阿里云
日志服务

数据投递

文档版本：20190910

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。
2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。
3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。
4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的”现状“、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。
5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含”阿里云”、Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。
6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **禁止:** 重置操作将丢失用户配置数据。
	该类警示信息可能导致系统重大变更甚至故障，或者导致人身伤害等结果。	 **警告:** 重启操作将导致业务中断，恢复业务所需时间约10分钟。
	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 **说明:** 您也可以通过按Ctrl + A选中全部文件。
>	多级菜单递进。	设置 > 网络 > 设置网络类型
粗体	表示按键、菜单、页面名称等UI元素。	单击 确定。
`courier` 字体	命令。	执行 `cd /d C:/windows` 命令，进入Windows系统文件夹。
*## *	表示参数、变量。	`bae log list --instanceid` *`Instance_ID`*
[]或者[a|b]	表示可选项，至多选择一个。	`ipconfig` *`[-all|-t]`*
{}或者{a|b}	表示必选项，至多选择一个。	`swich` *`{stand | slave}`*

目录

1 简介

将日志源接入日志服务后，日志服务开始实时采集日志，并提供控制台或SDK/API方式的日志消费和日志投递功能。日志服务采集到LogHub的日志数据，可以实时投递至OSS、Table Store、MaxCompute等存储类阿里云产品，您只需要在控制台配置即可完成，同时，LogShipper提供完整状态API与自动重试功能。

应用场景

对接数据仓库

日志数据来源

日志服务的日志投递功能所投递的日志数据来源于日志服务采集到LogHub的日志。这部分日志生成之后，被日志服务实时采集并投递至其他云产品中进行存储与分析。

日志投递目标

- OSS（大规模对象存储）：
  - #unique_4
  - OSS 上格式可以通过 Hive 处理，推荐 E-MapReduce。
  - 投递后支持通过阿里云DLA分析数据。
- Table Store（NoSQL 数据存储服务）：#unique_5
- MaxCompute (大数据计算服务):
  - 直接投递-#unique_6
  - 通过DataWorks数据集成投递-#unique_7

2 管理日志投递任务

投递日志是日志服务的一个功能，能够帮助您最大化数据价值。您可以选择将收集到的日志数据通过控制台方式投递至MaxCompute，做数据长期存储或联合其它系统（如 E-MapReduce）消费数据。一旦启用日志投递功能，日志服务后台会定时把写入到该日志库内的日志投递到对应云产品中。为方便您及时了解投递进度，处理线上问题，日志服务控制台提供了 日志投递任务管理 页面，您可以查询指定时间内的数据投递状态。

在Logstore列表界面，单击日志投递列下的或MaxComputeOSS，进入日志投递任务管理页面。您可以执行以下操作管理您的投递任务：

开启/关闭投递任务

1. 日志投递任务管理页面左上角菜单中选择目标Logstore。
2. 单击Logstore菜单旁的开启投递或关闭投递。

   关闭投递任务后再次开启任务，需要重新配置投递规则。

关闭投递任务后再次开启任务，需要重新配置投递规则。

修改投递规则

成功创建投递任务之后，可以单击修改属性修改投递规则。

查看投递任务详情

根据Logstore、时间段和任务投递状态筛选出需要查看的投递任务后，可以在当前页面中查看指定投递任务状态、开始时间、结束时间、接受日志数据时间、任务类型等详细信息。

任务的投递状态有以下三种：

状态	含义	操作
成功	投递任务正常运行状态。	无须关注。
进行中	投递任务进行中。	请稍后查看是否投递成功。
失败	日志数据投递失败。投递任务因外部原因出现了错误，且无法重试。如MaxCompute表结构不符合日志服务规范、无授权等。	请排查投递问题。

删除投递配置

操作步骤

1. 在Logstore列表页面，单击删除规则。
2. 在弹出的对话框中单击确定。

   删除后将不能再创建同名的离线归档配置，请慎重选择。

3 投递日志到OSS

3.1 投递流程

日志服务采集到日志数据后，支持将日志数据投递至OSS中进行存储与分析。本文档主要介绍将日志数据投递至OSS的操作步骤。

前提条件

- 已开通日志服务，创建Project和Logstore，并成功采集到日志数据，请参见#unique_11和#unique_12。
- 已开通OSS服务，并在日志服务Project所在的地域创建Bucket，请参见#unique_13。
- 已开通访问控制RAM。

背景信息

日志服务可以把Logstore中的数据自动归档到OSS，以发挥日志更多的效用。

- OSS 数据支持自由设置生命周期，可以对日志进行长期存储。
- 可以通过自建程序和更多系统（如E-MapReduce和DLA）消费OSS数据。

功能优势

通过日志服务投递日志数据到OSS具有如下优势：

- 操作简单。仅需在控制台上做简单配置即可将日志服务Logstore的数据投递到OSS。
- 效率提升。日志服务的日志收集过程已经完成不同机器上的日志集中化，无需在不同机器上重复收集日志导入OSS。
- 便于管理。投递日志到OSS可以充分复用日志服务内的日志分类管理功能。用户可让日志服务不同项目（Project）、不同类型（Logstore）的日志自动投递到不同的OSS Bucket目录，方便管理OSS数据。

注意事项

- 日志服务Project和OSS的Bucket必须位于相同Region，不支持跨Region投递数据。
- 金融云和公共云之间可以投递。

操作步骤

1. 访问控制（RAM）授权。

   开启投递任务之前，您需要为日志服务授权，允许日志数据写入OSS。

   单击快捷授权，在弹出页面中单击同意授权。成功授权后，日志服务具备OSS的数据写入权限。

   > 说明:
   > - 修改授权策略、跨阿里云账号配置投递任务，请参见RAM授权。
   > - 不涉及跨阿里云账号时，子账号配置投递任务请参见#unique_15，为子账号授予权限。

2. 配置OSS投递规则。

   a) 登录日志服务控制台，单击Project名称。

   b) 单击  依次展开Lostore名称 > 数据处理 > 导出。

   c) 单击OSS（对象存储），打开OSS投递管理页面。

   d) 单击开启投递，设置OSS投递配置并单击确认。

   请参考下表设置OSS投递配置。

配置项	说明	取值范围
OSS投递名称	您所创建的投递的名称。	只能包含小写字母，数字，连字符（-）和下划线（_），必须以小写字母和数字开头和结尾，且名称长度为3~63字节。
OSS Bucket	OSS Bucket名称。	必须是已存在的Bucket名称，且需要保证OSS的Bucket与日志服务Project位于相同Region。
OSS Prefix	OSS前缀，从日志服务同步到OSS的数据将存放到Bucket的该目录下。	必须是已存在的OSS Prefix名称。

配置项	说明	取值范围
分区格式	将投递任务创建时间使用%Y，%m，%d，%H，%M等格式化生成分区字符串，以此来定义写到OSS的Object文件所在的目录层次结构，其中斜线/表示一级OSS目录。如下表举例说明OSS Prefix和分区格式如何定义OSS目标文件路径。	格式化参考strptime API。
RAM角色	RAM角色ARN及角色名称，用于访问权限控制，OSS Bucket拥有者创建角色的标识，如何获得ARN请参见图 3-2: 获取角色ARN。	如`acs:ram::45643:role/aliyunlogdefaultrole`。
投递大小	自动控制投递任务创建间隔并设置OSS的一个Object大小（以未压缩计算）上限。	取值范围为5~256，单位为MB。
存储格式	日志数据投递OSS的存储格式。	支持JSON/Parquet/CSV三种格式，配置细节请单击查看：JSON格式、Parquet格式、CSV格式。
是否压缩	OSS数据存储的压缩方式。	· 不压缩：表示原始数据不压缩。 · 压缩（snappy）：表示使用snappy算法对数据做压缩，可以减少OSS Bucket存储空间使用量。
是否投递tag	选择是否投递日志的标签。	选择是或者否。

配置项	说明	取值范围
投递时间	投递任务的间隔时长。	取值范围为300~900，默认值300。单位为秒。

图 3-1: 投递日志

图 3-2: 获取角色ARN

> **说明:**
> 日志服务在后端并发执行数据投递，对写入每个shard的数据单独进行服务。每一个Shard都会根据投递大小、投递时间决定任务生成的频率，当任一条件满足时，即会创建投递任务。

分区格式

每个投递任务会写入OSS一个文件，路径格式是`oss:// OSS-BUCKET/OSS-PREFIX/PARTITION-FORMAT_RANDOM-ID`。PARTITION-FORMAT是根据投递任务创建时间格式化而得到的，以创建时间为2017/01/20 19:50:43的投递任务为例，说明分区格式的用法：

OSS Bucket	OSS Prefix	分区格式	OSS文件路径
test-bucket	test-table	%Y/%m/%d/%H/%M	*oss://test-bucket/test-table/2017/01/20/19/50_1484913043351525351_2850008*
test-bucket	log_ship_oss_example	year=%Y/mon=%m/day=%d/log_%H%M%s	*oss://test-bucket/log_ship_oss_example/year=2017/mon=01/day=20/log_195043_1484913043351525351_2850008.parquet*
test-bucket	log_ship_oss_example	ds=%Y%m%d/%H	*oss://test-bucket/log_ship_oss_example/ds=20170120/19_1484913043351525351_2850008.snappy*
test-bucket	log_ship_oss_example	%Y%m%d/	*oss://test-bucket/log_ship_oss_example/20170120/_1484913043351525351_2850008*

OSS Bucket	OSS Prefix	分区格式	OSS文件路径
test-bucket	log_ship_oss_example	%Y%m%d%H	*oss://test-bucket/log_ship_oss_example/20170120 19_148491304 3351525351_2850008*

使用Hive、MaxCompute等大数据平台或阿里云DLA产品分析OSS数据时，如果希望使用Partition信息，可以设置每一层目录上为key=value格式（Hive-style partition）。

例如：`oss://test-bucket/log_ship_oss_example/year=2017/mon=01/day=20/log_195043_1484913043351525351_2850008.parquet`可以设置三层分区列，分别为：year、mon、day。

日志投递任务管理

在启动OSS投递功能后，日志服务后台会定期启动投递任务。您可以在控制台OSS投递管理界面上看到投递任务的状态。

通过日志投递任务管理，您可以：

- 查看过去两天内的所有日志投递任务，了解其状态。投递任务状态可以是“成功”、“进行中”和“失败”。“失败”状态则表示您的投递任务出现了因外部原因而无法重试的错误，需要您参与解决问题。

- 一般情况下，日志数据在写入Logstore后的30分钟内同步到OSS。如果投递任务执行失败，控制台上会显示相应的错误信息，系统会按照策略默认为您重试，您也可以手动重试。
  - 日志服务默认会按照退火策略重试最近两天之内的任务，重试等待的最小间隔是15分钟。当任务执行失败时，第一次失败需要等待15分钟再试，第二次失败需要等待30分钟（2 乘以 15）再试，第三次失败需要等待60分钟（2 乘以 30）再试，以此类推。
  - 如需立即重试失败任务，可以通过控制台单击重试全部失败任务或通过API/SDK方式指定任务进行重试。

  常见失败任务的错误信息如下：

错误信息	错误原因	处理方法
UnAuthorized	没有权限。	请确认以下配置： - OSS 用户是否已创建角色。 - 角色描述的账号 ID 是否正确。 - 角色是否授予OSS Bucket写权限。 - role-arn是否配置正确。
ConfigNotExist	配置不存在。	一般是由于删除投递规则导致，如又重新创建了规则，可以通过重试来解决。
InvalidOssBucket	OSS Bucket 不存在。	请确认以下配置： - OSS Bucket所在Region是否与日志服务Project一致。 - Bucket名称是否配置正确。
InternalServerError	日志服务内部错误。	通过重试解决。

OSS 数据存储

可以通过控制台、API/SDK或其它方式访问OSS数据。

如使用Web管理控制台访问，进入OSS服务，选择Bucket，单击 文件管理即可看到有日志服务投递过来的数据。

更多OSS使用步骤请参见OSS文档。

Object 地址

```
oss:// OSS-BUCKET/OSS-PREFIX/PARTITION-FORMAT_RANDOM-ID
```

- 路径字段说明
  - OSS-BUCKET、OSS-PREFIX表示OSS的Bucket名称和目录前缀，由用户配置，INCREMENTID是系统添加的随机数。
  - PARTITION-FORMAT定义为%Y/%m/%d/%H/%M，其中%Y，%m，%d，%H，%M分别表示年、月、日、小时、分钟，由本次投递任务的服务端创建时间通过strptime API计算得到。
  - RANDOM-ID是一个投递任务的唯一标识。
- 目录的时间含义

  OSS数据目录是按照投递任务创建时间设置的，假设5分钟数据投递一次OSS，2016-06-23 00:00:00创建的投递任务，它投递的数据是2016-06-22 23:55后写入日志服务的数据。如需分析完整的2016-06-22全天日志，除了2016/06/22目录下的全部object以外，还需要检查`2016/06/23/00/`目录下前十分钟的Object是否有包含2016-06-22时间的日志。

Object存储格式

- JSON

  请参见JSON格式。
- Parquet

  请参见Parquet格式。
- CSV

  请参见CSV格式。

3.2 JSON格式

本文档主要介绍日志服务投递OSS使用JSON存储的相关配置，关于投递日志到OSS的其它内容请参考投递流程。

OSS文件压缩类型及文件地址见下表。

压缩类型	文件后缀	OSS文件地址举例
不压缩	无	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937

压缩类型	文件后缀	OSS文件地址举例
snappy	.snappy	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.snappy

不压缩

Object由多条日志拼接而成，文件的每一行是一条JSON格式的日志，样例如下：

```
{"__time__":1453809242,"__topic__":"","__source__":"10.170.***.***","
ip":"10.200.**.***","time":"26/Jan/2016:19:54:02 +0800","url":"POST
                /PutData?Category=YunOsAccountOpLog&AccessKeyId=<
yourAccessKeyId>&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&
Topic=raw&Signature=<yourSignature>
                HTTP/1.1","status":"200","user-agent":"aliyun-sdk-java"}
```

snappy压缩

当前支持通过以下方式解压缩，解压后为原格式文件。

详细说明请查看#unique_21：

- 使用 C++ Lib 解压缩
- 使用 Java Lib解压缩
- 使用Linux 环境解压工具
- Python解压方案

3.3 CSV格式

本文介绍日志服务投递OSS使用CSV存储的相关细节，其它内容请参考投递流程。

CSV存储字段配置

配置页面

您可以在日志服务数据预览或索引查询页面查看一条日志的多个Key-Value，将你需要投递到OSS的字段名（Key）有序填入。

如您配置的Key名称在日志中找不到，CSV行中这里一列值将设置为空值字符串（null）。

图 3-3: 配置项

配置项

配置项	取值	备注
分隔符 delimiter	字符	长度为1的字符串，用于分割不同字段。
转义符 quote	字符	长度为1的字符串，字段内出现分隔符（delimiter）或换行符等情况时，需要用quote前后包裹这个字段，避免读数据时造成字段错误切分。

配置项	取值	备注
跳出符 escape	字符	长度为1的字符串，默认设置与quote相同，暂不支持修改。字段内部出现quote（当成正常字符而不是转义符）时需要在quote前面加上escape做转义。
无效字段内容 null	字符串	当指定Key值不存在时，字段填写该字符串表示该字段无值。
投递字段名称 header	布尔	是否在csv文件的首行加上字段名的描述。

更多内容请参考CSV标准、postgresql CSV说明。

可配置的保留字段

在投递OSS过程中，除了使用日志本身的Key-Value外，日志服务同时提供以下几个保留字段可供选择：

保留字段	语义
`__time__`	日志的Unix时间戳（是从1970年1月1日开始所经过的秒数），由用户日志字段的time计算得到。
`__topic__`	日志的topic。
`__source__`	日志来源的客户端IP。

JSON格式存储会默认带上以上字段内容。

CSV存储可以根据您的需求自行选择。例如您需要日志的topic，那么可以填写字段名：`__topic__`。

OSS存储地址

压缩类型	文件后缀	OSS文件地址举例
无压缩	.csv	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.csv
snappy	.snappy.csv	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.snappy.csv

- CSV是可读格式，未压缩的文件可以直接从OSS下载以文本形式打开查看。
- 如果使用了snappy压缩，解压缩的详细说明请查看#unique_21。

数据消费

HybridDB

建议配置如下：

- 分隔符delimiter：逗号（,）。
- 转义符quote：双引号（“）。
- 无效字段内容null：不填写（空）。
- 投递字段名称header：不勾选（HybirdDB默认csv文件行首无字段说明）。

3.4 Parquet格式

本文档主要介绍日志服务投递OSS使用Parquet存储的相关配置，关于投递日志到OSS的其它内容请参考投递流程。

Parquet存储字段配置

数据类型

Parquet存储支持6种类型：string、boolean、int32、int64、float、double。

日志投递过程中，会将日志服务数据由字符串转换为Parquet目标类型。如果转换到非String类型失败，则该列数据为null。

列配置

请依次填写Parquet中需要的日志服务数据字段名和目标数据类型，在投递时将按照该字段顺序组织Parquet数据，并使用日志服务的字段名称作为Parquet数据列名，以下两种情况发生时将置数据列值为null：

- 该字段名在日志服务数据中不存在。

- 该字段由string转换非string（如double、int64等）失败。

图 3-4: 字段配置

可配置的保留字段

在投递OSS过程中，除了使用日志本身的Key-Value外，日志服务保留同时提供以下几个保留字段可供选择：

保留字段	语义
`__time__`	日志的Unix时间戳（是从1970年1月1日开始所经过的秒数），由用户日志字段的time计算得到。
`__topic__`	日志的Topic。
`__source__`	日志来源的客户端IP。

JSON格式存储会默认带上以上字段内容。

Parquet、CSV存储可以根据您的需求自行选择。例如您需要日志的Topic，那么可以填写字段名：`__topic__`，字段类型string。

OSS存储地址

压缩类型	文件后缀	OSS文件地址举例
无外部压缩	.parquet	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.parquet
snappy	.snappy.parquet	oss://oss-shipper-shenzhen/ecs_test/2016/01/26/20/54_1453812893059571256_937.snappy.parquet

数据消费

E-MapReduce / Spark / Hive

请参考社区文档。

单机校验工具

开源社区提供的parquet-tools可以用来文件级别验证Parquet格式、查看schema、读取数据内容。

您可以自行编译该工具或者点击下载日志服务提供的版本。

- 查看Parquet文件schema

```
$ java -jar parquet-tools-1.6.0rc3-SNAPSHOT.jar schema -d 00_1490803532136470439_124353.snappy.parquet | head -n 30
message schema {
  optional int32 __time__;
  optional binary ip;
  optional binary __source__;
  optional binary method;
  optional binary __topic__;
  optional double seq;
  optional int64 status;
  optional binary time;
  optional binary url;
  optional boolean ua;
}
creator: parquet-cpp version 1.0.0
file schema: schema
--------------------------------------------------------------------------------
__time__: OPTIONAL INT32 R:0 D:1
ip: OPTIONAL BINARY R:0 D:1
.......
```

- 查看Parquet文件全部内容

```
$ java -jar parquet-tools-1.6.0rc3-SNAPSHOT.jar head -n 2 00_1490803532136470439_124353.snappy.parquet
__time__ = 1490803230
ip = 10.200.98.220
__source__ = *.*.*.*
method = POST
```

```
__topic__ =
seq = 1667821.0
status = 200
time = 30/Mar/2017:00:00:30 +0800
url = /PutData?Category=YunOsAccountOpLog&AccessKeyId=**********&
Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&
Signature=******************************** HTTP/1.1
__time__ = 1490803230
ip = 10.200.98.220
__source__ = *.*.*.*
method = POST
__topic__ =
seq = 1667822.0
status = 200
time = 30/Mar/2017:00:00:30 +0800
url = /PutData?Category=YunOsAccountOpLog&AccessKeyId=**********&
Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&Topic=raw&
Signature=******************************** HTTP/1.1
```

更多用法请执行：`java -jar parquet-tools-1.6.0rc3-SNAPSHOT.jar -h`，参考帮助。

3.5 Snappy压缩文件

将日志服务中的日志数据投递到OSS中时，可以设置压缩方式为snappy压缩，解压时可以通过C++ Lib、Java Lib、Python Lib和Linux 环境解压工具进行解压缩。

投递日志到OSS时选择snappy算法对数据做压缩，可以减少 OSS Bucket 存储空间使用量。本文档提供以下解压方案。

- 使用 C++ Lib 解压缩
- 使用 Java Lib解压缩
- 使用 Python Lib解压缩
- 使用Linux 环境解压工具

使用 C++ Lib 解压缩

snappy 官网 右侧下载 Lib，执行 Snappy.Uncompress 方法解压。

使用 Java Lib解压缩

下载xerial snappy-java，使用 Snappy.Uncompress 或 Snappy.SnappyInputStream解压缩，不支持 SnappyFramedInputStream。

说明：

1.1.2.1 版本存在 bug，可能无法解压部分压缩文件，1.1.2.6及以后版本已修复该问题，建议使用最新的Java Lib解压缩。

```
<dependency>
<groupId>org.xerial.snappy</groupId>
<artifactId>snappy-java</artifactId>
```

```
<version>1.0.4.1</version>
<type>jar</type>
<scope>compile</scope>
</dependency>
```

- Snappy.Uncompress

```
String fileName = "C:\\我的下载\\36_1474212963188600684_4451886.
snappy";
RandomAccessFile randomFile = new RandomAccessFile(fileName, "r");
int fileLength = (int) randomFile.length();
randomFile.seek(0);
byte[] bytes = new byte[fileLength];
int byteread = randomFile.read(bytes);
System.out.println(fileLength);
System.out.println(byteread);
byte[] uncompressed = Snappy.uncompress(bytes);
String result = new String(uncompressed, "UTF-8");
System.out.println(result);
```

- Snappy.SnappyInputStream

```
String fileName = "C:\\我的下载\\36_1474212963188600684_4451886.
snappy";
SnappyInputStream sis = new SnappyInputStream(new FileInputStream(
fileName));
byte[] buffer = new byte[4096];
int len = 0;
while ((len = sis.read(buffer)) != -1) {
    System.out.println(new String(buffer, 0, len));
}
```

使用 Python Lib解压缩

1. 下载并安装Python压缩库。
2. 执行解压代码。

   解压缩代码示例：

```
import snappy
compressed = open('/tmp/temp.snappy').read()
snappy.uncompress(compressed)
```

说明：

不支持通过以下命令行工具解压OSS投递的snappy压缩文件，命令行模式下仅支持hadoop模式（hadoop_stream_decompress）与流模式（stream_decompress）。

```
$ python -m snappy -c uncompressed_file compressed_file.snappy
```

```
$ python -m snappy -d compressed_file.snappy uncompressed_file
```

Linux 环境解压工具

针对 Linux 环境，日志服务提供可以解压 snappy文件的工具，单击下载 snappy_tool。

```
./snappy_tool 03_1453457006548078722_44148.snappy 03_14534570065480787
22_44148
compressed.size: 2217186
snappy::Uncompress return: 1
uncompressed.size: 25223660
```

3.6 RAM授权

配置OSS投递任务之前，OSS Bucket的拥有者需要配置快捷授权，授权完成后，当前账号的日志服务有权限对OSS Bucket进行写入操作。

本文为您介绍配置OSS投递任务中多种场景下的RAM授权操作。

- 如您需要对OSS Bucket进行更细粒度的访问控制，请参考修改授权策略。
- 如果日志服务Project和OSS Bucket不是同一阿里云账号创建的，请参考跨阿里云账号投递。
- 如果子账号需要将当前主账号的日志数据投递至同一账号下的OSS Bucket，请参考子账号直接投递。
- 如果子账号需要将日志数据投递至其他阿里云账号下的OSS Bucket，请参考子账号的跨账号投递。

修改授权策略

通过快捷授权，角色 AliyunLogDefaultRole 默认被授予 AliyunLogRolePolicy，日志服务具有所有 OSS Bucket 的写入权限。

如您需要更精细的访问控制粒度，请解除角色 AliyunLogDefaultRole 的 AliyunLogRolePolicy 授权，并参考 《OSS用户指南》 创建更细粒度的权限策略，授权给角色 AliyunLogDefaultRole。

跨阿里云账号投递

如您的日志服务Project和OSS Bucket不是同一阿里云账号创建的，您需要按照以下步骤配置授权策略。

例如，需要将A账号下的日志服务数据投递至B账号创建的OSS Bucket中。

1. 主账号B通过快捷授权创建角色AliyunLogDefaultRole，并授予其OSS的写入权限。

2. 在访问控制RAM控制台中单击左侧导航栏RAM角色管理，找到AliyunLogDefaultRole，并单击角色名称以查看该角色的基本信息。

   其中，该角色描述中Service配置项定义了角色的合法使用者，例如log.aliyuncs.com表示当前账号可扮演此角色以获取OSS的写入权限。

3. 在Service配置项中添加角色描述A_ALIYUN_ID@log.aliyuncs.com。账号A的账号ID请在账号管理 > 安全设置中查看。

   例如A 的主账号 ID 为165421896534****，更改后的账号描述为：

   ```
   {
   "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "165421896534****@log.aliyuncs.com",
          "log.aliyuncs.com"
        ]
      }
    }
   ],
   "Version": "1"
   }
   ```

   该角色描述表示账号A有权限通过日志服务获取临时 Token 来操作 B 的资源，关于角色描述的更多信息请参考#unique_31。

4. A账号创建投递任务，并在配置投递任务时，RAM角色一栏填写OSS Bucket拥有者的RAM角色标识ARN，即账号B通过快速授权创建的RAM角色AliyunLogDefaultRole。

   RAM角色的ARN可以在该角色的基本信息中查看，格式为acs:ram::13234:role/logrole。

子账号直接投递

如果希望授予名下子账号创建OSS投递任务等权限，那么需要由主账号登录RAM控制台，为子账号授权。授权范围除OSS投递相关权限之外，还需要包括 PassRole 权限。所以，您需要按照以下步骤创建自定义授权策略并绑定到子账号上。

1. 主账号在RAM控制台创建自定义授权策略。

   示例授权策略如下：

   > **说明:**
   > 授权策略中需要包含PassRole权限。

   ```
   {
     "Version": "1",
   ```

```
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "log:*",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ram:PassRole",
      "Resource": "*"
    }
  ]
}
```

2. 将此自定义授权策略赋予子账号。

   a. 在用户页面，单击指定子账号右侧的添加权限。

   b. 在选择权限中查找以上自定义授权策略，并将其加入已选择，并单击确定。

子账号的跨账号投递

如果主账号A的子账号A1创建投递任务，将A账号的日志数据投递至B账号的OSS Bucket中，需要由主账号A为子账号A1授予PassRole权限。

配置步骤如下：

1. 参考跨阿里云账号投递，主账号B配置快捷授权，并添加角色描述。
2. 主账号A登录访问控制RAM控制台，为子账号A1授予AliyunRAMFullAccess权限。

   a. 主账号A在用户页面，单击子账号A1右侧的添加权限。

   b. 在选择权限中查找AliyunRAMFullAccess，将其加入已选择，并单击确定。

      授权成功后，子账号A1将具备RAM所有权限。

      如果您需要控制A1的权限范围，只授予投递 OSS 的必要权限，可以修改授权策略的`Action`和`Resource`参数。

      示例授权策略如下，您需要将`Resource`的内容替换为AliyunLogDefaultRole的角色ARN。

```
{
"Statement": [
{
"Action": "ram:PassRole",
"Effect": "Allow",
"Resource": "acs:ram::1111111:role/aliyunlogdefaultrole"
}
],
"Version": "1"
}
```

   c. 子账号A1创建投递任务，并在配置投递任务时，RAM角色一栏填写OSS Bucket拥有者的RAM角色标识ARN，即账号B通过快速授权创建的RAM角色AliyunLogDefaultRole。

4 投递日志到MaxCompute

4.1 通过日志服务投递日志到MaxCompute

投递日志到 MaxCompute 是日志服务的一个功能，能够帮助您最大化数据价值。您可以自己决定对某个日志库是否启用该功能。一旦启用该功能，日志服务后台会定时把写入到该日志库内的日志投递到 MaxCompute 对应的表格中。

使用限制

- 数加控制台创建、修改投递配置必须由主账号完成，不支持子账号操作。
- 不同Logstore的数据请勿导入到同一个MaxCompute表中，否则会造成分区冲突、丢失数据等后果。
- 投递MaxCompute是批量任务，请谨慎设置分区列及其类型：保证一个同步任务内处理的数据分区数小于512个；用作分区列的字段值不能为空或包括/等MaxCompute保留字段 。配置细节请参考下文投递配置说明。
- 不支持海外Region的MaxCompute投递，海外Region的MaxCompute请使用Dataworks进行数据同步。

  支持数据投递的国内Region如下：

日志服务Region	MaxCompute Region
华北1	华东2
华北2	华北2、华东2
华北3	华东2
华北5	华东2
华东1	华东2
华东2	华东2
华南1	华南1、华东2
香港	华东2

功能优势

日志服务收集的日志除了可以被实时查询外，还可以把日志数据投递到大数据计算服务MaxCompute（原ODPS），进一步进行个性化BI分析及数据挖掘。通过日志服务投递日志数据到MaxCompute具有如下优势：

- 使用便捷

  您只需要完成2步配置即可以把日志服务Logstore的日志数据迁移到MaxCompute中。

- 避免重复收集工作

  由于日志服务的日志收集过程已经完成不同机器上的日志集中化，无需重复在不同机器上收集一遍日志数据后再导入到MaxCompute。

- 充分复用日志服务内的日志分类管理工作

  用户可让日志服务中不同类型的日志（存在不同Logstore中）、不同Project的日志自动投递到不同的MaxCompute表格，方便管理及分析MaxCompute内的日志数据。

 说明:

一般情况下日志数据在写入Logstore后的1个小时导入到MaxCompute，您可以在控制台投递任务管理查看导入状态。导入成功后即可在MaxCompute内查看到相关日志数据。判断数据是否已完全投递请参考文档。

结合日志服务的实时消费，投递日志数据到MaxCompute的数据通道以及日志索引功能，可以让用户按照不同的场景和需求、以不同的方式复用数据，充分发挥日志数据的价值。

配置流程

举例日志服务的一条日志如下:

```
16年01月27日20时50分13秒
10.10.*.*
ip:10.10.*.*
status:200
thread:414579208
time:27/Jan/2016:20:50:13 +0800
url:POST /PutData?Category=YunOsAccountOpLog&AccessKeyId
=****************&Date=Fri%2C%2028%20Jun%202013%2006%3A53%3A30%20GMT&
Topic=raw&Signature=******************************** HTTP/1.1
user-agent:aliyun-sdk-java
```

日志左侧的ip、status、thread、time、url、user-agent等是日志服务数据的字段名称，需要在下方配置中应用到。

1. 初始化数加平台

   a. 在日志服务控制台的日志库页面选择需要投递的日志库名称并依次展开节点，日志库名称 > 数据处理 > 导出 > MaxCompute(原ODPS)，单击开启投递。

   自动跳转到初始化数加平台的页面。MaxCompute默认为按量付费模式，具体参见MaxCompute文档说明。

   b. 查看服务协议和条款后单击确定，初始化数加平台。

   初始化开通需10~20秒左右，请耐心等待。如果已经开通数加及大数据计算服务MaxCompute（原ODPS），将直接跳过该步骤。

2. 数据模型映射

   在日志服务和大数据计算服务MaxCompute（原ODPS）之间同步数据，涉及两个服务的数据模型映射问题。您可以参考日志服务日志数据结构了解数据结构。

   将样例日志导入MaxCompute，分别定义MaxCompute数据列、分区列与日志服务字段的映射关系：

MaxComp 列类型	MaxComp 列名（可自定义）	MaxComp 列类型（可自定义）	日志服务字段名（投递配置里填写）	日志服务字段类型	日志服务字段语义
数据列	log_source	string	__source__	系统保留字段	日志来源的机器IP。
	log_time	bigint	__time__	系统保留字段	日志的Unix时间戳（是从1970年1月1日开始所经过的秒数），对应数据模型中的Time域。
	log_topic	string	__topic__	系统保留字段	日志主题。
	time	string	time	日志内容字段	解析自日志，对应数据模型中的key-value，例如Logtail采集的数据在很多时候__time__与time取值相同。
	ip	string	ip	日志内容字段	解析自日志。
	thread	string	thread	日志内容字段	解析自日志。

MaxComp 列类型	MaxComp 列名（可自定义）	MaxComp 列类型（可自定义）	日志服务字段名（投递配置里填写）	日志服务字段类型	日志服务字段语义
	log_extrac t_others	string	__extract_ others__	系统保留字段	未在配置中进行映射的其他日志内字段会通过key-value序列化到json，该json是一层结构，不支持字段内部json嵌套。
分区列	log_partit ion_time	string	__partitio n_time__	系统保留字段	由日志的 __time__ 字段对齐计算而得，分区粒度可配置，在配置项部分详述。
	status	string	status	日志内容字段	解析自日志，该字段取值应该是可以枚举的，保证分区数目不会超出上限。

- MaxCompute表至少包含一个数据列、一个分区列。
- 系统保留字段中建议使用 __partition_time__，__source__，__topic__。
- MaxCompute单表有分区数目6万的限制，分区数超出后无法再写入数据，所以日志服务导入MaxCompute表至多支持3个分区列。请谨慎选择自定义字段作为分区列，保证其值是可枚举的。
- 系统保留字段__extract_others__历史上曾用名_extract_others_，填写后者也是兼容的。
- MaxCompute分区列的值不支持”/“等特殊字符，这些是 MaxCompute 的保留字段。
- MaxCompute分区列取值不支持空，所以映射到分区列的字段必须出自保留字段或日志字段，且可以通过cast运算符将string类型字段值转换为对应分区列类型，空分区列的日志会在投递中被丢弃。
- 日志服务数据的一个字段最多允许映射到一个MaxCompute表的列（数据列或分区列），不支持字段冗余，同一个字段名第二次使用时其投递的值为null，如果null出现在分区列会导致数据无法被投递。

3. 配置投递规则

a. 开启投递。

初始化数加平台之后，根据页面提示进入LogHub —— 数据投递页面，选择需要投递的Logstore，并单击确定。

您也可以在日志库页面选择需要投递的日志库名称并依次展开节点，日志库名称 > 数据处理 > 导出 > MaxCompute(原ODPS)，进入MaxCompute（原ODPS）投递管理页面。单击开启投递以进入LogHub —— 数据投递页面。

图 4-1: 开启投递

b. 配置投递规则。

在LogHub —— 数据投递页面配置字段关联等相关内容。

图 4-2: 配置投递规则

配置项含义:

参数	语义
投递名称	自定义一个投递的名称，方便后续管理。
MaxCompute Project	MaxCompute项目名称，该项默认为新创建的Project，如果已经是MaxCompute老客户，可以下拉选择已创建其他Project。
MaxCompute Table	MaxCompute表名称，请输入自定义的新建的MaxCompute表名称或者选择已有的MaxCompute表。
MaxCompute 普通列	按序，左边填写与MaxCompute表数据列相映射的日志服务字段名称，右边填写或选择MaxCompute表的普通字段名称及字段类型。
MaxCompute 分区列	按序，左边填写与MaxCompute表分区列相映射的日志服务字段名称，右边填写或选择MaxCompute表的普通字段名称及字段类型。
分区时间格式	__partition_time__输出的日期格式，参考 Java SimpleDate Format。

参数	语义
导入MaxCompute间隔	MaxCompute数据投递间隔，默认1800，单位：秒。

- 该步会默认为客户创建好新的MaxCompute Project和Table，其中如果已经是MaxCompute老客户，可以下拉选择其他已创建Project。
- 日志服务投递MaxCompute功能按照字段与列的顺序进行映射，修改MaxCompute表列名不影响数据导入，如更改MaxCompute表schema，请重新配置字段与列映射关系。

参考信息

__partition_time__ 格式

将日志时间作为分区字段，通过日期来筛选数据是MaxCompute常见的过滤数据方法。

`__partition_time__` 是根据日志__time__值计算得到（不是日志写入服务端时间，也不是日志投递时间），结合分区时间格式，向下取整（为避免触发MaxCompute单表分区数目的限制，日期分区列的值会按照导入MaxCompute间隔对齐）计算出日期作为分区列。

举例来说，日志提取的time字段是“27/Jan/2016:20:50:13 +0800”，日志服务据此计算出保留字段__time__为1453899013（Unix时间戳），不同配置下的时间分区列取值如下：

导入MaxCompute间隔	分区时间格式	__partition_time__
1800	yyyy_MM_dd_HH_mm_00	2016_01_27_20_30_00
1800	yyyy-MM-dd HH:mm	2016-01-27 20:30
1800	yyyyMMdd	20160127
3600	yyyyMMddHHmm	201601272000
3600	yyyy_MM_dd_HH	2016_01_27_20

- 请勿使用精确到秒的日期格式：1. 很容易导致单表的分区数目超过限制（6万）；2. 单次投递任务的数据分区数目必须在512以内。
- 以上分区时间格式是测试通过的样例，您也可以参考Java SimpleDateFormat自己定义日期格式，但是该格式不得包含斜线字符”/“（这是MaxCompute的保留字段）。

__partition_time__ 使用方法

使用MaxCompute的字符串比较筛选数据，可以避免全表扫描。比如查询2016年1月26日一天内日志数据：

```
select * from {ODPS_TABLE_NAME} where log_partition_time >= "
2015_01_26" and log_partition_time < "2016_01_27";
```

__extract_others__使用方法

log_extract_others为一个json字符串，如果想获取该字段的user-agent内容，可以进行如下查询：

```
select get_json_object(sls_extract_others, "$.user-agent") from {
ODPS_TABLE_NAME} limit 10;
```

说明：

- get_json_object是MaxCompute提供的标准UDF。请联系MaxCompute团队开通使用该标准UDF的权限。
- 示例供参考，请以MaxCompute产品建议为最终标准。

其他操作

编辑投递配置

在投递管理页面，单击修改即可针对之前的配置信息进行编辑。其中如果想新增列，可以在大数据计算服务MaxCompute（原ODPS）修改投递的数据表列信息，则单击修改后会加载最新的数据表信息。

投递任务管理

在启动投递功能后，日志服务后台会定期启动离线投递任务。用户可以在控制台上看到这些投递任务的状态和错误信息。具体请参考#unique_36。

如果投递任务出现错误，控制台上会显示相应的错误信息：

错误信息	建议方案
MaxCompute项目空间不存在	在MaxCompute控制台中确认配置的MaxCompute项目是否存在，如果不存在则需要重新创建或配置。
MaxCompute表不存在	在MaxCompute控制台中确认配置的MaxCompute表是否存在，如果不存在则需要重新创建或配置。
MaxCompute项目空间或表没有向日志服务授权	在MaxCompute控制台中确认授权给日志服务账号的权限是否还存在，如果不存在则需要重新添加上相应权限。

错误信息	建议方案
MaxCompute错误	显示投递任务收到的MaxCompute错误，请参考MaxCompute相关文档或联系MaxCompute团队解决。日志服务会自动重试最近两天时间的失败任务。
日志服务导入字段配置无法匹配MaxCompute表的列	重新配置MaxCompute表格的列与日志服务数据字段的映射配置。

当投递任务发生错误时，请查看错误信息，问题解决后可以通过云控制台中日志投递任务管理或SDK来重试失败任务。

MaxCompute中消费日志

MaxCompute用户表中示例数据如下：

```
| log_source | log_time | log_topic | time | ip | thread | log_extrac
t_others | log_partition_time | status |
+------------+------------+-----------+-----------+-----------
+-----------+-----------------+--------------------+-----------+
| 10.10.*.* | 1453899013 | | 27/Jan/2016:20:50:13 +0800 | 10.10.*.*
 | 414579208 | {"url":"POST /PutData?Category=YunOsAccountOpLog&
AccessKeyId=****************&Date=Fri%2C%2028%20Jun%202013%2006%3A53%
3A30%20GMT&Topic=raw&Signature=****************************** HTTP/1
.1","user-agent":"aliyun-sdk-java"} | 2016_01_27_20_50 | 200 |
+------------+------------+-----------+-----------+-----------
+-----------+-----------------+--------------------+-----------+
```

同时，我们推荐您直接使用已经与MaxCompute绑定的大数据开发Data IDE来进行可视化的BI分析及数据挖掘，这将提高数据加工的效率。

授予MaxCompute数据投递权限

如果在数加平台执行表删除重建动作，会导致默认授权失效。请手动重新为日志服务投递数据授权。

在MaxCompute项目空间下添加用户：

```
ADD USER aliyun$shennong_open@aliyun.com;
```

shennong_open@aliyun.com 是日志服务系统账号（请不要用自己的账号），授权目的是为了能将数据写入到MaxCompute

MaxCompute项目空间Read/List权限授予：

```
GRANT Read, List ON PROJECT {ODPS_PROJECT_NAME} TO USER aliyun$
shennong_open@aliyun.com;
```

MaxCompute项目空间的表Describe/Alter/Update权限授予：

```
GRANT Describe, Alter, Update ON TABLE {ODPS_TABLE_NAME} TO USER
aliyun$shennong_open@aliyun.com;
```

确认MaxCompute授权是否成功：

```
SHOW GRANTS FOR aliyun$shennong_open@aliyun.com;

A       projects/{ODPS_PROJECT_NAME}: List | Read
A       projects/{ODPS_PROJECT_NAME}/tables/{ODPS_TABLE_NAME}:
Describe | Alter | Update
```

4.2 通过DataWorks投递数据到MaxCompute

本文将以LogHub数据同步至MaxCompute为例，为您介绍如何通过DataWorks数据集成同步LogHub数据至数据集成已支持的目的端数据源（如MaxCompute、OSS、OTS、RDBMS、DataHub等）。

除了将日志投递到OSS存储之外，您还可以选择将日志数据通过DataWorks的数据集成（Data Integration）功能投递至MaxCompute。数据集成是阿里集团对外提供的稳定高效、弹性伸缩的数据同步平台，为阿里云大数据计算引擎（包括MaxCompute、AnalyticDB）提供离线、批量的数据进出通道。

说明：

此功能已在华北2、华东2、华南1、香港、美西1、亚太东南1、欧洲中部1、亚太东南2、亚太东南3、亚太东北1、亚太南部1等多个Region发布上线。

应用场景

- 跨Region的LogHub与MaxCompute等数据源的数据同步。
- 不同阿里云账号下的LogHub与MaxCompute等数据源间的数据同步。
- 同一阿里云账号下的LogHub与MaxCompute等数据源间的数据同步。
- 公共云与金融云账号下的LogHub与MaxCompute等数据源间的数据同步。

跨阿里云账号的特别说明

以B账号进入数据集成配置同步任务，将A账号的LogHub数据同步至B账号的MaxCompute为例。

1. 用A账号的AccessId和Accesskey建LogHub数据源。

   此时B账号可以拖A账号下所有sls project的数据。

2. 用A账号下子账号A1的AccessId和Accesskey创建LogHub数据源。

   - A给A1赋权日志服务的通用权限，即`AliyunLogFullAccess`和`AliyunLogReadOnlyAccess`，详情请参见访问日志服务资源。
   - A给A1赋权日志服务的自定义权限。

     主账号A进入RAM控制台 > 策略管理 > 权限策略管理页面，选择新建授权策略。

     相关的授权请参见访问控制RAM和RAM子用户访问。

     根据下述策略进行授权后，B账号通过子账号A1只能同步日志服务project_name1以及project_name2的数据。

```
{
"Version": "1",
"Statement": [
{
"Action": [
"log:Get*",
"log:List*",
"log:CreateConsumerGroup",
"log:UpdateConsumerGroup",
"log:DeleteConsumerGroup",
"log:ListConsumerGroup",
"log:ConsumerGroupUpdateCheckPoint",
"log:ConsumerGroupHeartBeat",
"log:GetConsumerGroupCheckPoint"
],
"Resource": [
"acs:log:*:*:project/project_name1",
"acs:log:*:*:project/project_name1/*",
"acs:log:*:*:project/project_name2",
"acs:log:*:*:project/project_name2/*"
],
"Effect": "Allow"
}
]
```

```
}
```

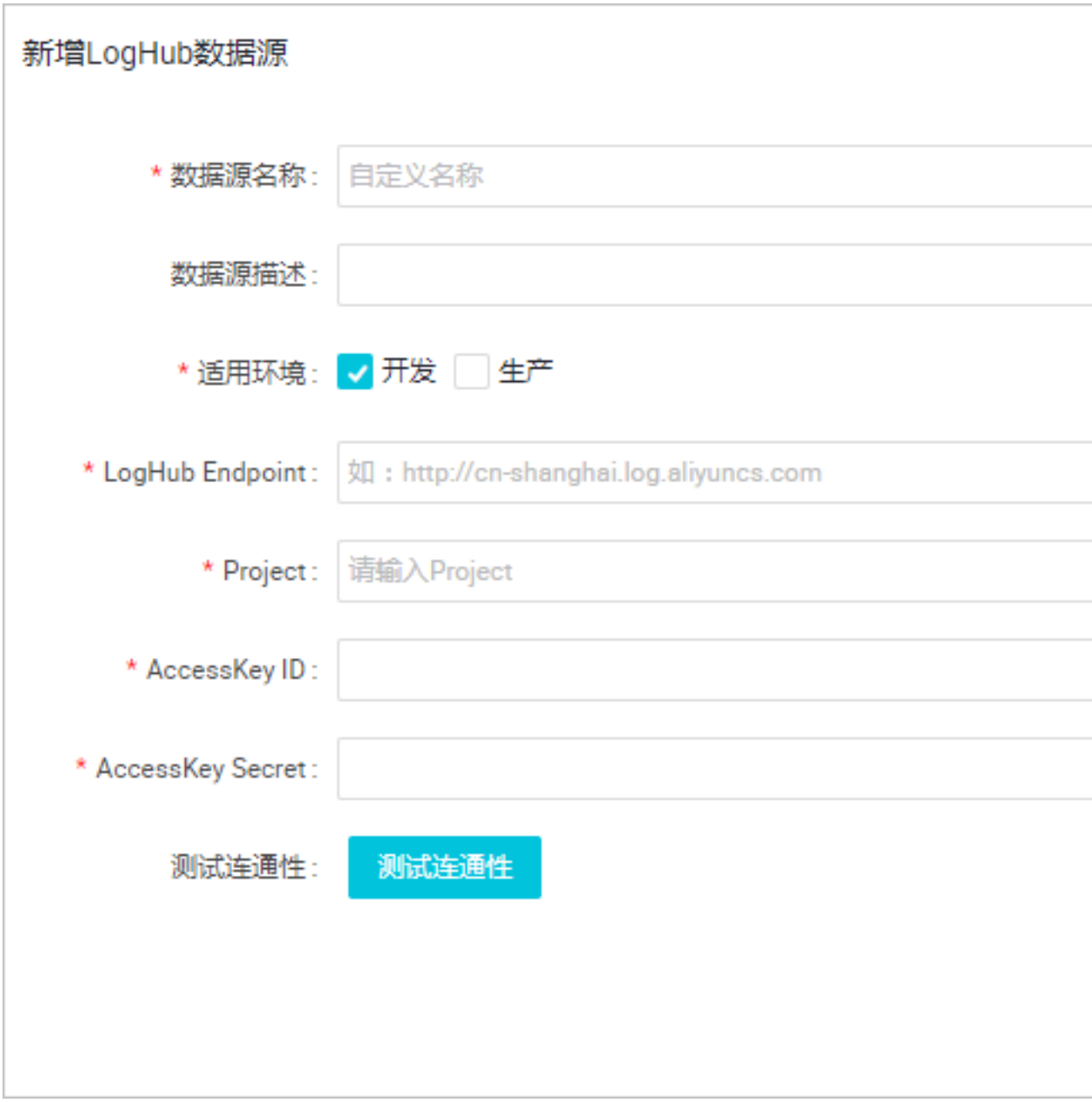
操作步骤

1. 新增数据源

   a. B账号或B的子账号以开发者身份登录DataWorks控制台，单击对应项目下的进入数据集成。

   b. 进入同步资源管理 > 数据源页面，单击右上角的新增数据源。

   c. 选择数据源类型为LogHub，填写新增LogHub数据源对话框中的配置。



配置	说明
数据源名称	数据源名称必须以字母、数字、下划线组合，且不能以数字和下划线开头。
数据源描述	对数据源进行简单描述，不得超过80个字符。
LogHub Endpoint	LogHub的Endpoint，格式为`http://yyy.com`。详细说明请参考#unique_41。
Project	您想要投递至MaxCompute的日志服务Project。必须是已创建的Project。
Access Id/Access Key	即访问密钥，相当于登录密码。您可以填写主账号或子账号的Access Id和Access Key。

d. 单击测试连通性。

e. 测试连通性通过后，单击确定。

2. 配置同步任务

单击左侧导航的同步任务，并单击第二步新建同步任务，进入配置同步任务流程。

您可选择向导模式，通过简单便捷的可视化页面完成任务配置；或者选择脚本模式，深度自定义配置您的同步任务。

通过向导模式配置同步任务

1. 进入数据开发 > 业务流程页面，在左上角单击新建数据同步节点。
2. 填写新建数据同步节点对话框中的配置，单击提交，进入数据同步任务配置页面。

3. 选择数据来源。

配置	说明
数据源	填写LogHub数据源的名称。
Logstore	导出增量数据的表的名称。该表需要开启Stream，可以在建表时开启，或者使用UpdateTable接口开启。
日志开始时间	数据消费的开始时间位点，为时间范围（左闭右开）的左边界，为yyyyMMddHHmmss格式的时间字符串（比如20180111013000），可以和DataWorks的调度时间参数配合使用。
日志结束时间	数据消费的结束时间位点，为时间范围（左闭右开）的右边界，为yyyyMMddHHmmss格式的时间字符串（比如20180111013010），可以和DataWorks的调度时间参数配合使用。
批量条数	一次读取的数据条数，默认为256。

数据预览默认收起，您可单击进行预览。

说明:

数据预览是选择LogHub中的几条数据展现在预览框，可能您同步的数据会跟您的预览的结果不一样，因为您同步的数据会指定开始时间可结束时间。

4. 选择数据去向。

选择MaxCompute数据源及目标表ok。

配置	说明
数据源	填写配置的数据源名称。
表	选择需要同步的表。
分区信息	此处需同步的表是非分区表，所以无分区信息。
清理规则	· 写入前清理已有数据：导数据之前，清空表或者分区的所有数据，相当于insert overwrite。 · 写入前保留已有数据：导数据之前不清理任何数据，每次运行数据都是追加进去的，相当于insert into。
压缩	默认选择不压缩。
空字符串作为null	默认选择否。

5. 字段映射。

选择字段的映射关系。需对字段映射关系进行配置，左侧源头表字段和右侧目标表字段为一一对应的关

系。

6. 通道控制。

配置作业速率上限和脏数据检查规则。

配置	说明
DMU	数据集成的计费单位。 说明: 设置DMU时，需注意DMU的值限制了最大并发数的值，请合理配置。
作业并发数	配置时会结合读取端指定的切分建，将数据分成多个Task，多个Task同时运行，以达到提速的效果。
同步速率	设置同步速率可保护读取端数据库，以避免抽取速度过大，给源库造成太大的压力。同步速率建议限流，结合源库的配置，请合理配置抽取速率。
错误记录数超过	脏数据，类似源端是Varchar类型的数据，写到Int类型的目标列中，导致因为转换不合理而无法写入的数据。同步脏数据的设置，主要在于控制同步数据的质量问题。建议根据业务情况，合理配置脏数据条数。
任务资源组	配置同步任务时，指定任务运行所在的资源组，默认运行在默认资源组上。当项目调度资源紧张时，也可以通过新增自定义资源组的方式来给调度资源进行扩容，然后将同步任务指定在自定义资源组上运行，新增自定义资源组的操作请参见#unique_42。 您可根据数据源网络情况、项目调度资源情况和业务重要程度，进行合理配置。

7. 运行任务。

您可通过以下两种方式运行任务。

- 直接运行（一次性运行）

  单击任务上方的运行按钮，将直接在数据集成页面运行任务，运行之前需要配置自定义参数的具体数值。

  如上图所示，代表同步10:10到17:30这段时间的LogHub记录到MaxCompute。

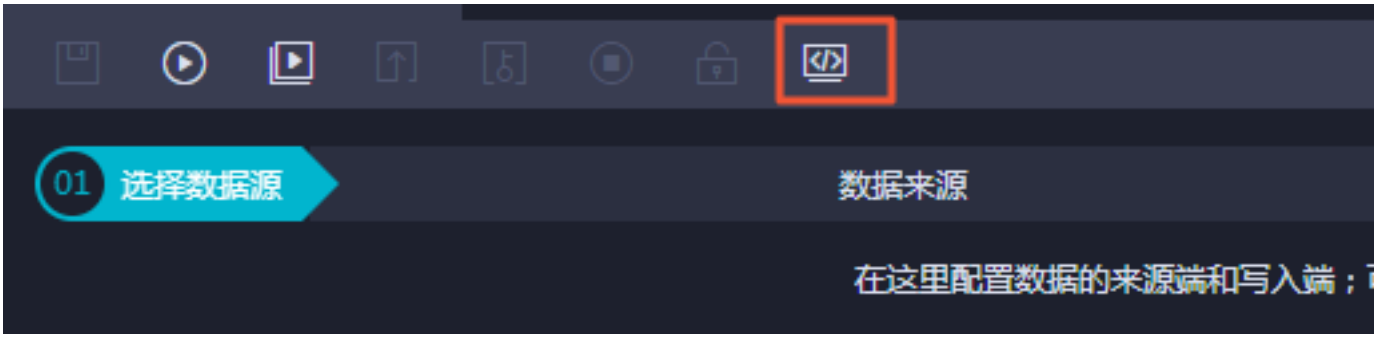
- 调度运行

单击提交按钮，将同步任务提交到调度系统中，调度系统会按照配置属性在从第二天开始自动定时执行。

如上图所示，设置按分钟调度，从00：00到23:59每5分钟调度一次。

通过脚本模式配置同步任务

如果您需要通过脚本模式配置此任务，单击工具栏中的转换脚本，选择确认即可进入脚本模式。



您可根据自身进行配置，示例脚本如下。

```
{
"type": "job",
"version": "1.0",
"configuration": {
"reader": {
"plugin": "loghub",
"parameter": {
"datasource": "loghub_lzz",//数据源名，保持跟您添加的数据源名一致
"logstore": "logstore-ut2",//目标日志库的名字，logstore是日志服务中日志数据的采集、存储和查询单元。
"beginDateTime": "${startTime}",//数据消费的开始时间位点，为时间范围（左闭右开）的左边界
"endDateTime": "${endTime}",//数据消费的开始时间位点，为时间范围（左闭右开）的右边界
"batchSize": 256,//一次读取的数据条数，默认为256。
"splitPk": "",
"column": [
"key1",
"key2",
"key3"
]
}
},
"writer": {
"plugin": "odps",
"parameter": {
"datasource": "odps_first",//数据源名，保持跟您添加的数据源名一致
"table": "ok",//目标表名
"truncate": true,
"partition": "",//分区信息
"column": [//目标列名
"key1",
"key2",
"key3"
]
}
},
"setting": {
"speed": {
"mbps": 8,//作业速率上限
"concurrent": 7//并发数
}
}
}
}
```

5 投递日志到SIEM

5.1 简介

日志服务支持将日志投递到 SIEM，以确保阿里云上的所有法规、审计、与其他相关日志能够导入到您的安全运维中心（SOC）中。

名词解释

- SIEM：安全信息与事件管理系统（Security Information and Event Management），如Splunk，IBM QRadar等。
- Splunk HEC：Splunk的http事件接收器（Splunk Http Event Collector），一个HTTP(s)接口，用于接收日志。

部署建议

- 硬件参数：
  - Linux（如Ubuntu x64）操作系统。
  - 2.0+ GHZ X 8核。
  - 16 GB 内存，推荐32 GB。
  - 1 Gbps网卡。
  - 至少2 GB可用磁盘空间，建议10 GB以上。
- 网络参数：

  从组织内的环境到阿里云的带宽应该大于数据在阿里云端产生的速度，否则日志无法实时消费。假设数据产生速度均匀，峰值为平时的2倍左右，每天产生1 TB原始日志。数据传输为5倍压缩的场景下，推荐带宽应该在4 MB/s（32 Mbps）左右。
- Python语言：我们用 Python 语言进行消费。Java 语言用法，请参考#unique_45。

Python SDK

- 推荐使用标准 CPython 解释器。
- 日志服务的 Python SDK 可以使用`python3 -m pip install aliyun-log-python-sdk -U`命令进行安装。
- 更多日志服务 Python SDK 的使用，请参考User Guide。

协同消费库

协同消费库（Consumer Library）是日志服务中对日志进行消费的高级模式，提供了消费组的概念对消费端进行抽象和管理，和直接使用SDK进行数据读取的区别在于，用户无需关心日志服务的实现细节，只需要专注于业务逻辑，另外，消费者之间的负载均衡、failover 等用户也都无需关心。

在日志服务中，一个日志库（Logstore）下面会有多个分区（Shard），协同消费库将 shard 分配给一个消费组下面的消费者，分配方式遵循以下原则：

- 每个shard只会分配到一个消费者。
- 一个消费者可以同时拥有多个shard。

新的消费者加入一个消费组，这个消费组的 shard 从属关系会调整，以达到消费负载均衡的目的，但是上面的分配原则不会变，分配过程对用户透明。

协同消费库的另一个功能是保存 checkpoint，方便程序故障恢复时能接着从断点继续消费，从而保证数据不会被重复消费。

Spark Streaming、Storm 以及 Flink Connector 都以 Consumer Library 作为基础实现。

投递方式

推荐使用日志服务消费组构建程序来从日志服务进行实时消费，然后通过HTTPS或者Syslog来发送日志给SIEM。

- HTTPS投递方式请参考#unique_46。
- Syslog投递方式请参考#unique_47。

5.2 通过HTTPS投递日志到SIEM

本文通过以 Splunk HEC 为例来展示如何通过HEC将阿里云相关日志投递到SIEM。

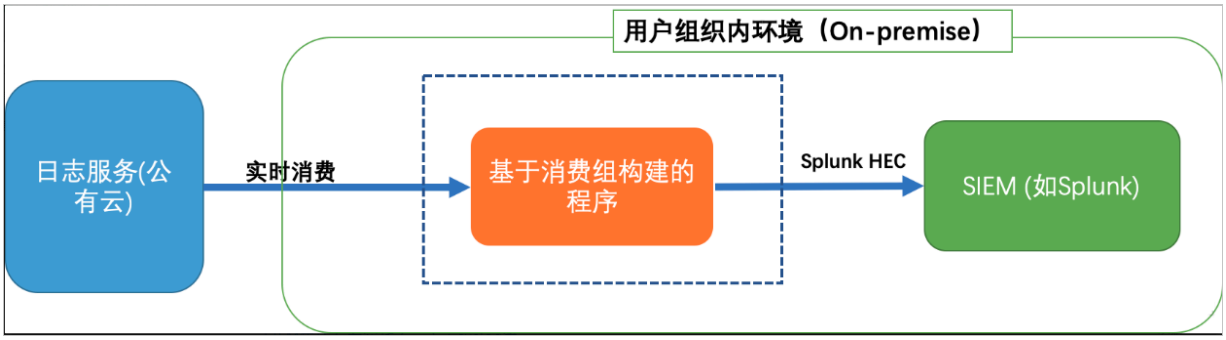
假设当前 SIEM（如Splunk）位于组织内部环境（on-premise），而不是云端。为了安全考虑，没有开放任何端口让外界环境来访问此SIEM。

说明:

本章节中的配置代码仅为示例，最新最全的代码示例请参考Github或Github（多源日志库时）。

投递流程

推荐使用日志服务消费组构建程序进行实时消费，然后通过 Splunk API（HEC）来发送日志给Splunk。



主程序示例

如下代码展示主程序控制逻辑：

```
def main():
    option, settings = get_monitor_option()

    logger.info("*** start to consume data...")
    worker = ConsumerWorker(SyncData, option, args=(settings,) )
    worker.start(join=True)

if __name__ == '__main__':
    main()
```

程序配置示例

- 配置内容：
  - 程序日志文件：以便后续测试或者诊断可能的问题。
  - 基本配置项：包括日志服务连接配置和消费组配置。
  - 消费组的高级选项：性能调参，不推荐修改。
  - SIEM（Splunk）相关参数与选项。
- 代码示例：

  请仔细阅读代码中相关注释并根据需要调整选项：

```
#encoding: utf8
import os
import logging
from logging.handlers import RotatingFileHandler

root = logging.getLogger()
handler = RotatingFileHandler("{0}_{1}.log".format(os.path.basename
(__file__), current_process().pid), maxBytes=100*1024*1024,
backupCount=5)
handler.setFormatter(logging.Formatter(fmt='[%(asctime)s] - [%(
threadName)s] - {%(module)s:%(funcName)s:%(lineno)d} %(levelname)s -
 %(message)s', datefmt='%Y-%m-%d %H:%M:%S'))
root.setLevel(logging.INFO)
root.addHandler(handler)
root.addHandler(logging.StreamHandler())

logger = logging.getLogger(__name__)
```

```
def get_option():
    ##########################
    # 基本选项
    ##########################

    # 从环境变量中加载日志服务参数与选项
    endpoint = os.environ.get('SLS_ENDPOINT', '')
    accessKeyId = os.environ.get('SLS_AK_ID', '')
    accessKey = os.environ.get('SLS_AK_KEY', '')
    project = os.environ.get('SLS_PROJECT', '')
    logstore = os.environ.get('SLS_LOGSTORE', '')
    consumer_group = os.environ.get('SLS_CG', '')

    # 消费的起点。这个参数在首次运行程序的时候有效，后续再次运行时将从上一次消费的保存点继续消费。
    # 可以使用“begin”，“end”，或者特定的 ISO 时间格式。
    cursor_start_time = "2018-12-26 0:0:0"

    ##########################
    # 一些高级选项
    ##########################

    # 一般不建议修改消费者名称，尤其是需要进行并发消费时。
    consumer_name = "{0}-{1}".format(consumer_group, current_process().pid)

    # 心跳时长，当服务器在2倍时间内没有收到特定 Shard 的心跳报告时，服务器会认为对应消费者离线并重新调配任务。
    # 所以当网络环境不佳时，不建议将时长设置的比较小。
    heartbeat_interval = 20

    # 消费数据的最大间隔，如果数据生成的速度很快，不需要调整这个参数。
    data_fetch_interval = 1

    # 构建一个消费组和消费者
    option = LogHubConfig(endpoint, accessKeyId, accessKey, project, logstore, consumer_group, consumer_name,
                          cursor_position=CursorPosition.SPECIAL_TIMER_CURSOR,
                          cursor_start_time=cursor_start_time,
                          heartbeat_interval=heartbeat_interval,
                          data_fetch_interval=data_fetch_interval)

    # Splunk选项
    settings = {
                "host": "10.1.2.3",
                "port": 80,
                "token": "a023nsdu123123123",
                'https': False,             # 可选，bool
                'timeout': 120,             # 可选，int
                'ssl_verify': True,         # 可选，bool
                "sourcetype": "",           # 可选，sourcetype
                "index": "",                # 可选，index
                "source": "",               # 可选，source
            }
```

```
        return option, settings
```

消费与投递示例

如下代码展示如何从日志服务获取数据并投递到 Splunk。

```
from aliyun.log.consumer import *
from aliyun.log.pulllog_response import PullLogResponse
from multiprocessing import current_process
import time
import json
import socket
import requests

class SyncData(ConsumerProcessorBase):
    """
    这个消费者从日志服务消费数据并发送给 Splunk
    """
    def __init__(self, splunk_setting):
      """初始化并验证 Splunk 连通性"""
        super(SyncData, self).__init__()

        assert splunk_setting, ValueError("You need to configure
settings of remote target")
        assert isinstance(splunk_setting, dict), ValueError("The
settings should be dict to include necessary address and confidentials
.")

        self.option = splunk_setting
        self.timeout = self.option.get("timeout", 120)

        # 测试 Splunk 连通性
        s = socket.socket()
        s.settimeout(self.timeout)
        s.connect((self.option["host"], self.option['port']))

        self.r = requests.session()
        self.r.max_redirects = 1
        self.r.verify = self.option.get("ssl_verify", True)
        self.r.headers['Authorization'] = "Splunk {}".format(self.
option['token'])
        self.url = "{0}://{1}:{2}/services/collector/event".format("
http" if not self.option.get('https') else "https", self.option['host
'], self.option['port'])

        self.default_fields = {}
        if self.option.get("sourcetype"):
            self.default_fields['sourcetype'] = self.option.get("
sourcetype")
        if self.option.get("source"):
            self.default_fields['source'] = self.option.get("source")
        if self.option.get("index"):
            self.default_fields['index'] = self.option.get("index")

    def process(self, log_groups, check_point_tracker):
        logs = PullLogResponse.loggroups_to_flattern_list(log_groups,
time_as_str=True, decode_bytes=True)
        logger.info("Get data from shard {0}, log count: {1}".format(
self.shard_id, len(logs)))
        for log in logs:
            # 发送数据到 Splunk
            event = {}
```

```
            event.update(self.default_fields)
            event['time'] = log[u'__time__']
            del log['__time__']

            json_topic = {"actiontrail_audit_event": ["event"] }
            topic = log.get("__topic__", "")
            if topic in json_topic:
                try:
                    for field in json_topic[topic]:
                        log[field] = json.loads(log[field])
                except Exception as ex:
                    pass
            event['event'] = json.dumps(log)

            data = json.dumps(event, sort_keys=True)

                try:
                    req = self.r.post(self.url, data=data, timeout=
self.timeout)
                    req.raise_for_status()
                except Exception as err:
                    logger.debug("Failed to connect to remote Splunk
server ({0}). Exception: {1}", self.url, err)

                    # TODO: 根据需要，添加一些重试或者报告

        logger.info("Complete send data to remote")

        self.save_checkpoint(check_point_tracker)
```

启动程序示例

假设程序命名为sync_data.py，启动如下：

```
export SLS_ENDPOINT=<Endpoint of your region>
export SLS_AK_ID=<YOUR AK ID>
export SLS_AK_KEY=<YOUR AK KEY>
export SLS_PROJECT=<SLS Project Name>
export SLS_LOGSTORE=<SLS Logstore Name>
export SLS_CG=<消费组名，可以简单命名为"syc_data">

python3 sync_data.py
```

多源日志库示例

针对多源日志库，需要公用一个 executor 以便避免进程过多，可以参考样例：多源日志库时发送日志到Splunk。核心的变化是主函数：

```
exeuctor, options, settings = get_option()

    logger.info("*** start to consume data...")
    workers = []

    for option in options:
        worker = ConsumerWorker(SyncData, option, args=(settings,) )
        workers.append(worker)
        worker.start()

    try:
        for i, worker in enumerate(workers):
```

```
            while worker.is_alive():
                worker.join(timeout=60)
            logger.info("worker project: {0} logstore: {1} exit
unexpected, try to shutdown it".format(
                options[i].project, options[i].logstore))
            worker.shutdown()
    except KeyboardInterrupt:
        logger.info("*** try to exit **** ")
        for worker in workers:
            worker.shutdown()

        # wait for all workers to shutdown before shutting down
executor
        for worker in workers:
            while worker.is_alive():
                worker.join(timeout=60)

    exeuctor.shutdown()
```

限制与约束

每一个日志库（logstore）最多可以配置10个消费组，如果提示`ConsumerGroupQuotaExceed`则表示遇到限制，建议在控制台删除一些不用的消费组。

消费状态监控

- 在控制台查看#unique_49。
- 通过云监控查看消费组延迟，并配置告警。

并发消费

基于消费组的程序，可以通过启动多次程序以达到并发的作用：

```
nohup python3 sync_data.py &
nohup python3 sync_data.py &
nohup python3 sync_data.py &
...
```

说明:

所有消费者的名称均不相同（消费者名以进程ID为后缀），且属于同一个消费组。 由于一个分区（Shard）只能被一个消费者消费，假设一个日志库有10个分区，那么最多有10个消费组同时消费。

吞吐量

基于测试，在没有带宽、接收端速率限制（如 Splunk 端）的情况下，用 python3 运行上述样例，单个消费者大约占用 20% 的单核 CPU 资源， 此时消费可以达到 10 MB/s 原始日志的速率。因此，10个消费者理论上可以达到 100 MB/s 原始日志每个 CPU 核，也就是每个 CPU 核每天可以消费 0.9 TB 原始日志。

高可用

消费组将检测点（check-point）保存在服务器端，当一个消费者停止，另外一个消费者将自动接管并从断点继续消费。可以在不同机器上启动消费者，这样在一台机器停止或者损坏的情况下，其他机器上的消费者可以自动接管并从断点进行消费。理论上，为了备用，也可以通过不同机器启动大于 Shard 数量的消费者。

HTTPS

如果服务入口（endpoint）配置为`https://`前缀，如`https://cn-beijing.log.aliyuncs.com`，程序与日志服务的连接将自动使用HTTPS加密。

服务器证书 *.aliyuncs.com 是 GlobalSign 签发，默认大多数 Linux/Windows 的机器会自动信任此证书。如果某些特殊情况，机器不信任此证书，可以参考这里下载并安装此证书。

5.3 通过Syslog投递日志到SIEM

Syslog是一个常见的日志通道，几乎所有的SIEM（如 IBM Qradar, HP Arcsight 等）都支持通过Syslog渠道接受日志。本文主要介绍如何通过Syslog将阿里云相关日志投递到 SIEM。

背景信息

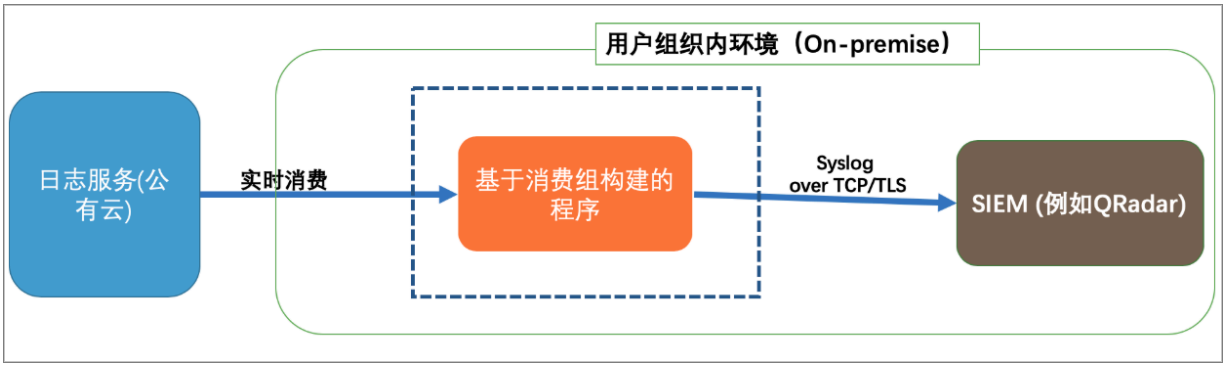
- Syslog主要是基于标准协议RFC5424和RFC3164定义的相关格式规范，RFC3164 协议是 2001 年发布，RFC5424 协议是 2009 年发布的升级版本。因为新版兼容旧版，且新版本解决了很多问题，因此推荐使用 RFC5424 协议。
- Syslog over TCP/TLS：Syslog 只是规定了日志格式，理论上 TCP 和 UDP 都可以支持 Syslog，可以较好的确保数据传输稳定性。协议RFC5425也定义了 TLS 的安全传输层，如果您的 SIEM 支持 TCP 通道，甚至是 TLS，那么建议优先使用。
- Syslog facility：早期 Unix 定义的程序组件，这里我们可以选择`user`作为默认组件。
- Syslog severity：定义的日志级别，可以根据需要设置指定内容的日志为较高的级别。默认一般用`info`。

说明:

本章节中的配置代码仅为示例，最新最全的代码示例请参考Github。

投递流程

推荐使用日志服务消费组构建程序来进行实时消费，然后通过 Syslog over TCP/TLS 来发送日志给 SIEM。如果 SIEM 支持 TCP 通道，甚至TLS，建议优先使用。



主程序示例

如下代码展示主程序控制逻辑：

```
def main():
    option, settings = get_monitor_option()

    logger.info("*** start to consume data...")
    worker = ConsumerWorker(SyncData, option, args=(settings,) )
    worker.start(join=True)

if __name__ == '__main__':
    main()
```

程序配置示例

- 配置内容：
  - 程序日志文件：以便后续测试或者诊断可能的问题。
  - 基本配置项：包括日志服务连接配置和消费组配置。
  - 消费组的高级选项：性能调参，不推荐修改。
  - SIEM 的 Syslog server 相关参数与选项。

> 说明：
> 如果 SIEM 支持基于 TCP 或者 TLS 的 Syslog 通道，那么需要在配置的时候配置额外的 `proto`为 TLS，并且配置正确的 SSL 的证书即可。

- 代码示例

  请仔细阅读代码中相关注释并根据需要调整选项：

```
#encoding: utf8
import os
import logging
from logging.handlers import RotatingFileHandler

root = logging.getLogger()
handler = RotatingFileHandler("{0}_{1}.log".format(os.path.basename
(__file__), current_process().pid), maxBytes=100*1024*1024,
backupCount=5)
```

```
handler.setFormatter(logging.Formatter(fmt='[%(asctime)s] - [%(
threadName)s] - {%(module)s:%(funcName)s:%(lineno)d} %(levelname)s -
 %(message)s', datefmt='%Y-%m-%d %H:%M:%S'))
root.setLevel(logging.INFO)
root.addHandler(handler)
root.addHandler(logging.StreamHandler())

logger = logging.getLogger(__name__)

def get_option():
    ##########################
    # 基本选项
    ##########################

    # 从环境变量中加载日志服务参数与选项
    endpoint = os.environ.get('SLS_ENDPOINT', '')
    accessKeyId = os.environ.get('SLS_AK_ID', '')
    accessKey = os.environ.get('SLS_AK_KEY', '')
    project = os.environ.get('SLS_PROJECT', '')
    logstore = os.environ.get('SLS_LOGSTORE', '')
    consumer_group = os.environ.get('SLS_CG', '')

    # 消费的起点。这个参数在首次运行程序的时候有效，后续再次运行时将从上一次消费
的保存点继续消费。
    # 可以使用 “begin”, “end”, 或者特定的 ISO 时间格式。
    cursor_start_time = "2018-12-26 0:0:0"

    ##########################
    # 一些高级选项
    ##########################

    # 一般不要修改消费者名称，尤其是需要并发消费时
    consumer_name = "{0}-{1}".format(consumer_group, current_process
().pid)

    # 心跳时长，当服务器在2倍时间内没有收到特定 Shard 的心跳报告时，服务器会认
为对应消费者离线并重新调配任务。
    # 所以当网络环境不佳的时候，不建议将时长设置的比较小。
    heartbeat_interval = 20

    # 消费数据的最大间隔，如果数据生成的速度很快，不需要调整这个参数。
    data_fetch_interval = 1

    # 构建一个消费组和消费者
    option = LogHubConfig(endpoint, accessKeyId, accessKey, project
, logstore, consumer_group, consumer_name,
                          cursor_position=CursorPosition.SPECIAL_TI
MER_CURSOR,
                          cursor_start_time=cursor_start_time,
                          heartbeat_interval=heartbeat_interval,
                          data_fetch_interval=data_fetch_interval)

    # syslog options
    settings = {
                "host": "1.2.3.4", # 必选
                "port": 514,       # 必选，端口
                "protocol": "tcp", # 必选，tcp, udp 或 tls  (仅
Python3)
                "sep": "||",       # 必选，key=value 键值对的分隔符，这
里用||分隔
                "cert_path": None, # 可选，TLS 的证书位置
                "timeout": 120,    # 可选，超时时间，默认 120 秒
                "facility": syslogclient.FAC_USER,  # 可选，可以参考其
他 syslogclient.FAC_* 的值
```

```
                "severity": syslogclient.SEV_INFO,  # 可选，可以参考其
他 syslogclient.SEV_* 的值
                "hostname": None,  # 可选，机器名，默认选择本机机器名
                "tag": None        # 可选，标签，默认是 -
            }

    return option, settings
```

消费与投递示例

如下代码展示如何从日志服务获取数据投递到 SIEM Syslog 服务器。请仔细阅读代码中相关注释并根据需要调整格式：

```
from syslogclient import SyslogClientRFC5424 as SyslogClient

class SyncData(ConsumerProcessorBase):
    """
    消费者从日志服务消费数据并发送给 Syslog server
    """
    def __init__(self, splunk_setting):
      """初始化并验证 Syslog server 连通性"""
        super(SyncData, self).__init__()   # remember to call base's
init

        assert target_setting, ValueError("You need to configure
settings of remote target")
        assert isinstance(target_setting, dict), ValueError("The
settings should be dict to include necessary address and confidentials
.")

        self.option = target_setting
        self.protocol = self.option['protocol']
        self.timeout = int(self.option.get('timeout', 120))
        self.sep = self.option.get('sep', "||")
        self.host = self.option["host"]
        self.port = int(self.option.get('port', 514))
        self.cert_path=self.option.get('cert_path', None)

        # try connection
        with SyslogClient(self.host, self.port, proto=self.protocol,
timeout=self.timeout, cert_path=self.cert_path) as client:
            pass

    def process(self, log_groups, check_point_tracker):
        logs = PullLogResponse.loggroups_to_flattern_list(log_groups,
time_as_str=True, decode_bytes=True)
        logger.info("Get data from shard {0}, log count: {1}".format(
self.shard_id, len(logs)))
        try:
            with SyslogClient(self.host, self.port, proto=self.
protocol, timeout=self.timeout, cert_path=self.cert_path) as client:
                for log in logs:
                    # Put your sync code here to send to remote.
                    # the format of log is just a dict with example as
 below (Note, all strings are unicode):
                    #    Python2: {"__time__": "12312312", "__topic__
": "topic", u"field1": u"value1", u"field2": u"value2"}
                    #    Python3: {"__time__": "12312312", "__topic__
": "topic", "field1": "value1", "field2": "value2"}
                    # suppose we only care about audit log
                    timestamp = datetime.fromtimestamp(int(log[u'
__time__']))
```

```
                    del log['__time__']

                    io = six.StringIO()
                    first = True
            # TODO:  这里可以根据需要修改格式化内容，这里使用 Key=Value 传输，并使用默认的||进行分割
                    for k, v in six.iteritems(log):
                        io.write("{0}{1}={2}".format(self.sep, k, v))

                    data = io.getvalue()

            # TODO: 这里可以根据需要修改 facility 或者 severity
                    client.log(data, facility=self.option.get("facility", None), severity=self.option.get("severity", None), timestamp=timestamp, program=self.option.get("tag", None), hostname=self.option.get("hostname", None))

        except Exception as err:
            logger.debug("Failed to connect to remote syslog server ({0}). Exception: {1}".format(self.option, err))

            # TODO: 需要添加一些错误处理的代码，例如重试或者通知等
            raise err

        logger.info("Complete send data to remote")

        self.save_checkpoint(check_point_tracker)
```

启动程序示例

假设程序命名为sync_data.py，启动如下：

```
export SLS_ENDPOINT=<Endpoint of your region>
export SLS_AK_ID=<YOUR AK ID>
export SLS_AK_KEY=<YOUR AK KEY>
export SLS_PROJECT=<SLS Project Name>
export SLS_LOGSTORE=<SLS Logstore Name>
export SLS_CG=<消费组名，可以简单命名为"syc_data">

python3 sync_data.py
```

限制与约束

每一个日志库（logstore）最多可以配置10个消费组，如果遇到ConsumerGroupQuotaExceed则表示遇到限制，建议在控制台删除一些不用的消费组。

消费状态与监控

- 在控制台查看#unique_49。
- 通过云监控查看消费组延迟，并配置告警。

并发消费

基于消费组的程序，可以直接启动多次程序以达到并发效果：

```
nohup python3 sync_data.py &
nohup python3 sync_data.py &
nohup python3 sync_data.py &
```

```
...
```

说明:

所有消费者的名称均不相同（消费者名以进程ID为后缀），且属于同一个消费组。 因为一个分区（Shard）只能被一个消费者消费，假设一个日志库有10个分区，那么最多有10个消费组同时消费。

吞吐量

基于测试，在没有带宽、接收端速率限制（如 Splunk 端）的情况下，用 python3 运行上述样例，单个消费者大约占用 20% 的单核 CPU 资源， 此时消费可以达到 10 MB/s 原始日志的速率。因此，10个消费者理论上可以达到 100 MB/s 原始日志每个 CPU 核，也就是每个 CPU 核每天可以消费 0.9 TB 原始日志。

高可用

消费组将检测点（check-point）保存在服务器端，当一个消费者停止，另外一个消费者将自动接管并从断点继续消费。可以在不同机器上启动消费者，这样在一台机器停止或者损坏的情况下，其他机器上的消费者可以自动接管并从断点进行消费。理论上，为了备用，也可以通过不同机器启动大于 Shard 数量的消费者。

6 最佳实践

6.1 投递-对接数据仓库

日志服务LogShipper功能可以便捷地将日志数据投递到 OSS、Table Store、MaxCompute 等存储类服务，配合 E-MapReduce（Spark、Hive）、MaxCompute 进行离线计算。

数仓（离线计算）

数据仓库+离线计算是实时计算的补充，两者针对目标不同：

模式	优势	劣势	使用领域
实时计算	快速	计算较为简单	增量为主，监控、实时分析
离线计算（数据仓库）	精准、计算能力强	较慢	全量为主，BI、数据统计、比较

目前对于数据分析类需求，同一份数据会同时做实时计算+数据仓库（离线计算）。例如对访问日志：

- 通过流计算实时显示大盘数据：当前PV、UV、各运营商信息。
- 每天晚上对全量数据进行细节分析，比较增长量、同步/环比，Top数据等。

互联网领域有两种经典的模式讨论：

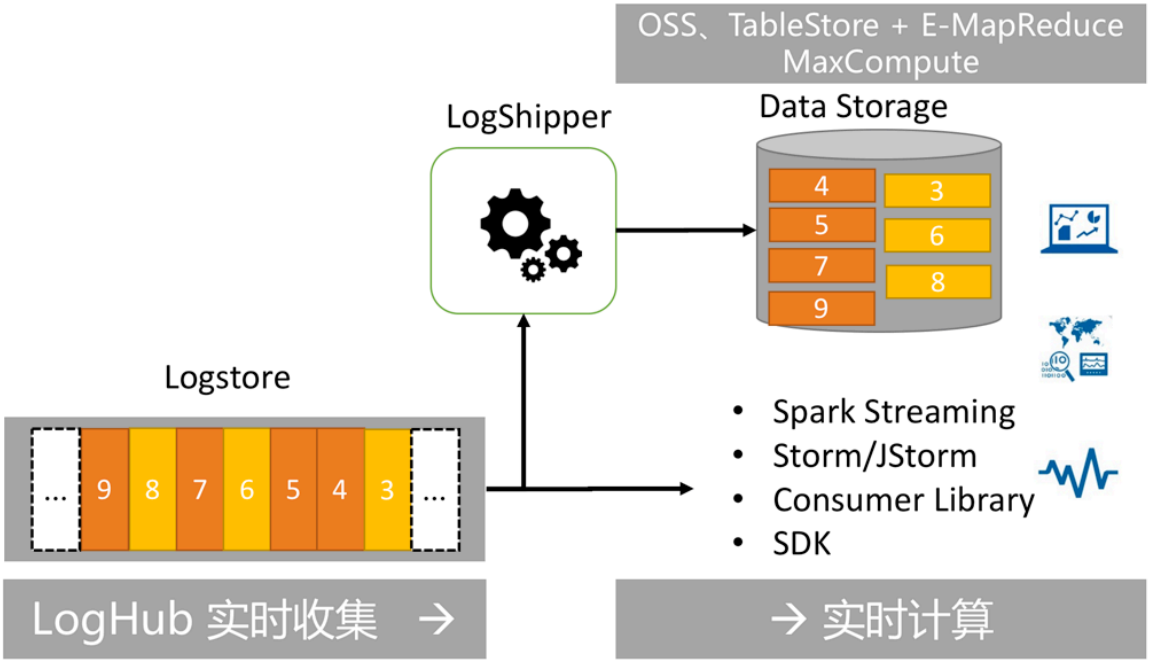
- Lamdba Architecture: 数据进来后，既支持流式处理、同时存入数仓。但用户发起查询时，会根据查询需求和复杂度从实时计算、离线计算拿结果返回。
- Kappa Architecture: kafka based Architecture。弱化离线计算部分，数据存储都在Kafka中，实时计算解决所有问题。

日志服务提供模式比较偏向Lamdba Architecture。

LogHub/LogShipper一站式解决实时+离线场景

在创建Logstore后，可以在控制台配置LogShipper支持数据仓库对接。当前支持如下：

- OSS（大规模对象存储）：
- TableStore(NoSQL数据存储服务):
- MaxCompute(大数据计算服务):



LogShipper提供如下功能：

- 准实时：分钟级进入数据仓库
- 数据量大：无需担心并发量
- 自动重试：遇到故障自动重试、也可以通过API手动重试
- 任务API：通过API可以获得时间段日志投递状态
- 自动压缩：支持数据压缩、节省存储带宽

典型场景

场景 1：日志审计

小A维护了一个论坛，需要对论坛所有访问日志进行审计和离线分析。

- G部门需要小A配合记录最近180天内用户访问情况，在有需求时，提供某个时间段的访问日志。
- 运营同学在每个季度需要对日志出一份访问报表。

小A使用日志服务（LOG）收集服务器上日志数据，并且打开了日志投递（LogShipper）功能，日志服务就会自动完成日志收集、投递、以及压缩。有审查需要时，可以将该时间段日志授权给第三方。需要离线分析时，利用E-MapReduce跑一个30分钟离线任务，用最少的成本办了两件事情。也可以使用阿里云DLA对投递到OSS中的日志数据进行数据分析。

场景 2：日志实时+离线分析

小B是一个开源软件爱好者，喜欢利用Spark进行数据分析。他的需求如下：

- 移动端通过API收集日志。
- 通过Spark Streaming对日志进行实时分析，统计线上用户访问。
- 通过Hive进行T+1离线分析。
- 将日志数据开放给下游代理商，进行其他维度分析。

通过今天LOG+OSS+EMR/DLA+RAM组合，可轻松应对这类需求。

7 FAQ

7.1 日志投递MaxCompute后，如何检查数据完整性

在日志服务数据投递MaxCompute场景下，需要在MaxCompute表分区维度上检查数据完整性，即MaxCompute表中某个分区中数据是否已经完整。

使用保留字段__partition_time__作为表分区列，如何判断分区数据是否已完整

__partition_time__由日志的time字段计算得到，由日志真实时间按照时间格式字符串向下取整得出。其中，日志真实时间既不是投递数据的时间，也不是日志写入服务端时间。

举例：日志时间为2017-05-19 10:43:00，分区字段格式字符串配置为yyyy_MM_dd_HH_mm，每1小时投递一次。那么：无论该日志是什么时刻写入服务端，这条日志会存入MaxCompute的2017_05_19_10_00分区，计算细节参考MaxCompute投递字段说明。

如果不考虑写入了历史数据等问题，在日志实时写入的情况下，有以下两种方法判断分区数据是否已完整：

- 通过控制台或API/SDK判断（推荐）

  使用API、SDK或者控制台获取指定Project/Logstore投递任务列表。例如API返回任务列表如下。控制台会对该返回结果进行可视化展示。

```
{
  "count" : 10,
  "total" : 20,
  "statistics" : {
      "running" : 0,
      "success" : 20,
      "fail" : 0
  }
  "tasks" : [
      ...
      {
          "id" : "abcdefghijk",
          "taskStatus" : "success",
          "taskMessage" : "",
          "taskCreateTime" : 1448925013,
          "taskLastDataReceiveTime" : 1448915013,
          "taskFinishTime" : 1448926013
      },
      {
          "id" : "xfegeagege",
          "taskStatus" : "success",
          "taskMessage" : "",
          "taskCreateTime" : 1448926813,
          "taskLastDataReceiveTime" : 1448930000,
          "taskFinishTime" : 1448936910
```

```
        }
    ]
}
```

taskLastDataReceiveTime表示该批任务中最后一条日志到达服务端时的机器系统时间，对应控制台的接收日志数据时间，根据该参数判断时间为T以前的数据是否已经全部投递到MaxCompute表。

- 如果taskLastDataReceiveTime < T + 300s（300秒是为了容忍数据发送服务端发生错误重试）以前的每个投递任务状态都是success，说明T时刻的数据都已经入库。
- 如果任务列表中有ready/running状态任务，说明数据还不完整，需要等待任务执行结束。
- 如果任务列表中有failed状态任务，请查看原因并在解决后重试任务。您可以尝试修改投递配置，以解决投递问题。

· 通过MaxCompute分区粗略估计

比如在MaxCompute中以半小时做一次分区，投递任务为每30分钟一次，当表中包含以下分区：

```
2017_05_19_10_00
2017_05_19_10_30
```

当发现分区2017_05_19_11_00出现时，说明11:00之前分区数据已经完整。

该方法不依赖API，判断方式简单但结果并不精确，仅用作粗略估计。

使用自定义日志字段作为表分区列，如何判断分区数据已完整

比如日志中有一个字段date，取值：20170518，20170519，在配置投递规则时将date列映射到表分区列。

这种情况下，需要考虑date字段与写入服务端时间差，结合使用保留字段方法，根据接收日志数据时间判断。

投递成功，但MaxCompute表数据有缺失，应如何解决

MaxCompute投递任务状态成功，但表中数据缺失，一般有以下原因：

· 表分区列映射的日志服务字段的名称不存在。此时投递过去的列值为null，而MaxCompute表不允许分区列值为null。
· 表分区列映射的日志服务字段的值包含/或其他特殊符号。MaxCompute将这些字符作为保留字，不允许在分区列中出现。

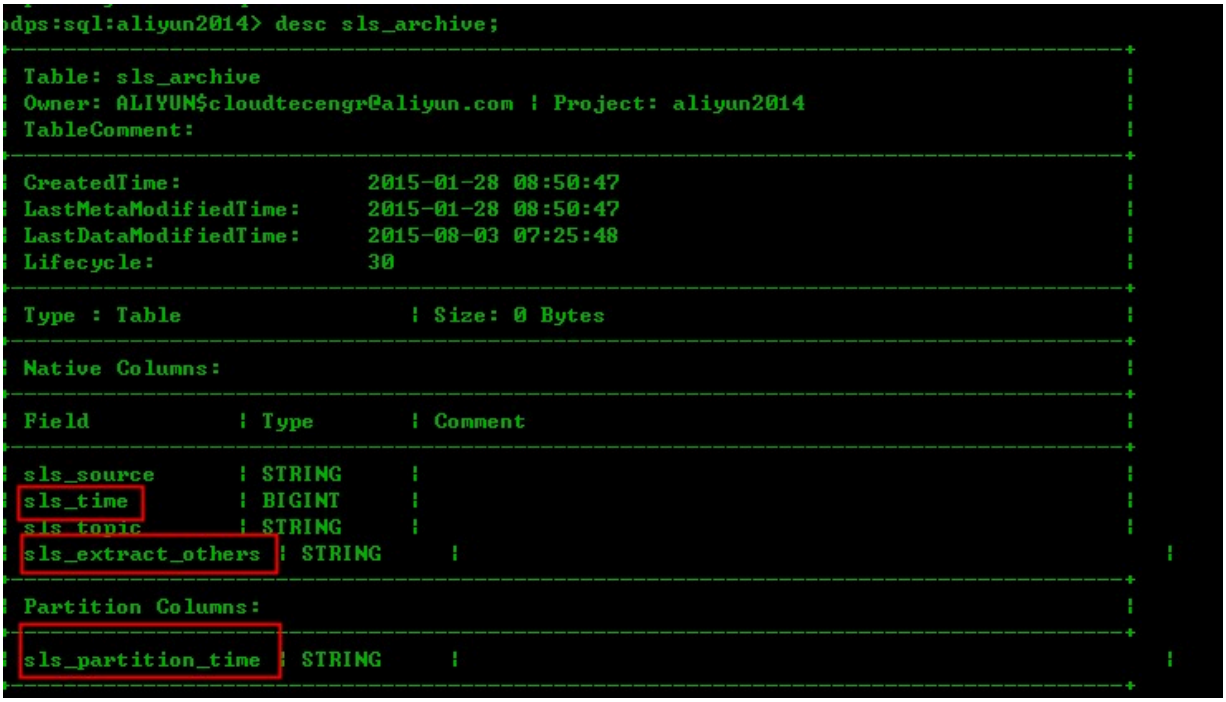
遇到这些情况时，投递策略为忽略异常的日志行，并继续任务，在该次任务中其它分区正确的日志行可以成功同步。

因此，在配置字段映射不当的情况下，可能出现任务成功但是表中缺少数据的情况，请修改分区列配置。建议使用保留字段`__partition_time__`作为分区。

更多细节请参考MaxCompute投递相关限制说明。

7.2 投递MaxCompute时的参数时间

日志服务投递日志至MaxCompute中有多个时间参数。



其中涉及到的时间参数及其含义如下：

时间参数	含义
sls_time	sls的服务器收到日志的时间
sls_extract_others	是用户真实的时间
sls-partition_time	MaxCompute归档的时间

> 说明：
> sls_time一般都是略晚于sls_extract_others的时间。

如您的问题仍未解决，请联系售后技术支持。