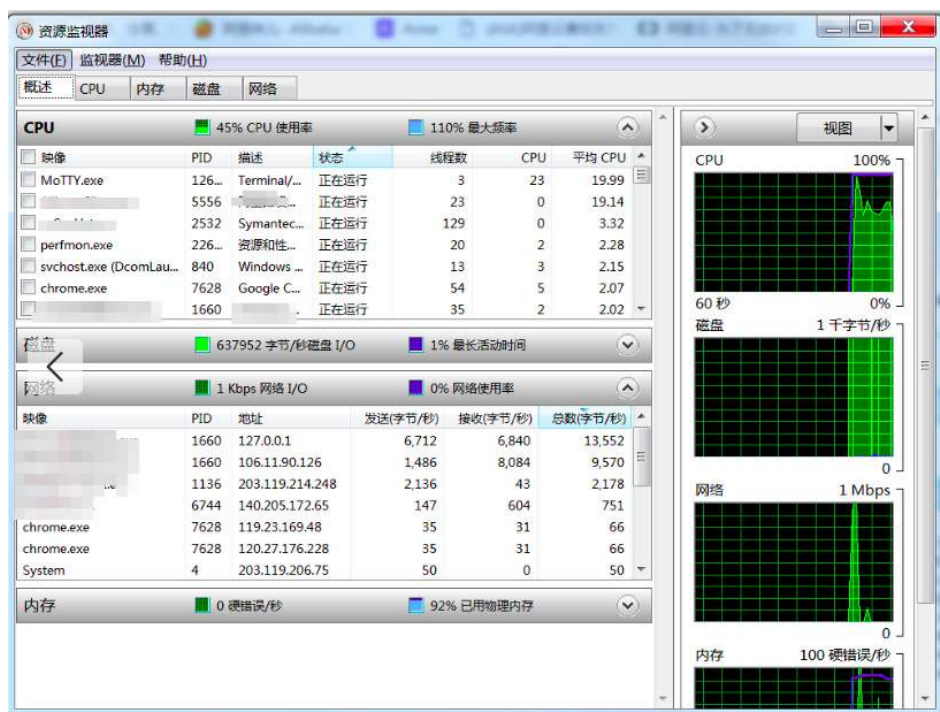
- 尝试降低 ping 包的 len 发现到 1412 ping 可以通过，说明客户端在网络上做限制，不允许 1460 大包的传输，影响了客户端的上行网络吞吐量。



尝试检查本机的网络负载

1. 通过本机资源监控看用户端当时的 CPU 负载和本机内存占用较高, 实际登陆到客户端机器上发现系统卡顿, 实际资源监控器看到 ossutil 工具线程并没有跑到和设置的 `--job=10 --parallel=10` ($10*10=100$) 一样, 只是运行了 24 个, 说明工具的线程受到了系统 CPU 调度影响, 无法将网络吞吐打上去, 于是让用户关系了一些占用 CPU 内存较高的应用后, ossutil 线程数终于打到 69。

调整前的测试



调整后的测试



灵活调整 ossutil 的线程数和分片的并发数

--bigfile-threshold=52428800 --jobs=10 --parallel=50 --part-size=52428800。

1. 遇到大文件数量多，以及单一的大文件时我们要灵活调整 ossutil 的相关参数能够提高我们的 ossutil 的效率。
2. 比如这个案例中客户的文件大小平均是 100M 以上，而且客户的出口带宽是共享 200M，也就是客户自己的上线速度理论上也要 20M。
3. 针对这种情况，我们可以把分片大小调整为 50M-10M，同时增加 parallel 分片的并发数量，尽可能的打满上行的吞吐。当客户端 CPU 核心比较少的时候不建议分的太小，比如分到 1M 时就会造成 CPU 切片过快，消耗 CPU 计算，影响系统性能。

4. 如果遇到文件数量单一文件，或者小文件时，可以不用加任何参数，直接上传即可；调整完成后用户的上行出口速率能打到 10M 大 B。

对比其他家的工具

比如七牛的 qfetch 以及 qshell 做了性能对比，上行的传输效率相差并不多，并没有用户反馈了七牛 20M (大 B) 阿里云 4M (大 B)，基本上 qshell 和 qfetch 的效果都是稳定在 10M 左右。

总结以及需要解决问题

1. 当使用 ossbrowser 上传速度低时可以切换到 ossutil 这个高效的工具进行测试。
2. 需要用户解决下为什么网络出口限制的大包 (1460) 的传输，这个问题不解决很难将网络带宽吞吐提上去。尽量本机不要在负载高的情况下上传一些大文件，如果传输的话可以关闭一些应用卡顿的程序，或者降低下 ossutil 工具的并发线程数量，不然即使设置了 100，但实际受到系统的 CPU 调用影响，不一定能跑到 100。降低线程数，上传的效率也会受到影响。

OSS 公网上调优

场景描述

OSS 公网上传分为两种，针对这两种场景分别描述。

- 跨境传输
- 国内传输

跨境传输

由于网络因素导致，国外国内互传 OSS 文件时，受到国际出口链路的瓶颈影响，传输速度以及稳定性都是无法保证 100% 可用，尤其是在特殊节日例如重要政治会议时期可能更容易出现封堵，针对此类情况用户可以考虑如下种优化手段。

SDK 层面

1. 客户可以用 SDK 的断点上传 / 下载 功能，对源文件进行切片传输，降低大文件的网络传输带宽、延迟开销；同时在网络带宽利用率不高的情况下可以增加分片的并发数量，提高带宽吞吐。
2. 客户在跨境传输时，可以采用增加重试，适当调大超时时间兼容弱网、丢包等环境的影响；但超时时间不宜太长，这样会影响重试。
3. 多线程上传时协程的效率会比较高，但是传输速率不一定高，协程是通过线程中断进行调度切换的，保证每个线程都可以上传但是切换过程中类似软中断一样都有影响传输速率。

网络层面

1. 可以开通全地域上传加速功能，通过智能 DNS 择优链路访问 OSS。

	EndPoint (地域节点)	Bucket 域名	HTTPS
外网访问	oss-cn-hangzhou.aliyuncs.com	hangzhou.oss-cn-hangzhou.aliyuncs.com	支持
ECS 的经典网络访问 (内网)	oss-cn-hangzhou-internal.aliyuncs.com	hangzhou.oss-cn-hangzhou-internal.aliyuncs.com	支持
ECS 的 VPC 网络访问 (内网)	oss-cn-hangzhou-internal.aliyuncs.com	hangzhou.oss-cn-hangzhou-internal.aliyuncs.com	支持
传输加速域名 (全地域上传下载加速)	未开启		支持

2. 使用 全站加速，开通 DCDN，将加速类型设置为全球加速，然后将 DCDN 的原站设置为 OSS 域名。DCDN 会通过就近的边缘节点（靠近网民）接入客户的上下行请求，然后通过智能选路的方式，探测一条最短的回源路径回到原站 OSS，起到上下行的加速作用。
3. 如果是 PC 端上传，保证本地的文件描述符、网络连接数不要被打满，避免出现 socket 连接等待引发超时。
4. 如果用户的场景是大量的 OSS 文件被下载，也可以使用 CDN 加速文件的下行链路，通过分布众多的 CDN 节点就近访问。

工具层面

用户如果不是擅长编码，最快的方式是通过阿里云的自动化工具进行上传、下载，如果是大量文件并发上传推荐使用 ossutil，支持并发、分片、debug 等场景。

如果客户端不会使用命令行，也可以使用 ossbrowser 工具，但是传输性能不如 ossutil 而且支持的文件大小是 5G。

如果用户上传 OSS 使用的 ECS 和 OSS 在同一个地区，建议用户使用内网的 endpoint 地址来上传文件到 OSS。

国内传输

SDK 层面

1. 客户可以用 SDK 的断点上传 / 下载功能，对源文件进行切片传输，降低大文件的网络传输带宽、延迟开销；同时在网络带宽利用率不高的情况下可以增加分片的并发数量，提高带宽吞吐。
2. 客户在跨境传输时，可以采用增加重试，适当调大超时时间兼容弱网、丢包等环境的影响；但超时时间不宜太长，这样会影响重试。
3. 多线程上传时协程的效率会比较高，但是传输速率不一定高，协程是通过线程中断进行调度切换的，保证每个线程都可以上传但是切换过程中类似软中断一样都有影响传输速率。

网络层面

1. 可以开通全地域上传加速功能，通过智能 DNS 择优链路访问 OSS。

	EndPoint (地域节点)	Bucket 域名	HTTPS
外网访问	oss-cn-hangzhou.aliyuncs.com	hangzhou.oss-cn-hangzhou.aliyuncs.com	支持
ECS 的经典网络访问 (内网)	oss-cn-hangzhou-internal.aliyuncs.com	hangzhou.oss-cn-hangzhou-internal.aliyuncs.com	支持
ECS 的 VPC 网络访问 (内网)	oss-cn-hangzhou-internal.aliyuncs.com	hangzhou.oss-cn-hangzhou-internal.aliyuncs.com	支持
传输加速域名 (全地域上传下载加速)	未开启		支持

2. 使用 全站加速，开通 DCDN，将加速类型设置为全球加速，然后将 DCDN 的原站设置为 OSS 域名。DCDN 会通过就近的边缘节点（靠近网民）接入客户的上下行请求，然后通过智能选路的方式，探测一条最短的回源路径回到原站 OSS，起到上下行的加速作用。
3. 如果是 PC 端上传，保证本地的文件描述符、网络连接数不要被打满，避免出现 socket 连接等待引发超时；

4. 如果用户的场景是大量的 OSS 文件被下载，也可以使用 CDN 加速文件的下行链路，通过分布众多的 CDN 节点就近访问。

工具层面

用户如果不是擅长编码，最快的方式是通过阿里云的自动化工具进行上传、下载，如果是大量文件并发上传推荐使用 ossutil，支持并发、分片、debug 等场景。

如果客户端不会使用命令行，也可以使用 ossbrowser 工具，但是传输性能不如 ossutil 而且支持的文件大小是 5G。

如果用户上传 OSS 使用的 ECS 和 OSS 在同一个地区，建议用户使用内网的 endpoint 地址来上传文件到 OSS。

OSS Java SDK 调优

环境准备

- 使用 Java 1.8 及以上版本。
- 查看版本执行

命令 `java -version` 查看 Java 版本

下载 SDK

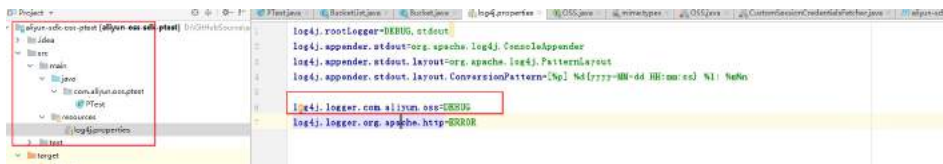
- 直接通过 GitHub 下载
- 安装 SDK，在 Maven 项目中加入依赖项（推荐方式）

```
<dependency>
<groupId>com.aliyun.oss</groupId>
<artifactId>aliyun-sdk-oss</artifactId>
<version>2.8.3</version>
</dependency>
```

初始化的参数设置

日志调优参数

一般情况下不会开启日志功能，很多用户都是开启日志 `log4j` 后写变慢了；网上有很多关闭的方法可以找下开源的处理文档关闭掉，如果用户配置了 `log4j` 的属性文件，需要在那个属性文件里 配置 `oss` 的 日志级别。



超时参数设置

弱网的环境下最好将超时时间设置长一些，增加重试次数，避免上传失败。

```
// 创建 ClientConfiguration。ClientConfiguration 是 OSSClient 的配置类，可配置代理、
// 连接超时、最大连接数等参数。
ClientConfiguration conf = new ClientConfiguration();

// 设置 OSSClient 允许打开的最大 HTTP 连接数，默认为 1024 个。
conf.setMaxConnections(2048);
// 设置 Socket 层传输数据的超时时间，默认为 50000 毫秒。
conf.setSocketTimeout(10000);
// 设置建立连接的超时时间，默认为 50000 毫秒。
conf.setConnectionTimeout(10000);
// 设置从连接池中获取连接的超时时间(单位：毫秒)，默认不超时。
conf.setConnectionRequestTimeout(1000);
// 设置连接空闲超时时间。超时则关闭连接，默认为 60000 毫秒。
conf.setIdleConnectionTime(10000);
// 设置失败请求重试次数，默认为 3 次。
conf.setMaxErrorRetry(5);
```

网络流传输

用户采用网络流传输，SDK 会通过公网先拉源文件，如果源拉文件很慢，直接影响到写 OSS 速度，尽量不要通过网络流上传文件，尽量使用本地上传。如果要用网络流的方式传输，先保证源文件网络的通畅和带宽充足。

```
// Endpoint 以杭州为例，其它 Region 请按实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 账号进行
// API 访问或日常运维，请登录 https://ram.console.aliyun.com 创建 RAM 账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建 OSSClient 实例。
OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
```

```

accessKeySecret);

// 上传网络流。
InputStream inputStream = new URL("https://www.aliyun.com/").openStream();
ossClient.putObject("<yourBucketName>", "<yourObjectName>", inputStream);

// 关闭 OSSClient。
ossClient.shutdown();

```

主机文件描述符

用户使用 SDK 配置的 connection 代表应用层处理的最大能力，要和主机的文件描述符相对成，如果主机设置的 FD 是 1000，SDK 应用层即便设置到 2000，最后效果也达不到 2000，最多只能处理 1000 个连接。

代码优化

java SDK 在 分片的基础上增加了断点传输，既可以在分片基础上增加并发，也支持断点记录的功能，在遇到网络问题断开时不需要每次从文件开始重传，从记录断点的位置继续传输。

分片大小建议

100M, 1M 分片

1000M, 10M 分片

10G, 100M 分片

100G, 1G 分片

setPartSize 设置分片大小，默认 1M

```

// Endpoint 以杭州为例，其它 Region 请根据实际情况填写。
String endpoint = "http://oss-cn-hangzhou.aliyuncs.com";
// 阿里云主账号 AccessKey 拥有所有 API 的访问权限，风险很高。强烈建议您创建并使用 RAM 账号进行 API 访问或日常运维，请登录 https://ram.console.aliyun.com 创建 RAM 账号。
String accessKeyId = "<yourAccessKeyId>";
String accessKeySecret = "<yourAccessKeySecret>";

// 创建 OSSClient 实例。

```

```

OSS ossClient = new OSSClientBuilder().build(endpoint, accessKeyId,
accessKeySecret);

ObjectMetadata meta = new ObjectMetadata();
// 指定上传的内容类型。
meta.setContentType("text/plain");

// 通过 UploadFileRequest 设置多个参数。
UploadFileRequest uploadFileRequest = new
UploadFileRequest("<yourBucketName>", "<yourObjectName>");

// 通过 UploadFileRequest 设置单个参数。
// 设置存储空间名称。
//uploadFileRequest.setBucketName("<yourBucketName>");
// 设置文件名称。
//uploadFileRequest.setKey("<yourObjectName>");
// 指定上传的本地文件。
uploadFileRequest.setUploadFile("<yourLocalFile>");
// 指定上传并发线程数，默认为 1。
uploadFileRequest.setTaskNum(5);
// 指定上传的分片大小，范围为 100KB~5GB，默认为文件大小 /10000。
uploadFileRequest.setPartSize(1 * 1024 * 1024);
// 开启断点续传，默认关闭。
uploadFileRequest.setEnableCheckpoint(true);
// 记录本地分片上传结果的文件。开启断点续传功能时需要设置此参数，上传过程中的进度信息会保存在
该文件中，如果某一分片上传失败，再次上传时会根据文件中记录的点继续上传。上传完成后，该文件
会被删除。默认与待上传的本地文件同目录，为 uploadFile.ucp。
uploadFileRequest.setCheckpointFile("<yourCheckpointFile>");
// 文件的元数据。
uploadFileRequest.setObjectMetadata(meta);
// 设置上传成功回调，参数为 Callback 类型。
uploadFileRequest.setCallback("<yourCallbackEvent>");

// 断点续传上传。
ossClient.uploadFile(uploadFileRequest);

// 关闭 OSSClient。
ossClient.shutdown();

```

常见问题

下文结合一些常见的使用案例说下注意问题，案例不定期补充，所有用户数据信息已经处理过。

SDK.ServerUnreachable : Specified endpoint or uri is not valid

```

SunshineE/android/01040181-2018-05-07173747-860769-online-1.jpg</cloudUrl><fileSize>4096</fileSize><lossName></lossName></oss>
... skipping...
com.aliyuncs.exceptions.ClientException: SDK.ServerUnreachable : Specified endpoint or uri is not valid.
    at com.aliyuncs.DefaultAcsClient.doAction(DefaultAcsClient.java:201)
    at com.aliyuncs.DefaultAcsClient.doAction(DefaultAcsClient.java:151)
    at com.aliyuncs.DefaultAcsClient.doAction(DefaultAcsClient.java:59)
    at com.aliyuncs.DefaultAcsClient.getResponse(DefaultAcsClient.java:103)
    at com.sunnyard.insurance.oss.aliyun.GetSts.assumeRole(GetSts.java:50)
    at com.sunnyard.insurance.oss.aliyun.GetSts.getSts(GetSts.java:60)
    at com.sunnyard.insurance.ecm.web.action.oss.OssInfoAction.getUpOssInfo(OssInfoAction.java:259)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:10)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:503)

```

出现这种问题意思是用户没有连接到阿里云网关，一般和一下几个原因有关系。

- 用户端的代码中并发请求 STS 过高，而用户端的 ECS 或者本地 PC 不足以承载当时的并发导致，降低 OSS 并发。
- 用户的网络到 server 端有超时现象可以进行抓包验证。
- 用户的 STS SDK 版本以及 SDK core 版本不是最新导致。更换到新版 SDK 测试一下。

访问 NoSuchKey

```

case:java.lang.Exception: com.aliyun.oss.OSSException:
Not Found[ErrorCode]: NoSuchKey

```

java SDK 出现这种问题，就是源文件不存在，可以参考排查系列中的 [404] (<https://yq.aliyun.com/articles/657166?spm=ata.13261165.0.0.181e32fbX-DYxLN>)

socketException

```

2018-10-10 16:21:11,127 ERROR com.qunhe.instdeco.plan.maxservice.convert.Converter
downloadOssObjecttaskId:LO63HXQKN4BMWM4BAU888888;
com.aliyun.oss.ClientException: SocketException
    at com.aliyun.oss.common.utils.ExceptionFactory.createNetworkException(ExceptionFactory.java:71)
    at com.aliyun.oss.common.comm.DefaultServiceClient.sendRequestCore(DefaultServiceClient.java:128)
    at com.aliyun.oss.common.comm.ServiceClient.sendRequestImpl(ServiceClient.java:123)
    at com.aliyun.oss.common.comm.ServiceClient.sendRequest(ServiceClient.java:68)
    at com.aliyun.oss.internal.OSSOperation.send(OSSOperation.java:94)
    at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:149)
    at com.aliyun.oss.internal.OSSOperation.doOperation(OSSOperation.java:113)
    at com.aliyun.oss.internal.OSSObjectOperation.getObject(OSSObjectOperation.java:273)

```

可以明显看到错误是 socket 异常了，这可能是 socket 在 init 阶段就失败了，请求都没有到 OSS，建议客户端检查下。

- 当时的网络是否出现抖动。可以用 `ping -c 100 -s 1024 -i 0.01` 或者 `mtr` 命令看下网络即时探测的数据。
- 主机的 socket 连接数是否占满。可以用 `netstat` 命令看下当前的连接数。
- SDK 中设置的 `maxconnection` 是多大，如果当时链接数超过 `maxconnection` 设置，也会出现 socket 异常。
- 如果以上没有没有问题，只能让用户部署 `tcpdump` 或者 `wireshark` 抓包，然后复现问题后，分下数据包。

Java 结合 web 实现通过 STS 上传 OSS

如果出现如下报错，可能和跨域或者代码中配置的 OSS 信息有关，可以按照如下思路处理。

```

1:throw()
2:body()
3:body()
4:body()
5:body()
6:body()
7:body()
8:body()
9:body()
10:body()
11:body()
12:body()
13:body()
14:body()
15:body()
16:body()
17:body()
18:body()
19:body()
20:body()
21:body()
22:body()
23:body()
24:body()
25:body()
26:body()
27:body()
28:body()
29:body()
30:body()
31:body()
32:body()
33:body()
34:body()
35:body()
36:body()
37:body()
38:body()
39:body()
40:body()
41:body()
42:body()
43:body()
44:body()
45:body()
46:body()
47:body()
48:body()
49:body()
50:body()
51:body()
52:body()
53:body()
54:body()
55:body()
56:body()
57:body()
58:body()
59:body()
60:body()
61:body()
62:body()
63:body()
64:body()
65:body()
66:body()
67:body()
68:body()
69:body()
70:body()
71:body()
72:body()
73:body()
74:body()
75:body()
76:body()
77:body()
78:body()
79:body()
80:body()
81:body()
82:body()
83:body()
84:body()
85:body()
86:body()
87:body()
88:body()
89:body()
90:body()
91:body()
92:body()
93:body()
94:body()
95:body()
96:body()
97:body()
98:body()
99:body()
100:body()
101:body()
102:body()
103:body()
104:body()
105:body()
106:body()
107:body()
108:body()
109:body()
110:body()
111:body()
112:body()
113:body()
114:body()
115:body()
116:body()
117:body()
118:body()
119:body()
120:body()
121:body()
122:body()
123:body()
124:body()
125:body()
126:body()
127:body()
128:body()
129:body()
130:body()
131:body()
132:body()
133:body()
134:body()
135:body()
136:body()
137:body()
138:body()
139:body()
140:body()
141:body()
142:body()
143:body()
144:body()
145:body()
146:body()
147:body()
148:body()
149:body()
150:body()
151:body()
152:body()
153:body()
154:body()
155:body()
156:body()
157:body()
158:body()
159:body()
160:body()
161:body()
162:body()
163:body()
164:body()
165:body()
166:body()
167:body()
168:body()
169:body()
170:body()
171:body()
172:body()
173:body()
174:body()
175:body()
176:body()
177:body()
178:body()
179:body()
180:body()
181:body()
182:body()
183:body()
184:body()
185:body()
186:body()
187:body()
188:body()
189:body()
190:body()
191:body()
192:body()
193:body()
194:body()
195:body()
196:body()
197:body()
198:body()
199:body()
200:body()
201:body()
202:body()
203:body()
204:body()
205:body()
206:body()
207:body()
208:body()
209:body()
210:body()
211:body()
212:body()
213:body()
214:body()
215:body()
216:body()
217:body()
218:body()
219:body()
220:body()
221:body()
222:body()
223:body()
224:body()
225:body()
226:body()
227:body()
228:body()
229:body()
230:body()
231:body()
232:body()
233:body()
234:body()
235:body()
236:body()
237:body()
238:body()
239:body()
240:body()
241:body()
242:body()
243:body()
244:body()
245:body()
246:body()
247:body()
248:body()
249:body()
250:body()
251:body()
252:body()
253:body()
254:body()
255:body()
256:body()
257:body()
258:body()
259:body()
260:body()
261:body()
262:body()
263:body()
264:body()
265:body()
266:body()
267:body()
268:body()
269:body()
270:body()
271:body()
272:body()
273:body()
274:body()
275:body()
276:body()
277:body()
278:body()
279:body()
280:body()
281:body()
282:body()
283:body()
284:body()
285:body()
286:body()
287:body()
288:body()
289:body()
290:body()
291:body()
292:body()
293:body()
294:body()
295:body()
296:body()
297:body()
298:body()
299:body()
300:body()
301:body()
302:body()
303:body()
304:body()
305:body()
306:body()
307:body()
308:body()
309:body()
310:body()
311:body()
312:body()
313:body()
314:body()
315:body()
316:body()
317:body()
318:body()
319:body()
320:body()
321:body()
322:body()
323:body()
324:body()
325:body()
326:body()
327:body()
328:body()
329:body()
330:body()
331:body()
332:body()
333:body()
334:body()
335:body()
336:body()
337:body()
338:body()
339:body()
340:body()
341:body()
342:body()
343:body()
344:body()
345:body()
346:body()
347:body()
348:body()
349:body()
350:body()
351:body()
352:body()
353:body()
354:body()
355:body()
356:body()
357:body()
358:body()
359:body()
360:body()
361:body()
362:body()
363:body()
364:body()
365:body()
366:body()
367:body()
368:body()
369:body()
370:body()
371:body()
372:body()
373:body()
374:body()
375:body()
376:body()
377:body()
378:body()
379:body()
380:body()
381:body()
382:body()
383:body()
384:body()
385:body()
386:body()
387:body()
388:body()
389:body()
390:body()
391:body()
392:body()
393:body()
394:body()
395:body()
396:body()
397:body()
398:body()
399:body()
400:body()
401:body()
402:body()
403:body()
404:body()
405:body()
406:body()
407:body()
408:body()
409:body()
410:body()
411:body()
412:body()
413:body()
414:body()
415:body()
416:body()
417:body()
418:body()
419:body()
420:body()
421:body()
422:body()
423:body()
424:body()
425:body()
426:body()
427:body()
428:body()
429:body()
430:body()
431:body()
432:body()
433:body()
434:body()
435:body()
436:body()
437:body()
438:body()
439:body()
440:body()
441:body()
442:body()
443:body()
444:body()
445:body()
446:body()
447:body()
448:body()
449:body()
450:body()
451:body()
452:body()
453:body()
454:body()
455:body()
456:body()
457:body()
458:body()
459:body()
460:body()
461:body()
462:body()
463:body()
464:body()
465:body()
466:body()
467:body()
468:body()
469:body()
470:body()
471:body()
472:body()
473:body()
474:body()
475:body()
476:body()
477:body()
478:body()
479:body()
480:body()
481:body()
482:body()
483:body()
484:body()
485:body()
486:body()
487:body()
488:body()
489:body()
490:body()
491:body()
492:body()
493:body()
494:body()
495:body()
496:body()
497:body()
498:body()
499:body()
500:body()
501:body()
502:body()
503:body()
504:body()
505:body()
506:body()
507:body()
508:body()
509:body()
510:body()
511:body()
512:body()
513:body()
514:body()
515:body()
516:body()
517:body()
518:body()
519:body()
520:body()
521:body()
522:body()
523:body()
524:body()
525:body()
526:body()
527:body()
528:body()
529:body()
530:body()
531:body()
532:body()
533:body()
534:body()
535:body()
536:body()
537:body()
538:body()
539:body()
540:body()
541:body()
542:body()
543:body()
544:body()
545:body()
546:body()
547:body()
548:body()
549:body()
550:body()
551:body()
552:body()
553:body()
554:body()
555:body()
556:body()
557:body()
558:body()
559:body()
560:body()
561:body()
562:body()
563:body()
564:body()
565:body()
566:body()
567:body()
568:body()
569:body()
570:body()
571:body()
572:body()
573:body()
574:body()
575:body()
576:body()
577:body()
578:body()
579:body()
580:body()
581:body()
582:body()
583:body()
584:body()
585:body()
586:body()
587:body()
588:body()
589:body()
590:body()
591:body()
592:body()
593:body()
594:body()
595:body()
596:body()
597:body()
598:body()
599:body()
600:body()
601:body()
602:body()
603:body()
604:body()
605:body()
606:body()
607:body()
608:body()
609:body()
610:body()
611:body()
612:body()
613:body()
614:body()
615:body()
616:body()
617:body()
618:body()
619:body()
620:body()
621:body()
622:body()
623:body()
624:body()
625:body()
626:body()
627:body()
628:body()
629:body()
630:body()
631:body()
632:body()
633:body()
634:body()
635:body()
636:body()
637:body()
638:body()
639:body()
640:body()
641:body()
642:body()
643:body()
644:body()
645:body()
646:body()
647:body()
648:body()
649:body()
650:body()
651:body()
652:body()
653:body()
654:body()
655:body()
656:body()
657:body()
658:body()
659:body()
660:body()
661:body()
662:body()
663:body()
664:body()
665:body()
666:body()
667:body()
668:body()
669:body()
670:body()
671:body()
672:body()
673:body()
674:body()
675:body()
676:body()
677:body()
678:body()
679:body()
680:body()
681:body()
682:body()
683:body()
684:body()
685:body()
686:body()
687:body()
688:body()
689:body()
690:body()
691:body()
692:body()
693:body()
694:body()
695:body()
696:body()
697:body()
698:body()
699:body()
700:body()
701:body()
702:body()
703:body()
704:body()
705:body()
706:body()
707:body()
708:body()
709:body()
710:body()
711:body()
712:body()
713:body()
714:body()
715:body()
716:body()
717:body()
718:body()
719:body()
720:body()
721:body()
722:body()
723:body()
724:body()
725:body()
726:body()
727:body()
728:body()
729:body()
730:body()
731:body()
732:body()
733:body()
734:body()
735:body()
736:body()
737:body()
738:body()
739:body()
740:body()
741:body()
742:body()
743:body()
744:body()
745:body()
746:body()
747:body()
748:body()
749:body()
750:body()
751:body()
752:body()
753:body()
754:body()
755:body()
756:body()
757:body()
758:body()
759:body()
760:body()
761:body()
762:body()
763:body()
764:body()
765:body()
766:body()
767:body()
768:body()
769:body()
770:body()
771:body()
772:body()
773:body()
774:body()
775:body()
776:body()
777:body()
778:body()
779:body()
780:body()
781:body()
782:body()
783:body()
784:body()
785:body()
786:body()
787:body()
788:body()
789:body()
790:body()
791:body()
792:body()
793:body()
794:body()
795:body()
796:body()
797:body()
798:body()
799:body()
800:body()
801:body()
802:body()
803:body()
804:body()
805:body()
806:body()
807:body()
808:body()
809:body()
810:body()
811:body()
812:body()
813:body()
814:body()
815:body()
816:body()
817:body()
818:body()
819:body()
820:body()
821:body()
822:body()
823:body()
824:body()
825:body()
826:body()
827:body()
828:body()
829:body()
830:body()
831:body()
832:body()
833:body()
834:body()
835:body()
836:body()
837:body()
838:body()
839:body()
840:body()
841:body()
842:body()
843:body()
844:body()
845:body()
846:body()
847:body()
848:body()
849:body()
850:body()
851:body()
852:body()
853:body()
854:body()
855:body()
856:body()
857:body()
858:body()
859:body()
860:body()
861:body()
862:body()
863:body()
864:body()
865:body()
866:body()
867:body()
868:body()
869:body()
870:body()
871:body()
872:body()
873:body()
874:body()
875:body()
876:body()
877:body()
878:body()
879:body()
880:body()
881:body()
882:body()
883:body()
884:body()
885:body()
886:body()
887:body()
888:body()
889:body()
890:body()
891:body()
892:body()
893:body()
894:body()
895:body()
896:body()
897:body()
898:body()
899:body()
900:body()
901:body()
902:body()
903:body()
904:body()
905:body()
906:body()
907:body()
908:body()
909:body()
910:body()
911:body()
912:body()
913:body()
914:body()
915:body()
916:body()
917:body()
918:body()
919:body()
920:body()
921:body()
922:body()
923:body()
924:body()
925:body()
926:body()
927:body()
928:body()
929:body()
930:body()
931:body()
932:body()
933:body()
934:body()
935:body()
936:body()
937:body()
938:body()
939:body()
940:body()
941:body()
942:body()
943:body()
944:body()
945:body()
946:body()
947:body()
948:body()
949:body()
950:body()
951:body()
952:body()
953:body()
954:body()
955:body()
956:body()
957:body()
958:body()
959:body()
960:body()
961:body()
962:body()
963:body()
964:body()
965:body()
966:body()
967:body()
968:body()
969:body()
970:body()
971:body()
972:body()
973:body()
974:body()
975:body()
976:body()
977:body()
978:body()
979:body()
980:body()
981:body()
982:body()
983:body()
984:body()
985:body()
986:body()
987:body()
988:body()
989:body()
990:body()
991:body()
992:body()
993:body()
994:body()
995:body()
996:body()
997:body()
998:body()
999:body()
1000:body()

```

```

String accessId = "<yourAccessKeyId>"; // 请填写您的AccessKeyId。
String accessKey = "<yourAccessKeySecret>"; // 请填写您的AccessKeySecret。
String endpoint = "oss-cn-hangzhou.aliyuncs.com"; // 请填写您的 endpoint。
String bucket = "bucket-name"; // 请填写您的 bucketname。
String host = "http://" + bucket + "." + endpoint; // host的格式为 bucketname.endpoint

// callbackUrl为 上传回调服务器的URL，请将下面的IP和Port配置为您自己的真实信息。
String callbackUrl = "http://88.88.88.88:8888";
String dir = "user-dir-prefix/"; // 用户上传文件时指定的前缀。

```


从浏览器向OSS发出的请求消息带有 `Origin` 的消息头，OSS对带有 `Origin` 头的请求消息会首先进行 **跨域规则** 的验证。

即客户端进行表单直接上传到OSS会产生跨域请求，需要为Bucket设置跨域规则（CORS），支持Post方法。

具体操作步骤请参见[设置跨域访问](#)。

跨域规则

* 来源 *

来源可以设置多个，每行一个，每行最多能有一个通配符[*]

* 允许 Methods ☐ GET ☒ POST ☐ PUT ☐ DELETE ☐ HEAD

允许 Headers *

如果是使用 java 的代码，可以看下

- 服务端生成上传信息时 host 和 bucket 是否填写正确。
- 是否对 OSS 配置了跨域文件。



阿里云 开发者社区



云服务技术大学
云产品干货高频分享



云服务技术课堂
和大牛零距离沟通



阿里云开发者“藏经阁”
海量免费电子书下载