天翼云
State Cloud

对象存储（经典版）I 型

(Object-Oriented Storage,OOS)

数据迁移工具

天翼云科技有限公司

目录

1 介绍

OOS 数据迁移工具是一款将数据迁移至 OOS 的工具。您可以将迁移工具部署在本地服务器或云主机上，将其它云存储（阿里云、腾讯云、华为云、AmazaonS3）的数据迁移到 OOS、本地数据迁移到 OOS、或者进行 OOS 各资源池间数据迁移。

OOS 数据迁移工具具有以下特点：

- 支持断点续传。迁移过程中，如果出现中断，重新启动工具后，可以继续执行迁移工作。已经迁移完成的部分数据，根据 isSkipExist 参数设置进行上传或跳过。
- 支持流量控制。迁移过程中，可以动态调整从源端获取数据的速率。
- 支持迁移指定前缀的文件。
- 支持文件并行下载和上传。
- 支持根据迁移工具日志（迁移成功日志、迁移失败日志、迁移忽略日志）进行文件迁移。

2 运行环境

迁移工具支持部署在 Windows 或 Linux 客户端，运行环境要求如下：

- Windows7 及以上版本或 Linux CentOS 7.x 及以上版本。
- Java 1.8 及以上版本。
- 迁移工具所在的服务器可以访问源云存储资源池和 OOS 资源池。

3 迁移步骤

迁移需要的步骤：

1. 迁移准备。
2. 安装迁移工具。
3. 修改配置文件。
4. 执行迁移。

3.1 迁移准备

- 已拥有 OOS 账户，获得 OOS 的 AccessKey 和 SecretKey，且在 OOS 中创建了目标存储桶 Bucket。

  注意：

  - 普通迁移时，srcAccessKey 和 srcSecretKey 需要具备 srcBucket 的 ListBucket 和 GetObject 权限；destAccessKey 和 destSecretKey 需要具备 destBucket 中 PutObject 的权限。
  - srcType 为 OOS 时，可以使用加速迁移。源和目的资源池为同类资源池时才可以启用加速迁移，如源和目的资源池为 5.0 的同一地域、源和目的资源池同为香港资源池、或源和目的资源池同为对象存储网络。另外 srcAccessKey 和 srcSecretKey 需要具备 srcBucket 的 ListBucket 和 GetObject 权限；destAccessKey 和 destSecretKey 需要具备 destBucket 中 PutObject 的权限、srcBucket 的 GetObject 权限。
- 下载 **OOS 数据迁移工具**。

3.2 安装

- 对于 Windows 客户端，直接解压缩迁移工具 zip 包即可。
- 对于 Linux 客户端，执行 unzip CTYUN_OOS_Import_1.3.2.zip，解压缩安装包。

迁移工具解压缩后的文件结构如下：

```
CTYUN_OOS_Import_1.3.2
|----config
|-------log4j2.xml
|-------migrate.conf
|-------system.conf
|----lib
|----import.sh
|----import.bat
```

3.3 修改配置文件

- 更新迁移任务配置文件 migrate.conf，配置源和目的资源池信息、迁移配置项。

  说明：

  - 迁移过程中，如果想停止正在迁移的任务，可以修改系统配置 system.conf 中的 stopScan

为 true 停止本次迁移，重启时需要将系统配置 system.conf 中的 stopScan 修改为 false。

- 如果迁移过程中想修改迁移任务，需停止本次迁移，然后修改 migrate.conf，并重启。如果迁移过程中未停止本次迁移任务，进行修改迁移任务配置文件 migrate.conf，修改后的迁移任务配置文件不生效。

- （可选）根据需要更新系统配置文件 system.conf，在迁移过程中可以修改此配置文件中的配置项。仅影响修改后加入迁移任务的迁移文件（Object），修改前已加入迁移任务的迁移文件，不受本次修改影响。

3.3.1 migrate.conf

表1 迁移任务配置文件（migrate.conf）参数

参数	描述	是否必须
srcType	迁移源类型： ● OOS：天翼云对象存储（经典版）I 型。 ● OSS：阿里云。 ● COS：腾讯云。 ● OBS：华为云。 ● S3：AmazonS3。 ● LOCAL：本地。	是
srcEndpoint	源资源池 Endpoint。 ● 迁移源类型为 COS、LOCAL：不配置此参数。 ● 迁移源类型为 OOS、OSS、OBS、S3：此项必须填。	条件
srcAccessKey	源资源池账户 AccessKey。 ● 迁移源类型为 LOCAL：不配置此参数。 ● 迁移源类型为 OOS、OSS、COS、OBS、S3：此项必须填。	条件
srcSecretKey	源资源池账户 SecretKey。 ● 迁移源类型为 LOCAL：不配置此参数。 ● 迁移源类型为 OOS、OSS、COS、OBS、S3：此项	条件

	必须填。	
srcBucket	源资源池 Bucket。 ● 迁移源类型为 LOCAL：不配置此参数。 ● 迁移源类型为 OOS、OSS、COS、OBS、S3：必须填。	条件
srcRegionName	源资源池 RegionName ● 迁移源类型为 COS、S3：必须填。 ● 迁移源类型为 OOS、OSS、OBS、LOCAL：不配置此参数。	条件
localFolderPath	本地文件夹路径，或以执行脚本的相对路径。 需要完整路径，以单个正斜线（/）进行分割并且以单个正斜线（/）结尾，仅支持如 c:/example/ 或者 /data/example/ 的格式。 ● 迁移源类型为 LOCAL：必须填。 ● 迁移类型为 OOS、OSS、COS、OBS、S3：不配置此参数。	条件
destEndpoint	目标资源池 Endpoint,即要迁移到的 OOS 资源池 Endpoint，参见域名（Endpoint）列表。	是
destAccessKey	目标资源池 AccessKey，可以从 OOS 控制台中获取，参见密钥。	是
destSecretKey	目标资源池 SecretKey。	是
destBucket	目标资源池 Bucket。	是
srcPrefix	源文件（Object）名前缀，默认为空。该前缀不包括 Bucket 名称，仅为文件（Object）名前缀。 **说明**：此参数空或者未配置此参数，表示迁移所有的文件。	否
destPrefix	目标前缀，即为迁移文件（Object）指定具体存放文件夹。 **注意**：如果不需要目标前缀，此参数不要写在配置文件	否

	中。 **取值**：字符串形式，以“/”结尾，且不能以“/”开头；不能包含以下字符：'、"、//。 **说明**： ● 如果不指定，迁移文件直接存放在目标资源池 Bucket 下。 ● 如果指定前缀，则将迁移文件存放在目标资源池 Bucket 下以前缀命名的文件夹中；如果前缀命名的文件夹不存在，则在该 Bucket 下创建以前缀命名的文件夹。 ● 如果目标前缀包含字符“\”，会识别为转义字符，不建议使用字符 “\”。	
srcMarker	● 迁移源类型为 OOS，表示按字典序，从 srcMarker 文件（Object）开始迁移。默认为空，表示从第一个文件开始迁移。 ● 迁移源类型为 OSS、OBS、COS、S3，表示按字典序，从 srcMarker 文件（Object）后的下一个文件开始迁移。默认为空，表示从第一个文件开始迁移。 ● 迁移源类型为 LOCAL，此项不起作用，不配置此参数。	否
srcStopObject	迁移的截止文件（Object）名，默认为空。 ● 如果配置了 srcStopObject，则迁移到配置的文件后停止迁移，即迁移到该文件的前一个文件，此文件及后续文件都不迁移。 ● 如果指定的 srcStopObject 不存在，则迁移满足迁移条件的所有文件（Object）。 **注意**：迁移源类型为 LOCAL，此项不起作用，不配置此参数。	否
isSkipExistFile	是否跳过目标资源池中已有的文件（Object）。	否

	● true：跳过已有文件，根据 Etag 和 size 进行判断数据是否为已有文件。 ● false：覆盖已有文件。 默认值为 false。	
migrateLogFile	表示是否根据日志文件加载迁移文件（Object）。 日志文件仅支持通过本迁移工具生成的四种类型的日志文件：successObjectLog_*time*.txt、errorObjectLog_*time*.txt、skipObjectLog_*time*.txt、otherObjectLog_*time*.txt。 ● true：仅处理日志文件中的项，不扫描源资源池或者 local 文件夹、不加载备份文件。 ● false：扫描资源池或 local 文件夹，加载备份文件。 默认值为 false。	否
logFile	表示日志文件路径。 migrateLogFile 为 true 时，此项必须填。	条件
importSince	表示迁移大于此时间的数据，Unix 时间戳。即仅迁移此时间点后的文件（Object）。 **取值**：大于等于 0 的整数，单位是秒。默认值为 0，表示迁移所有的数据。	否
objectSize	表示迁移文件（Object）的大小范围。格式是 N-M，表示迁移 N 至 M 大小的文件。 **取值**：N 和 M 是大于等于 0 的整数，且 N≤M，单位是字节。默认不配置此项，表示迁移所有大小的文件。	否
storageClass	设置迁移后文件（Object）的存储类型。 **取值**： ● STANDARD：标准存储。 ● STANDARD_IA：低频访问存储。 默认值为 STANDARD。 **注意**：仅对象存储网络、香港节点支持	否

	STANDARD_IA，其他地域不支持。	
contentType	迁移之后，文件（Object）的标准 MIME 类型。 **注意**：如果 srcType 为 OOS，且 isAcceleratedMigration 为 true，contentType 配置会失效。	否
isAcceleratedMigration	是否加速迁移文件（Object）。 取值： ● true：加速迁移文件。 ● false：不加速迁移文件。 默认值为 false。 **说明**：srcType 为 OOS 时，该参数才生效。建议源和目的资源池为同类资源池时使用该参数，如源和目的资源池为 5.0 的同一地域、源和目的资源池同为香港资源池、或源和目的资源池同为对象存储网络。 使用加速迁移文件时，srcAccessKey 和 srcSecretKey 需要具备 srcBucket 的 ListBucket 和 GetObject 权限；destAccessKey 和 destSecretKey 需要具备 destBucket 中 PutObject 的权限、srcBucket 的 GetObject 权限。	否

3.3.2 system.conf

表2 系统配置文件（system.conf）参数

参数	描述	是否必须
threadNum	并发数。 取值范围：整数形式，取值为[1, 3000]，默认值为 1。	否
maxSimpleObjectSizeM	文件（Object）大小限制，单位是 MiB。 取值范围：大于等于 5 的整数，默认值为 100。 超过文件大小限制，源文件将被拆分成分段文件进行迁移。	否

partSizeM	拆分为分段文件时的分片大小，单位 MiB。 取值范围：大于等于 5 的整数，默认值为 50。	否
stopScan	是否终止遍历源文件（Object）。 ● true：终止。 ● false：不终止。 默认 false。	否
maxThroughput	对源端流量进行限制，单位是 KiB/s。 取值：整数形式，[100, 102400]。默认不限速。 **说明：** ● 如果填写负数，则表示不进行限速。 ● 如果 0≤maxThroughput<100，则按 100KiB/s 限速。 ● 如果 maxThroughput>102400，则按 102400KiB/s 限速。	否
moveFailRetryTimes	指定迁移对象遇到非 4xx 类的错误时可以重试次数。 取值：整数形式，取值范围[-1, 2147483647]。默认值-1,代表一直重试。 **注意：**失败重试次数过大可能会引起下行流量和请求次数的增加。	否

3.3.3 log4j2.xml

迁移工具的日志配置文件 log4j2.xml，可以通过修改该配置文件，配置日志文件的大小（size）和日志文件保存的最大数量（max）等。

参数	描述
size	日志文件的大小。 取值：大于 0 的整数，单位为 KB、MB 或 GB。
max	日志文件保存的最大数量。 取值：大于 0 的整数。

默认的日志配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration status="warn" monitorInterval="30">
    <appenders>
        <Console name="stdout" target="SYSTEM_OUT">
            <PatternLayout charset="UTF-8">
                <Pattern>[%4p]( %F,%L ) [%d{yyyy-MM-dd HH:mm:ss}] [%t] %c - %m%n</Pattern>
            </PatternLayout>
        </Console>
        <!-- 记录迁移成功文件（Object）明细 -->
        <RollingFile name="successObjectInfo"
fileName="migratelog/successObjectLog_${sys:log4j.migratetime}.log"
            filePattern="migrateLog/successObjects_${sys:log4j.migratetime}_%i.log.zip">
            <PatternLayout charset="UTF-8">
                <Pattern>%m%n</Pattern>
            </PatternLayout>
            <Policies>
                <!--SizeBasedTriggeringPolicy:日志文件按照大小备份 -->
                <!--size:指定日志文件最大为 20MB，单位可以为 KB、MB 或 GB -->
                <SizeBasedTriggeringPolicy size="20MB" />
            </Policies>
            <!--DefaultRolloverStrategy:翻转策略决定如何执行备份 -->
            <!--max:最多保存 10 个备份文件 -->
            <DefaultRolloverStrategy max="10" fileIndex="min"/>
        </RollingFile>
        <!-- 记录跳过文件（Object）明细 -->
        <RollingFile name="skipObjectInfo"
fileName="migratelog/skipObjectLog_${sys:log4j.migratetime}.log"
            filePattern="migrateLog/skipObjects_${sys:log4j.migratetime}_%i.log.zip">
            <PatternLayout charset="UTF-8">
                <Pattern>%m%n</Pattern>
            </PatternLayout>
            <Policies>
                <SizeBasedTriggeringPolicy size="20MB" />
            </Policies>
            <DefaultRolloverStrategy max="10" fileIndex="min"/>
        </RollingFile>
        <!-- 记录迁移失败文件（Object）明细 -->
        <RollingFile name="errorObjectInfo"
fileName="migratelog/errorObjectLog_${sys:log4j.migratetime}.log"
                     filePattern="migrateLog/errorObjectLog_${sys:log4j.migratetime}_%i.log">
            <PatternLayout charset="UTF-8">
```

```
            <Pattern>%m%n</Pattern>
        </PatternLayout>
        <Policies>
            <SizeBasedTriggeringPolicy size="20MB" />
        </Policies>
        <DefaultRolloverStrategy max="10" fileIndex="min"/>
    </RollingFile>
    <!-- 记录其他文件（Object）明细 -->
    <RollingFile name="otherObjectInfo"
fileName="migratelog/otherObjectLog_${sys:log4j.migratetime}.log"
                 filePattern="migrateLog/errorObjectLog_${sys:log4j.migratetime}_%i.log">
        <PatternLayout charset="UTF-8">
            <Pattern>%m%n</Pattern>
        </PatternLayout>
        <Policies>
            <SizeBasedTriggeringPolicy size="20MB" />
        </Policies>
        <DefaultRolloverStrategy max="10" fileIndex="min"/>
    </RollingFile>

    <RollingFile name="info" fileName="logs/${sys:log4j.log.app}/server.log"
filePattern="logs/${sys:log4j.log.app}/server_%i.log.zip">
        <!--添加过滤器 ThresholdFilter,可以有选择的输出某个级别以上的类别 onMatch="ACCEPT"
onMismatch="DENY"意思是匹配就接受,否则直接拒绝 -->
        <ThresholdFilter level="info" onMatch="ACCEPT" onMismatch="DENY" />
        <PatternLayout charset="UTF-8">
            <Pattern>[%4p]( %F,%L ) [%d{yyyy-MM-dd HH:mm:ss.SSS}] [%t] %c - %m%n</Pattern>
        </PatternLayout>
        <Policies>
            <SizeBasedTriggeringPolicy size="100MB" />
        </Policies>
        <DefaultRolloverStrategy max="100" fileIndex="min"/>
    </RollingFile>

    <RollingFile name="debug" fileName="logs/${sys:log4j.log.app}/debug.log"
filePattern="logs/${sys:log4j.log.app}/debug_%i.log.zip">
        <ThresholdFilter level="debug" onMatch="ACCEPT"
            onMismatch="DENY" />
        <PatternLayout charset="UTF-8">
            <Pattern>[%4p]( %F,%L ) [%d{yyyy-MM-dd HH:mm:ss.SSS}] [%t] %c - %m%n</Pattern>
        </PatternLayout>
        <Policies>
            <SizeBasedTriggeringPolicy size="100MB" />
        </Policies>
```

```
            <DefaultRolloverStrategy max="100" fileIndex="min"/>
        </RollingFile>

        <RollingFile name="error" fileName="logs/${sys:log4j.log.app}/error.log"
filePattern="logs/${sys:log4j.log.app}/error_%i.log.zip">
            <ThresholdFilter level="error" onMatch="ACCEPT"
                onMismatch="DENY" />
            <PatternLayout charset="UTF-8">
                <Pattern>[%4p]( %F,%L ) [%d{yyyy-MM-dd HH:mm:ss.SSS}] [%t] %c - %m%n</Pattern>
            </PatternLayout>
            <Policies>
                <SizeBasedTriggeringPolicy size="100MB" />
            </Policies>
            <DefaultRolloverStrategy max="100" fileIndex="min"/>
        </RollingFile>

    </appenders>

    <!--然后定义 logger，只有定义了 logger 并引入的 appender，appender 才会生效 -->
    <loggers>
        <!-- 迁移日志配置 -->
        <AsyncLogger name="successObjectLog" level="info" additivity="false">
            <AppenderRef ref="successObjectInfo" />
        </AsyncLogger>
        <AsyncLogger name="skipObjectLog" level="info" additivity="false">
            <AppenderRef ref="skipObjectInfo" />
        </AsyncLogger>
        <AsyncLogger name="errorObjectLog" level="info" additivity="false">
            <AppenderRef ref="errorObjectInfo" />
        </AsyncLogger>
        <AsyncLogger name="otherObjectLog" level="info" additivity="false">
            <AppenderRef ref="otherObjectInfo" />
        </AsyncLogger>
        <!-- 系统日志配置 -->
        <AsyncRoot level="info" includeLocation="true">
            <!-- 如果需要输出到控制台，取消下面一行的注释 -->
            <!-- <AppenderRef ref="stdout"/> -->
            <!-- 如果需要输出 debug 日志，取消下面一行的注释，且将父节点 AsyncRoot 中的 level 属性的值改为
debug -->
            <!-- <AppenderRef ref="debug"/> -->
            <AppenderRef ref="info"/>
            <AppenderRef ref="error"/>
        </AsyncRoot>
    </loggers>
```

```
</configuration>
```

3.4 执行迁移

- 对于 Windows 客户端，执行 import.bat 启动迁移。
  1. 打开**开始**菜单，搜索 **cmd**，打开命令提示符窗口。
  2. cd 到压缩工具所在的目录下，例如 cd c:\CTYUN_OOS_Import_1.3.2。
  3. 执行 import.bat。

- 对于 Linux 客户端，执行 import.sh 启动迁移。
  1. 为 import.sh 增加执行权限，执行命令：chmod +x import.sh。
  2. 运行 import.sh，执行命令：./import.sh。

3.5 断点续传

数据迁移工具支持断点续传，如果迁移过程中程序被终止了，可以重新启动迁移任务，从之前中断的位置继续开始迁移。继续迁移的步骤如下：

1. 保留上次迁移执行过程中产生的 backup 文件。如果是在另外一台服务器上重新开始迁移任务，需要将 backup 文件拷贝到新服务器的迁移工具所在目录下。
2. （可选）查看 nextMarker.txt 文件中记录的上次数据迁移位置。修改 migrate.conf，设置 srcMarker 为上次迁移位置。

3.6 日志

数据迁移工具执行过程中，会生成一个 migratelog 文件夹，用于记录迁移情况。日志类型分为异常日志、成功日志、跳过文件（Object）日志、其他日志。

运行日志生成在 logs/migrate 文件夹下，根据 log4j2.xml 配置可生成 server.log、error.log、dubug.log 等。

4 迁移配置文件示例

- 迁移任务配置文件（migrate.conf）示例。
  - 数据从 OSS 迁移至 OOS 示例

```
{
    "srcType":"OSS", # 从阿里云迁移文件（Object）
    "srcEndpoint":"oss-cn-hangzhou.aliyuncs.com", # 阿里云的 Endpoint
    "srcAccessKey":"your oss accessKey", # 阿里云的 AccessKey
    "srcSecretKey":"your oss secretKey",  # 阿里云的 SecretKey
    "srcBucket":"ossbucket", # 阿里云的 Bucket
    "destEndpoint":"oos-cn.ctyunapi.cn", # OOS 的 Endpoint
    "destAccessKey":"your oos accesKey", # OOS 的 AccessKey
    "destSecretKey":"your oos secretKey", # OOS 的 SecretKey
    "destBucket":"oosbucket", # OOS 的 Bucket
    "srcPrefix":"logs/", # 阿里云上要迁移文件（Object）的前缀
    "srcMarker":"", # 从第一个文件（Object）开始迁移
    "srcStopObject":"", # 阿里云上要迁移的截止文件（Object）
    "isSkipExistFile":false # 是否跳过目标资源池中已有的文件（Object）
}
```

  - 数据从 OOS 迁移至 OOS 示例

```
{
    "srcType":"OOS", # 从 OOS 迁移文件（Object）
    "srcEndpoint":"oos-cn.ctyunapi.cn", # 源 OOS 的 Endpoint
    "srcAccessKey":"your oos accessKey",# 源 OOS 的 AccessKey
    "srcSecretKey":"your oos secretKey", # 源 OOS 的 SecretKey
    "srcBucket":"srcoosbucket", # 源 OOS 的 Bucket
    "destEndpoint":"oos-cn.ctyunapi.cn", # 目标 OOS 的 Endpoint
    "destAccessKey":"your oos accesKey", # 目标 OOS 的 AccessKey
    "destSecretKey":"your oos secretKey", # 目标 OOS 的 SecretKey
    "destBucket":"destoosbucket", # 目标 OOS 的 Bucket
    "srcPrefix":"logs/", # OOS 上要迁移文件（Object）的前缀
    "srcMarker":"", # 从第一个文件（Object）开始迁移
    "srcStopObject":"", # OOS 上要迁移的截止文件（Object）
    "isSkipExistFile":false # 是否跳过目标资源池中已有的文件（Object）
}
```

  - 数据从 COS 迁移至 OOS 示例

```
{
    "srcType":"COS", #从 COS 迁移文件（Object）
    "srcAccessKey":"your cos accessKey", # COS 的 AccessKey
    "srcSecretKey":"your cos secretKey", # COS 的 SecretKey
```

```
    "srcBucket":"cosbucket", # COS 的 Bucket
    "srcRegionName":"ap-beijing", # COS 的 region
   "destEndpoint":"oos-cn.ctyunapi.cn", # OOS 的 Endpoint
   "destAccessKey":"your oos accesKey", # OOS 的 AccessKey
   "destSecretKey":"your oos secretKey", # OOS 的 SecretKey
   "destBucket":"oosbucket", # OOS 的 Bucket
   "srcPrefix":"logs/", # COS 上要迁移文件（Object）的前缀
   "srcMarker":"", # 从第一个文件（Object）开始迁移
   "srcStopObject":"", # COS 上要迁移的截止文件（Object）
   "isSkipExistFile":false #是否跳过目标资源池中已有的文件（Object）
}
```

■ 数据从 OBS 迁移至 OOS 示例

```
{
    "srcType":"OBS", # 从 OBS 迁移文件（Object）
    "srcEndpoint":"obs.cn-north-4.myhuaweicloud.com", # OBS 的 Endpoint
   "srcAccessKey":"your obs accessKey", # OBS 的 AccessKey
   "srcSecretKey":"your obs secretKey",  # OBS 的 SecretKey
    "srcBucket":"obsbucket", # OBS 的 bucket
   "destEndpoint":"oos-cn.ctyunapi.cn", # OOS 的 Endpoint
   "destAccessKey":"your oos accesKey", # OOS 的 AccessKey
   "destSecretKey":"your oos secretKey", # OOS 的 SecretKey
   "destBucket":"oosbucket", # OOS 的 Bucket
   "srcPrefix":"logs/", # OBS 上要迁移文件（Object）的前缀
   "srcMarker":"", # 从第一个文件（Object）开始迁移
   "srcStopObject":"", # OBS 上要迁移的截文件（Object）
   "isSkipExistFile":false # 是否跳过目标资源池中已有的文件（Object）
}
```

■ 数据从 S3 迁移至 OOS 示例

```
{
    "srcType":"S3", # 从 S3 迁移文件（Object）
    "srcEndpoint":"s3.amazonaws.com", # S3 的 Endpoint
   "srcAccessKey":"your s3 accessKey", # S3 的 AccessKey
   "srcSecretKey":"your s3 secretKey", # S3 的 SecretKey
    "srcBucket":"s3bucket", # S3 的 Bucket
    "srcRegionName":"us-east-1", # S3 的 region
   "destEndpoint":"oos-cn.ctyunapi.cn", # OOS 的 Endpoint
   "destAccessKey":"your oos accesKey", # OOS 的 AccessKey
   "destSecretKey":"your oos secretKey", # OOS 的 SecretKey
   "destBucket":"oosbucket", # OOS 的 Bucket
   "srcPrefix":"logs/", # S3 上要迁移文件（Object）的前缀
   "srcMarker":"", # 从第一个文件（Object）开始迁移
   "srcStopObject":"", # S3 上要迁移的截止文件（Object）
```

```
    "isSkipExistFile":false # 是否跳过目标资源池中已有的文件（Object）
}
```

- 数据从 LOCAL 迁移至 OOS

```
{
    "srcType":"LOCAL", # 从 LOCAL 迁移文件（Object）
    "localFolderPath":"F:/test/test1/", # 本地目录
    "destEndpoint":"oos-cn.ctyunapi.cn", # OOS 的 Endpoint
    "destAccessKey":"your oos accesKey", # OOS 的 AccessKey
    "destSecretKey":"your oos secretKey", # OOS 的 SecretKey
    "destBucket":"oosbucket", # OOS 的 Bucket
    "isSkipExistFile":false # 是否跳过目标资源池中已有的文件（Object）

}
```

- 根据日志文件迁移源文件（Object）至 OOS（以 LOCAL 迁移至 OOS 为例）

```
{
    "srcType":"LOCAL", # 从 LOCAL 迁移文件（Object）
    "localFolderPath":"F:/test/test1/", # 本地目录
    "destEndpoint":"oos-cn.ctyunapi.cn", # OOS 的 Endpoint
    "destAccessKey":"your oos accesKey", # OOS 的 AccessKey
    "destSecretKey":"your oos secretKey", # OOS 的 SecretKey
    "destBucket":"oosbucket", # OOS 的 Bucket
    "isSkipExistFile":false # 是否跳过目标资源池中已有的文件（Object）
    "migrateLogFile":true,
    "logFile":"F:/errorlog.txt"
}
```

- 系统配置文件（system.conf）示例

```
{
    "threadNum":1,  # 迁移时的读写并发数
    "maxSimpleObjectSizeM":100, # 超过文件（Object）大小限制时，将被拆分成分段文件（Object）迁移
    "partSizeM":50, # 拆分为分段文件（Object）时的分片大小
    "maxThroughput":-1, # 对源端流量进行限制，负数表示不限流量
    "stopScan":false # 是否终止遍历源文件（Object）
}
```

5 常见问题

1. 怎么查看迁移进度？

   A：可以在 statistics.txt 中查看迁移进度。例如

```
<html>
<body>
<div>已加载备份 object 数:0<div>
<div>已扫描 object 数/已扫描 object 量:31/32.26(MB)<div>
<div>已完成 object 数/已完成 object 量:31/32.26(MB)<div>
<div>成功 object 数/成功 object 量:29/32.16(MB)<div>
<div>失败 object 数/失败 object 量:2/100.01(KB)<div>
<div>跳过 object 数/跳过 object 量:0/0.00(B)<div>
<div>其他 object 数/其他 object 量:0/0.00(B)<div>
<div>总体速度：扫描速度/已完成速度/成功速度:275(KB/s)/275(KB/s)/274(KB/s)<div>
<div>分钟速度：扫描速度/已完成速度/成功速度:75(KB/s)/75(KB/s)/74(KB/s)<div>
<div>成功数占完成比/成功量占完成比/单位成功量:80%/80%/5M<div>
<div>失败数占完成比/失败量占完成比/单位失败量:20%/20%/1M<div>
<div>待迁移队列长度/正在迁移队列长度:0/0<div>
<div>启动时间/完成时间:2020-07-20 09:15:13/2020-07-20 09:16:55<div>
<div>扫描状态:已停止<div>
<div>加速模式:已生效<div>
<div>并发数(每个客户端):100<div>
<div>客户端数量:0<div>
<div>客户端编号:<div>
</body>
</html>
```

2. 此工具支持 OOS 同一个 Bucket 内文件（Object）的迁移吗？

   A：如果在 migrate.conf 中增加参数 destPrefix，可以使用此迁移工具实现 OOS 同一个 Bucket 内文件（Object）的迁移。

3. 对源数据的存储类型是否有要求？

   A:不支持 S3 归档存储类型的数据，其他基本都支持。

4. 如果文件（Object）存储在中国大陆外的 S3 region，是否可以迁移？

   A：可以迁移，但可能读取数据较慢，导致迁移较慢甚至有超时风险。

5. 如果数据有加密，会影响迁移吗？

   A：不支持迁移使用了服务端加密且加密方式为 SSE-C 的文件（Object）。

6. 对于分段文件（Object）有什么要求？

   A：OSS、COS、OBS、S3，对于分段文件（Object），仅可以遍历到已合并的分段文件，即仅迁移已合并的分段文件，未合并、已 abort 的不迁移。

7. 是否支持文件（Object）的多版本迁移？

   A：OSS、COS、OBS、S3、OOS 文件（Object），仅迁移当前可读取到的文件（Object），不支持按 object version Id 迁移。

8. 迁移源类型为 LOCAL 的，OOS 上文件（Object）名称是否和本地的名称一样？

   A：不一定。迁移工具迁移数据时会遍历给定目录下的所有文件、子目录及子目录下文件，文件（Object）名为文件路径去掉目录参数部分：

   - 如果文件（Object）直接属于指定的目录，OOS 上文件（Object）名称和本地的名称一致。如指定目录参数为 F:/test/，文件 test1.txt 在本地的文件目录为 F:/test/test1.txt，则 OOS 上的文件名称为 test1.txt。
   - 如果文件（Object）不直属于指定的目录，则 OOS 上文件名称需要增加未指定的目录名部分。如指定目录参数为 F:/test/，文件 test2.txt 在本地的文件目录为 F:/test/sub/test2.txt，则 OOS 上的文件名称为 sub/test2.txt。

9. 是否支持配置迁移文件（Object）的大小？

   A：支持，可以通过配置文件 migrate.conf 中参数 objectSize 进行配置迁移文件（Object）的范围。格式是 N-M，表示迁移 N 至 M 大小的文件。取值：N 和 M 是大于等于 0 的整数，且 N ≤M，单位是字节。默认不配置此项，表示迁移所有大小的文件。

10. 迁移过程中任务被终止了如何继续？

    A：可以按照下列步骤继续终止的任务：

1) 保留上次迁移执行过程中产生的 backup 文件。

说明：如果不在原服务器上继续执行迁移任务，而是换一台服务器上重新开始迁移任务，则需要将 backup 文件拷贝到新服务器的迁移工具所在目录下。

2) 查看 nextMarker.txt 文件中记录的上次数据迁移位置。修改迁移任务配置文件(migrate.conf)，设置 srcMarker 为上次迁移位置，然后运行迁移工具。

11. 如何提升迁移速度？

A：在客户端网络环境不变的情况下，可以通过调整如下参数来提升迁移速度：

- 将系统配置文件（system.conf）中的 threadNum 参数调大，执行多线程并发迁移，譬如调整到 50。
- 将迁移任务配置文件（migrate.conf）中的 isSkipExistFile 参数设置为 true，当目标资源池中已有同名文件则跳过不再迁移。

  说明：这个参数根据实际情况而定。
- 如果 srcType 为 OOS 时，且迁移源端和目的端的资源池为同类型，可以将迁移任务配置文件（migrate.conf）中的 isAcceleratedMigration 参数设置为 true，使用加速迁移。
- 将迁移任务配置文件（migrate.conf）中的 importSince 参数设置为增量迁移的时间戳，可以只迁移该时间戳之后的文件。

12. 如何开始一个全新的迁移任务？

A：按下列步骤开始一个全新的迁移任务：

1) 确认迁移工具目录下没有 backup 目录或者 backup 目录下没有文件。
2) 确认系统配置文件（system.conf）和任务配置文件（migrate.conf）中的参数正确。
3) 在客户端执行迁移任务：
   - 如果客户端为 Windows，执行 import.bat。
   - 如果客户端为 Linux，执行 import.sh。