

ALIBABA CLOUD

阿里云

专有云企业版

日志服务
开发指南

产品版本：v3.16.2

文档版本：20220915

 阿里云

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
 危险	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。	 危险 重置操作将丢失用户配置数据。
 警告	该类警示信息可能会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，恢复业务时间约十分钟。
 注意	用于警示信息、补充说明等，是用户必须了解的内容。	 注意 权重设置为0，该服务器不会再接受新请求。
 说明	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。	单击设置> 网络> 设置网络类型。
粗体	表示按键、菜单、页面名称等UI元素。	在结果确认页面，单击确定。
Courier字体	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <code>Instance_ID</code>
[] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

目录

1.准备工作	08
1.1. 登录API与工具控制台	08
1.2. 查看API信息	08
1.3. 获取AccessKey	09
1.4. 获取公共Header参数	11
1.5. 获取日志服务的Endpoint	12
1.6. 获取日志服务的SDK包	13
2.SDK参考	14
2.1. 基本介绍	14
2.1.1. 概述	14
2.1.2. 配置	15
2.1.3. 错误处理	16
2.1.4. 接口规范	21
2.2. Java SDK	23
2.3. .NET SDK	28
2.4. .NET Core SDK	29
2.5. PHP SDK	30
2.6. Python SDK	33
2.7. Android SDK	37
2.8. C SDK	37
2.9. GO SDK	37
2.10. iOS SDK	37
2.11. C++ SDK	38
2.12. Node.js SDK	38
3.API参考	42
3.1. 概览	42

3.2. 公共请求头	43
3.3. 公共响应头	44
3.4. 请求签名	44
3.5. 通用错误码	49
3.6. 日志项目接口	51
3.6.1. CreateProject	51
3.6.2. DeleteProject	52
3.6.3. UpdateProject	54
3.6.4. GetProject	56
3.6.5. ListProject	57
3.6.6. GetProjectLogs	60
3.7. 日志库相关接口	63
3.7.1. CreateLogstore	63
3.7.2. DeleteLogstore	65
3.7.3. UpdateLogstore	67
3.7.4. GetLogstore	69
3.7.5. ListLogstore	71
3.7.6. ListShards	73
3.7.7. SplitShard	75
3.7.8. MergeShards	78
3.7.9. GetCursor	81
3.7.10. GetCursorTime	84
3.7.11. PullLogs	86
3.7.12. PostLog	88
3.7.13. GetShipperStatus	92
3.7.14. RetryShipperTask	96
3.7.15. GetLogs	98
3.7.16. GetHistograms	102

3.7.17. UpdateIndex	106
3.7.18. CreateIndex	110
3.7.19. DeleteIndex	113
3.7.20. GetIndex	115
3.7.21. PutWebtracking	119
3.8. Logtail 机器组相关接口	122
3.8.1. CreateMachineGroup	122
3.8.2. DeleteMachineGroup	125
3.8.3. UpdateMachineGroup	127
3.8.4. ListMachineGroup	129
3.8.5. GetMachineGroup	131
3.8.6. ApplyConfigToMachineGroup	133
3.8.7. RemoveConfigFromMachineGroup	135
3.8.8. ListMachines	136
3.8.9. GetAppliedConfigs	139
3.9. Logtail 配置相关接口	141
3.9.1. CreateConfig	141
3.9.2. ListConfig	145
3.9.3. GetAppliedMachineGroups	146
3.9.4. GetConfig	148
3.9.5. DeleteConfig	150
3.9.6. UpdateConfig	152
3.10. 消费组接口	156
3.10.1. CreateConsumerGroup	156
3.10.2. DeleteConsumerGroup	158
3.10.3. GetCheckPoint	160
3.10.4. HeartBeat	162
3.10.5. ListConsumerGroup	164

3.10.6. UpdateCheckPoint	166
3.10.7. UpdateConsumerGroup	168
3.11. RAM/STS	170
3.11.1. 概览	170
3.11.2. 资源列表	171
3.11.3. 动作列表	173
3.11.4. 鉴权规则	175
3.12. 公共资源说明	178
3.12.1. 数据加密	178
3.12.2. 数据模型	180
3.12.3. 数据编码方式	182
3.12.4. 日志库	183
3.12.5. 分区	184
3.12.6. Logtail 配置	184
3.12.7. Logtail 机器组	187

1. 准备工作

1.1. 登录API与工具控制台

您可以通过Apsara Uni-manager运营控制台登录API与工具控制台。


前提条件

- 登录Apsara Uni-manager运营控制台前，确认您已从部署人员处获取Apsara Uni-manager运营控制台的服务域名地址。
- 推荐使用Chrome浏览器

操作步骤

1. 在浏览器地址栏中，输入Apsara Uni-manager运营控制台的服务域名地址，按回车键。
2. 输入正确的用户名及密码。


请向运营管理员获取登录控制台的用户名和密码。

 **说明** 首次登录Apsara Uni-manager运营控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。为提高安全性，密码长度必须为10~32位，且至少包含以下两种类型：

- 英文大写或小写字母（A~Z、a~z）
- 阿拉伯数字（0~9）
- 特殊符号（感叹号（!）、at（@）、井号（#）、美元符号（\$）、百分号（%）等）

3. 单击登录。
4. 如果账号已激活MFA多因素认证，请根据以下两种情况进行操作：
 - 管理员强制开启MFA后的首次登录：
 - a. 在绑定虚拟MFA设备页面中，按页面提示步骤绑定MFA设备。
 - b. 按照步骤2重新输入账号和密码，单击登录。
 - c. 输入6位MFA码后单击认证。
 - 您已开启并绑定MFA：

输入6位MFA码后单击认证。

 **说明** 绑定并开启MFA的操作请参见Apsara Uni-manager运营控制台用户指南中的绑定并开启虚拟MFA设备章节。

5. 在页面顶部的菜单栏中，选择产品 > 其他 > API与工具。

1.2. 查看API信息

您需要获取产品名称、API名称和API版本，使用SDK时需要填写这些信息。

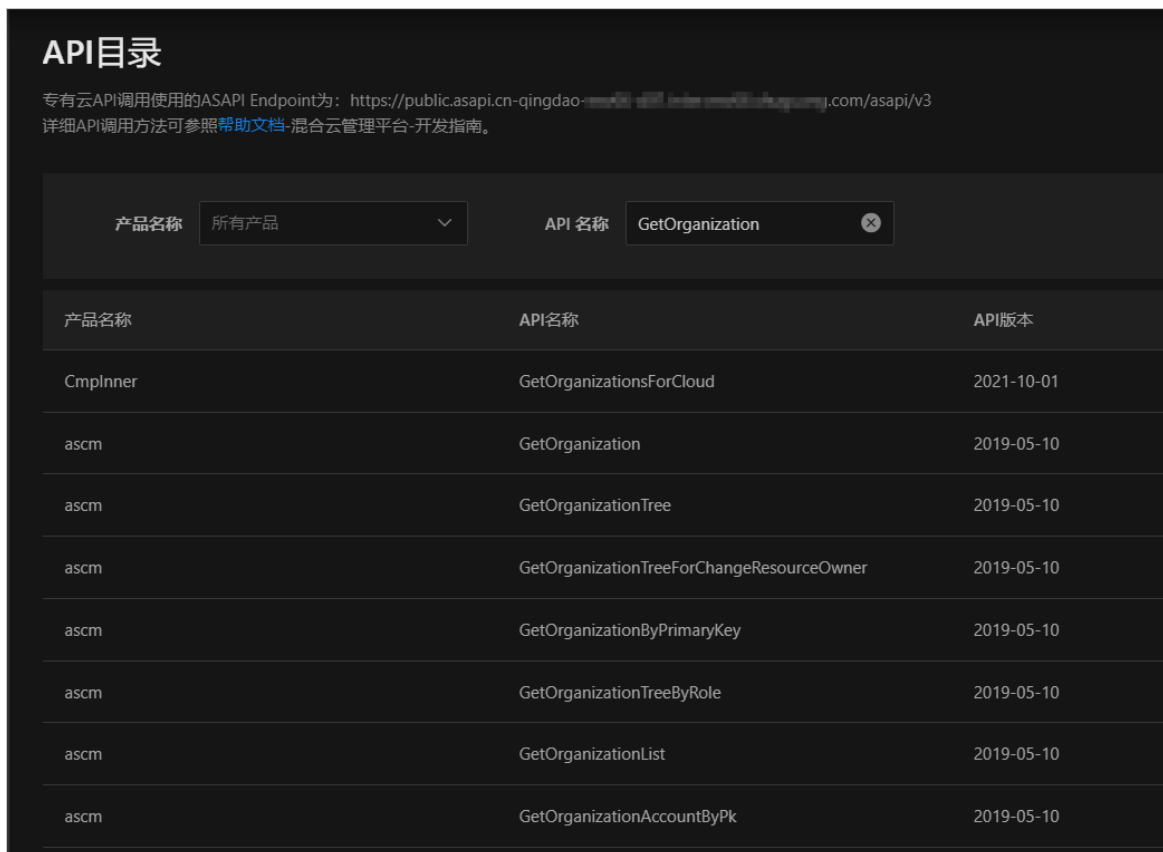
背景信息

API目录中的API仅供调用参考，具体API的使用方式需要参见云产品的开发指南。如果发生不一致情况，以开

发指南支持情况为准。

操作步骤

1. [登录API与工具控制台](#)。
2. 在左侧导航栏中选择**API目录**。
3. 在**API目录**页面，在下拉列表框中选择对应的产品，输入API名称的关键词进行搜索。



4. 查看并记录对应的产品名称、API名称和API版本。

1.3. 获取AccessKey

AccessKey支持RAM和STS两种授权模式，在发起调用时选择使用其中一种AccessKey即可。本文将为您介绍如何获取通过RAM授权的AccessKey。


背景信息

AccessKey通过使用AccessKey ID和AccessKey Secret对称加密，实现验证某个请求发送者的身份的功能。其中AccessKey ID用于标识用户，AccessKey Secret是用户用于加密签名的字符串。

AccessKey的两种授权模式如下：

- 通过RAM授权给第三方请求者的AccessKey ID和AccessKey Secret。
- 通过STS授权给第三方请求者的AccessKey ID、AccessKey Secret和Security Token。

Apsara Uni-manager运营控制台提供了个人AccessKey和组织AccessKey（只有运营管理员和一级组织管理员可以获取组织AccessKey），推荐使用个人账号AccessKey调用Apsara Uni-manager运营控制台及云产品相关接口。如果使用个人账号AccessKey，需要在Header中添加以下限制性参数，否则可能提示权限不足：



 **警告** 个人账号AccessKey是由Apsara Uni-manager运营控制台权限体系管控的受限AccessKey，组织AccessKey相对权限较大，需要管理员确认操作的安全性。


参数名称	描述
x-acs-regionid	地域，如：cn-hangzhou-*
x-acs-organizationid	Apsara Uni-manager运营控制台中对应的组织ID。获取方法可参见 <i>Apsara Uni-manager运营控制台开发指南</i> 中的 <i>GetOrganizationList</i> 章节。
x-acs-resourcegroupid	Apsara Uni-manager运营控制台中对应的资源集ID。获取方法可参见 <i>Apsara Uni-manager运营控制台开发指南</i> 中的 <i>ListResourceGroup</i> 章节。
x-acs-instanceid	进行操作的实例ID，例如，ECS的实例ID可以从ECS列表中查询到。

获取个人账号AccessKey

获取个人账号AccessKey的方法如下：

1. 登录Apsara Uni-manager运营控制台。
2. 在系统界面右上角，单击当前登录用户头像，单击**个人信息**。
3. 在**AccessKey**区域，您可以查看个人账户的AccessKey信息。

AccessKey + 创建AccessKey				
 AccessKey ID和AccessKey Secret是您访问云服务资源时的密钥，具有该账号完整的权限，请您妥善保管。 ×				
AccessKey ID	AccessKey Secret	状态	创建时间	
z*****w	*****	 启用	2022年01月13日 20:45:35	

 **说明** AccessKey ID和AccessKey Secret是您访问云资源时的密钥，具有该账号完整的权限，请您妥善保管。

获取组织AccessKey

获取组织AccessKey的方法如下：

1. 管理员登录Apsara Uni-manager运营控制台。
2. 在顶部菜单栏，单击**企业**。
3. 在左侧导航栏中，选择**资源管理 > 组织管理**。
4. 在组织结构中，单击目标一级组织名称。
5. 单击**管理Accesskey**。
6. 在弹出的对话框中，查看组织AccessKey信息。

1.4. 获取公共Header参数

调用专有云API时需要提供多个Header参数。本文为您介绍这些参数的含义及其获取方式，并对调用后的一些返回值做出解释。

Header参数描述

名称	描述
x-acs-regionid	环境Region，获取方式请参见 获取Region ID 。
x-acs-organizationid	Apsara Uni-manager运营控制台上的组织ID，获取方式请参见 获取组织 ID 。
x-acs-resourcegroupid	Apsara Uni-manager运营控制台上的资源集ID，获取方式请参见 获取资源集 ID 。
x-acs-instanceid	进行操作的实例ID，获取方式请参见 获取Project 。

获取Region ID

1. 管理员登录Apsara Uni-manager运营控制台。
2. 在顶部菜单栏中，单击企业。
3. 在左侧导航栏中，选择资源管理>地域管理。
4. 单击目标组织名称。
5. 在右侧Region列表中查看Region ID。

获取组织ID

组织ID需要通过API进行获取，获取方式请参见Apsara Uni-manager运营控制台开发指南中的[Get OrganizationList](#)章节。

获取资源集ID

资源集ID需要通过API进行获取，获取方式请参见Apsara Uni-manager运营控制台开发指南中的[List ResourceGroup](#)章节。

获取Project

Project可以通过Apsara Uni-manager运营控制台中的Project列表进行获取，获取步骤如下：

1. 登录日志服务控制台。
具体操作，请参见[日志服务用户指南中的登录日志服务控制台章节](#)。
2. 在Project列表中，获取Project名称。

返回参数

对于列表类的云产品资源实例接口，调用后会默认增加如下返回值。

名称	描述
ResourceGroupName	资源集名称。
ResourceGroup	资源集ID。
DepartmentName	组织名称。
Department	组织ID。

1.5. 获取日志服务的Endpoint

本文介绍如何获取日志服务Endpoint。

背景信息

调用专有云API时需要使用云产品业务域的Endpoint。您可以在阿里云驻场运维工程师处获取。您也可以通过如下方法自行获取Endpoint。

操作步骤

登录飞天基础运维平台。

1. 登录Apsara Uni-manager运维控制台。
2. 在顶部菜单栏单击产品运维，选择云平台运维>飞天基础运维平台。
3. 在左侧导航栏中单击报表。
4. 在全部报表页面搜索 服务注册变量，单击报表名称。



5. 在服务注册变量页面，单击Service旁的  图标，搜索 SLS。
6. 在 sls-backend-server 服务的Service Registration列中，单击鼠标右键，选择显示更多。

在详情页面，查看的sls_data.endpoint值，即 Endpoint。

1.6. 获取日志服务的SDK包

通过POP网关调用专有云API时，需使用相应云产品的SDK。本文为您介绍如何获取阿里云开发工具包（SDK）。

软件开发工具包（SDK）包括代码以及示例，用户可以自行选择语言创建云应用。

目前SDK的语言支持：Java、Python、Go等，您可以前往[阿里云开发工具包（SDK）](#)中进行选择。

说明

推荐使用 Java、Python、Go三种语言的SDK包，使用其他语言的SDK包可能发生不兼容情况。

SDK核心库推荐使用阿里云官网的SDK核心库，其内容所有云产品共用，且适用于专有云。

下表列举了日志服务常用语言的SDK的Git Hub源码地址。

SDK语言	Git Hub源码
Java	Log Service Java SDK
Python	Log Service Python SDK
Go	Log Service Go SDK
PHP	Log Service PHP SDK

2.SDK参考

2.1. 基本介绍

2.1.1. 概述

为方便开发人员更高效地使用日志服务，日志服务提供了多个语言版本（Java、.NET、Python、PHP、C）的 SDK（Software Development Kit），您可以根据自己需求选择合适版本使用。

日志服务 SDK 基于日志服务 API 实现，且提供和日志服务 API 同样的能力。如果您需要了解日志服务 API 的更多细节，参见 API 参考。

类似于使用日志服务 API，您首先需要拥有一个处于 Active 状态的访问密钥（AccessKeyId/AccessKeySecurity）。

为使用日志服务 SDK，您需要了解日志服务的服务入口。具体如何在 SDK 中指定这个根服务入口，请参考[配置](#)。

尽管不同语言的日志服务 SDK 具体实现细节会有所不同，但是它们都是日志服务 API 在不同语言上的封装，实现的功能也基本一致，具体包括如下几个方面：

- 实现对日志服务 API 接口的 **统一封装**，让您不需要关心具体的 API 请求构建和响应解析。而且各个不同语言的接口也非常接近，方便您在不同语言间切换。
- 实现日志服务 API 的数字签名逻辑，让您不需要关心 API 的签名逻辑细节，大大降低使用日志服务 API 的难度。
- 实现日志服务日志的 ProtoBuffer 格式封装，让您在写入日志时不需要关心 ProtoBuffer 格式的具体细节。
- 实现日志服务 API 中定义的压缩方法，让您不用关心压缩实现的细节。部分语言的 SDK 可以让您指定是否启用压缩模式写入日志（默认为使用压缩方式）。
- 提供统一的 **错误处理机制**，让您可以使用语言所熟悉的方式处理请求异常。
- 目前所有语言实现的 SDK 仅提供同步请求方式。

如下表所示，各个不同语言的 SDK 的下载地址、详细使用说明及完整的编程参考如下。

SDK相关文档及源码

SDK 语言	相关文档	源码
Java	Java SDK, 接口参考	GitHub
.NET	.NET SDK, 接口参考	GitHub
PHP	PHP SDK, 接口参考	GitHub
Node.js	Node.js SDK	GitHub
Python	Python SDK, 接口参考	GitHub
C	C SDK	GitHub
GO	GO SDK	GitHub

SDK 语言	相关文档	源码
iOS	iOS SDK	GitHub
Android	Android SDK	GitHub
C++	C++ SDK	GitHub

2.1.2. 配置

与使用API和日志服务服务端进行交互相同，使用SDK也需要指定一些基本配置。本文介绍SDK的基本配置信息。

具体包括如下几项：

- 服务入口（Endpoint）：确认 Client 需要访问的服务入口。
- 访问密钥（AccessKeyId/AccessKeySecret）：指定 Client 访问日志服务时使用的访问密钥。

下面详细说明这两个配置的使用方式。

服务入口（Endpoint）

当使用 SDK 时，首先需要确认您的接入方式和服务入口。

1. 当选择 Client 的 Endpoint 时，请根据您的部署按照正确的方式接入服务，否则 SDK 将无法访问您的项目。
2. 目前，所有 API 的服务入口仅支持 HTTP 协议。

访问密钥（AccessKey）

正如 API 参考文档中的访问密钥所述，所有和日志服务端交互的请求都必须经过安全验证，而访问密钥就是用来对请求进行安全验证的关键因子，且以 AccessKeyId 和 AccessKeySecret 方式成对出现。在 Client 构造时需要指定两个参数（AccessKeyId, AccessKeySecret）即为该访问密钥对。所以，在使用 SDK 前，请在云控制台获取合适的密钥对。

说明

- 您的账号下可以拥有多组访问密钥对，但在构造 Client 时指定的 AccessKeyId 和 AccessKeySecret 必须成对，否则无法通过服务端的安全验证。
- 指定的访问密钥对必须处于“启用”状态，否则会被服务端拒绝请求。同样，您也可以到云控制台查看访问密钥的状态。

示例

如果您需要访问某个 Project，且当前已经拥有一对处于“启用”状态的访问密钥对。如下：

```
AccessKeyId = "<yourAccessKeyId>"
AccessKeySecret = "<yourAccessKeySecret>"
```

则可以用如下方式创建 Client 实例：

- Java


```
String endpoint = "regionid.example.com";           //在实际使用中，请按照您实际的服务入口和接入方式编写。
String accessKeyId = "<youraccessKeyId>";           //用户访问密钥对中的 AccessKeyId。
String accessKeySecret = "<youraccessKeySecret>"; //用户访问密钥对中的 AccessKeySecret。
Client client = new Client(endpoint, accessKeyId, accessKeySecret);
//use client to operate log service project.....
```

- .NET(C#)

```
String endpoint = "regionid.example.com";           // 在实际使用中，请按照您实际的服务入口和接入方式编写。
String accessKeyId = "<youraccessKeyId>";           //用户访问密钥对中的 AccessKeyId。
String accessKeySecret = "<youraccessKeySecret>"; //用户访问密钥对中的 AccessKeySecret。
SLSCClient client = new SLSCClient(endpoint, accessKeyId, accessKeySecret);
//use client to operate sls project.....
```

- PHP

```
$endpoint = 'regionid.example.com'; //此处以hangzhou Region为例，在实际使用中，请按照您实际的服务入口和接入方式编写。
$accessKeyId = '<youraccessKeyId>'; //用户访问密钥对中的 AccessKeyId。
$accessKey = '<youraccessKey>'; //用户访问密钥对中的 AccessKeySecret。
$client = new Aliyun_Sls_Client($endpoint, $accessKeyId, $accessKey);
//use client to operate sls project.....
```

- Python

```
# // 在实际使用中，请按照您实际的服务入口和接入方式编写。
endpoint = 'regionid.example.com'
# 用户访问密钥对中的 AccessKeyId
accessKeyId = '<youraccessKeyId>'
# 用户访问密钥对中的 AccessKeySecret。
accessKey = '<youraccessKey>'
client = LogClient(endpoint, accessKeyId, accessKey)
#use client to operate log project.....
```

2.1.3. 错误处理

SDK 可能出现的异常错误可以分成如下几类：

- 由日志服务端返回的错误。这类错误由日志服务端返回并由 SDK 处理。关于这类错误的详细信息可以参考日志服务 API 的通用错误码和各个 API 接口的具体说明。
- 由 SDK 在向服务端发出请求时出现的网络错误。这类错误包括网络连接不通，服务端返回超时等。
- 由 SDK 自身产生的、与平台及语言相关的错误，如内存溢出等。

目前，各个语言 SDK 的实现都采取抛出异常的方式处理错误。具体原则如下：

- 由如上第一或者第二类错误将会被 SDK 处理并包装在统一的 LogException 类抛出给用户处理。
- 由如上第三类错误不会被 SDK 处理，而是直接抛出平台及语言的 Native Exception 类给用户处理。

LogException

LogException 类是 SDK 定义的、用于处理日志服务自身逻辑错误的异常类。它继承自各个语言的异常基类，提供如下异常信息：

- 错误代码（Error Code）：标示错误类型。如果是来自服务端的返回错误，则这个错误代码与 API 返回的

错误代码一致。如果是 SDK 网络请求错误，则其错误代码为 “RequestError”。具体请参考各个语言的完整 API 参考。

- 错误消息（Error Message）：标示错误消息。如果是来自服务端的响应错误，则这个错误消息 API 返回的错误消息一致。如果是 SDK 网络请求错误，则其错误消息为 “request is failed.”。具体请参考各个语言的完整 API 参考。
- 错误请求 ID（Request Id）：标示当前错误对应于服务端的请求 ID。该 ID 只有在服务端返回错误消息时有效，否则为空字符串。用户可以在遇到错误请求时记下该请求 ID 并提供给日志服务团队进行问题追踪和定位。

请求失败与重试

在使用 SDK 访问日志服务端时，有可能会因为网络临时中断、传输延时过程、服务端处理过慢等一系列原因导致请求失败。目前，这类错误都直接以异常抛出，日志服务内部并未对此做任何重试逻辑。所以，您在使用 SDK 时需要自己定义相应的处理逻辑（重试请求或者直接报错等）。

示例

假设您需要访问名字为 **big-game** 的 Project，且在出现网络异常时主动重试指定次数。各语言的代码片段如下：

- Java

```
//其他代码.....
String accessId = "<youraccessId>"; //TODO: 用您的真实阿里云 AccessKeyId 替代。
String accessKey = "<youraccessKey>"; //TODO: 用您的真实阿里云 AccessKeySecret 替代。
String project = "big-game";
String endpoint = "regionid.example.com";
int max_retries = 3;
/*
 * 构建一个 client
 */
Client client = new Client(accessId, accessKey, endpoint);
ListLogStoresRequest lsRequest = new ListLogStoresRequest(project);
for (int i = 0; i < max_retries; i++)
{
    try
    {
        ListLogStoresResponse res = client.ListLogStores(lsRequest)
        //TODO: 处理返回的 response.....
        break;
    }
    catch(LogException ex)
    {
        if (e.GetErrorCode() == "RequestError")
        {
            if ( i == max_retries - 1)
            {
                System.out.println("request is still failed after all retries.");
                break;
            }
            else
            {
                System.out.println("request error happens, retry it!");
            }
        }
        else
        {
            System.out.println("error code :" + e.GetErrorCode());
            System.out.println("error message :" + e.GetErrorMessage());
            System.out.println("error requestId :" + e.GetRequestId());
            break;
        }
    }
    catch(...)
    {
        System.out.println("unrecoverable exception when listing logstores.");
        break;
    }
}
//其他代码.....
```

- .NET (C#)

```
//其他代码.....
String accessId = "<youraccessId>";    //TODO: 用您的真实阿里云 AccessKeyId 替代。
String accessKey = "<youraccessKey>";  //TODO: 用您的真实阿里云 AccessKeySecret 替代。
String project = "big-game";
String endpoint = "regionid.example.com";
int max_retries = 3;
//创建一个 Client
SLSClient client = new SLSClient(endpoint, accessId, accessKey);
ListLogstoresRequest request = new ListLogstoresRequest();
request.Project = project;
for (int i = 0; i < max_retries; i++)
{
    try
    {
        ListLogstoresResponse response = client.ListLogstores(request);
        //TODO: 处理返回的 response.....
        break;
    }
    catch(LogException ex)
    {
        if (e.errorCode == "SLSRequestError")
        {
            if ( i == max_retries - 1)
            {
                Console.WriteLine("request is still failed after all retries.");
                break;
            }
            else
            {
                Console.WriteLine("request error happens, retry it!");
            }
        }
        else
        {
            Console.WriteLine("error code :" + e.errorCode;
            Console.WriteLine("error message :" + e.Message;
            Console.WriteLine("error requestId :" + e.RequestId;
            break;
        }
    }
    catch(...)
    {
        Console.WriteLine("unrecoverable exception when listing logstores.");
        break;
    }
}
//其他代码.....
```

- PHP

```
<?php
//其他代码.....
$endpoint = 'regionid.example.com';
$accessId = '<youraccessId>'; // TODO: 用你的真实阿里云 AccessKeyId 替代
$accessKey = '<youraccessKey>'; //TODO: 用你的真实阿里云 AccessKeySecret 替代
$maxRetries = 3;
// 构建一个 sls client
$client = new Aliyun_Sls_Client($endpoint, $accessId, $accessKey);
$project = 'big-game';
$request = new Aliyun_Sls_Models_ListLogstoresRequest($project);
for($i = 0; $i < $maxRetries; ++$i)
{
    try
    {
        $response = $client->ListLogstores($request);
        //TODO: 处理返回的 response.....
        break;
    }
    catch (Aliyun_Sls_Exception $e)
    {
        if ($e->getErrorCode()=='RequestError')
        {
            if ($i+1 == $maxRetries)
            {
                echo "error code :" . $e->getErrorCode() . PHP_EOL;
                echo "error message :" . $e->getErrorMessage() . PHP_EOL;
                break;
            }
            echo 'request error happens, retry it!' . PHP_EOL;
        }
        else
        {
            echo "error code :" . $e->getErrorCode() . PHP_EOL;
            echo "error message :" . $e->getErrorMessage() . PHP_EOL;
            echo "error requestId :" . $e->getRequestId() . PHP_EOL;
            break;
        }
    }
}
catch (Exception $ex)
{
    echo 'unrecoverable exception when listing logstores.' . PHP_EOL;
    var_dump($ex);
    break;
}
}
//其他代码.....
```

- Python

```
//其他代码.....
endpoint = 'regionid.example.com'
accessId = '<youraccessId>' # TODO: 用你的真实阿里云 AccessKeyId 替代
accessKey = '<youraccessKey>' # TODO: 用你的真实阿里云 AccessKeySecret 替代
maxRetries = 3
# 构建一个 client
client = Client(endpoint, accessId, accessKey)
project = 'big-game'
lsRequest = ListLogstoresRequest(project)
for i in xrange(maxRetries):
    try:
        res = client.ListLogstores(lsRequest)
        # TODO: 处理返回的 response.....
        break
    except LogException as e:
        if e.getErrorCode() == "RequestError":
            if i+1 == maxRetries:
                print "error code :" + e.getErrorCode()
                print "error message :" + e.getErrorMessage()
                break
            else:
                print "request error happens, retry it!"
        else:
            print "error code :" + e.getErrorCode()
            print "error message :" + e.getErrorMessage()
            print "error requestId :" + e.getRequestId()
            break
    except Exception as e:
        print 'unrecoverable exception when listing logstores.'
        break
//其他代码.....
```

2.1.4. 接口规范

尽管不同语言的 SDK 实现有所不同，但其接口都遵循 Request-Response 原则，即对 API 的调用按照如下方式进行：

1. 利用请求参数构建相应的 Request 实例。
2. 调用 SDK 中的相应接口并传入上一步的 Request 实例。
3. SDK 接口的返回结果以相应的 Response 实例返回给用户。

如下代码片段解释如何按照上面流程获取一个 Project 下的所有 Logstore 的名称。

Java

```
// 其他代码.....
String accessId = "<youraccessId>"; //TODO: 用您的真实阿里云 AccessKeyId 替代。
String accessKey = "<youraccessKey>"; //TODO: 用您的真实阿里云 AccessKeySecret 替代。
String project = "your_project"; //TODO: 用您的真实 project 名称替代。
String endpoint = "region_endpoint"; //TODO: 此处以hangzhou Region为例, 在实际使用中, 请按照您实际的服务入口和接入方式编写。
//构建一个 Client 实例。
Client client = new Client(endpoint, accessId, accessKey);
//用请求参数"project"初始化 ListLogstores 的请求类。
ListLogStoresRequest lsRequest = new ListLogStoresRequest(project);
//使用 request 实例调用 ListLogstores 接口, 且返回参数为对应的 Response 实例
ListLogStoresResponse res = client.ListLogStores(lsRequest);
//访问 Response 实例获取请求结果
ArrayList<String> names = res.GetLogStores();
// 其他代码.....
```

.NET(C#)

```
// 其他代码.....
String accessId = "<youraccessId>"; //TODO: 用您的真实阿里云 AccessKeyId 替代。
String accessKey = "<youraccessKey>"; //TODO: 用您的真实阿里云 AccessKeySecret 替代。
String project = "your_project"; //TODO: 用您的真实 project 名称替代。
String endpoint = "region_endpoint"; //TODO: 此处以hangzhou Region为例, 在实际使用中, 请按照您实际的服务入口和接入方式编写。
//构建一个 Client 实例。
SLSCClient client = new SLSCClient(endpoint, accessId, accessKey);
//用请求参数"project"初始化 ListLogstores 的请求类。
ListLogStoresRequest lsRequest = new ListLogStoresRequest();
lsRequest.Project = project;
//使用 request 实例调用 ListLogstores 接口, 且返回参数为对应的 Response 实例
ListLogStoresResponse res = client.ListLogStores(lsRequest);
//访问 Response 实例获取请求结果
List<String> names = res.Logstores;
// 其他代码.....
```

PHP


```
// 其他代码.....
$accessId = "<youraccessId>"; //TODO: 用您的真实阿里云 AccessKeyId 替代。
$accessKey = "<youraccessKey>"; //TODO: 用您的真实阿里云 AccessKeySecret 替代。
$project = "your_project"; //TODO: 用您的真实 project 名称替代。
$endpoint = "region_endpoint";////TODO: 此处以hangzhou Region为例, 在实际使用中, 请按照您实际的服务入口和接入方式编写。
//构建一个 SLS Client 实例。
$client = new Aliyun_Sls_Client($endpoint, $accessId, $accessKey);
//用请求参数“project”初始化 ListLogstores 的请求类。
$request = new Aliyun_Sls_Models_ListLogstoresRequest($project);
//使用 request 实例调用 ListLogstores 接口, 且返回参数为对应的 Response 实例
$response = $client->listLogstores($request);
//访问 Response 实例获取请求结果
$names = $response->getLogstores();
// 其他代码.....
```

Python

```
// 其他代码.....
accessId = '<youraccessId>'; //TODO: 用您的真实阿里云 AccessKeyId 替代。
accessKey = '<youraccessKey>'; //TODO: 用您的真实阿里云 AccessKeySecret 替代。
project = 'your_project'; //TODO: 用您的真实 project 名称替代。
endpoint = 'region_endpoint';////TODO: 此处以hangzhou Region为例, 在实际使用中, 请按照您实际的服务入口和接入方式编写。
# 构建一个 client
client = LogClient(endpoint, accessId, accessKey)
# 用请求参数 “project” 初始化 ListLogstores 的请求类。
lsRequest = ListLogstoresRequest(project)
# 使用 request 实例调用 ListLogstores 接口, 且返回参数为对应的 Response 实例
res = client.list_logstores(lsRequest)
# 访问 Response 实例获取请求结果
names = res.get_logstores();
// 其他代码.....
```

SDK 实现了多组类似 ListLogStores 的接口, 也定义了相应的 Request 和 Response 类。除去 Request-Response 风格的基础接口外, 各个不同语言的 SDK 还会提供一些包装了这些基础接口的辅助接口, 让您无需自己构建 Request 及解析最终 Response 内容。这类接口的细节请见各 SDK 的 API 参考。

2.2. Java SDK

Log Service 的 Java SDK 让 Java 开发人员可以非常方便地使用 Java 程序操作阿里云日志服务。开发者可以直接使用Maven依赖添加SDK, 也可以下载包到本地。

下载地址

SDK 支持 J2SE 6.0 及以上版本, 单击[此处](#)下载最新版SDK。

步骤 1 创建账号

为了访问日志服务, 您需要有一个控制台账号。

步骤 2 获取访问密钥

为了使用 Log Service Java SDK，您必须有一对访问密钥。

登录控制台。选择一对用于 SDK 的访问密钥对。

该密钥对会在下面的步骤使用，且需要保管好，不能对外泄露。另外，您可以参考 [配置](#) 了解更多 SDK 如何使用访问密钥的信息。

步骤 3 创建一个日志服务项目和日志库

使用 Log Service Java SDK 创建日志项目（Project）和日志库（Logstore）。

? 说明

- 请确保使用同一账号获取访问密钥和创建日志项目及日志库。
- Log 的 Project 名称为日志服务全局唯一，而 Logstore 名称在一个 Project 下面唯一。
- Log 的 Project 一旦创建则无法更改它的所属区域。目前也不支持在不同 Region 间迁移 Log Project。

步骤 4 安装 Java 开发环境

目前，Log Java SDK 支持 J2SE 6.0 及以上的 Java 运行环境，您可以从 [Java 官方网站](#) 下载并按说明安装 Java 开发环境。

步骤 5 安装 Log Service Java SDK

在安装完 Java 开发环境后，您需要安装 Log Service Java SDK。目前，我们提供两种方式安装日志服务的 Java SDK：

建议使用 [Apache Maven](#) 获取最新版本的 SDK，您可以添加如下配置到您的 Maven 项目。

```
<dependency>
  <groupId>com.google.protobuf</groupId>
  <artifactId>protobuf-java</artifactId>
  <version>2.5.0</version>
</dependency>
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>aliyun-log</artifactId>
<version>0.6.7</version>
<exclusions>
  <exclusion>
    <groupId>com.google.protobuf</groupId>
    <artifactId>protobuf-java</artifactId>
  </exclusion>
</exclusions>
</dependency>
```

您也可以完整下载 Java SDK 软件包，然后在自己的 Java 项目中直接引用本地软件包。

1. 下载 Java SDK 包（版本会定期更新，如需使用最新版本请使用 Maven）。
2. 解压完整下载的包到指定的目录即可。Java SDK 是一个软件开发包，不需要额外的安装操作。
3. 把 SDK 包中的所有 Jar 包（包括依赖的第三方包）添加到您的 Java 工程（具体操作请参照不同的 IDE 文档）。

步骤 6 开始一个新的 Java 项目

现在，您可以开始使用 SDK Java SDK。使用任何文本编辑器或者 Java IDE，运行如下示例代码即可与 Log Service 服务端交互并得到相应输出。

```
package sdksample;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import java.util.Date;
import com.aliyun.openservices.log.Client;
import com.aliyun.openservices.log.common.*;
import com.aliyun.openservices.log.exception.*;
import com.aliyun.openservices.log.request.*;
import com.aliyun.openservices.log.response.*;
import com.aliyun.openservices.log.common.LogGroupData;
import com.aliyun.openservices.log.common.LogItem;
import com.aliyun.openservices.log.common.Logs.Log;
import com.aliyun.openservices.log.common.Logs.Log.Content;
import com.aliyun.openservices.log.common.Logs.LogGroup;
import com.aliyun.openservices.log.common.Consts.CursorMode;
public class sdksample {
    public static void main(String args[]) throws LogException, InterruptedException {
        String endpoint = "<log_service_endpoint>"; // 选择与上面步骤创建 project 所属区域匹配的
        // Endpoint
        String accessKeyId = "<youraccessKeyId>"; // 使用您的访问密钥 AccessKeyId
        String accessKeySecret = "<youraccessKeySecret>"; // 使用您的访问密钥AccessKey Secret
        // AccessKeySecret
        String project = "<project_name>"; // 上面步骤创建的项目名称
        String logstore = "<logstore_name>"; // 上面步骤创建的日志库名称
        // 构建一个客户端实例
        Client client = new Client(endpoint, accessKeyId, accessKeySecret);
        // 列出当前 project 下的所有日志库名称
        int offset = 0;
        int size = 100;
        String logStoreSubName = "";
        ListLogStoresRequest req1 = new ListLogStoresRequest(project, offset, size, logStoreSubName);
        ArrayList<String> logStores = client.ListLogStores(req1).GetLogStores();
        System.out.println("ListLogs:" + logStores.toString() + "\n");
        // 写入日志
        String topic = "";
        String source = "";
        // 连续发送 10 个数据包，每个数据包有 10 条日志
        for (int i = 0; i < 10; i++) {
            Vector<LogItem> logGroup = new Vector<LogItem>();
            for (int j = 0; j < 10; j++) {
                LogItem logItem = new LogItem((int) (new Date().getTime() / 1000));
                logItem.PushBack("index"+String.valueOf(j), String.valueOf(i * 10 + j));
                logGroup.add(logItem);
            }
            PutLogsRequest req2 = new PutLogsRequest(project, logstore, topic, source, logGroup);
        }
    }
}
```

```

client.PutLogs(req);
/*
 * 发送的时候也可以指定将数据发送至有一个特定的 shard，只要设置 shard 的 hashkey，则数
据会写入包含该
 * hashkey 的 range 所对应的 shard，具体 API 参考以下接口：PutLogsResponse
 * PutLogs( String project, String logStore, String topic,
 * List<LogItem> logItems, String source, String shardHash // 根据
 * hashkey 确定写入 shard, hashkey 可以是 MD5(ip) 或 MD5(id) 等 );
 */
}
// 把 0 号 shard 中，最近 1 分钟写入的数据都读取出来。
int shard_id = 0;
long curTimeInSec = System.currentTimeMillis() / 1000;
GetCursorResponse cursorRes = client.GetCursor(project, logstore, shard_id, curTime
InSec - 60);
String beginCursor = cursorRes.GetCursor();
cursorRes = client.GetCursor(project, logstore, shard_id, CursorMode.END);
String endCursor = cursorRes.GetCursor();
String curCursor = beginCursor;
while (curCursor.equals(endCursor) == false) {
    int loggroup_count = 2; // 每次读取两个 loggroup
    BatchGetLogResponse logDataRes = client.BatchGetLog(project, logstore, shard_id
, loggroup_count, curCursor,
        endCursor);
    // 读取LogGroup的List
    List<LogGroupData> logGroups = logDataRes.GetLogGroups();
    for(LogGroupData logGroup: logGroups){
        FastLogGroup flg = logGroup.GetFastLogGroup();
        System.out.println(String.format("\tcategory\t:\t%s\n\tsource\t:\t%s\n\ttop
ic\t:\t%s\n\tmachineUUID\t:\t%s",
            flg.getCategory(), flg.getSource(), flg.getTopic(), flg.getMachineU
UID()));
        System.out.println("Tags");
        for (int tagIdx = 0; tagIdx < flg.getLogTagsCount(); ++tagIdx) {
            FastLogTag logtag = flg.getLogTags(tagIdx);
            System.out.println(String.format("\t%s\t:\t%s", logtag.getKey(), logtag
.getValue()));
        }
        for (int lIdx = 0; lIdx < flg.getLogsCount(); ++lIdx) {
            FastLog log = flg.getLogs(lIdx);
            System.out.println("-----\nLog: " + lIdx + ", time: " + log.getTime(
) + ", GetContentCount: " + log.getContentsCount());
            for (int cIdx = 0; cIdx < log.getContentsCount(); ++cIdx) {
                FastLogContent content = log.getContents(cIdx);
                System.out.println(content.getKey() + "\t:\t" + content.getValue())
;
            }
        }
    }
    String next_cursor = logDataRes.GetNextCursor();
    System.out.println("The Next cursor:" + next_cursor);
    curCursor = next_cursor;
}
// !!! 重要提示：只有打开索引功能，才能调用以下接口 !!!
// 等待 1 分钟让日志可查询

```

```

try {
    Thread.sleep(60 * 1000);
} catch (InterruptedException e) {
    e.printStackTrace();
}
// 查询日志分布情况
String query = "<此处为需要查询的关键词，如果查询全部内容设置为空字符串即可>";
int from = (int) (new Date().getTime() / 1000 - 300);
int to = (int) (new Date().getTime() / 1000);
GetHistogramsResponse res3 = null;
while (true) {
    GetHistogramsRequest req3 = new GetHistogramsRequest(project, logstore, topic,
query, from, to);
    res3 = client.GetHistograms(req3);
    if (res3 != null && res3.IsCompleted()) // IsCompleted() 返回
// true，表示查询结果是准确的，如果返回
// false，则重复查询
    {
        break;
    }
    Thread.sleep(200);
}
System.out.println("Total count of logs is " + res3.GetTotalCount());
for (Histogram ht : res3.GetHistograms()) {
    System.out.printf("from %d, to %d, count %d.\n", ht.GetFrom(), ht.GetTo(), ht.G
etCount());
}
// 查询日志数据
long total_log_lines = res3.GetTotalCount();
int log_offset = 0;
int log_line = 10;
while (log_offset <= total_log_lines) {
    GetLogsResponse res4 = null;
    // 对于每个 log offset,一次读取 10 行 log，如果读取失败，最多重复读取 3 次。
    for (int retry_time = 0; retry_time < 3; retry_time++) {
        GetLogsRequest req4 = new GetLogsRequest(project, logstore, from, to, topic
, query, log_offset,
            log_line, false);
        res4 = client.GetLogs(req4);
        if (res4 != null && res4.IsCompleted()) {
            break;
        }
        Thread.sleep(200);
    }
    System.out.println("Read log count:" + String.valueOf(res4.GetCount()));
    log_offset += log_line;
}
//打开分析功能,只有打开分析功能，才能使用SQL 功能。 可以在控制台开通分析功能，也可以使用SDK开
启分析功能
IndexKeys indexKeys = new IndexKeys();
ArrayList<String> tokens = new ArrayList<String>();
tokens.add(",");
tokens.add(".");
tokens.add("#");

```

```
IndexKey keyContent = new IndexKey(tokens,false,"text");
indexKeys.AddKey("index0",keyContent);
keyContent = new IndexKey(new ArrayList<String>(),false,"long");
indexKeys.AddKey("index1",keyContent);
keyContent = new IndexKey(new ArrayList<String>(),false,"double");
indexKeys.AddKey("index2",keyContent);
IndexLine indexLine = new IndexLine(new ArrayList<String>(),false);
Index index = new Index(7,indexKeys,indexLine);
CreateIndexRequest createIndexRequest = new CreateIndexRequest(project,logstore,index);

client.CreateIndex(createIndexRequest);
//使用分析功能
GetLogsRequest req4 = new GetLogsRequest(project, logstore, from, to, "", " index0:
value | select avg(index1) as v1,sum(index2) as v2, index0 group by index0");
GetLogsResponse res4 = client.GetLogs(req4);
if(res4 != null && res4.IsCompleted()){
    for (QueriedLog log : res4.GetLogs()){
        LogItem item = log.GetLogItem();
        for(LogContent content : item.GetLogContents()){
            System.out.print(content.GetKey()+" "+content.GetValue());
        }
        System.out.println();
    }
}
}
```

注意事项

- 为了提高您的系统的IO效率，请尽量不要直接使用SDK往日志服务中写数据，写数据标准做法参考 *用户指南* 中 *Producer Library* 章节。
- 要消费日志服务中的数据，请尽量不要直接使用SDK的拉数据接口，我们提供了一个高级消费库，该库屏蔽了日志服务的实现细节，并且提供了负载均衡、按序消费等高级功能。关于该消费库的详细信息，请参考 *用户指南* 中 *Consumer Library* 章节。

2.3. .NET SDK

为方便开发人员更高效地使用日志服务，日志服务提供了 .NET SDK。

下载地址

日志服务的 .NET SDK让Windows 平台上的开发人员可以非常方便地使用 .NET 平台操作日志服务。目前，SDK 支持 .NET Framework 3.5/4.0/4.5 版本。尽管针对不同 .NET Framework 版本的 SDK 文件不同，但是其接口和实现功能完全一致。

该 SDK GitHub 地址如下：[单击此处进入GitHub](#)

步骤 1 创建账号

为了访问日志服务，您需要有一个控制台账号。

步骤 2 获取访问密钥

为了使用 Log Service .NET SDK，您必须有一对访问密钥。

登录云控制台，选择一对用于 SDK 的访问密钥对。

该密钥对会在下面的步骤使用，且需要保管好，不能对外泄露。另外，您可以参考 [配置](#) 了解更多 SDK 如何使用访问密钥的信息。

步骤 3 创建一个日志服务项目和日志库

使用 Log Service Java SDK 创建日志项目（Project）和日志库（Logstore）。

说明

- 请确保使用同一账号获取访问密钥和创建日志项目及日志库。
- Log 的 Project 名称为日志服务全局唯一，而 Logstore 名称在一个 Project 下面唯一。
- Log 的 Project 一旦创建则无法更改它的所属区域。目前也不支持在不同 Region 间迁移 Log project。

步骤 4 安装 .NET 开发环境

目前，日志服务 SDK 支持 .NET 3.5 和 .NET 4.0/4.5 运行环境。为支持日志服务 SDK 开发，建议安装：

- Microsoft .NET Framework 3.5/4.0/4.5（具体版本依赖于您的程序需要运行的目标环境）
- Visual Studio 2010 及其以后版本

步骤 5 下载并安装日志服务 .NET SDK

搭建好 .NET 开发环境后，您需要安装日志服务的 .NET SDK。具体如下：

1. 下载。
 - 通过 Github 下载：<https://github.com/aliyun/aliyun-log-csharp-sdk>
 - 历史版下载：从 [这里](#) 下载最新的日志服务 .NET SDK 包
2. 解压完整下载的包到指定的目录即可。日志服务 .NET SDK 是一个软件开发包，不需要额外的安装操作。您可以按照下面的步骤直接在自己的 Visual Studio 项目中使用。

步骤 6 开始一个新的日志服务 .NET 项目

安装好 .NET 开发环境及日志服务 .NET SDK 后，您就可以开始创建日志服务的 .NET 项目。具体请参见 [Github](#) SLS SDK 4.0 解决方案中的 LOG SDK Sample 项目。

2.4. .NET Core SDK

背景信息

日志服务的 .NET Core SDK 让跨平台的开发人员可以非常方便地使用 .NET Core 框架操作阿里云日志服务。

.NET Core SDK GitHub 地址：[单击进入 GitHub](#)。

操作步骤

1. 创建账号。

为了访问日志服务，您需要有一个控制台账号。
2. 获取访问密钥。

为了使用 Log Service .NET Core SDK，您必须拥有一对访问密钥。

登录控制台，选择一对用于 SDK 的访问密钥对。

该密钥对会在下面的步骤使用，且需要保管好，不能对外泄露。关于SDK如何使用访问密钥的信息，详情请参见[配置](#)。

3. 创建日志服务项目和日志库。

在进行SDK操作之前，请先在控制台上创建Project和Logstore。

- i. 登录日志服务控制台。
- ii. 单击右上角的创建Project。
- iii. 填写Project名称和所属区域，单击**确认**。
- iv. 在Project列表页面，单击项目的名称，然后单击**创建**。
或者在创建完项目后，根据系统提示创建日志库，单击**创建**。
- v. 填写日志库的配置信息并单击**确认**。

您需要在此处填写Logstore名称、数据保存时间、Shard数目等。请按照您的需求合理设置Shard数目，例如本文实例中，您需要设置4个Shard。

说明

- 请确保使用同一账号获取访问密钥和创建日志项目及日志库。
- Log的Project名称为日志服务全局唯一，而Logstore名称在一个Project下面唯一。
- Log的Project一旦创建则无法更改它的所属区域。目前也不支持在不同Region间迁移Log project。

4. 安装.NET Core开发环境。

目前，日志服务.NET Core SDK支持以下平台版本：

- .NET Core 2.0
- .NET Framework (with .NET Core 1.x SDK) 4.6.2
- .NET Framework (with .NET Core 2.0 SDK) 4.6.1

支持的全部版本请参见[Git Hub](#)。

5. 下载并安装日志服务.NET Core SDK。

搭建好.NET Core开发环境后，您需要安装日志服务的.NET Core SDK。

- i. 下载.NET Core SDK。
单击[此处](#)进入Git hub下载。
- ii. 解压安装包到指定的目录。

日志服务.NET Core SDK是一个软件开发包，不需要额外的安装操作。您可以直接在自己的Visual Studio项目中使用。

6. 开始一个新的日志服务.NET Core项目。

安装好.NET开发环境及日志服务.NET Core SDK后，您就可以开始创建日志服务的.NET Core项目。具体请参见[Git hub Wiki](#)。

2.5. PHP SDK

为方便开发人员更高效地使用日志服务，日志服务提供了 PHP SDK。

下载地址

该 SDK Git Hub 地址如下：[GitHub](#)。

步骤 1 创建账号

为了访问日志服务，您需要有一个控制台账号。

步骤 2 获取访问密钥

为了使用 Log Service PHP SDK，您必须申请访问密钥。

登录控制台，选择一对用于 SDK 的访问密钥对。

该密钥对会在下面的步骤使用，且需要保管好，不能对外泄露。另外，您可以参考[配置](#)了解更多 SDK 如何使用访问密钥的信息。

步骤 3 创建一个日志服务项目和日志库

使用 Log Service PHP SDK 创建日志项目（Project）和日志库（Logstore）。

? 说明

- 请确保使用同一账号获取访问密钥和创建日志项目及日志库。
- Log 的 Project 名称为日志服务全局唯一，而 Logstore 名称在一个 Project 下面唯一。
- Log 的 Project 一旦创建则无法更改它的所属区域。目前也不支持在不同 Region 间迁移 Log Project。

步骤 4 安装 PHP 开发环境

PHP SDK 支持 PHP 5.2.1 及以上版本，您可以在本地安装 SDK 所支持的任何 PHP 版本并搭建好相应的 PHP 开发环境。

步骤 5 下载并安装 PHP SDK

搭建好 PHP 开发环境后，您需要安装 PHP SDK。具体如下：

1. 从 [GitHub](#) 下载最新的 PHP SDK 包。
2. 解压完整下载的包到指定的目录即可。PHP SDK 是一个软件开发包，不需要额外的安装操作。这个软件开发包除了包括 SDK 自身的代码外，还有一组第三方依赖包和一个 autoloader 类帮助你简化使用流程。您可以按照下面的步骤直接在自己的 PHP 项目中使用。

步骤 6 开始一个新的 PHP 项目

使用任何文本编辑器或者 PHP IDE，运行如下示例代码即可与日志服务端交互并得到相应输出。

```
<?php
/* 使用 autoloader 类自动加载所有需要的 PHP 模块。注意使用合适的路径指向 autoloader 类所在文件*/
require_once realpath(dirname(__FILE__) . '/../Log_Autoload.php');
$endpoint = 'regionid.example.com'; // 此处以hangzhou Region为例，在实际使用中，请按照您实际的服务入口和接入方式编写。
$accessKeyId = '<youraccessKeyId>'; // 使用你的访问秘钥 AccessKeyId
$accessKey = '<youraccessKey>'; // 使用你的访问秘钥 AccessKeySecret
$project = 'your_project'; // 上面步骤创建的项目名称
$logstore = 'your_logstore'; // 上面步骤创建的日志库名称
$client = new Aliyun_Log_Client($endpoint, $accessKeyId, $accessKey);
#列出当前 project 下的所有日志库名称
$req1 = new Aliyun_Log_Models_ListLogstoresRequest($project);
$res1 = $client->listLogstores($req1);
```

```

var_dump($res1);
#创建 logstore
$req2 = new Aliyun_Log_Models_CreateLogstoreRequest($project,$logstore,3,2);
$res2 = $client->createLogstore($req2);
#等待 logstore 生效
sleep(60);
#写入日志
$topic = "";
$source = "";
$logitems = array();
for ($i = 0; $i < 5; $i++)
{
    $contents = array('index1'=> strval($i));
    $logItem = new Aliyun_Log_Models_LogItem();
    $logItem->setTime(time());
    $logItem->setContents($contents);
    array_push($logitems, $logItem);
}
$req2 = new Aliyun_Log_Models_PutLogsRequest($project, $logstore, $topic, $source, $logitems);
$res2 = $client->putLogs($req2);
var_dump($res2);
#不必等待，立即拖数据
#首先遍历有哪些 shardId
$listShardRequest = new Aliyun_Log_Models_ListShardsRequest($project,$logstore);
$listShardResponse = $client->listShards($listShardRequest);
foreach($listShardResponse->getShardIds() as $shardId)
{
    #对每一个 ShardId，先获取 Cursor
    $getCursorRequest = new Aliyun_Log_Models_GetCursorRequest($project,$logstore,$shardId,
    null, time() - 60);
    $response = $client->getCursor($getCursorRequest);
    $cursor = $response->getCursor();
    $count = 100;
    while(true)
    {
        #从 cursor 开始读数据
        $batchGetDataRequest = new Aliyun_Log_Models_BatchGetLogsRequest($project,$logstore
        ,$shardId,$count,$cursor);
        var_dump($batchGetDataRequest);
        $response = $client->batchGetLogs($batchGetDataRequest);
        if($cursor == $response->getNextCursor())
        {
            break;
        }
        $logGroupList = $response->getLogGroupList();
        foreach($logGroupList as $logGroup)
        {
            print ($logGroup->getCategory());
            foreach($logGroup->getLogsArray() as $log)
            {
                foreach($log->getContentsArray() as $content)
                {
                    print ($content->getKey().":".$content->getValue()."\t");
                }
            }
        }
    }
}

```

```
        }
        print("\n");
    }
}
$cursor = $response -> getNextCursor();
}
}
#等待 1 分钟让日志可查询
sleep(60);
#查询日志分布情况询（注意，要查询日志，必须保证已经创建了索引，PHP SDK 不提供该接口，请在控制台创建）
$topic = "";
$query='';
$from = time()-3600;
$to = time();
$res3 = NULL;
while (is_null($res3) || (! $res3->isCompleted()))
{
    $req3 = new Aliyun_Log_Models_GetHistogramsRequest($project, $logstore, $from, $to, $topic, $query);
    $res3 = $client->getHistograms($req3);
}
var_dump($res3);
#查询日志数据
$res4 = NULL;
while (is_null($res4) || (! $res4->isCompleted()))
{
    $req4 = new Aliyun_Log_Models_GetLogsRequest($project, $logstore, $from, $to, $topic, $query, 5, 0, False);
    $res4 = $client->getLogs($req4);
}
var_dump($res4);
```

2.6. Python SDK

为方便开发人员更高效地使用日志服务，日志服务提供了 Python SDK。

下载地址

下载地址：[单击下载](#)。

该 SDK GitHub 地址如下：

<https://github.com/aliyun/aliyun-log-python-sdk>

1 创建账号

为了访问日志服务，您需要有一个控制台账号。

2 获取访问密钥

为了使用 Log Service Python SDK，您必须拥有一对访问密钥。

登录控制台，选择一对用于 SDK 的访问密钥对。

该密钥对会在下面的步骤使用，且需要保管好，不能对外泄露。另外，您可以参考[配置](#)了解更多 SDK 如何使用访问密钥的信息。

3 创建一个日志服务项目和日志库

在进行SDK操作之前，请先在控制台上创建Project和Logstore。

1. 登录日志服务控制台。
2. 单击右上角的 **创建Project**。
3. 填写 **Project名称和所属区域**，单击 **确认**。
4. 在**Project列表**页面，单击项目的名称，然后单击**创建**。
或者在创建完项目后，根据系统提示创建日志库，单击**创建**。
5. 填写日志库的配置信息并单击**确认**。

您需要在此处填写Logstore名称、数据保存时间、Shard数目等。请按照您的需求合理设置Shard数目，例如本文实例中，您需要设置4个Shard。

? 说明

- 请确保使用同一账号获取访问密钥和创建日志项目及日志库。
- Log 的 Project 名称为日志服务全局唯一，而 Logstore 名称在一个 Project 下面唯一。
- Log 的 Project 一旦创建则无法更改它的所属区域。目前也不支持在不同 Region 间迁移 Log project。

4 安装 Python 环境

Python SDK是一个纯Python的库，支持在所有运行Python的操作系统。目前主要为Linux、Mac OS X和Windows。请如下安装Python：

1. 下载并安装最新的Python [安装包](#)。

? 说明

- 目前，Python SDK支持Python 2.6/2.7和Python 3.3/3.4/3.5/3.6环境。您可以运行 `python -V` 查询当前的Python运行版本。
- Python官方不再支持Python 2.6和Python 3.3，推荐您使用Python 2.7、Python 3.4及以后版本。

2. 下载并安装Python的包管理工具 [pip](#)。

完成pip安装后，你可以运行 `pip -V` 确认pip是否安装成功，并查看当前pip版本。

5 安装Python SDK

Shell下以管理员权限执行以下命令，安装Python SDK。

```
pip install -U aliyun-log-python-sdk
```

6 开始一个 Python 程序

现在，你可以开始使用 SDK Python SDK。使用任何文本编辑器或者 Python IDE，运行如下示例代码即可与服务端交互并得到相应输出。

更多信息请参考[Github 项目地址](#)以及[更多详细说明](#)。

```
# encoding: utf-8
import time
from aliyun.log.logitem import LogItem
from aliyun.log.logclient import LogClient
from aliyun.log.getlogsrequest import GetLogsRequest
from aliyun.log.putlogsrequest import PutLogsRequest
from aliyun.log.listlogstoresrequest import ListLogstoresRequest
from aliyun.log.gethistogramsrequest import GetHistogramsRequest

def main():
    endpoint = ''          # 选择与上面步骤创建Project所属区域匹配的Endpoint
    accessKeyId = ''       # 使用您的访问密钥AccessKeyId
    accessKey = ''        # 使用您的访问密钥AccessKeySecret
    project = ''          # 上面步骤创建的项目名称
    logstore = ''         # 上面步骤创建的日志库名称
    # 重要提示：创建的logstore请配置为4个shard以便于后面测试通过
    # 构建一个client
    client = LogClient(endpoint, accessKeyId, accessKey)
    # list 所有的logstore
    req1 = ListLogstoresRequest(project)
    res1 = client.list_logstores(req1)
    res1.log_print()
    topic = ""
    source = ""
    # 发送10个数据包，每个数据包有10条log
    for i in range(10):
        logitemList = [] # LogItem list
        for j in range(10):
            contents = [('index', str(i * 10 + j))]
            logItem = LogItem()
            logItem.set_time(int(time.time()))
            logItem.set_contents(contents)
            logitemList.append(logItem)
        req2 = PutLogsRequest(project, logstore, topic, source, logitemList)
        res2 = client.put_logs(req2)
        res2.log_print()
    # list所有的shard，读取上1分钟写入的数据全部读取出来
    listShardRes = client.list_shards(project, logstore)
    for shard in listShardRes.get_shards_info():
        shard_id = shard["shardID"]
        start_time = int(time.time() - 60)
        end_time = start_time + 60
        res = client.get_cursor(project, logstore, shard_id, start_time)
        res.log_print()
        start_cursor = res.get_cursor()
        res = client.get_cursor(project, logstore, shard_id, end_time)
        end_cursor = res.get_cursor()
        while True:
            loggroup_count = 100 # 每次读取100个包
            res = client.pull_logs(project, logstore, shard_id, start_cursor, loggroup_count, end_cursor)
```

```

        res.log_print()
        next_cursor = res.get_next_cursor()
        if next_cursor == start_cursor:
            break
        start_cursor = next_cursor
# 重要提示： 只有打开索引功能，才可以使用以下接口来查询数据
time.sleep(60)
topic = ""
query = "index"
From = int(time.time()) - 600
To = int(time.time())
res3 = None
# 查询最近10分钟内，满足query条件的日志条数，如果执行结果不是完全正确，则进行重试
while (res3 is None) or (not res3.is_completed()):
    req3 = GetHistogramsRequest(project, logstore, From, To, topic, query)
    res3 = client.get_histograms(req3)
res3.log_print()
# 获取满足query的日志条数
total_log_count = res3.get_total_count()
log_line = 10
# 每次读取10条日志，将日志数据查询完，对于每一次查询，如果查询结果不是完全准确，则重试3次
for offset in range(0, total_log_count, log_line):
    res4 = None
    for retry_time in range(0, 3):
        req4 = GetLogsRequest(project, logstore, From, To, topic, query, log_line, offset, False)
        res4 = client.get_logs(req4)
        if res4 is not None and res4.is_completed():
            break
        time.sleep(1)
    if res4 is not None:
        res4.log_print()
listShardRes = client.list_shards(project, logstore)
shard = listShardRes.get_shards_info()[0]
# 分裂shard
if shard["status"] == "readwrite":
    shard_id = shard["shardID"]
    inclusiveBeginKey = shard["inclusiveBeginKey"]
    midKey = inclusiveBeginKey[:-1] + str((int(inclusiveBeginKey[-1:])) + 1)
    client.split_shard(project, logstore, shard_id, midKey)
# 合并shard
shard = listShardRes.get_shards_info()[1]
if shard["status"] == "readwrite":
    shard_id = shard["shardID"]
    client.merge_shard(project, logstore, shard_id)
# 删除shard
shard = listShardRes.get_shards_info()[-1]
if shard["status"] == "readonly":
    shard_id = shard["shardID"]
    client.delete_shard(project, logstore, shard_id)
# 创建外部数据源 def sample_external_store(client, project): res = client.create_external_store(
project, ExternalStoreConfig("rds_store", "cn-qingdao", "rds-vpc", "vpc-m5eq4irc1pucpk85frr5j", "i-m5eeo2whsnfg4kzq54ah", "47.104.78.128", "3306", "root", "sfdsfldsfksfldsfs", "meta", "join_meta")); res.log_print() res = client.update_external_store(project, ExternalStoreConfig("rd

```



```
s_store","cn-qingdao","rds-vpc","vpc-m5eq4irc1pucpk85frr5j","i-m5eeo2whsnfg4kzq54ah","47.104.78.128","3306","root","sfdsfldsfksfldsdfs","meta","join_meta")); res.log_print() res = client.get_external_store(project,"rds_store"); res.log_print() res = client.list_external_store(project,""); res.log_print(); res = client.delete_external_store(project,"rds_store") res.log_print(); # 使用python sdk进行查询分析 req4 = GetLogsRequest(project, logstore, From, To, topic, "* | select count(1)", 10,0, False) res4 = client.get_logs(req4) # 使用python sdk进行join rds查询 req4 = GetLogsRequest(project, logstore, From, To, topic, "* | select count(1) from "+logstore+" l join rds_store r on l.ikey =r.ekey", 10,0, False) res4 = client.get_logs(req4) # 使用python sdk把查询结果写入rds req4 = GetLogsRequest(project, logstore, From, To, topic, "* | insert into rds_store select count(1) ", 10,0, False) res4 = client.get_logs(req4)

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

2.7. Android SDK

日志服务 Android SDK 主要解决 Android 平台用户数据采集问题，目前提供以下功能：写入日志数据。

下载地址

<https://github.com/aliyun/aliyun-log-android-sdk>

2.8. C SDK

日志服务C SDK主要解决各种平台的日志接入问题，比如兼容MIPS芯片、OpenWrt系统等。

C SDK使用libCurl作为网络库，*apr/apr-util*库解决内存管理以及跨平台问题。您只需要基于源码进行简单编译便可以使用。

此外提供C版本的Producer Library和Producer Lite Library，为您提供跨平台、简洁、高性能、低资源消耗的一站式日志采集方案。

Git hub项目地址以及更多详细说明参见：

- [C Producer Library](#)（推荐服务端使用）
- [C Producer Lite Library](#)（推荐IOT、智能设备使用）

2.9. GO SDK

为方便开发人员更高效地使用日志服务，日志服务提供了Go SDK。

日志服务Go SDK支持以下功能：

- 批量写入、消费数据
- 通过关键词查询索引
- 操作Logtail采集配置、机器组

Git Hub地址：<https://github.com/aliyun/aliyun-log-go-sdk>

使用帮助与代码示例请参考项目README。

2.10. iOS SDK

日志服务 iOS SDK 基于 API 实现，目前提供写入日志的功能。

GitHub 地址:

- Swift: <https://github.com/aliyun/aliyun-log-ios-sdk>
- Object-C: <https://github.com/lujiating1126/AliyunLogObjc> (经过测试/生产考验第三方 SDK。可以给作者提 Issue, 响应很快)

基于Swift的iOS SDK示例

```
/*
 通过EndPoint、accessKeyID、accessKeySecret 构建日志服务客户端
@endPoint: 服务访问入口, 参见 https://help.aliyun.com/document_detail/29008.html
*/
let myClient = try! LOGClient(endPoint: "",
                              accessKeyID: "",
                              accessKeySecret: "",
                              projectName:"")

/* 创建logGroup */
let logGroup = try! LogGroup(topic: "mTopic",source: "mSource")
/* 存入一条log */
let log1 = Log()
    try! log1.PutContent("K11", value: "V11")
    try! log1.PutContent("K12", value: "V12")
    try! log1.PutContent("K13", value: "V13")
logGroup.PutLog(log1)
/* 存入一条log */
let log2 = Log()
    try! log2.PutContent("K21", value: "V21")
    try! log2.PutContent("K22", value: "V22")
    try! log2.PutContent("K23", value: "V23")
logGroup.PutLog(log2)
/* 发送 log */
myClient.PostLog(logGroup,logStoreName: ""){ response, error in
    // handle response however you want
    if error?.domain == NSErrorDomain && error?.code == NSErrorDomainTimedOut {
        print("timed out") // note, `response` is likely `nil` if it timed out
    }
}
```

Objective-C

参见 Github: [Objective-C](#)

2.11. C++ SDK

阿里云日志服务C++ SDK帮助您在C++ 程序中调用日志服务的API接口, 仅用于Linux平台。

日志服务C++ SDK支持日志服务全量API接口, 提供资源创建、数据读写等多种功能。

GitHub地址: <https://github.com/aliyun/aliyun-log-cpp-sdk>

2.12. Node.js SDK

Node.js SDK使开发人员可以非常方便地使用Node.js程序操作阿里云日志服务。本文档通过一个示例来介绍如何使用Node.js SDK将日志数据写入到日志服务。

操作步骤

1. 创建控制台账号。
2. 获取阿里云访问密钥。

该密钥对会在下面的步骤使用，且需要保管好，不能对外泄露。另外，您可以参考[配置](#)了解更多 SDK 如何使用访问密钥的信息。

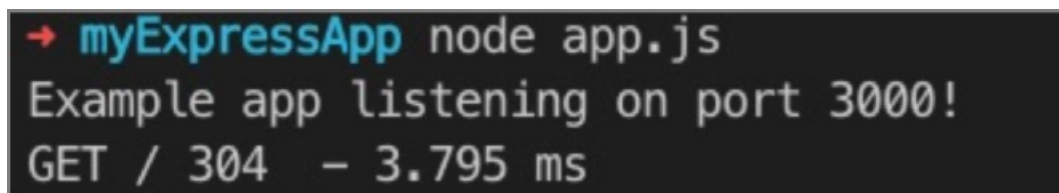
3. 搭建项目。
 - i. 使用基于Node.js的web开发框架Express搭建项目。
关于搭建项目的详细操作步骤请参见[installing](#)。
 - ii. 安装[morgan](#)。
 - iii. 在搭建好的项目中创建app.js。

app.js代码如下：

```
var express = require('express')
var morgan = require('morgan')
var app = express()
const logger = morgan(function (tokens, req, res) {
  return [
    tokens.method(req, res),
    tokens.url(req, res),
    tokens.status(req, res),
    tokens.res(req, res, 'content-length'), '-',
    tokens['response-time'](req, res), 'ms'
  ].join(' ')
})
app.use(logger)
app.get('/', (req, res) => res.send('Hello World!'))
app.listen(3000, () => console.log('Example app listening on port 3000!'))
```

- iv. 执行node app.js启动项目。

打开 `http://localhost:3000`，访问后产生访问日志。



4. 将日志写入到日志服务。
 - i. 安装aliyun-sdk。
单击[这里](#)下载aliyun-sdk。执行`npm install aliyun-sdk`命令进行安装。
 - ii. 在日志服务控制台创建Project和Logstore。
 - iii. 将Node.js SDK引用到项目。

代码如下：

```
var express = require('express')
```

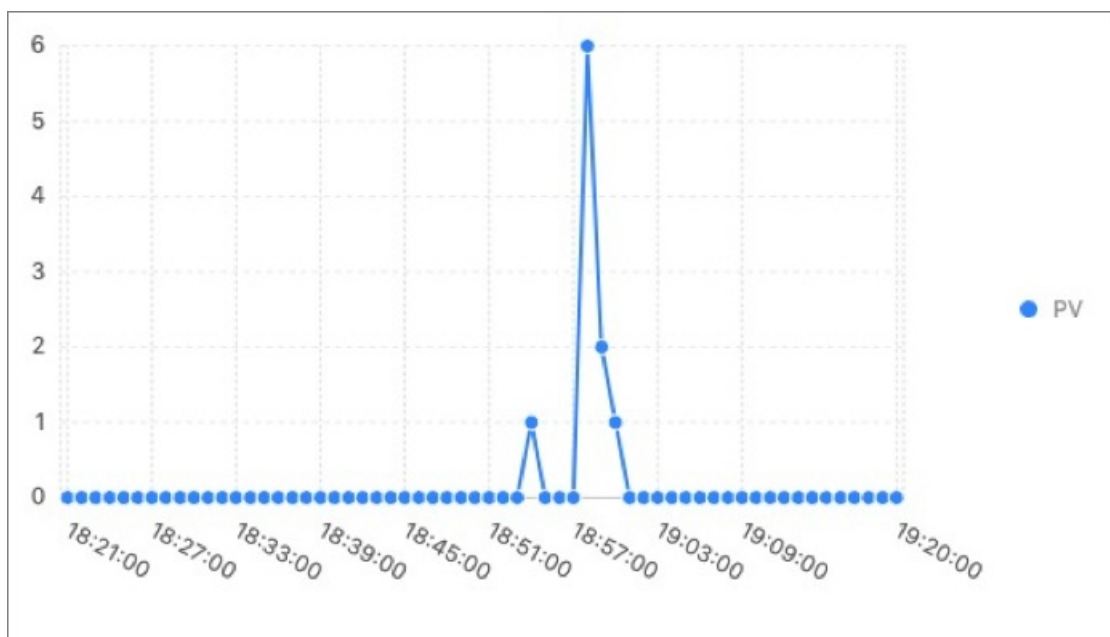
```
var morgan = require('morgan')
var ALY = require('aliyun-sdk') // 引入aliyun-sdk
var app = express()
const logger = morgan(function (tokens, req, res) {
  // 获取到日志信息
  const log = [
    tokens.method(req, res),
    tokens.url(req, res),
    tokens.status(req, res),
    tokens.res(req, res, 'content-length'), '-',
    tokens['response-time'](req, res), 'ms'
  ]
  var sls = new ALY.SLS({
    "accessKeyId": "LTAI*****G9", // 步骤2获取的密钥
    "secretAccessKey": "MMf*****o1", // 步骤2获取的密钥值
    endpoint: 'http://cn-hangzhou.sls.aliyuncs.com',
    apiVersion: '2015-06-01'
  })
  var logGroup = {
    logs: [
      {
        time: Math.floor(new Date().getTime()/1000), //单位秒
        contents: [ //key-value对
          {
            key: 'method',
            value: log[0]
          },
          {
            key: 'url',
            value: log[1]
          },
          {
            key: 'status',
            value: log[2]
          },
          {
            key: 'time',
            value: log[5]
          }
        ]
      }
    ],
    topic: 'topicName', //optional
    source: '127.0.0.1' //optional
  };
  var opt = {
    projectName: 'wangdan-test',
    logStoreName: 'test',
    logGroup
  }
  sls.putLogs(opt,function(error, data) {
    console.log(error, data)
  })
  return [
    tokens.method(req, res),
```

```
tokens.url(req, res),
tokens.status(req, res),
tokens.res(req, res, 'content-length'), '-',
tokens['response-time'](req, res), 'ms'
].join(' ')
})
app.use(logger)
app.get('/', (req, res) => res.send('Hello World!'))
app.listen(3000, () => console.log('Example app listening on port 3000!'))
```

5. 查看日志。

在日志服务控制台，找到对应Project下面的Logstore，单击Logstore名称，在查询/分析界面输入查询语句，查看日志数据。

通过对时间段的限制，可以看到在9月12日18:30到19点之间的访问量是最多的，从而可以把这段时间的访问量做成仪表盘记录下来。



3.API参考

3.1. 概览

除了通过专有云控制台进行操作外，LOG 还提供了 API（Application Programming Interface）方式写入、查询日志数据，管理自己的项目及日志库等。

对象	方法
Log（日志）	日志、日志组表示等基本概念
Project（项目）	ListProject、CreateProject、DeleteProject、GetProject、UpdateProject
	GetProjectLogs（统计Project下所有日志）
Config（配置）	ListConfig、CreateConfig、DeleteConfig、GetConfig、UpdateConfig
	GetAppliedMachineGroups（查询应用到的机器组）
MachineGroup（机器组）	ListMachineGroup、CreateMachineGroup、DeleteMachineGroup、GetMachineGroup、UpdateMachineGroup
	ApplyConfigToMachineGroup、RemoveConfigFromMachineGroup（删除配置）
	GetAppliedConfigs（查询已应用配置列表）
Logstore（日志库）	ListLogstore、CreateLogstore、DeleteLogstore、GetLogstore、UpdateLogstore
	GetLogs（查询日志）、GetHistograms（查询日志分布）
Index（索引）	GetIndex、CreateIndex、DeleteIndex、UpdateIndex
Shard（分区）	ListShards、SplitShard、MergeShards
	PostLog（写入日志）
	GetCursor（定位日志位置）
	PullLogs（消费日志）
ConsumerGroup（消费组）	CreateConsumerGroup、DeleteConsumerGroup、ListConsumerGroup、UpdateConsumerGroup
	HeartBeat（发送心跳）、GetCheckPoint、UpdateCheckPoint

通过 API 可以操作下列服务：

- 根据[Logtail 配置](#)和[Logtail 机器组](#) 信息收集日志。
- 创建[日志库](#)、向日志库写入、读取日志。

?

说明

- API 目前提供 Rest 风格。
- API 所有请求都需要做安全验证，请参见[请求签名](#) 解释了具体的 API 请求签名机制及流程。

3.2. 公共请求头

Log Service API是基于HTTP协议的Rest 风格接口。它支持一组可以在所有API请求中使用的公共请求头（除特别说明，每个 Log Service API 请求都必须提供这些公共请求头）。

Header 名称	类型	说明
Accept	字符串	客户端希望服务端返回的类型，目前支持 application/json、application/x-protobuf 两种，该字段为非必选参数，仅对 GET 请求有效。具体取值以各个接口定义为准。
Accept-Encoding	字符串	客户端希望服务端返回的压缩算法，目前支持 lz4、deflate 或空（不压缩）。该字段为非必选参数，仅对 GET 类请求有效。具体取值以各个接口定义为准。
Authorization	字符串	签名内容，更多细节请参见 请求签名 。
Content-Length	数值	RFC 2616 中定义的 HTTP 请求 Body 长度。如果请求无 Body 部分，则不需要提供该字段。
Content-MD5	字符串	请求 Body 经过 MD5 计算后的字符串，计算结果为大写。如果没有 Body 部分，则不需要提供该字段。
Content-Type	字符串	RFC 2616 中定义的 HTTP 请求 Body 类型。目前 Log Service API 请求只支持 application/x-protobuf。如果没有 Body 部分，则不需要提供该字段。具体取值以各个接口定义为准。
Date	字符串	当前发送时刻的时间，参数目前只支持 RFC 822 格式，使用 GMT 标准时间。格式化字符串如下： <code>%a, %d %b %Y %H:%M:%S GMT</code> (如： <code>Mon, 3 Jan 2010 08:33:47 GMT</code>)。
Host	字符串	HTTP 请求的完整 HOST 名字（不包括如 <code>http://</code> 这样的协议头）。例如， <code>big-game.regionid.example.com</code> 。
x-log-apiversion	字符串	API 的版本号，当前版本为 0.6.0。
x-log-bodyrawsize	数值	请求的 Body 原始大小。当无 Body 时，该字段为 0；当 Body 是压缩数据，则为压缩前的原始数据大小。该域取值范围为 0~3x1024x1024。该字段为非必选字段，只在压缩时需要。
x-log-compress-type	字符串	API 请求中 Body 部分使用的压缩方式。目前支持 lz4 压缩类型和 deflate 压缩类型，请参见 RFC 1951。如果使用 zlib 格式，请参见 RFC 1950。如果不压缩可以不提供该字段。

Header 名称	类型	说明
x-log-date	字符串	当前发送时刻的时间，格式和 Date 头一致。该请求头为可选项。如果请求中包含该公共请求头，它的值会取代 Date 标准头的值用于服务端请求验证。无论是否有 x-log-date 头，HTTP 标准 Date 头都必须提供。
x-log-signaturemethod	字符串	签名计算方式，目前仅支持 <code>hmac-sha1</code> 。

说明

- 请求中 Date 所表示的时间与服务器接收到该请求的时间最大可接受误差为 15 分钟，如果超过 15 分钟服务器端会拒绝该请求。如果请求中设置了 x-log-date 头部，则该时间误差计算基于 x-log-date 头的值。
- 如果请求指明了压缩算法（在 x-log-compress-type 中指定），则需要把原始数据压缩后放到 HTTP Body 部分，而对应的 Content-Length、Content-MD5 头部也是按照压缩后的 Body 部分计算。
- 由于某些平台上发送 HTTP 请求时无法指定 Date 头（由平台自身的库内部自动指定为发送当前时间），造成无法使用正确的 Date 值计算请求签名。在这种情况下，请指定 x-log-date 头并用该请求头的值参与请求签名计算。Log Service 服务端在接收到 API 请求后会首先判断是否有 x-log-date 头。如果有，则用它的值来做签名验证，否则就用 HTTP 的标准头 Date 做签名验证。

3.3. 公共响应头

Log Service API 是基于 HTTP 协议的 Rest 风格接口。所有的 Log Service API 响应都提供一组公共响应头。


Header 名称	类型	说明
Content-Length	数值	RFC 2616 中定义的 HTTP 响应内容长度。
Content-MD5	字符串	RFC 2616 中定义的 HTTP 响应内容的 MD5 值。Body 经过 MD5 计算后的字符串，为大写字母。
Content-Type	字符串	RFC 2616 中定义的 HTTP 响应内容类型。目前 Log Service 服务端响应类型支持 application/json, application/x-protobuf 两种类型。
Date	字符串	当前返回时刻的时间，参数目前只支持 RFC 822 格式，使用 GMT 标准时间。格式化字符串如下：%a, %d %b %Y %H:%M:%S GMT，例如：Mon, 3 Jan 2010 08:33:47 GMT。
x-log-requestid	字符串	服务端产生的标识该请求的唯一 ID。该响应头与具体应用无关，主要用于跟踪和调查问题。如果用户希望调查出现问题的 API 请求，可以向 Log Service 团队提供该 ID。

3.4. 请求签名

为保证用户日志数据的安全，Log Service API 的所有 HTTP 请求都必须经过安全验证。目前，该安全验证基于访问密钥，使用对称加密算法完成的。

其工作流程如下：

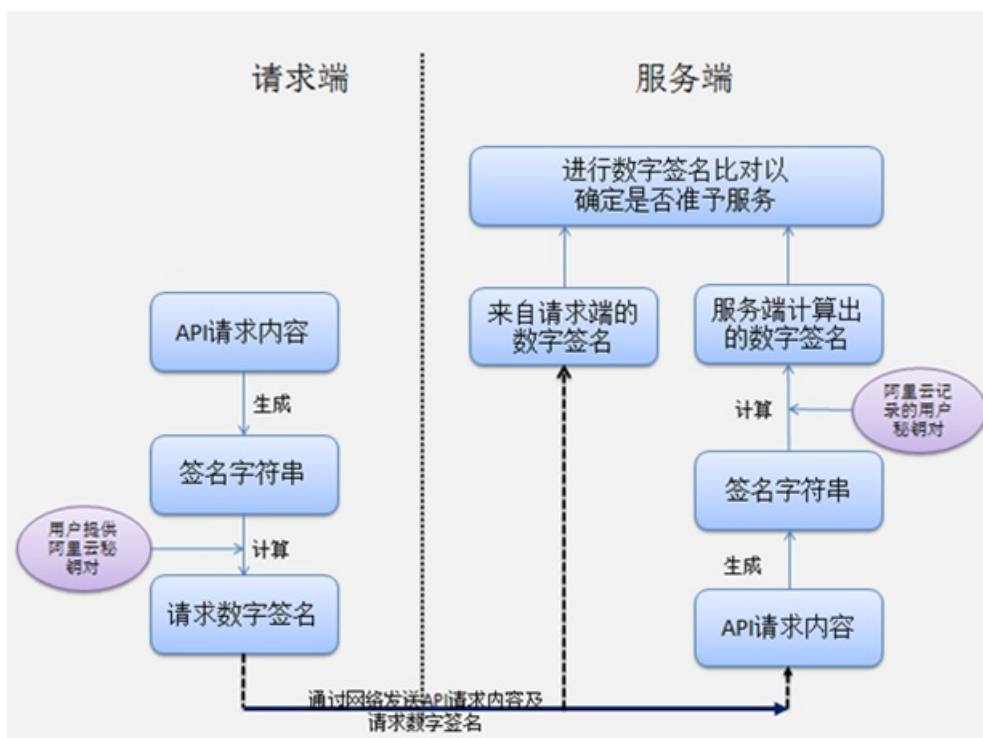
1. 请求端根据API请求内容（包括 HTTP Header 和 Body）生成签名字符串。
2. 请求端使用访问密钥对（AccessKeyId和AccessKeySecret）对第一步生成的签名字符串进行签名，形成该 API 请求的数字签名。
3. 请求端把API请求内容和数字签名一同发送给服务端。
4. 服务端在接到请求后会重复如上的第一、二步工作。

 **说明** 服务端会在后台取得该请求使用的用户访问密钥对，并在服务端计算出该请求期望的数字签名。

5. 服务端用期望的数字签名和请求端发送过来的数字签名做比对，如果完全一致则认为该请求通过安全验证。否则直接拒绝该请求。

如下图所示，上面的整个流程也可以直观描述。

安全验证流程



上面的安全验证流程可以达到如下目的：

- 确认哪位用户在做API请求。因为在发送请求前需要用户指定生成数字签名的密钥对，在服务端即可通过该密钥对确定用户身份，进而可做访问权限管理。
- 确认用户请求在网络传输过程中有无被篡改。因为服务端会对接收到的请求内容重新计算数字签名，一旦请求内容在网络上被篡改，则无法通过数字签名比对。

签名API请求

为了通过API请求的安全验证，您需要在请求端对其API请求进行签名（即生成正确的数字签名），并且使用HTTP头Authorization在网络上传输该请求的数字签名。Authorization头的具体格式如下：

```
Authorization:LOG <AccessKeyId>:<Signature>
```

如上格式所示，Authorization头的值包含用户访问密钥对中的AccessKeyId，且与之对应的AccessKeySecret将用于Signature的构造。下面将详细解释如何构造该Signature。

1. 准备合适的访问密钥

如上所述，给API请求生成签名，需使用一对访问密钥（AccessKeyId/AccessKeySecret）。

2. 生成请求的签名字符串

Log Service API的签名字符串由HTTP请求中的Method，Header和Body信息一同生成，具体方式如下：

```
SignString = VERB + "\n"
            + CONTENT-MD5 + "\n"
            + CONTENT-TYPE + "\n"
            + DATE + "\n"
            + CanonicalizedLOGHeaders + "\n"
            + CanonicalizedResource
```

上面公式中的 `\n` 表示换行转义字符，`+`（加号）表示字符串连接操作，其他各个部分定义如下表所示。

签名字符串定义

名称	定义	示例
VERB	HTTP请求的方法名称	PUT、GET、POST等
CONTENT-MD5	HTTP请求中Body部分的MD5值（必须为大写字符串）	875264590688CA6171F6228AF5BBB3D2
CONTENT-TYPE	HTTP请求中Body部分的类型	application/x-protobuf
DATE	HTTP请求中的标准时间戳头（遵循RFC 1123格式，使用GMT标准时间）	Mon, 3 Jan 2010 08:33:47 GMT
CanonicalizedLOGHeaders	由HTTP请求中以 <code>x-log</code> 和 <code>x-ac</code> 为前缀的自定义头构造的字符串（具体构造方法见下面详述）	x-log-apiversion:0.6.0\nx-log-bodyrawsize:50\nx-log-signaturemethod:hmac-sha1
CanonicalizedResource	由HTTP请求资源构造的字符串（具体构造方法见下面详述）	/logstores/app_log

对于部分无Body的HTTP请求，其CONTENT-MD5和CONTENT-TYPE两个域为空字符串，这时整个签名字符串的生成方式如下：

```
SignString = VERB + "\n"
            + "\n"
            + "\n"
            + DATE + "\n"
            + CanonicalizedLOGHeaders + "\n"
            + CanonicalizedResource
```

正如[公共请求头](#)中描述，Log Service API中引入了一个自定义请求头 `x-log-date`。如果您在请求中指定了该请求头，则其值会替代HTTP标准请求头Date加入签名计算。

CanonicalizedLOGHeaders的构造方式如下：

- i. 将所有以 `x-log` 和 `x-acs` 为前缀的HTTP请求头的名字转换成小写字母。
- ii. 将上一步得到的所有LOG自定义请求头按照字典序进行升序排序。
- iii. 删除请求头和内容之间分隔符两端出现的任何空格。
- iv. 将所有的头和内容用 `\n` 分隔符组合成最后的CanonicalizedLOGHeader。

CanonicalizedResource的构造方式如下：

- i. 将CanonicalizedResource设置为空字符串（`" "`）。
- ii. 放入要访问的LOG资源，如 `/logstores/logstorename`（无 `logstorename` 则不填）。
- iii. 如请求包含查询字符串（`QUERY_STRING`），则在CanonicalizedResource字符串尾部添加 `?` 和查询字符串。

其中，`QUERY_STRING` 是URL中请求参数按字典序排序后的字符串，其中参数名和值之间用 `=` 相隔组成字符串，并对参数名-值对按照字典序升序排序，然后以 `&` 符号连接构成字符串。其公式化描述如下：

```
QUERY_STRING = "KEY1=VALUE1" + "&" + "KEY2=VALUE2"
```

3. 生成请求的数字签名

目前，Log Service API只支持一种数字签名算法，即默认签名算法 `hmac-sha1`。其完整签名公式如下：

```
Signature = base64(hmac-sha1(UTF8-Encoding-Of(SignString),AccessKeySecret))
```

签名的方法用 [RFC 2104](#) 中定义的HMAC-SHA1方法。如上公式用的AccessKeySecret必须和最终的Authorization头中使用的AccessKeyId相对应。否则，请求将无法通过服务端验证。

在计算出数字签名后，使用该值按本节最前面描述的Authorization头格式构建完整的Log Service API请求安全验证头，并填入HTTP请求中即可发送。

请求签名过程示例

为更好地理解整个请求签名的流程，我们用两个示例来演示整个过程。首先，假设您用做Log Service API签名的访问密钥对如下：

```
AccessKeyId = "<yourAccessKeyId>"
AccessKeySecret = "<yourAccessKeySecret>"
```

- 您需要发送如下GET请求列出ali-test-project项目下的所有Logstores，其HTTP请求如下：

```
GET /logstores HTTP 1.1
Mon, 09 Nov 2015 06:11:16 GMT
Host: ali-test-project.regionid.example.com
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

如上Log Service API请求生成的签名字符串为：

```
GET\n\n\nMon, 09 Nov 2015 06:11:16 GMT\nx-log-apiversion:0.6.0\nx-log-signaturemethod:hmac-sha1\n/logstores?logstoreName=&offset=0&size=1000
```

由于是GET请求，该请求无任何HTTP Body，所以生成的签名字符串中CONTENT-TYPE与CONTENT-MD5域为空字符串。如果以前面指定的AccessKeySecret做签名运算后得到的签名为：

```
jEYOTCJs2e88o+y5F4/S5IsnBJQ=
```

最后发送经数字签名的HTTP请求内容如下：

```
GET /logstores HTTP 1.1
Mon, 09 Nov 2015 06:11:16 GMT
Host: ali-test-project.regionid.example.com
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
```

- 您需要给同上例ali-test-project项目中名为test-logstore的Logstore写入下面的日志：

```
topic=""
time=1447048976
source="10.10.10.1"
"TestKey": "TestContent"
```

为此，按照Log Service API定义需要构建如下HTTP请求：

```
POST /logstores/test-logstore HTTP/1.1
Date: Mon, 09 Nov 2015 06:03:03 GMT
Host: test-project.regionid.example.com
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
Content-MD5: 1DD45FA4A70A9300CC9FE7305AF2C494
Content-Length: 52
x-log-apiversion:0.6.0
x-log-bodyrawsize:50
x-log-compresstype:lz4
x-log-signaturemethod:hmac-sha1
<日志内容序列化成 ProtoBuffer 格式的字节流>
```

在这个HTTP请求中，写入的日志内容首先被序列化成 ProtoBuffer 格式（请参见[数据编码方式](#)了解该格式的更多细节）后作为请求Body。所以该请求的Content-Type头的值指定为application/x-protobuf。类似，Content-MD5头的值是请求body对应的MD5值。按照上面的签名字符串构造方式，这个请求对应的签名字符串为：

```
POST\n1DD45FA4A70A9300CC9FE7305AF2C494\napplication/x-protobuf\nMon, 09 Nov 2015 06:03:03
GMT\nx-log-apiversion:0.6.0\nx-log-bodyrawsize:50\nx-log-compresstype:lz4\nx-log-signatur
emethod:hmac-sha1\n/logstores/test-logstore
```

同样，以前面示例中的AccessKeySecret做签名运算，得到的最终签名为：

```
XWLGyHGg2F2hcfxWxMLiNkGki6g=
```

最后发送经数字签名的HTTP请求内容如下：

```
POST /logstores/test-logstore HTTP/1.1
Date: Mon, 09 Nov 2015 06:03:03 GMT
Host: test-project.regionid.example.com
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
Content-MD5: 1DD45FA4A70A9300CC9FE7305AF2C494
Content-Length: 52
x-log-apiversion:0.6.0
x-log-bodyrawsize:50
x-log-compresstype:lz4
x-log-signaturemethod:hmac-sha1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
<日志内容序列化后 ProtoBuffer 格式的字节流>
```

3.5. 通用错误码

本文介绍日志服务API的通用错误码。

当API请求发生错误的时候，服务端会返回错误信息，包括HTTP的Status Code和响应Body中的具体错误细节。其中响应Body中的错误细节为如下格式。

```
{
  "errorCode" : <ErrorCode>,
  "errorMessage" : <ErrorMessage>
}
```

在日志服务可能返回的错误消息中，有些消息适用于大多数API，这部分消息被定义为通用错误码。其他消息仅适用于某些API，这部分消息则被定义为独有错误码。

通用错误码

HTTP状态码 (Status Code)	错误码 (Error Code)	错误消息 (Error Message)	描述 (Description)
411	MissingContentLength	Content-Length does not exist in http header when it is necessary.	没有提供必须的Content-Length请求头。
415	InvalidContentType	Content-Type {type} is unsupported.	不支持Content-Type指定的类型。
400	MissingContentType	Content-Type does not exist in http header when body is not empty.	没有为Body不为空的HTTP请求指定Content-Type头。
400	MissingBodyRawSize	x-log-bodyrawsize does not exist in header when it is necessary.	压缩场景下没有提供必须的x-log-bodyrawsize请求头。
400	InvalidBodyRawSize	x-log-bodyrawsize is invalid.	x-log-bodyrawsize的值无效。

HTTP状态码 (Status Code)	错误码 (Error Code)	错误消息 (Error Message)	描述 (Description)
400	InvalidCompressType	x-log-compresstype {type} is unsupported.	x-log-compresstype指定的压缩方式不支持。
400	MissingHost	Host does not exist in http header.	没有提供HTTP标准请求头Host。
400	MissingDate	Date does not exist in http header.	没有提供HTTP标准请求头Date。
400	InvalidDateFormat	Date {date} must follow RFC822.	Date请求头的值不符合RFC822标准。
400	MissingAPIVersion	x-log-apiversion does not exist in http header.	没有提供HTTP请求头x-log-apiversion。
400	InvalidAPIVersion	x-log-apiversion {version} is unsupported.	HTTP请求头x-log-apiversion的值不支持。
400	MissAccessKeyId	x-log-accesskeyid does not exist in header.	没有在Authorization头部提供AccessKeyId。
401	Unauthorized	The AccessKeyId is unauthorized.	提供的AccessKeyId值未授权。
400	MissingSignatureMethod	x-log-signaturemethod does not exist in http header.	没有提供HTTP请求头x-log-signaturemethod。
400	InvalidSignatureMethod	signature method {method} is unsupported.	x-log-signaturemethod头部指定的签名方法不支持。
400	RequestTimeTooSkewed	Request time exceeds server time more than 15 minutes.	请求的发送时间超过当前服务处理时间前后15分钟的范围。
404	ProjectNotExist	Project {name} does not exist.	日志项目 (Project) 不存在。
401	SignatureNotMatch	Signature {signature} is not matched.	请求的数字签名不匹配。
403	WriteQuotaExceed	Write quota is exceeded.	超过写入日志限额。
403	ReadQuotaExceed	Read quota is exceeded.	超过读取日志限额。
500	InternalServerError	Internal server error message.	服务器内部错误。
503	ServerBusy	The server is busy, please try again later.	服务器正忙，请稍后再试。

错误消息中包括 {...} 部分为出错相关的具体信息。例如，ProjectNotExist 的错误消息中包括{name}，表示错误消息中该部分会被具体的Project Name来替换。

3.6. 日志项目接口

3.6.1. CreateProject

调用CreateProject创建Project。

请求语法

```
POST / HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Content-MD5: <Content-MD5>
Content-Length: <ContentLength>
Connection: Keep-Alive
{
  "projectName": <ProjectName>,
  "description": <Description>
}
```

请求元素

- 请求头
CreateProject接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

属性名称	类型	是否必须	描述
projectName	string	是	Project 名称。
description	string	是	Project 描述。

响应元素

- 响应头
CreateProject接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应元素
HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
400	ProjectAlreadyExist	Project {project} already exist
400	ParameterInvalid	The body is not valid json string
500	InternalServerError	Specified Server Error Message

示例

● 请求示例

```
POST / HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project-test.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 27 May 2018 07:43:26 GMT
Content-Type: application/json
Content-MD5: A7967D81EFF5E3CD447FB6D8DF294E20
Content-Length: 80
Connection: Keep-Alive
{
  "projectName": "my-project-test",
  "description": "Description of my-project-test"
}
```

● 响应示例

```
HTTP/1.1 200
Server: nginx
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 27 May 2018 07:43:27 GMT
x-log-requestid: 5B0A619F205DC3F30EDA9322
```

3.6.2. DeleteProject

调用DeleteProject删除一个指定的Project。

请求语法


```
DELETE / HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

请求元素

- 请求头

DeleteProject接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
projectName	string	是	Project的名称，作为Header中Host的一部分。

响应元素

- 响应头

DeleteProject接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应元素

HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	The Project does not exist : {Project}
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
DELETE / HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project-test.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 27 May 2018 08:25:04 GMT
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

- 响应示例

```
HTTP/1.1 200
Server: nginx
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 27 May 2018 08:25:04 GMT
x-log-requestid: 5B0A6B60BB6EE39764D458B5
```

3.6.3. UpdateProject

调用UpdateProject修改指定的Project。

请求语法

```
PUT / HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Content-MD5: <Content-MD5>
Content-Length: <ContentLength>
Connection: Keep-Alive
```

请求元素

- 请求头

UpdateProject接口无特有请求头，关于Log Service API的公共请求头，请参考[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
projectName	String	是	Project名称，作为Header中Host的一部分。

属性名称	类型	是否必须	描述
description	String	否	Project描述，默认为空字符串。

响应元素

- 响应头
UpdateProject接口无特有响应头，关于Log Service API的公共响应头，请参考[公共响应头](#)。
- 响应元素
HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	The Project does not exist : <Project>
400	ParameterInvalid	The body is not valid json string
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
PUT / HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project-test.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 27 May 2018 07:43:26 GMT
Content-Type: application/json
Content-MD5: A7967D81EFF5E3CD447FB6D8DF294E20
Content-Length: 40
Connection: Keep-Alive
{
  "description": "Description of my-project-test"
}
```

- 响应示例

```
HTTP/1.1 200
Server: nginx
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 27 May 2018 07:43:27 GMT
x-log-requestid: 5B0A619F205DC3F30EDA9322
```

3.6.4. GetProject

调用GetProject根据Project名称查询Project。

请求语法

```
GET / HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

请求元素

- 请求头
GetProject接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

属性名称	类型	是否必须	描述
projectName	string	是	Project 的名称，作为Header中Host的一部分。

响应元素

- 响应头
GetProject接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应参数

GetProject请求成功，其响应Body中包含Project的详细信息，具体格式如下所示。

属性名称	类型	描述
createTime	string	创建时间。
description	string	Project描述

属性名称	类型	描述
lastModifyTime	string	最后一次更新时间。
owner	string	创建人的账户ID。
projectName	string	Project名称。
status	string	Project状态。
region	string	Project所属的区域。

错误码

除了返回Log Service API的通用错误码，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	The Project does not exist : {Project}
500	InternalServerError	Specified Server Error Message

示例

● 请求示例

```
GET / HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project-test.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 27 May 2018 08:25:04 GMT
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

● 响应示例

```
HTTP/1.1 200
Server: nginx
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 27 May 2018 08:25:04 GMT
x-log-requestid: 5B0A6B60BB6EE39764D458B5
```

3.6.5. ListProject

调用ListProject查询所有的Project列表。

请求语法

```
GET /?offset={offset}&size={size} HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

请求元素

- 请求头
ListProject接口无特有请求头，关于Log Service API的公共请求头，请参考[公共请求头](#)。
- 请求参数

属性名称	类型	是否必须	描述
offset	integer	否	返回记录的起始位置，默认值：0。
size	integer	否	每页返回最大条目，默认值：500（最大值）。

响应元素

- 响应头
ListProject接口无特有响应头，关于Log Service API的公共响应头，请参考[公共响应头](#)。
- 响应参数
ListProject请求成功，其响应Body中包含Project表，具体格式如下所示。

属性名称	类型	描述
count	integer	返回的Project个数。
total	integer	Project总数。
projects	array	Project列表。

projects中每个元素的格式如下所示。

属性名称	类型	描述
createTime	string	创建时间。
description	string	Project描述
lastModifyTime	string	最后一次更新时间。

属性名称	类型	描述
owner	string	创建人的账户ID。
projectName	string	Project名称。
status	string	Project状态。
region	string	Project所属的区域。

错误码

除了返回Log Service API的通用错误码，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
500	InternalServerError	Specified Server Error Message

示例

● 请求示例

```
GET /?offset=0&size=2&projectName= HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 27 May 2018 09:03:33 GMT
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

● 响应示例

```
HTTP/1.1 200
Server: nginx
Content-Type: application/json
Content-Length: 345
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 27 May 2018 09:03:33 GMT
x-log-requestid: 5B0A7465AAEA20CA70DE3064
{
  "count": 2,
  "total": 11,
  "projects": [
    {
      "projectName": "project1",
      "status": "Normal",
      "owner": "",
      "description": "",
      "region": "cn-shanghai",
      "createTime": "1524222931",
      "lastModifyTime": "1524539357"
    },
    {
      "projectName": "project123456",
      "status": "Normal",
      "owner": "",
      "description": "",
      "region": "cn-shanghai",
      "createTime": "1471963876",
      "lastModifyTime": "1524539357"
    }
  ]
}
```

3.6.6. GetProjectLogs

GetProjectLogs是Project级别的SQL查询接口。

请求语法

GetProjectLogs接口请求语法如下所示。

```
GET /logs query=SELECT avg(latency) as avg_latency FROM  where __date__ >'2017-09-01 00:00:00' and __date__ < '2017-09-02 00:00:00'
Authorization: <AuthorizationString>
Date: Wed, 3 Sept. 2014 08:33:46 GMT
Host: big-game.cn-hangzhou.log.aliyuncs.com
x-log-bodyrawsize: 0
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

GetProjectLogs接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
query	string	是	查询SQL条件。

响应元素

- 响应头

关于Log Service API的公共响应头，请参见[公共响应头](#)。该API特有的响应头如下所示。

名称	类型	描述
x-log-progress	字符串	查询结果的状态。可以有Incomplete 和 Complete两个选值，表示本次是否完整。
x-log-count	整型	当前查询结果的日志总数。
x-log-processed-rows	整型	本次查询处理的行数。
x-log-elapsed-millisecond	整型	本次查询消耗的毫秒时间。

- 响应参数

GetProjectLogs的响应body是一个数组，数组中每个元素是一条日志结果。

名称	类型	描述
__time__	整型	日志的时间戳，精度为秒，从1970-1-1 00:00:00 UTC开始计算的秒数。
__source__	字符串	日志的来源，写入日志时指定。
[content]	Key-Value对	日志原始内容。

细节描述

- 该接口的query是一个标准的SQL查询语句。
- 查询的Project在请求的域名中指定。
- 查询的Logstore在查询语句的from条件中指定。可以将Logstore看做是SQL中的表。
- 在查询的SQL条件中必须指定要查询的时间范围，时间范围由__date__（timestamp类型）或__time__（int类型，单位是unix_time）来指定。
- 当查询涉及的日志数量变化非常大时，日志服务API无法预测需要调用多少次该接口来获取完整结果。所以需要您查看每次请求返回结果中的x-log-progress成员状态值，根据成员状态值来确定是否需要重复调用该接口来获取最终完整结果。

 **说明** 每次重复调用该接口都会重新消耗相同数量的查询CU。

错误码

GetProjectLogs接口除了返回Log Service API的通用错误码，还可能返回如下特有错误码。

HTTP状态码 (Status Code)	错误码 (Error Code)	错误消息 (Error Message)	描述 (Description)
400	ParameterInvalid	parameter is invalid	请求的参数错误，详情请参见具体的错误message。

示例

以杭州地域内名为big-game的Project为例，查询该Project内名为app_log的Logstore中，主题为groupA的日志数据。查询区间为2014-09-01 00:00:00到2014-09-01 22:00:00，查询关键字为error，且从时间区间头开始查询，最多返回20条日志数据。

● 请求示例

```
GET /logs/?query=SELECT * FROM <logStoreName> where __line__ = 'abc' and __date__ >'2017-09-01 00:00:00' and __date__ < '2017-09-02 00:00:00'&line=20&offset=0 HTTP/1.1
Authorization: <AuthorizationString>
Date: Wed, 3 Sept. 2014 08:33:46 GMT
Host: big-game.cn-hangzhou.log.aliyuncs.com
x-log-bodyrawsize: 0
x-log-apiversion: 0.4.0
x-log-signaturemethod: hmac-sha1
```

● 响应示例

```
HTTP/1.1 200 OK
Content-MD5: 36F9F7F0339BEAF571581AF1B0AAAFB5
Content-Type: application/json
Content-Length: 269
Date: Wed, 3 Sept. 2014 08:33:47 GMT
x-log-requestid: efag01234-12341-15432f
x-log-progress : Complete
x-log-count : 10000
x-log-processed-rows: 10000
x-log-elapsed-millisecond:5
{
  "progress": "Complete",
  "count": 2,
  "logs": [
    {
      "__time__": 1409529660,
      "__source__": "10.237.0.17",
      "Key1": "error",
      "Key2": "Value2"
    },
    {
      "__time__": 1409529680,
      "__source__": "10.237.0.18",
      "Key3": "error",
      "Key4": "Value4"
    }
  ]
}
```

响应示例中，x-log-progress的值为Complete，表明整个日志查询已经完成，返回结果为完整结果。本次请求共查询到2条符合条件的日志数据。如果响应结果中的x-log-progress的值为Incomplete，则需要重复请求以获得完整结果。

3.7. 日志库相关接口

3.7.1. CreateLogstore

CreateLogstore接口用于创建Logstore。

请求语法

CreateLogstore接口请求语法如下。

```
POST /logstores HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
{
  "logstoreName" : <logStoreName>,
  "ttl": <ttl>,
  "shardCount": <shardCount>,
  "autoSplit": <autoSplit>,
  "maxSplitShard": <maxSplitShard>
}
```

请求元素

- 请求头

CreateLogstore接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	Logstore的名称，在Project下必须唯一。
ttl	integer	是	数据的保存时间，单位为天，范围1~3600。
shardCount	integer	是	Shard个数，单位为个，范围为1~100。
enable_tracking	bool	否	是否开启WebTracking，默认为false。
autoSplit	bool	否	是否自动分裂shard，默认为false。
maxSplitShard	int	否	自动分裂时shard最大个数，范围为1~64，默认为64。当autoSplit为true时必须设置。

响应元素

- 响应头

CreateLogstore接口无特有响应头，关于Log Service API的公共响应头，请参见[公共请求头](#)。

- 响应元素

HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
---------	-----------	--------------

HTTP状态码	ErrorCode	ErrorMessage
400	LogstoreAlreadyExist	logstore {logstoreName} already exists
500	InternalServerError	Specified Server Error Message
400	LogstoreInfoInvalid	logstore info is invalid
400	ProjectQuotaExceed	Project Quota Exceed

示例

● 请求示例

```
POST /logstores HTTP/1.1
Header :
{
  x-log-apiversion=0.6.0,
  Authorization=LOG <yourAccessKeyId>:<yourSignature>,
  Host=ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com,
  Date=Wed, 11 Nov 2015 07:35:00 GMT,
  Content-Length=55,
  x-log-signaturemethod=hmac-sha1,
  Content-MD5=7EF43D0B8F4A807B95E775048C911C72,
  User-Agent=sls-java-sdk-v-0.6.0,
  Content-Type=application/json
}
Body :
{
  "logstoreName": "test-logstore",
  "ttl": 1,
  "shardCount": 2,
  "autoSplit": true,
  "maxSplitShard": 64
}
```

● 响应示例

```
HTTP/1.1 200 OK
Header:
{
  Date=Wed, 11 Nov 2015 07:35:00 GMT,
  Content-Length=0,
  x-log-requestid=5642EFA499248C827B012B39,
  Connection=close,
  Server=nginx/1.6.1
}
```

3.7.2. DeleteLogstore

调用DeleteLogstore删除Logstore，包括所有的Shard数据、索引等。

请求语法

```
DELETE /logstores/{logstoreName} HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求参数

- 请求头

DeleteLogstore接口无特有请求头，关于Log Service API的公共请求头，请参考 [公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	日志库名称，同一Project下唯一。

响应参数

- 响应头

DeleteLogstore接口无特有响应头，关于Log Service API的公共响应头，请参考 [公共响应头](#)。

- 响应元素

HTTP状态码返回200。

错误码

除了返回Log Service API的 [通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	LogStoreNotExist	logstore {logstoreName} does not exist
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
DELETE /logstores/test_logstore HTTP/1.1
Header :
{
  x-log-apiversion=0.6.0,
  Authorization=LOG <yourAccessKeyId>:<yourSignature>,
  Host=ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com,
  Date=Wed, 11 Nov 2015 08:09:38 GMT,
  Content-Length=0,
  x-log-signaturemethod=hmac-sha1,
  User-Agent=sls-java-sdk-v-0.6.0,
  Content-Type=application/json
}
```

- 响应示例

```
HTTP/1.1 200 OK
Header:
{
  Date=Wed, 11 Nov 2015 08:09:39 GMT,
  Content-Length=0,
  x-log-requestid=5642F7C399248C817B013A07,
  Connection=close,
  Server=nginx/1.6.1
}
```

3.7.3. UpdateLogstore

更新Logstore的属性。目前只支持更新TTL和Shard属性。

请求语法

```
PUT /logstores/{logstoreName} HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
{
  "logstoreName": <logstoreName>,
  "ttl": <ttl>,
  "shardCount": <shardCount>,
  "autoSplit": <autoSplit>,
  "maxSplitShard": <maxSplitShard>
}
```

请求元素

- 请求头

UpdateLogstore接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	日志库名称，同一Project下唯一。
ttl	integer	是	日志数据生命周期（TTL），单位为天，范围1~3600。
shardCount	integer	是	查看Shard的个数。更新Logstore的属性后，Shard的个数不变。Shard个数的修改只能通过SplitShard或MergeShards完成。
enable_tracking	bool	否	是否开启WebTracking。
autoSplit	bool	否	是否自动分裂shard。
maxSplitShard	int	否	自动分裂时最大的shard个数，范围为1~64。当autoSplit为true时必须指定。

响应元素

- 响应头

UpdateLogstore接口无特有响应头。关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应元素

HTTP 状态码返回 200。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	Project {ProjectName} does not exist
404	LogStoreNotExist	logstore {logstoreName} does not exist
400	LogStoreAlreadyExist	logstore {logstoreName} already exists
500	InternalServerError	Specified Server Error Message
400	ParameterInvalid	invalid shard count,you can only modify shardCount by split& merge shard

细节描述

目前，Shard的数量只能通过SplitShard或MergeShards操作进行调整。

示例

● 请求示例

```
PUT /logstores/test-logstore HTTP/1.1
Header:
{
  x-log-apiversion=0.6.0,
  Authorization=LOG <yourAccessKeyId>:<yourSignature>,
  Host=ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com,
  Date=Wed, 11 Nov 2015 08:28:19 GMT,
  Content-Length=55,
  x-log-signaturemethod=hmac-sha1,
  Content-MD5=757C60FC41CC7D3F60B88E0D916D051E,
  User-Agent=sls-java-sdk-v-0.6.0,
  Content-Type=application/json
}
Body :
{
  "logstoreName": "test-logstore",
  "ttl": 1,
  "shardCount": 2,
  "autoSplit": true,
  "maxSplitShard": 64
}
```

● 响应示例

```
HTTP/1.1 200 OK
Header:
{
  Date=Wed, 11 Nov 2015 08:28:20 GMT,
  Content-Length=0,
  x-log-requestid=5642FC2399248C8F7B0145FD,
  Connection=close,
  Server=nginx/1.6.1
}
```

3.7.4. GetLogstore

调用GetLogstore查看Logstore属性。

请求语法

```
GET /logstores/{logstoreName} HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头
GetLogstore接口无特有请求头。关于Log Service API的公共请求头，请参见 [公共请求头](#)。
- 请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	日志库名称，同一 Project 下唯一。

响应元素

- 响应头
GetLogstore接口无特有响应头。关于Log Service API的公共响应头，请参考 [公共响应头](#)。
- 响应元素
HTTP状态码返回200。

```
{
  "logstoreName" : <logstoreName>,
  "ttl": <ttl>,
  "shardCount": <shardCount>,
  "createTime": <createTime>,
  "lastModifyTime": <lastModifyTime>
}
```

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	Project {ProjectName} does not exist
404	LogstoreNotExist	logstore {logstoreName} does not exist
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
GET /logstores/test-logstore HTTP/1.1
Header :
{
  x-log-apiversion=0.6.0,
  Authorization=LOG <yourAccessKeyId>:<yourSignature>,
  Host=ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com,
  Date=Wed, 11 Nov 2015 07:53:29 GMT,
  Content-Length=0,
  x-log-signaturemethod=hmac-sha1,
  User-Agent=sls-java-sdk-v-0.6.0,
  Content-Type=application/json
}
```

- 响应示例

```
HTTP/1.1 200 OK
Header :
{
  Date=Wed, 11 Nov 2015 07:53:30 GMT,
  Content-Length=107,
  x-log-requestid=5642F3FA99248C817B01352D,
  Connection=close,
  Content-Type=application/json,
  Server=nginx/1.6.1
}
Body :
{
  "logstoreName": test-logstore,
  "ttl": 1,
  "shardCount": 2,
  "createTime": 1447833064,
  "lastModifyTime": 1447833064
}
```

3.7.5. ListLogstore

调用ListLogstores接口列出指定Project下的所有Logstore。

请求语法

```
GET /logstores HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

无特有请求头。关于 Log Service API 的公共请求头，请参考 [公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
offset	int	否	返回记录的起始位置，从0开始。
size	int	否	每页返回最大条目，默认500（最大值）。
logstoreName	string	是	用于请求的Logstore名称（支持部分匹配）。

响应元素

- 响应头
无特有响应头。关于Log Service API的公共响应头，请参考[公共响应头](#)。

- 响应元素
List LogStores请求成功，其响应Body会包括指定Project下的所有Logstore名称列表。

名称	类型	描述
count	整型	返回的Logstore数目。
total	整型	Logstore总数。
logstores	字符串数组	返回的 Logstore名称列表。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	Project {ProjectName} does not exist
500	InternalServerError	Specified Server Error Message
400	ParameterInvalid	Invalid parameter size, (0.6.0]
400	InvalidLogStoreQuery	logstore Query is invalid

示例

- 请求示例

```
GET /logstores HTTP/1.1
Header:
{
  x-log-apiversion=0.6.0,
  Authorization=LOG <yourAccessKeyId>:<yourSignature>,
  Host=ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com,
  Date=Wed, 11 Nov 2015 08:09:39 GMT,
  Content-Length=0,
  x-log-signaturemethod=hmac-sha1,
  User-Agent=sls-java-sdk-v-0.6.0,
  Content-Type=application/json
}
```

- 响应示例

```
HTTP/1.1 200 OK
Header:
{
  Date=Wed, 11 Nov 2015 08:09:39 GMT,
  Content-Length=52,
  x-log-requestid=5642F7C399248C8D7B01342F,
  Connection=close,
  Content-Type=application/json,
  Server=nginx/1.6.1
}
Body:
{
  "count":1,
  "logstores":["test-logstore"],
  "total":1
}
```

3.7.6. ListShards

列出Logstore下当前所有可用的Shard。

请求语法

```
GET /logstores/<logstorename>/shards HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

ListShards接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	日志库名称

响应元素


- 响应头
List Shards接口无特有响应头。关于Log Service API的公共响应头，请参考[公共响应头](#)。
- 响应元素
shard元素组成的数组。

```
[
  {
    "shardID": 0,
    "status": "readwrite",
    "inclusiveBeginKey": "00000000000000000000000000000000",
    "exclusiveEndKey": "80000000000000000000000000000000",
    "createTime": 1453949705
  },
  {
    ...
  },
  {
    ...
  }
]
```

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	LogStoreNotExist	logstore {logstoreName} does not exist
500	InternalServerError	Specified Server Error Message
400	LogStoreWithoutShard	logstore has no shard

 **说明** 上表错误消息中 {name} 表示该部分会被具体的LogstoreName来替换。

示例

- 请求示例

```
GET /logstores/sls-test-logstore/shards
Header :
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Thu, 12 Nov 2015 03:40:31 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

● 响应示例

```
Header:
{
  "content-length": "57",
  "server": "nginx/1.6.1",
  "connection": "close",
  "date": "Thu, 12 Nov 2015 03:40:31 GMT",
  "content-type": "application/json",
  "x-log-requestid": "56440A2F99248C050600C74C"
}
Body :
[
  {
    "shardID": 1,
    "status": "readwrite",
    "inclusiveBeginKey": "00000000000000000000000000000000",
    "exclusiveEndKey": "80000000000000000000000000000000",
    "createTime": 1453949705
  },
  {
    "shardID": 2,
    "status": "readwrite",
    "inclusiveBeginKey": "80000000000000000000000000000000",
    "exclusiveEndKey": "ffffffffffffffffffffffffffffffff",
    "createTime": 1453949705
  },
  {
    "shardID": 0,
    "status": "readonly",
    "inclusiveBeginKey": "00000000000000000000000000000000",
    "exclusiveEndKey": "ffffffffffffffffffffffffffffffff",
    "createTime": 1453949705
  }
]
```

3.7.7. SplitShard

分裂一个指定readwrite状态的Shard。

请求语法

```
POST /logstores/<logstorename>/shards/<shardid>?action=split&key=<splitkey> HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

SplitShard接口无特有请求头。关于 Log Service API 的公共请求头，请参考 [公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	日志库名称
shardid	int	是	Shard ID
splitkey	string	是	split切分位置

响应元素

- 响应头

Content-Type: application/json

SplitShard接口无特有响应头。关于Log Service API的公共响应头，请参考 [公共响应头](#)。

- 响应参数


3个Shard元素组成的数组，第一个Shard为分裂之前的Shard，后两个为分裂的结果。


```
[
  {
    "shardID": 33,
    "status": "readonly",
    "inclusiveBeginKey": "ee000000000000000000000000000000",
    "exclusiveEndKey": "ffffffffffffffffffffffffffffffff",
    "createTime": 1453949705
  },
  {
    "shardID": 163,
    "status": "readwrite",
    "inclusiveBeginKey": "ee000000000000000000000000000000",
    "exclusiveEndKey": "ef000000000000000000000000000000",
    "createTime": 1453949705
  },
  {
    "shardID": 164,
    "status": "readwrite",
    "inclusiveBeginKey": "ef000000000000000000000000000000",
    "exclusiveEndKey": "ffffffffffffffffffffffffffffffff",
    "createTime": 1453949705
  }
]
```

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
404	LogStoreNotExist	logstore {logstoreName} does not exist
400	ParameterInvalid	invalid shard id
400	ParameterInvalid	invalid mid hash
500	InternalServerError	Specified Server Error Message
400	LogStoreWithoutShard	logstore has no shard

 **说明** 上表错误消息中 {name} 表示该部分会被具体的LogstoreName来替换。

示例

- 请求示例

```
POST /logstores/logstorename/shards/33?action=split&key=ef000000000000000000000000000000
Header :
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou.sls.aliyuncs.com",
  "Date": "Thu, 12 Nov 2015 03:40:31 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

● 响应示例

```
Header:
{
  "content-length": "57",
  "server": "nginx/1.6.1",
  "connection": "close",
  "date": "Thu, 12 Nov 2015 03:40:31 GMT",
  "content-type": "application/json",
  "x-log-requestid": "56440A2F99248C050600C74C"
}
Body :
[
  {
    "shardID": 33,
    "status": "readonly",
    "inclusiveBeginKey": "ee000000000000000000000000000000",
    "exclusiveEndKey": "ffffffffffffffffffffffffffffffff",
    "createTime": 1453949705
  },
  {
    "shardID": 163,
    "status": "readwrite",
    "inclusiveBeginKey": "ee000000000000000000000000000000",
    "exclusiveEndKey": "ef000000000000000000000000000000",
    "createTime": 1453949705
  },
  {
    "shardID": 164,
    "status": "readwrite",
    "inclusiveBeginKey": "ef000000000000000000000000000000",
    "exclusiveEndKey": "ffffffffffffffffffffffffffffffff",
    "createTime": 1453949705
  }
]
```

3.7.8. MergeShards

合并两个相邻的状态为readwrite的Shards。在参数中指定一个ShardId，服务端自动查找相邻的下一个Shard。

请求语法

```
POST /logstores/<logstorename>/shards/<shardid>?action=merge HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

MergeShards接口无特有请求头。关于Log Service API的公共请求头，请参考[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	日志库名称
shardid	int	是	shard ID

响应元素

- 响应头

Content-Type: application/json

MergeShards接口无特有响应头。关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应元素


3个Shard元素组成的数组，第一个shard为合并之后的Shard，后两个为合并之前的Shard。

```
[
  {
    'shardID': 167,
    'status': 'readwrite',
    'inclusiveBeginKey': 'e0000000000000000000000000000000',
    'createTime': 1453953105,
    'exclusiveEndKey': 'ffffffffffffffffffffffffffffffff'
  },
  {
    'shardID': 30,
    'status': 'readonly',
    'inclusiveBeginKey': 'e0000000000000000000000000000000',
    'createTime': 0,
    'exclusiveEndKey': 'e7000000000000000000000000000000'
  },
  {
    'shardID': 166,
    'status': 'readonly',
    'inclusiveBeginKey': 'e7000000000000000000000000000000',
    'createTime': 1453953073,
    'exclusiveEndKey': 'ffffffffffffffffffffffffffffffff'
  }
]
```

错误码

除了返回Log Service API的 通用错误码，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	LogStoreNotExist	logstore {logstoreName} does not exist
400	ParameterInvalid	invalid shard id
400	ParameterInvalid	can not merge the last shard
500	InternalServerError	Specified Server Error Message
400	LogStoreWithoutShard	logstore has no shard

 说明 上表错误消息中 {name} 表示该部分会被具体的LogstoreName来替换。

示例

- 请求示例

```
POST /logstores/logstorename/shards/30?action=merge
Header :
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou.sls.aliyuncs.com",
  "Date": "Thu, 12 Nov 2015 03:40:31 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

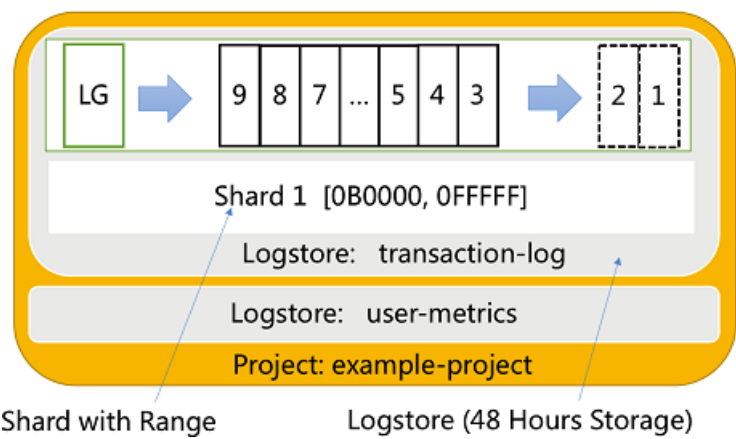
● 响应示例

```
Header:
{
  "content-length": "57",
  "server": "nginx/1.6.1",
  "connection": "close",
  "date": "Thu, 12 Nov 2015 03:40:31 GMT",
  "content-type": "application/json",
  "x-log-requestid": "56440A2F99248C050600C74C"
}
Body :
[
  {
    'shardID': 167,
    'status': 'readwrite',
    'inclusiveBeginKey': 'e0000000000000000000000000000000',
    'createTime': 1453953105,
    'exclusiveEndKey': 'ffffffffffffffffffffffffffffffff'
  },
  {
    'shardID': 30,
    'status': 'readonly',
    'inclusiveBeginKey': 'e0000000000000000000000000000000',
    'createTime': 0,
    'exclusiveEndKey': 'e7000000000000000000000000000000'
  },
  {
    'shardID': 166,
    'status': 'readonly',
    'inclusiveBeginKey': 'e7000000000000000000000000000000',
    'createTime': 1453953073,
    'exclusiveEndKey': 'ffffffffffffffffffffffffffffffff'
  }
]
```

3.7.9. GetCursor

GetCursor接口用于根据时间获取对应的游标（cursor）。

通过游标（cursor）可以获取特定日志对应的位置。关于cursor与Project、Logstore、Shard的关系如下图所示。



- Project下有多个Logstore。
- 每个Logstore会有多个Shard。
- 通过cursor可以获得特定日志对应的位置。

请求语法

Get Cursor接口的请求语法如下所示。

```
GET /logstores/ay42/shards/2?type=cursor&from=1402341900 HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
```

请求元素


- 请求头
Get Cursor接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

参数名称	类型	是否必须	描述
shard	string	是	Shard名称。
type	string	是	此处为cursor。
from	string	是	时间点（Unix时间戳）或者字符串 <code>begin</code> 、 <code>end</code> 。

通过from可以在Shard中定位生命周期内的日志，假设Logstore的生命周期为 `[begin_time,end_time)` ， `from=from_time`，那么：

- 当 `from_time ≤ begin_time` or `from_time = "begin"` 时：返回时间点为begin_time对应的cursor位置。

- 当 `from_time ≥ end_time` or `from_time = "end"` 时：返回当前时间点下一条将被写入的cursor位置（当前该 cursor 位置上无数据）。
- 当 `from_time > begin_time` and `from_time < end_time` 时：返回第一个服务端接收时间大于等于 `from_time` 的数据包对应的 cursor。

 **说明** Logstore生命周期由属性中TTL字段指定。例如，当前时间为2018-11-11 09:00:00，TTL=5。则每个Shard中可以消费的数据时间段为 [2018-11-05 09:00:00, 2018-11-11 09:00:00)，这里的时间指的是Server端时间。

响应元素

- 响应头

Get Cursor接口无特有响应头。关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应元素

```
{
  "cursor": "MTQ0NzI5OTYwNjg5NjYzMjM1Ng=="
}
```

错误码

Get Cursor接口除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	LogStoreNotExist	Logstore {Name} does not exist
400	ParameterInvalid	Parameter From is not valid
400	ShardNotExist	Shard {ShardID} does not exist
500	InternalServerError	Specified Server Error Message
400	LogStoreWithoutShard	the logstore has no shard

示例

- 请求示例

```
GET /logstores/sls-test-logstore/shards/0?type=cursor&from=begin
Header:
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Thu, 12 Nov 2015 03:56:57 GMT",
  "x-log-apiversion": "0.6.0",
  "Content-Type": "application/json",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

- 响应示例

```
Header:
{
  "content-length": "41",
  "server": "nginx/1.6.1",
  "connection": "close",
  "date": "Thu, 12 Nov 2015 03:56:57 GMT",
  "content-type": "application/json",
  "x-log-requestid": "56440E0999248C070600C6AA"
}
Body:
{
  "cursor": "MTQ0NzI5OTYwNjg5NjYzMjM1Ng=="
}
```

3.7.10. GetCursorTime

GetCursorTime API根据Cursor获取服务端时间。

请求语法

```
GET /logstores/{logstoreName}/shards/{shard}?cursor={cursor}&type=cursor_time HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求参数

- 请求头

GetCursorTime接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	Logstore名称
shard	int	是	shard ID
cursor	string	是	获取时间戳的Cursor。
type	string	是	固定为cursor_time

响应参数

- 响应头
GetCursorTime接口无特有响应头。关于Log Service API的公共响应头，请参见公共响应头。
- 响应元素

```
{
  "cursor_time": 1554260243
}
```

错误码

除了返回Log Service API的通用错误码，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	LogStoreNotExist	Logstore {Name} does not exist
400	ParameterInvalid	Parameter From is not valid
400	ShardNotExist	Shard {ShardID} does not exist
500	InternalServerError	Specified Server Error Message
400	LogStoreWithoutShard	the logstore has no shard

示例

- 请求示例

```
GET /logstores/internal-operation_log/shards/0?cursor=MTU0NzQ3MDY4MjM3NjUxMzQ0Ng%3D%3D&type=cursor_time HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-hangzhou.sls.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Wed, 03 Apr 2019 02:57:23 GMT
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

- 响应示例

```
HTTP/1.1 200
Server: nginx
Content-Type: application/json
Content-Length: 26
Connection: close
Access-Control-Allow-Origin: *
Date: Wed, 03 Apr 2019 02:57:23 GMT
x-log-requestid: 5CA42113D2F00378A74B9051
{
  "cursor_time": 1554260243
}
```

3.7.11. PullLogs

根据游标、数量获取日志，获得日志时必须指定shard。

目前仅支持读取PB格式数据。

在storm等情况下可以通过LoghubClientLib进行选举与协同消费。

请求语法

```
GET /logstores/ay42/shards/0?type=logs&cursor=MTQ0NzMyOTQwMTEwMjEzMDkwNA==&count=100 HTTP/1.1
Accept: application/x-protobuf
Accept-Encoding: lz4
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求参数

- 请求头
 - Accept: application/x-protobuf
 - Accept-Encoding: lz4或deflate或 ""
 - Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

参数名称	类型	是否必须	描述
type	string	是	此处为logs。
cursor	string	是	游标，用以表示从什么位置开始读取数据，相当于起点。
count	int	是	返回的loggroup数目，范围为 1~1000。

响应参数

- 响应头
 - x-log-cursor: 当前读取数据下一条 cursor
 - x-log-count: 当前返回数量
 - Log Service API的公共响应头, 请参见[公共响应头](#)。
- 响应元素

protobuf 格式序列化后的数据 (可能经过压缩)。

错误码

除了返回Log Service API的[通用错误码](#), 还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	LogStoreNotExist	Logstore {Name} does not exist
400	ParameterInvalid	Parameter Cursor is not valid
400	ParameterInvalid	ParameterCount should be in [0-1000]
400	ShardNotExist	Shard {ShardID} does not exist
400	InvalidCursor	this cursor is invalid
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

读取 0 号 shard 上的数据

```
GET /logstores/sls-test-logstore/shards/0?cursor=MTQ0NzMyOTQwMTEwMjEzMDkwNA==&count=1000&type=log
Header:
{
  "Authorization"="LOG <yourAccessKeyId>:<yourSignature>",
  "x-log-bodyrawsize"=0,
  "User-Agent" : "sls-java-sdk-v-0.6.0",
  "x-log-apiversion" : "0.6.0",
  "Host" : "ali-test-project.cn-hangzhou-failover-intranet.sls.aliyuncs.com",
  "x-log-signaturemethod" : "hmac-sha1",
  "Accept-Encoding" : "lz4",
  "Content-Length": 0,
  "Date" : "Thu, 12 Nov 2015 12:03:17 GMT",
  "Content-Type" : "application/x-protobuf",
  "accept" : "application/x-protobuf"
}
```

- 响应示例

```
Header:
{
  "x-log-count" : "1000",
  "x-log-requestid" : "56447FB20351626D7C000874",
  "Server" : "nginx/1.6.1",
  "x-log-bodyrawsize" : "34121",
  "Connection" : "close",
  "Content-Length" : "4231",
  "x-log-cursor" : "MTQ0NzMyOTQwMTEwMjEzMDkwNA==",
  "Date" : "Thu, 12 Nov 2015 12:01:54 GMT",
  "x-log-compresstype" : "lz4",
  "Content-Type" : "application/x-protobuf"
}
Body:
<protobuf 格式 loggroup list 内容> 压缩后结果
```

- 翻页

如果只为了翻页（拿到下一组 Token），不返回数据，可以通过HTTP HEAD方式进行请求。

3.7.12. PostLog

PutLogs接口用于向指定的Logstore中写入日志数据。

目前仅支持写入PB格式的日志数据。

 说明 日志数据以LogGroup的形式写入。

写入时有两种模式：

- 负载均衡模式（LoadBalance）：自动根据Logstore下所有可写的Shard进行负载均衡写入。该方法写入可用性较高（SLA：99.95%），适合写入消费数据与Shard无关的场景，例如不保序。
- Key路由Shard 模式（KeyHash）：在URL参数中增加key字段，用来判断数据写入哪个Shard的range中。该参数为可选参数，不设置时自动切换负载均衡写入模式。例如：可以将某个生产者（例如 instance）根据名称Hash到固定Shard上，这样就能保证写入与消费在该Shard上的数据是严格有序的（在Merge/Split过程中能够严格保证对于Key在一个时间点只会出现在一个Shard上，请参见分区）。

PB数据

PB格式日志压缩数据字段描述如下所示，详情请参见数据模型和数据编码方式。

- Log

字段名称	类型	是否必须	描述
Time	Int	是	日志时间。
Contents	List	是，至少包含一个元素	日志字段列表，每个元素类型请参见Content字段表格。

- Content

字段名称	类型	是否必须	描述
Key	String	是	自定义key名称。
Value	String	是	自定义key对应的值。

- LogTag

字段名称	类型	是否必须	描述
Key	String	是	自定义key名称。
Value	String	是	自定义key对应的值。

- LogGroup

字段名称	类型	是否必须	描述
Logs	List	是	日志列表，每个元素请参见Log字段表格。
Topic	String	否	用户自定义字段，用以标记一批日志。
Source	String	否	日志的来源。例如产生该日志机器的IP地址。
LogTags	List	是	日志的标签列表，每个元素请参见LogTag。

请求语法

- 负载均衡模式

```
POST /logstores/<logstorename>/shards/lb HTTP/1.1
Authorization: <AuthorizationString>
Content-Type: application/x-protobuf
Content-Length: <Content Length>
Content-MD5: <Content MD5>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-bodyrawsize: <BodyRawSize>
x-log-compresstype: lz4
x-log-signaturemethod: hmac-sha1
<PB 格式日志压缩数据>
```

- Key路由Shard模式

```
POST /logstores/<logstorename>/shards/route?key=14d2f850ad6ea48e46e4547edbbb27e0
Authorization: <AuthorizationString>
Content-Type: application/x-protobuf
Content-Length: <Content Length>
Content-MD5: <Content MD5>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-bodyrawsize: <BodyRawSize>
x-log-compresstype: lz4
x-log-signaturemethod: hmac-sha1
<PB 格式日志压缩数据>
```

请求元素

- 请求头

关于Log Service API 的公共请求头，请参见[公共请求头](#)。

- 请求参数

名称	类型	必选	描述
logstorename	string	是	需要写入日志的Logstore名称。

响应元素

- 响应头

PutLogs接口无特有响应头。关于Log Service API 的公共响应头，请参见[公共响应头](#)。

- 响应元素

成功后无任何响应元素。

细节描述

- PutLogs接口每次可以写入的日志组数据量上限为3MB或者4096条，日志组中每条日志下的Value部分不超过1MB大小。只要日志数据量超过这两条上限中的任意一条则整个请求失败，且无任何日志数据成功写入。
- 服务端会对每次PutLogs写入的日志数据做格式检查，只要日志数据中有任何一条日志不符合规范，则整个请求失败且无任何日志数据成功写入。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码 (Status Code)	错误码 (Error Code)	错误消息 (Error Message)	描述 (Description)
400	PostBodyInvalid	Protobuffer content cannot be parsed.	Protobuffer内容不能够解析。
400	InvalidTimestamp	Invalid timestamps are in logs.	日志内容中有无效的日志时间戳。

HTTP 状态码 (Status Code)	错误码 (Error Code)	错误消息 (Error Message)	描述 (Description)
400	InvalidEncoding	Non-UTF8 characters are in logs.	日志内容中有非UTF8字符。
400	InvalidKey	Invalid keys are in logs.	日志内容中有无效的key。
400	PostBodyTooLarge	Logs must be less than or equal to 3 MB and 4096 entries.	日志内容包含的日志必须小于3MB和4096条。
400	PostBodyUncompressError	Failed to decompress logs.	日志内容解压失败。
499	PostBodyInvalid	The post data time is out of range	日志中时间范围不在 [-7*24Hour, +15Min] 有效范围内。
404	LogStoreNotExist	logstore {Name} does not exist.	日志库 (Logstore) 不存在。

 **说明** 上表错误消息中 {name} 表示该部分会被具体的LogstoreName来替换。

示例

● 请求示例

```
POST /logstores/sls-test-logstore
{
  "Content-Length": 118,
  "Content-Type": "application/x-protobuf",
  "x-log-bodyrawsize": 1356,
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Content-MD5": "6554BD042149C844761C2C094A8FECCE",
  "Date": "Thu, 12 Nov 2015 06:54:26 GMT",
  "x-log-apiversion": "0.6.0",
  "x-log-compresstype": "lz4",
  "x-log-signaturemethod": "hmac-sha1",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
<PB 格式日志使用 Lz4 压缩后的二进制数据>
```

● 响应示例

```
Header
{
  "date": "Thu, 12 Nov 2015 06:53:03 GMT",
  "connection": "close",
  "x-log-requestid": "5644160399248C060600D216",
  "content-length": "0",
  "server": "nginx/1.6.1"
}
```

3.7.13. GetShipperStatus

查询日志投递任务状态。

请求语法

```
GET /logstores/{logstoreName}/shipper/{shipperName}/tasks?from=1448748198&to=1448948198&status=succcess&offset=0&size=100 HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求参数

请求参数如所示。

GetShipperStatus请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	日志库名称，同一 Project 下唯一
shipperName	string	是	日志投递规则名称，同一 Logstore 下唯一
from	integer	是	日志投递任务创建时间区间
to	integer	是	日志投递任务创建时间区间
status	string	否	默认为空，表示返回所有状态的任务，目前支持 successful/running/failed 等状态
offset	integer	否	返回指定时间区间内投递任务的起始数目，默认值为 0
size	integer	否	返回指定时间区间内投递任务的数目，默认值为 100，最大为 500

请求头

GetShipperStatus 接口无特有请求头。关于 Log Service API 的公共请求头，请参考 [公共请求头](#)。

响应头

GetShipperStatus 接口无特有响应头。关于 Log Service API 的公共响应头，请参考 [公共响应头](#)。

响应元素

请求成功，其响应 Body 会包括指定的日志投递任务列表：

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

GetShipperStatus响应元素如所示。

GetShipperStatus响应元素

名称	类型	描述
count	integer	返回的任务个数。
total	integer	指定范围内任务总数。
statistics	json	指定范围内任务汇总状态，具体请参考下表。
tasks	array	指定范围内投递任务具体详情，具体请参考下表。

statistics内容请参考。

statistics参数详情

名称	类型	描述
running	integer	指定范围内状态为 running 的任务个数。
successful	integer	指定范围内状态为 success 的任务个数。
failed	integer	指定范围内状态为 fail 的任务个数。

tasks内容请参考。

tasks参数详情

名称	类型	描述
id	string	具体投递任务的任务唯一 ID。
taskStatus	string	投递任务的具体状态，可能为 running/success/fail 中的一种。
taskMessage	string	投递任务失败时的具体错误信息。
taskCreateTime	integer	投递任务创建时间。
taskLastDataReceiveTime	integer	投递任务中的最新日志数据时间。
taskFinishTime	integer	投递任务结束时间。

错误码

如所示，除了返回 Log Service API 的 [通用错误码](#)，还可能返回如下特有错误码：

GetShipperStatus特有错误码

HTTP 状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	Project {ProjectName} does not exist
404	LogStoreNotExist	logstore {logstoreName} does not exist
400	ShipperNotExist	shipper {logstoreName} does not exist
500	InternalServerError	internal server error
400	ParameterInvalid	start time must be earlier than end time
400	ParameterInvalid	only support query last 48 hours task status

HTTP 状态码	ErrorCode	ErrorMessage
400	ParameterInvalid	status only contains success/running/fail

细节描述

投递任务状态查询时间区间只能指定为最近 24 小时之内。

示例

请求示例：

```
GET /logstores/test-logstore/shipper/test-shipper/tasks?from=1448748198&to=1448948198&status=success&offset=0&size=100 HTTP/1.1
Header:
{
  x-log-apiversion=0.6.0,
  Authorization=LOG <yourAccessKeyId>:<yourSignature>,
  Host=ali-test-project.regionid.example.com,
  Date=Wed, 11 Nov 2015 08:28:19 GMT,
  Content-Length=55,
  x-log-signaturemethod=hmac-sha1,
  Content-MD5=757C60FC41CC7D3F60B88E0D916D051E,
  User-Agent=sls-java-sdk-v-0.6.0,
  Content-Type=application/json
}
```

响应示例：

```
HTTP/1.1 200 OK
Header:
{
  Date=Wed, 11 Nov 2015 08:28:20 GMT,
  Content-Length=0,
  x-log-requestid=5642FC2399248C8F7B0145FD,
  Connection=close,
  Server=nginx/1.6.1
}
Body:
{
  "count" : 10,
  "total" : 20,
  "statistics" : {
    "running" : 0,
    "success" : 20,
    "fail" : 0
  }
  "tasks" : [
    {
      "id" : "abcdefghijk",
      "taskStatus" : "success",
      "taskMessage" : "",
      "taskCreateTime" : 1448925013,
      "taskLastDataReceiveTime" : 1448915013,
      "taskFinishTime" : 1448926013
    }
  ]
}
```

3.7.14. RetryShipperTask

重新执行失败的日志投递任务。

请求语法

```
PUT /logstores/{logstoreName}/shipper/{shipperName}/tasks HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
["task-id-1", "task-id-2", "task-id-2"]
```

请求参数

请求参数如所示。

RetryShipperTask请求参数

参数名称	类型	是否必须	描述
logstoreName	string	是	日志库名称，同一 Project 下唯一
shipperName	string	是	日志投递规则名称，同一 Logstore 下唯一

请求头

RetryShipperTask 接口无特有请求头。关于 Log Service API 的公共请求头，请参考 [公共请求头](#)。

响应头

RetryShipperTask 接口无特有响应头。关于 Log Service API 的公共响应头，请参考 [公共响应头](#)。

响应元素

HTTP 状态码返回 200。

错误码

如所示，除了返回 Log Service API 的 [通用错误码](#)，还可能返回如下特有错误码：

RetryShipperTask特有错误码

HTTP 状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	Project {ProjectName} does not exist
404	LogStoreNotExist	logstore {logstoreName} does not exist
400	ShipperNotExist	shipper {logstoreName} does not exist
500	InternalServerError	Specified Server Error Message
400	ParameterInvalid	only allow retry 10 tasks every time

细节描述

每次最多只能重新执行 10 个失败的投递任务。

示例

请求示例：

```
PUT /logstores/test-logstore/shipper/test-shipper/tasks HTTP/1.1
Header:
{
  x-log-apiversion=0.6.0,
  Authorization=LOG <yourAccessKeyId>:<yourSignature>,
  Host=ali-test-project.regionid.example.com,
  Date=Wed, 11 Nov 2015 08:28:19 GMT,
  Content-Length=55,
  x-log-signaturemethod=hmac-sha1,
  Content-MD5=757C60FC41CC7D3F60B88E0D916D051E,
  User-Agent=sls-java-sdk-v-0.6.0,
  Content-Type=application/json
}
Body :
["task-id-1", "task-id-2", "task-id-2"]
```

响应示例：

```
HTTP/1.1 200 OK
Header:
{
  Date=Wed, 11 Nov 2015 08:28:20 GMT,
  Content-Length=0,
  x-log-requestid=5642FC2399248C8F7B0145FD,
  Connection=close,
  Server=nginx/1.6.1
}
```

3.7.15. GetLogs

GetLogs接口查询指定Project下某个Logstore中的日志数据。还可以通过指定相关参数仅查询符合指定条件的日志数据。

当日志写入到Logstore中，日志服务的查询接口（GetHistograms 和 GetLogs）能够查到该日志的延时因写入日志类型不同而异。日志服务按日志时间戳把日志分为如下两类：

- 实时数据：日志中时间点为服务器当前时间点（-180秒，900秒]。例如，日志时间为UTC 2014-09-25 12:03:00，服务器收到时为UTC 2014-09-25 12:05:00，则该日志被作为实时数据处理，一般出现在正常场景下。
- 历史数据：日志中时间点为服务器当前时间点 [-7 x 86400秒， -180秒)。例如，日志时间为UTC 2014-09-25 12:00:00，服务器收到时为UTC 2014-09-25 12:05:00，则该日志被作为历史数据处理，一般出现在补数据场景下。

其中，实时数据写入至可查询的最大延时为3秒（99.9%情况下1秒内即可查询）。

请求语法

GetLogs 接口的请求语法如下所示。

```
GET /logstores/<logstorename>?type=log&topic=<logtopic>&from=<starttime>&to=<endtime>&query
=<querystring>&line=<linenum>&offset=<startindex>&reverse=<ture|false> HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-bodyrawsize: 0
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头
GetLogs接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

名称	类型	必选	描述
logstorename	string	是	需要查询日志的Logstore名称。
type	string	是	查询Logstore数据的类型。在GetLogs接口中该参数必须为log。
from	int	是	查询开始时间点（精度为秒，从 1970-1-1 00:00:00 UTC 计算起的秒数）。
to	int	是	查询结束时间点（精度为秒，从 1970-1-1 00:00:00 UTC 计算起的秒数）。
topic	string	否	查询日志主题。
query	string	否	查询表达式。关于查询表达式的详细语法，请参见《日志服务用户指南》中的 查询与分析 > 《 查询语法与功能 》 > 《 查询语法 》章节。
line	int	否	请求返回的最大日志条数。取值范围为 0~100，默认值为 100。
offset	int	否	请求返回日志的起始点。取值范围为0或正整数，默认值为0。
reverse	bool	否	是否按日志时间戳逆序返回日志，精确到分钟级别。true表示逆序，false表示顺序，默认值为false。

响应元素

- 响应头
GetLogs接口无特有响应头。关于Log Service API的公共响应头，请参见[公共响应头](#)。
响应头中有专门成员表示请求返回结果是否完整。具体响应元素格式如下所示。

名称	类型	描述
x-log-progress	string	查询结果的状态。可以有Incomplete和Complete两个选值，表示本次是否完整。
x-log-count	int	当前查询结果的日志总数。

● 响应元素

GetLogs请求成功，其响应Body会包括查询命中的日志数据。当需要查询的日志数据量非常大（T级别）的时候，该接口的响应结果可能并不完整，GetLogs的响应body是一个数组，数组中每个元素是一条日志结果。数组中的每个元素结构如下。

名称	类型	描述
__time__	int	日志的时间戳（精度为秒，从 1970-1-1 00:00:00 UTC 计算起的秒数）。
__source__	string	日志的来源，写入日志时指定。
[content]	Key-Value对	日志原始内容。

细节描述

- 该接口中由请求参数from和to定义的时间区间遵循“左闭右开”原则，即该时间区间包括区间开始时间点，但不包括区间结束时间点。如果from和to的值相同，则为无效区间，函数直接返回错误。
- 当查询涉及的日志数量变化非常大时，日志服务API无法预测需要调用多少次该接口来获取完整结果。所以需要您查看每次请求返回结果中的x-log-progress成员状态值，根据成员状态值来确定是否需要重复调用该接口来获取最终完整结果。


 说明 每次重复调用该接口都会重新消耗相同数量的查询CU。

错误码

除了返回Log Service API的 [通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码（Status Code）	错误码（Error Code）	错误消息（Error Message）	描述（Description）
404	LogStoreNotExist	logstore {Name} does not exist	日志库（logstore）不存在。
400	InvalidTimeRange	request time range is invalid	请求的时间区间无效。
400	InvalidQueryString	query string is invalid	请求的查询字符串无效。
400	InvalidOffset	offset is invalid	请求的offset参数无效。
400	InvalidLine	line is invalid	请求的line参数无效。
400	InvalidReverse	Reverse value is invalid	Reverse参数的值无效。

HTTP状态码（Status Code）	错误码（Error Code）	错误消息（Error Message）	描述（Description）
400	IndexConfigNotExist	logstore without index config	日志库（logstore）未开启索引。

 **说明** 上表错误消息中 {name} 表示该部分会被具体的LogstoreName来替换。

示例

以杭州地域内名为big-game的Project为例，查询该project内名为app_log的Logstore中，主题为groupA的日志数据。查询区间为 2014-09-01 00:00:00到2014-09-01 22:00:00，查询关键字为error，且从时间区间头开始查询，最多返回20条日志数据。

● 请求示例：

```
GET /logstores/app_log? type=log&topic=groupA&from=1409529600&to=1409608800&query=error&line=20&offset=0 HTTP/1.1
Authorization: <AuthorizationString>
Date: Wed, 3 Sept. 2014 08:33:46 GMT
Host: big-game.cn-hangzhou.log.aliyuncs.com
x-log-bodyrawsize: 0
x-log-apiversion: 0.4.0
x-log-signaturemethod: hmac-sha1
```

● 响应示例：

```
HTTP/1.1 200 OK
Content-MD5: 36F9F7F0339BEAF571581AF1B0AAAFB5
Content-Type: application/json
Content-Length: 269
Date: Wed, 3 Sept. 2014 08:33:47 GMT
x-log-requestid: efag01234-12341-15432f
x-log-progress : Complete
x-log-count : 10000
x-log-processed-rows: 10000
x-log-elapsed-millisecond:5
{
  "progress": "Complete",
  "count": 2,
  "logs": [
    {
      "__time__": 1409529660,
      "__source__": "10.237.0.17",
      "Key1": "error",
      "Key2": "Value2"
    },
    {
      "__time__": 1409529680,
      "__source__": "10.237.0.18",
      "Key3": "error",
      "Key4": "Value4"
    }
  ]
}
```

在这个响应示例中，x-log-progress成员的状态为Complete，表明整个日志查询已经完成，返回结果为完整结果。在这次请求中共查询到2条符合条件的日志，且日志数据在logs成员中。如果响应结果中的x-log-progress成员的状态为Incomplete，则需要重复相同请求以获得完整结果。

3.7.16. GetHistograms

GetHistograms接口查询指定的Project下某个Logstore中日志的分布情况。还可以通过指定相关参数仅查询符合指定条件的日志分布情况。

从日志写入日志库到查询接口（GetHistograms 和 GetLogs）查到该日志，延时时长因写入日志类型不同而异。日志服务按日志时间戳把日志分为如下两类：

- 实时数据：日志中时间点为服务器当前时间点（-180秒，900秒）。例如，日志时间为UTC 2014-09-25 12:03:00，服务器收到时为UTC 2014-09-25 12:05:00，则该日志被视作实时数据处理。实时数据从写入到在日志查询界面查询到该数据的延迟为3秒。
- 历史数据：日志中时间点为服务器当前时间点 [-7 x 86400秒，-180秒)。例如，日志时间为UTC 2014-09-25 12:00:00，服务器收到时为UTC 2014-09-25 12:05:00，则该日志被作为历史数据处理，一般出现在补数据场景下。

请求语法

GetHistograms接口请求语法如下。

```
GET /logstores/<logstorename>?type=histogram&topic=<logtopic>&from=<starttime>&to=<endtime>
&query=<querystring> HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-bodyrawsize: 0
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头
GetHistograms接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

名称	类型	必选	描述
logstorename	string	是	需要查询日志的Logstore名称。
type	string	是	查询Logstore数据的类型，在GetHistograms 接口中该参数必须为 histogram。
from	int	是	查询开始时间点。精度为秒，从1970-1-1 00:00:00 UTC计算起的秒数。
to	int	是	查询结束时间点。精度为秒，从1970-1-1 00:00:00 UTC计算起的秒数。
topic	string	否	查询日志主题。
query	string	否	查询表达式。关于查询表达式的详细语法，请参见 用户指南 中的 查询与分析 > 查询语法与功能 > 查询语法 。

响应元素

- 响应头
GetHistograms接口无特有响应头。关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应参数
GetHistograms请求成功，其响应Body会包括查询命中的日志数量在时间轴上的分布情况。响应结果会把您查询的时间范围均匀分割成若干个子时间区间，并返回每个子时间区间内命中的日志条数。具体响应元素格式如下：

名称	类型	描述
progress	string	查询结果的状态。可以有Incomplete和Complete两个选值，表示结果是否完整。
count	int	当前查询结果中所有日志总数。

名称	类型	描述
histograms	array	当前查询结果在划分的子时间区间上的分布状况，具体结构见下面的描述。

上表中的histograms数组中的每个元素结构如下。

名称	类型	描述
from	int	子时间区间的开始时间点。精度为秒，从1970-1-1 00:00:00 UTC计算起的秒数。
to	int	子时间区间的结束时间点。精度为秒，从1970-1-1 00:00:00 UTC计算起的秒数。
count	int	当前查询结果在该子时间区间内命中的日志条数。
progress	string	当前查询结果在该子时间区间内的结果是否完整，可以有Incomplete和Complete两个选值。

细节描述


- 该接口中涉及的时间区间（无论是请求参数from和to定义的时间区间，还是返回结果中各个子时间区间）都遵循“左闭右开”原则，即该时间区间包括区间开始时间点，但不包括区间结束时间点。如果from和to的值相同，则为无效区间，函数直接返回错误。
- 该接口的响应中子区间划分方式是一致稳定的。如果您在请求查询的时间区间不变，则响应中子区间划分结果也不会改变。
- 当查询涉及的日志数量变化非常大时，日志服务API无法预测需要调用多少次该接口来获取完整结果。所以需要您查看每次请求返回结果中的progress成员状态值，根据成员状态值来确定是否需要重复调用该接口来获取最终完整结果。

 **说明** 每次重复调用该接口都会重新消耗相同数量的查询CU。

错误码

除了返回 Log Service API 的 [通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码 (Status Code)	错误码 (Error Code)	错误消息 (Error Message)	描述 (Description)
404	LogStoreNotExist	logstore {Name} does not exist	日志库 (logstore) 不存在。
400	InvalidTimeRange	request time range is invalid	请求的时间区间无效。
400	InvalidQueryString	query string is invalid	请求的查询字符串无效。

 **说明** 上表错误消息中 {name} 表示该部分会被具体的LogstoreName替换。

示例

以杭州地域内名为big-game的project为例，查询该project内名为app_log的Logstore中，主题为groupA的日志分布情况。查询区间为2014-09-01 00:00:00到2014-09-01 22:00:00，查询关键字是error。

- 请求示例

```
GET /logstores/app_log?type=histogram&topic=groupA&from=1409529600&to=1409608800&query=error HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
Date: Wed, 3 Sept. 2014 08:33:46 GMT
Host: big-game.cn-hangzhou.log.aliyuncs.com
x-log-bodyrawsize: 0
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

- 响应示例

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-MD5: E6AD9C21204868C2DE84EE3808AAA8C8
Content-Type: application/json
Date: Wed, 3 Sept. 2014 08:33:47 GMT
Content-Length: 232
x-log-requestid: efag01234-12341-15432f
[
  {
    "from": 1409529600,
    "to": 1409569200,
    "count": 2,
    "progress": "Complete"
  },
  {
    "from": 1409569200,
    "to": 1409608800,
    "count": 1,
    "progress": "Incomplete"
  }
]
```

在这个响应示例中，服务端把整个 Histogram 划分到两个均等的时间区间，分别为[2014-09-01 00:00:00, 2014-09-01 11:00:00) 和[2014-09-01 11:00:00, 2014-09-01 22:00:00)。由于您的数据量过大，第一次查询会返回不完整数据。响应结果告知您命中日志条数为3，但整体结果不完整，仅在时间段[2014-09-01 00:00:00, 2014-09-01 11:00:00) 结果完整且命中2条，而在另外一个时间段结果不完整但已经命中1条。在这个情况下，如果您希望得到完整结果，则需要重复调用上面的请求示例若干次直到整个响应中的progress成员值变成Complete。响应示例如下。

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-MD5: E6AD9C21204868C2DE84EE3808AAA8C8
Content-Type: application/json
Date: Wed, 3 Sept. 2014 08:33:48 GMT
Content-Length: 232
x-log-requestid: afag01322-1e241-25432e
[
  {
    "from": 1409529600,
    "to": 1409569200,
    "count": 2,
    "progress": "Complete"
  },
  {
    "from": 1409569200,
    "to": 1409608800,
    "count": 2,
    "progress": "complete"
  }
]
```

3.7.17. UpdateIndex

更新指定Logstore的索引。

请求语法

```
PUT /logstores/<logstoreName>/index HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Content-MD5: <Content-MD5>
Content-Length: <ContentLength>
{
  "line": <full text index>,
  "keys": <key-value index>,
  "ttl": <ttl>
}
```

请求元素

- 请求头

UpdateIndex接口无特有请求头，关于Log Service API的公共请求头请参见[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	Logstore名称。
ttl	integer	否	索引文件生命周期，支持7天，30天和90天。
keys	object	否	键值索引配置，key为字段名称，value为字段索引配置。
line	object	否	全文索引配置。

属性keys和line必须至少指定一个。全文索引配置包含如下属性。

属性名称	类型	是否必须	描述
token	array	是	分词符列表。
caseSensitive	bool	否	是否大小写敏感。
chn	bool	否	是否包含中文。
include_keys	array	否	包含的字段列表，不能与exclude_keys同时指定。
exclude_keys	array	否	排除的字段列表，不能与include_keys同时指定。

字段索引配置包含如下属性。

属性名称	类型	是否必须	描述
type	string	是	字段类型。
alias	string	否	别名。
chn	bool	否	是否包含中文，只有type为text时才有意义。
token	array	否，当type为text时必须	分词符列表，只有type为text时才有意义。
caseSensitive	bool	否	是否大小写敏感，只有type为text时才有意义。
doc_value	bool	否	是否开启统计。

响应元素

- 响应头

UpdateIndex 接口无特有响应头，关于Log Service API的公共响应头请参见[公共响应头](#)。

- 响应元素

HTTP 状态码返回200。

错误码

除了返回 Log Service API 的[通用错误码](#)，还可能返回如下特有错误码：

HTTP状态码	ErrorCode	ErrorMessage
400	ParameterInvalid	log store index is not created
400	ParameterInvalid	key config must has type
400	IndexInfoInvalid	required field token is lacking or of error format
400	IndexInfoInvalid	required fields line/keys are lacking or of error format
404	ProjectNotExist	The Project does not exist : {Project}
404	LogStoreNotExist	logstore {logstoreName} dose not exist
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
PUT /logstores/logstore-4/index HTTP/1.1
Header:
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Mon, 07 May 2018 09:47:20 GMT
Content-Type: application/json
Content-MD5: 1860B805A6AA5288B97B32CF3B519112
Content-Length: 316
Connection: Keep-Alive
Body:
{
  "line": {
    "token": [
      ",",
      " ",
      "'",
      "\"",
      "\\",
      ";",
      "=",
      "(",
      ")"
    ],
```



```
    "[",
    "]",
    "{",
    "}",
    "?",
    "@",
    "&",
    "<",
    ">",
    "/",
    ":",
    "\\n",
    "\\t",
    "\\r"
  ]
},
"keys": {
  "agent": {
    "doc_value": true,
    "caseSensitive": true,
    "alias": "agent_alias",
    "type": "text",
    "token": [
      ",",
      " ",
      "'",
      "\"",
      ";",
      "=",
      "(",
      ")",
      "[",
      "]",
      "{",
      "}",
      "?",
      "@",
      "&",
      "<",
      ">",
      "/",
      ":",
      "\\n",
      "\\t",
      "\\r"
    ]
  }
},
"ttl": 90
}
```

- 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Mon, 07 May 2018 09:47:21 GMT
x-log-requestid: 5AF020A98CBAA2EF52AB66C9
```

3.7.18. CreateIndex

为指定的Logstore创建索引。

请求语法

```
POST /logstores/<logstoreName>/index HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: <x-log-bodyrawsize>
User-Agent: <User-Agent>
x-log-apiversion: <APIVersion>
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Content-MD5: <Content-MD5>
Content-Length: <Content-Length>
{
  "line": <full text index>,
  "keys": <key-value index>
}
```

请求元素

- 请求头
CreateIndex接口无特有请求头，关于Log Service API的公共请求头，请参考[公共请求头](#)。
- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	Logstore名称。
keys	object	否	字段索引配置，key为字段名称，value为字段索引配置。
line	object	否	全文索引配置。

属性keys和line必须至少指定一个。全文索引配置包含如下属性。

属性名称	类型	是否必须	描述
caseSensitive	bool	否	是否大小写敏感。

属性名称	类型	是否必须	描述
chn	bool	否	是否包含中文。
token	array	是	分词符列表。
include_keys	array	否	包含的字段列表，不能与exclude_keys同时指定。
exclude_keys	array	否	排除的字段列表，不能与include_keys同时指定。

字段索引配置包含如下属性。

属性名称	类型	是否必须	描述
type	string	是	字段类型。
alias	string	否	字段别名。
chn	bool	否	是否包含中文，只有type为text时才有意义。
token	array	仅当type为text时必须	分词符列表，只有type为text时才有意义。
caseSensitive	bool	否	是否大小写敏感，只有type为text时才有意义。
doc_value	bool	否	该字段是否开启统计。

响应元素

- 响应头
CreateIndex接口无特有响应头，关于Log Service API的公共响应头，请参考[公共响应头](#)。
- 响应元素
HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
400	IndexInfoInvalid	required field token is lacking or of error format
400	IndexAlreadyExist	log store index is already created
404	ProjectNotExist	The Project does not exist : {Project}

HTTP状态码	ErrorCode	ErrorMessage
404	LogStoreNotExist	logstore {logstoreName} dose not exist
500	InternalServerError	Specified Server Error Message

② 说明 上表错误消息中 {Project} 和 {logstoreName} 表示该部分会被具体的ProjectName和LogstoreName来替换。

示例

请求示例

```
POST /logstores/my-logstore/index HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Mon, 07 May 2018 09:43:16 GMT
Content-Type: application/json
Content-MD5: 22876515FC311F857AD6C7902F6A0148
Content-Length: 316
Connection: Keep-Alive
{
  "line": {
    "token": [
      ",",
      " ",
      "'",
      "\"",
      ";",
      "=",
      "(",
      ")",
      "[",
      "]",
      "{",
      "}",
      "?",
      "@",
      "&",
      "<",
      ">",
      "/",
      ":",
      "\n",
      "\t",
      "\r"
    ]
  }
}
```

```
,
"keys": {
  "agent": {
    "doc_value": true,
    "caseSensitive": true,
    "alias": "agent_alias",
    "type": "text",
    "token": [
      ",",
      " ",
      "'",
      "\"",
      ";",
      "=",
      "(",
      ")",
      "[",
      "]",
      "{",
      "}",
      "?",
      "@",
      "&",
      "<",
      ">",
      "/",
      ":",
      "\n",
      "\t",
      "\r"
    ]
  }
}
```

3.7.19. DeleteIndex

调用DeleteIndex删除指定Logstore的索引。

请求语法

```
DELETE /logstores/<logstoreName>/index HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

请求参数

- 请求头
DeleteIndex接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	Logstore名称。

响应元素

- 响应头
DeleteIndex接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应元素
HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
400	ParameterInvalid	log store index not created
404	ProjectNotExist	The Project does not exist : {Project}
404	LogStoreNotExist	logstore {logstoreName} does not exist
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
DELETE /logstores/my-logstore/index HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 06 May 2018 13:22:57 GMT
Content-Type: application/x-protobuf
Connection: Keep-Alive
```
- 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 06 May 2018 13:22:57 GMT
x-log-requestid: 5AEF01B1A458E41C2F8326A8
```

3.7.20. GetIndex

调用GetIndex查询指定Logstore的索引。

请求语法

GetIndex接口的请求语法如下所示。

```
GET /logstores/<logstoreName>/index HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

请求元素

- 请求头
GetIndex接口无特有请求头，关于Log Service API的公共请求头，请参见[公共响应头](#)。
- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	Logstore名称。

响应元素

- 响应头
GetIndex接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应元素

GetIndex请求成功，其响应Body会包括指定Project和Logstore的索引，具体格式如下所示。

属性名称	类型	描述
index_mode	string	索引类型。
keys	dict	字段索引配置。key为字段名称，value为索引配置。

属性名称	类型	描述
line	object	全文索引配置。
storage	string	存储类型，目前为pg。
ttl	int	索引文件生命周期，支持7天、30天、90天。
lastModifyTime	int	索引最后更新时间。UNIX时间戳。

◦ 全文索引配置属性

属性名称	类型	描述
caseSensitive	bool	大小写是否敏感。
chn	bool	是否包含中文。
token	array	分词符列表。
include_keys	array	包含的字段列表。
exclude_keys	array	排除的字段列表。

◦ 字段索引配置属性

属性名称	类型	描述
type	string	字段类型。
alias	string	字段别名。
chn	bool	是否包含中文，只有type为text时才存在。
token	array	分词符列表，只有type为text时才存在。
caseSensitive	bool	大小写是否敏感，只有type为text时才存在。
doc_value	bool	是否开启字段统计。

错误码

GetIndex接口除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP状态码	ErrorCode	ErrorMessage	描述
400	IndexConfigNotExist	index config doesn't exist	查询的索引不存在。

HTTP状态码	ErrorCode	ErrorMessage	描述
404	ProjectNotExist	The Project does not exist : {Project}	Project不存在。
404	LogStoreNotExist	logstore {logstoreName} dose not exist	Logstore不存在。
500	InternalServerError	Specified Server Error Message	服务器错误信息。

示例

● 请求示例

```
GET /logstores/logstore-4/index HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 06 May 2018 13:08:42 GMT
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

● 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Type: application/json
Content-Length: 712
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 06 May 2018 13:08:42 GMT
x-log-requestid: 5AEEFE5A8B8AEB5E6C82B395
{
  "index_mode": "v2",
  "keys": {
    "agent": {
      "alias": "",
      "caseSensitive": false,
      "chn": false,
      "doc_value": true,
      "token": [
        ",",
        " ",
        "'",
        "\"",
        ";",
        "=",
        "(",
        ")",
        "[",
        "]"
      ]
    }
  }
}
```

```
    ],
    "]:",
    "{",
    "}",
    "?",
    "@",
    "&",
    "<",
    ">",
    "/",
    ":",
    "\\n",
    "\\t",
    "\\r"
  ],
  "type": "text"
},
"bytes": {
  "alias": "",
  "doc_value": true,
  "type": "long"
},
"remote_ip": {
  "alias": "",
  "caseSensitive": false,
  "chn": false,
  "doc_value": true,
  "token": [
    ",",
    " ",
    "'",
    "\"",
    ";",
    "=",
    "(",
    ")",
    "[",
    "]",
    "{",
    "}",
    "?",
    "@",
    "&",
    "<",
    ">",
    "/",
    ":",
    "\\n",
    "\\t",
    "\\r"
  ],
  "type": "text"
},
"response": {
  "alias": "",
```

```
        "doc_value": true,
        "type": "long"
    }
},
"line": {
    "caseSensitive": false,
    "chn": false,
    "token": [
        ",",
        " ",
        "'",
        "\"",
        ";",
        "=",
        "(",
        ")",
        "[",
        "]",
        "{",
        "}",
        "?",
        "@",
        "&",
        "<",
        ">",
        "/",
        ":",
        "\n",
        "\t",
        "\r"
    ]
},
"storage": "pg",
"ttl": 30,
"lastModifyTime": 1524155379
}
```

3.7.21. PutWebtracking

Web Tracking采集方式，单个请求只能写入一条日志，针对日志量较大的场景，可以使用PutWebTracking API将多条日志合并为一次请求。使用PutWebtracking接口写入日志需要为Logstore打开Web Tracking开关。适用于在网页或者客户端采集日志的场景。

请求语法

```
POST {project}.{endpoint}/logstores/{logstoreName}/track HTTP/1.1
x-log-apiversion: 0.6.0
x-log-bodyrawsize: 1234
x-log-compresstype: gzip
{
  "__topic__": "topic",
  "__source__": "source",
  "__logs__": [
    {
      "key1": "value1",
      "key2": "value2"
    },
    {
      "key1": "value1",
      "key2": "value2"
    }
  ],
  "__tags__": {
    "tag1": "value1",
    "tag2": "value2"
  }
}
```

请求元素

● 请求头

仅需要如下三个请求头：

- x-log-apiversion: 0.6.0
- x-log-bodyrawsize: 1234
- x-log-compresstype: gzip


前两个为必选，第三个是可选，格式和含义请参见[公共请求头](#)。

● 请求参数

- URL参数

参数名称	类型	是否必须	描述
logstoreName	string	是	Logstore名称。
endpoint	string	是	日志服务的Endpoint，请参考 获取日志服务的Endpoint 。
project	string	是	Project名称。

◦ 请求Body

参数名称	类型	是否必须	描述
__topic__	string	否	日志主题。
__source__	string	否	日志来源。
__logs__	list	是	日志内容列表。每个元素为一个JSON对象，表示一条日志。 <div><p> 说明</p><p>WebTracking采集的日志时间为日志到达服务端的时间，每条日志中无需设置__time__字段，如果存在该字段，将被服务端使用日志到达的时间覆盖。</p></div>
__tags__	object	否	日志标签。

响应元素

- 响应头
PutWebtracking接口无特有响应头。关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应元素
成功后无任何响应元素。

示例

- 请求示例

```
POST my-project.cn-hangzhou.log.aliyuncs.com/logstores/logstore-test/track HTTP/1.1
x-log-apiversion: 0.6.0
x-log-bodyrawsize: 1234
x-log-compresstype: gzip
{
  "__topic__": "topic",
  "__source__": "source",
  "__logs__": [
    {
      "foo": "bar",
      "foo1": "bar1"
    },
    {
      "bar": "foo",
      "bar1": "foo1"
    }
  ],
  "__tags__": {
    "tag1": "value1",
    "tag2": "value2"
  }
}
```

- 响应示例

```
HTTP/1.1 200 OK
Header:
{
  Date=Wed, 20 May 2019 07:35:00 GMT,
  Content-Length=0,
  x-log-requestid=5642EFA499248C827B012B39,
  Connection=close,
  Server=nginx/1.6.1
}
```

3.8. Logtail 机器组相关接口

3.8.1. CreateMachineGroup

您可以根据需求创建一组机器，用于日志采集时下发采集配置。

请求语法

```
POST /machinegroups HTTP/1.1
Authorization: <AuthorizationString>
Content-Type:application/json
Content-Length:<Content Length>
Content-MD5<:<Content MD5>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
{
  "groupName" : "testgroup",
  "groupType" : "",
  "groupAttribute" : {
    "externalName" : "testgroup",
    "groupTopic": "testgrouptopic"
  },
  "machineIdentifyType" : "ip",
  "machineList" : [
    "test-ip1",
    "test-ip2"
  ]
}
```

请求元素

- 请求头
CreateMachineGroup接口无特有请求头。关于Log Service API的公共请求头，请参见公共请求头。
- 请求参数

Body参数

属性名称	类型	是否必须	描述
groupName	string	是	机器分组名称，Project下唯一
groupType	string	否	机器分组类型，默认为空
machineIdentifyType	string	是	机器标识类型，分为IP和userdefined两种
groupAttribute	object	是	机器分组的属性，默认为空
machineList	array	是	具体的机器标识，可以是IP或userdefined-id

groupAttribute说明如下所示。

属性名称	类型	是否必须	描述
------	----	------	----

属性名称	类型	是否必须	描述
groupTopic	string	否	机器分组的topic，默认为空
externalName	string	否	机器分组所依赖的外部管理标识，默认为空

响应元素

- 响应头
CreateMachineGroup接口无特有响应头。关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应参数
HTTP 状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
400	MachineGroupAlreadyExist	group {GroupName} already exists
400	InvalidParameter	invalid group resource json
500	InternalServerError	Internal server error

示例

- 请求示例


```
POST /machinegroups HTTP/1.1
Header :
{
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Tue, 10 Nov 2015 17:57:33 GMT",
  "Content-Length": "187",
  "x-log-signaturemethod": "hmac-sha1",
  "Content-MD5": "82033D507DEAAD72067BB58DFDCB590D",
  "User-Agent": "sls-java-sdk-v-0.6.0",
  "Content-Type": "application/json",
  "x-log-bodyrawsize": "0"
}
Body :
{
  "groupName": "test-machine-group",
  "groupType": "",
  "machineIdentifyType": "ip",
  "groupAttribute": {
    "groupTopic": "testtopic",
    "externalName": "testgroup"
  },
  "machineList": [
    "127.0.0.1",
    "127.0.0.2"
  ]
}
```

- 响应示例

```
HTTP/1.1 200 OK
Header :
{
  "Date": "Tue, 10 Nov 2015 17:57:33 GMT",
  "Content-Length": "0",
  "x-log-requestid": "5642300D99248CB76D005D36",
  "Connection": "close",
  "Server": "nginx/1.6.1"
}
```

3.8.2. DeleteMachineGroup

调用DeleteMachineGroup删除机器组。如果机器组上有配置，则Logtail上对应的配置也会被删除。

请求语法

```
DELETE /machinegroups/{groupName} HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

DeleteMachineGroup接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
groupName	string	是	机器分组名称

响应元素

- 响应头

DeleteMachineGroup接口无特有响应头。关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应参数

HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	GroupNotExist	group {GroupName} does not exist
500	InternalServerError	internal server error

示例

- 请求示例

```
DELETE /machinegroups/test-machine-group-4 HTTP/1.1
Header :
{
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Tue, 10 Nov 2015 19:13:28 GMT",
  "Content-Length": "0",
  "x-log-signaturemethod": "hmac-sha1",
  "User-Agent": "sls-java-sdk-v-0.6.0",
  "Content-Type": "application/x-protobuf",
  "x-log-bodyrawsize": "0"
}
```

- 响应示例

```
HTTP/1.1 200 OK
Header :
{
  "Date": "Tue, 10 Nov 2015 19:13:28 GMT",
  "Content-Length": "0",
  "x-log-requestid": "564241D899248C827B000CFE",
  "Connection": "close",
  "Server": "nginx/1.6.1"
}
```

3.8.3. UpdateMachineGroup

调用UpdateMachineGroup更新机器组信息。如果机器组已应用配置，则新加入、减少机器会自动增加、移除配置。

请求语法

```
PUT /machinegroups/{groupName} HTTP/1.1
Authorization: <AuthorizationString>
Content-Type:application/json
Content-Length:<Content Length>
Content-MD5<:<Content MD5>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
{
  "groupName": "test-machine-group",
  "groupType" : "",
  "groupAttribute" : {
    "externalName" : "testgroup",
    "groupTopic": "testgrouptopic"
  },
  "machineIdentifyType" : "ip",
  "machineList" : [
    "test-ip1",
    "test-ip2"
  ]
}
```

请求元素

- 请求头

UpdateMachineGroup接口无特有请求头。关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

- URL 参数

参数名称	类型	是否必须	描述
groupName	string	是	机器分组名称

o

属性名称	类型	是否必须	描述
groupName	string	是	机器分组名称，Project下唯一
groupType	string	否	机器分组类型，默认为空
machineIdentifyType	string	是	机器标识类型，分为IP和userdefined两种
groupAttribute	object	是	机器分组的属性，默认为空
machineList	array	是	具体的机器标识，可以是IP或userdefined-id

groupAttribute 说明如下所示。

属性名称	类型	是否必须	描述
groupTopic	string	否	机器分组的topic，默认为空
externalName	string	否	机器分组所依赖的外部管理系统标识，默认为空

响应元素

- 响应头
UpdateMachineGroup接口无特有响应头。关于Log Service API的公共响应头，请参见公共响应头。
- 响应元素
HTTP状态码返回200。

错误码

除了返回Log Service API的通用错误码，还可能返回如下特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	GroupNotExist	group {GroupName} does not exist
400	InvalidParameter	invalid group resource json
500	InternalServerError	internal server error

示例

- 请求示例

```

PUT /machinegroups/test-machine-group HTTP/1.1
Header :
{
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Tue, 10 Nov 2015 18:41:43 GMT",
  "Content-Length": "194",
  "x-log-signaturemethod": "hmac-sha1",
  "Content-MD5": "2CEBAEBE53C078891527CB70A855BAF4",
  "User-Agent": "sls-java-sdk-v-0.6.0",
  "Content-Type": "application/json",
  "x-log-bodyrawsize": "0"
}
Body :
{
  "groupName": "test-machine-group",
  "groupType": "",
  "machineIdentifyType": "userdefined",
  "groupAttribute": {
    "groupTopic": "testtopic2",
    "externalName": "testgroup2"
  },
  "machineList": [
    "uu_id_1",
    "uu_id_2"
  ]
}

```

● 响应示例

```

HTTP/1.1 200 OK
Header :
{
  "Date": "Tue, 10 Nov 2015 18:41:43 GMT",
  "Content-Length": "0",
  "x-log-requestid": "56423A6799248CA57B00035C",
  "Connection": "close",
  "Server": "nginx/1.6.1"
}

```

3.8.4. ListMachineGroup

调用ListMachineGroup查看机器组列表。

请求语法

```

GET /machinegroups?offset=1&size=100 HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1

```

请求元素

- 请求头

ListMachineGroup接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
offset	int	否	返回记录的起始位置，默认为 0
size	int	否	每页返回最大条目，默认 500（最大值）
groupName	string	否	用于过滤的机器组名称（支持部分匹配）

响应元素

- 响应头

ListMachineGroup接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应元素

请求成功，其响应Body会包括指定Project下的所有机器组列表。具体格式如下所示。

名称	类型	描述
count	int	返回的机器组数目
total	int	返回的机器组总数
machinegroups	json array	返回的机器组列表

```
{
  "machinegroups": [
    "test-machine-group",
    "test-machine-group-2"
  ],
  "count": 2,
  "total": 2
}
```

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
500	InternalServerError	internal server error

示例

- 请求示例

```
GET /machinegroups?groupName=test-machine-group&offset=0&size=3 HTTP/1.1
Header :
{
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Tue, 10 Nov 2015 18:34:44 GMT",
  "Content-Length": "0",
  "x-log-signaturemethod": "hmac-sha1",
  "User-Agent": "sls-java-sdk-v-0.6.0",
  "Content-Type": "application/x-protobuf",
  "x-log-bodyrawsize": "0"
}
```

- 响应示例

```
HTTP/1.1 200 OK
Header :
{
  "Date": "Tue, 10 Nov 2015 18:34:44 GMT",
  "Content-Length": "83",
  "x-log-requestid": "564238C499248C8F7B0001DE",
  "Connection": "close",
  "Content-Type": "application/json",
  "Server": "nginx/1.6.1"
}
Body :
{
  "machinegroups": [
    "test-machine-group",
    "test-machine-group-2"
  ],
  "count": 2,
  "total": 2
}
```

3.8.5. GetMachineGroup

调用GetMachineGroup查看具体的机器组信息。

请求语法

```
GET /machinegroups/{groupName} HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

● 请求头

GetMachineGroup接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

● 请求参数

参数名称	类型	是否必须	描述
groupName	string	是	机器分组名称

响应元素

● 响应头

GetMachineGroup接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

● 响应参数

属性名称	类型	描述
groupName	string	机器分组名称，Project 下唯一
groupType	string	分组类型（空或者 Armory），默认为空
machineIdentifyType	string	机器标识类型，分为IP和 userdefined两种
groupAttribute	json object	机器分组的属性，默认为空
machineList	json array	具体的机器标识，可以是IP或 userdefined-id
createTime	int	机器分组创建时间
lastModifyTime	int	机器分组最近更新时间

groupAttribute说明如下：

属性名称	类型	是否必须	描述
groupTopic	string	否	机器分组的topic，一般不设置
externalName	string	否	机器分组所依赖的外部管理系统（Armory）标识

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	MachineGroupNotExist	group {GroupName} does not exist

HTTP 状态码	ErrorCode	ErrorMessage
500	InternalServerError	internal server error

示例

● 请求示例

```
GET /machinegroups/test-machine-group HTTP/1.1
Header :
{
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Tue, 10 Nov 2015 18:15:24 GMT",
  "Content-Length": "0",
  "x-log-signaturemethod": "hmac-sha1",
  "User-Agent": "sls-java-sdk-v-0.6.0",
  "Content-Type": "application/x-protobuf",
  "x-log-bodyrawsize": "0"
}
```

● 响应示例

```
HTTP/1.1 200 OK
Header :
{
  "Date": "Tue, 10 Nov 2015 18:15:23 GMT",
  "Content-Length": "239",
  "x-log-requestid": "5642343B99248CB36D0060B8",
  "Connection": "close",
  "Content-Type": "application/json",
  "Server": "nginx/1.6.1"
}
Body :
{
  "groupName": "test-machine-group",
  "groupType": "",
  "groupAttribute": {
    "externalName": "testgroup",
    "groupTopic": "testtopic"
  },
  "machineIdentifyType": "ip",
  "machineList": [
    "127.0.0.1",
    "127.0.0.2"
  ],
  "createTime": 1447178253,
  "lastModifyTime": 1447178253
}
```

3.8.6. ApplyConfigToMachineGroup

调用ApplyConfigToMachineGroup将配置应用到机器组。

请求语法

```
GET /machinegroups/{groupName} HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头
ApplyConfigToMachineGroup接口无特有请求头，关于Log Service API的公共请求头，请参见公共请求头。
- 请求参数

参数名称	类型	是否必须	描述
GroupName	string	是	机器分组名称
ConfigName	string	是	日志配置名称

响应元素

- 响应头
ApplyConfigToMachineGroup接口无特有响应头，关于Log Service API的公共响应头，请参见公共响应头。
- 响应元素
HTTP状态码返回200。

错误码

除了返回Log Service API的通用错误码，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	GroupNotExist	group {GroupName} does not exist
404	ConfigNotExist	config {ConfigName} does not exist
500	InternalServerError	internal server error

示例

- 请求示例

```
PUT /machinegroups/sample-group/configs/logtail-config-sample
Header :
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Mon, 09 Nov 2015 09:44:43 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

- 响应示例

```
{
  "date": "Mon, 09 Nov 2015 09:44:43 GMT",
  "connection": "close",
  "x-log-requestid": "56406B0B99248CAA230BA094",
  "content-length": "0",
  "server": "nginx/1.6.1"
}
```

3.8.7. RemoveConfigFromMachineGroup

调用RemoveConfigFromMachineGroup删除机器组中的配置。

请求语法

```
GET /machinegroups/{groupName} HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

RemoveConfigFromMachineGroup接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
GroupName	string	是	机器分组名称
ConfigName	string	是	日志配置名称

响应元素

- 响应头

RemoveConfigFromMachineGroup接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应元素

HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	GroupNotExist	group {GroupName} does not exist
404	ConfigNotExist	config {ConfigName} does not exist
500	InternalServerError	internal server error

示例

- 请求示例

```
DELETE /machinegroups/sample-group/configs/logtail-config-sample
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Mon, 09 Nov 2015 09:48:48 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

- 响应示例

```
{
  "date": "Mon, 09 Nov 2015 09:48:48 GMT",
  "connection": "close",
  "x-log-requestid": "56406C0099248CAA230BE135",
  "content-length": "0",
  "server": "nginx/1.6.1"
}
```

3.8.8. ListMachines

调用ListMachines获得机器组下属于用户并与服务器端连接的机器状态信息。

请求语法

```
GET /machinegroups/{groupName}/machines HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头
ListMachines接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

名称	类型	必须	描述
groupName	string	是	机器分组名称
offset	int	否	返回记录的起始位置，默认为0
size	int	否	每页返回最大条目，默认500（最大值）

响应元素

- 响应头
ListMachines接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应参数

名称	类型	描述
count	int	返回的机器数目
total	int	机器总数
machines	json array	返回的机器列表

machines说明如下：

名称	类型	描述
ip	string	机器的IP地址
machine-uniqueid	string	机器的DMI UUID
userdefined-id	string	机器的用户自定义标识

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	GroupNotExist	group {GroupName} does not exist
500	InternalServerError	internal server error

细节描述

该接口只获取与服务器端连接正常的机器列表。

示例

● 请求示例

```
GET /machinegroups/test-machine-group-5/machines?offset=0&size=3 HTTP/1.1
Header :
{
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Tue, 10 Nov 2015 19:04:57 GMT",
  "Content-Length": "0",
  "x-log-signaturemethod": "hmac-sha1",
  "User-Agent": "sls-java-sdk-v-0.6.0",
  "Content-Type": "application/x-protobuf",
  "x-log-bodyrawsize": "0"
}
```

● 响应示例

```
HTTP/1.1 200 OK
Header :
{
  "Date": "Tue, 10 Nov 2015 19:04:58 GMT",
  "Content-Length": "324",
  "x-log-requestid": "56423FD999248C827B000A57",
  "Connection": "close",
  "Content-Type": "application/json",
  "Server": "nginx/1.6.1"
}
Body :
{
  "machines": [
    {
      "ip": "10.101.166.116",
      "machine-uniqueid": "",
      "userdefined-id": "",
      "lastHeartbeatTime": 1447182247
    },
    {
      "ip": "10.101.165.193",
      "machine-uniqueid": "",
      "userdefined-id": "",
      "lastHeartbeatTime": 1447182246
    },
    {
      "ip": "10.101.166.91",
      "machine-uniqueid": "",
      "userdefined-id": "",
      "lastHeartbeatTime": 1447182248
    }
  ],
  "count": 3,
  "total": 8
}
```

3.8.9. GetAppliedConfigs

调用GetAppliedConfigs获得机器组上已经被应用的配置名称。

请求语法

```
GET /machinegroups/{groupName}/configs HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

GetAppliedConfigs接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
groupName	string	是	机器分组名称

响应元素

- 响应头

GetAppliedConfigs接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应元素

请求成功后，其响应Body会包括指定机器组下的所有机器列表，具体格式如下所示。

名称	类型	描述
count	整型	返回的配置数目。
configs	字符串数组	返回的配置列表。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	GroupNotExist	group {GroupName} does not exist
500	InternalServerError	internal server error

示例

- 请求示例

```
GET /machinegroups/test-machine-group/configs HTTP/1.1
Header :
{
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Tue, 10 Nov 2015 19:45:48 GMT",
  "Content-Length": "0",
  "x-log-signaturemethod": "hmac-sha1",
  "User-Agent": "sls-java-sdk-v-0.6.0",
  "Content-Type": "application/x-protobuf",
  "x-log-bodyrawsize": "0"
}
```

- 响应示例


```
HTTP/1.1 200 OK
Header :
{
  "Date": "Tue, 10 Nov 2015 19:45:48 GMT",
  "Content-Length": "53",
  "x-log-requestid": "5642496C99248C8C7B00173F",
  "Connection": "close",
  "Content-Type": "application/json",
  "Server": "nginx/1.6.1"
}
Body :
{
  "configs": [
    "two",
    "three",
    "test_logstore"
  ],
  "count": 3
}
```

3.9. Logtail 配置相关接口

3.9.1. CreateConfig

调用CreateConfig接口用于创建日志采集配置。

请求语法

```
POST /configs HTTP/1.1
Authorization: <AuthorizationString>
Content-Type:application/json
Content-Length:<Content Length>
Content-MD5<:<Content MD5>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
{
  "configName": "testcategory1",
  "inputType": "file",
  "inputDetail": {
    "logType": "common_reg_log",
    "logPath": "/var/log/httpd/",
    "filePattern": "access*.log",
    "localStorage": true,
    "timeFormat": "%Y/%m/%d %H:%M:%S",
    "logBeginRegex": ".*",
    "regex": "(\\w+) (\\s+)",
    "key": ["key1", "key2"],
    "filterKey": ["key1"],
    "filterRegex": ["regex1"],
    "fileEncoding": "utf8",
    "topicFormat": "none"
  },
  "outputType": "LogService",
  "outputDetail": {
    {
      "logstoreName": "perfcounter"
    }
  }
}
```

请求元素

- 请求头

CreateConfig接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

关于请求参数和各种模式的Logtail配置样例，请参见[Logtail 配置](#)。

响应元素

- 响应头

CreateConfig接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应元素

HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
400	ConfigAlreadyExist	config {Configname} already exists
400	InvalidParameter	invalid config resource json
500	InternalServerError	internal server error

创建过程中遇到配置已经存在、格式错误、必要参数遗失或者quota超过限制等错误，则会创建失败。

示例

- 请求示例

```

POST /configs HTTP/1.1
Header :
{
  'Content-Length': 737,
  'Host': 'ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com',
  'x-log-bodyrawsize': 737,
  'Content-MD5': 'FBA01ECF7255BE143379BC70C56BBF68',
  'x-log-signaturemethod': 'hmac-sha1',
  'Date': 'Mon, 09 Nov 2015 07:45:30 GMT',
  'x-log-apiversion': '0.6.0',
  'User-Agent': 'log-python-sdk-v-0.6.0',
  'Content-Type': 'application/json',
  'Authorization': 'LOG <yourAccessKeyId>:<yourSignature>'
}
Body:
{
  "configName": "sample-logtail-config",
  "inputType": "file",
  "inputDetail": {
    "logType": "common_reg_log",
    "logPath": "/var/log/httpd/",
    "filePattern": "access*.log",
    "localStorage": true,
    "timeFormat": "%d/%b/%Y:%H:%M:%S",
    "logBeginRegex": "\\d+\\.\\.\\d+\\.\\.\\d+\\.\\.\\d+ - .*",
    "regex": "([\\d\\.]+) \\S+ \\S+ \\[([\\S+]) \\S+\\] \"([\\w+]) ([^\"\\"]*)\" ([\\d\\.]+) ([\\d+]) ([\\d+|-]) \"([^\"\\"]*)\" \"([^\"\\"]*)\".*",
    "key": ["ip", "time", "method", "url", "request_time", "request_length", "status", "length", "ref_url", "browser"],
    "filterKey": [],
    "filterRegex": [],
    "topicFormat": "none",
    "fileEncoding": "utf8"
  },
  "outputType": "LogService",
  "outputDetail": {
    {
      "logstoreName": "sls-test-logstore"
    }
  }
}

```

● 响应示例

```

HTTP/1.1 200 OK
Header
{
  'date': 'Mon, 09 Nov 2015 07:45:30 GMT',
  'connection': 'close',
  'x-log-requestid': '56404F1A99248CA26C002180',
  'content-length': '0',
  'server': 'nginx/1.6.1'
}

```

3.9.2. ListConfig

调用ListConfig接口列出Project下所有的配置信息，支持通过参数进行翻页显示。

请求语法

```
GET /configs?offset=0&size=100 HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头
ListConfig接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

参数名称	类型	是否必须	描述
offset	integer	否	返回记录的起始位置，默认为0。
size	integer	否	每页返回最大条目，默认为最大值500。

响应元素

- 响应头
ListConfig接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应参数
ListConfig接口的响应Body会包含该project下所有的config列表，具体格式如下所示。

名称	类型	描述
count	整型	每页返回的配置数目。
total	整型	返回的配置总数。
configs	字符串数组	返回的配置列表。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
----------	-----------	--------------

HTTP 状态码	ErrorCode	ErrorMessage
404	ConfigNotExist	config {Configname} does not exist
500	InternalServerError	internal server error

示例

● 请求示例

```
GET /configs?offset=0&size=10 HTTP/1.1
Header :
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Mon, 09 Nov 2015 09:19:13 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

● 响应示例

```
Header :
{
  "content-length": "103",
  "server": "nginx/1.6.1",
  "connection": "close",
  "date": "Mon, 09 Nov 2015 09:19:13 GMT",
  "content-type": "application/json",
  "x-log-requestid": "5640651199248CAA2300C2BA"
}
Body:
{
  "count": 3,
  "configs":
  [
    "logtail-config-sample",
    "logtail-config-sample-2",
    "logtail-config-sample-3"
  ],
  "total": 3
}
```

3.9.3. GetAppliedMachineGroups

调用GetAppliedMachineGroups接口获取应用到日志采集配置的机器列表。

请求语法

```
GET /configs/{configName}/machinegroups HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头
GetAppliedMachineGroups接口无特有请求头，关于Log Service API的公共请求头，请参见公共请求头。

参数名称	类型	是否必须	描述
ConfigName	string	是	配置名称。

响应元素

- 响应头
GetAppliedMachineGroups接口无特有响应头，关于Log Service API的公共响应头，请参见公共响应头。

- 响应参数
请求成功后其响应Body会包含指定机器组下的所有机器列表，具体格式如下所示。

名称	类型	描述
count	整型	返回的机器组数目。
machinegroups	字符串数组	返回的机器组列表。

错误码

除了返回Log Service API的通用错误码，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	GroupNotExist	group {GroupName} does not exist
500	InternalServerError	internal server error

示例

- 请求示例

```
GET /configs/logtail-config-sample/machinegroups
Header:
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Mon, 09 Nov 2015 09:51:38 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

- 响应示例

```
Header :
{
  "content-length": "44",
  "server": "nginx/1.6.1",
  "connection": "close",
  "date": "Mon, 09 Nov 2015 09:51:38 GMT",
  "content-type": "application/json",
  "x-log-requestid": "56406CAA99248CAA230BE828"
}
Body:
{
  "count": 1,
  "machinegroups":
  [
    "sample-group"
  ]
}
```

3.9.4. GetConfig

调用GetConfig接口获取采集配置的详细信息。

请求语法

```
GET /configs/<configName> HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头

GetConfig接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

参数名称	类型	是否必须	描述
ConfigName	String	是	日志配置名称。

响应元素

- 响应头

Get Config接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应参数

响应内容中包含的参数信息和各种模式的Logtail配置样例，请参见[Logtail 配置](#)。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	ConfigNotExist	Config {Configname} does not exist
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
GET /configs/logtail-config-sample
Header :
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Mon, 09 Nov 2015 08:29:15 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

- 响应示例

```
Header :
{
  "content-length": "730",
  "server": "nginx/1.6.1",
  "connection": "close",
  "date": "Mon, 09 Nov 2015 08:29:15 GMT",
  "content-type": "application/json",
  "x-log-requestid": "5640595B99248CAA23004A59"
}
Body :
{
  "configName": "logtail-config-sample",
  "outputDetail": {
    "endpoint": "http://cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
    "logstoreName": "sls-test-logstore"
  },
  "outputType": "LogService",
  "inputType": "file",
  "inputDetail": {
    "regex": "([\\d\\.]+) \\S+ \\S+ \\[(\\S+) \\S+\\] \"(\\w+)\" ([^\"\\\"]*)\" ([\\d\\.]+)
(\\d+) (\\d+) (\\d+|-) \"([^\"]*)\" \"([^\"]*)\".*",
    "filterKey": [],
    "logPath": "/var/log/httpd/",
    "logBeginRegex": "\\d+\\.\\d+\\.\\d+\\.\\d+ - .*",
    "logType": "common_reg_log",
    "topicFormat": "none",
    "localStorage": true,
    "key": [
      "ip",
      "time",
      "method",
      "url",
      "request_time",
      "request_length",
      "status",
      "length",
      "ref_url",
      "browser"
    ],
    "filePattern": "access*.log",
    "timeFormat": "%d/%b/%Y:%H:%M:%S",
    "filterRegex": []
  },
  "createTime": 1447040456,
  "lastModifyTime": 1447050456
}
```

3.9.5. DeleteConfig

调用DeleteConfig接口删除指定的日志采集配置。如果该配置已经被应用到机器组，则Logtail配置也会被删除。

请求语法

```
DELETE /configs/<configName> HTTP/1.1
Authorization: <AuthorizationString>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
```

请求元素

- 请求头
DeleteConfig接口无特有请求头，关于Log Service API的公共请求头，请参见公共请求头。
- 请求参数

参数名称	类型	是否必须	描述
ConfigName	String	是	Logtail采集配置名称

响应元素

- 响应头
DeleteConfig接口无特有响应头，关于Log Service API的公共响应头，请参见公共响应头。
- 响应参数
HTTP状态码返回200。

错误码

除了返回Log Service API的通用错误码，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	ConfigNotExist	config {Configname} does not exist
400	InvalidParameter	invalid config resource json
500	InternalServerError	internal server error

示例

- 请求示例

```
DELETE /configs/logtail-config-sample
Header :
{
  "Content-Length": 0,
  "x-log-signaturemethod": "hmac-sha1",
  "x-log-bodyrawsize": 0,
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "Date": "Mon, 09 Nov 2015 09:28:21 GMT",
  "x-log-apiversion": "0.6.0",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
```

- 响应示例

```
Header :
{
  "date": "Mon, 09 Nov 2015 09:28:21 GMT",
  "connection": "close",
  "x-log-requestid": "5640673599248CAA230836C6",
  "content-length": "0",
  "server": "nginx/1.6.1"
}
```

3.9.6. UpdateConfig

调用UpdateConfig接口更新配置内容。如果配置被应用到机器组，则对应机器组也会同时更新。

请求语法

```
PUT /configs/<configName> HTTP/1.1
Authorization: <AuthorizationString>
Content-Type:application/json
Content-Length:<Content Length>
Content-MD5<:<Content MD5>
Date: <GMT Date>
Host: <Project Endpoint>
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
{
  "configName": "testcategory1",
  "inputType": "file",
  "inputDetail": {
    "logType": "common_reg_log",
    "logPath": "/var/log/httpd/",
    "filePattern": "access.log",
    "localStorage": true,
    "timeFormat": "%Y/%m/%d %H:%M:%S",
    "logBeginRegex": ".*",
    "regex": "(\\w+) (\\s+)",
    "key": ["key1", "key2"],
    "filterKey": ["key1"],
    "filterRegex": ["regex1"],
    "topicFormat": "none"
  },
  "outputType": "LogService",
  "outputDetail": {
    "logstoreName": "perfcounter"
  }
}
```

请求元素

- 请求头

UpdateConfig接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

请求参数和各种模式的Logtail配置样例，请参见[Logtail 配置](#)。

响应元素

- 响应头

UpdateConfig接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应参数

成功后返回200状态码。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP 状态码	ErrorCode	ErrorMessage
404	ConfigNotExist	config {Configname} does not exist
400	InvalidParameter	invalid config resource json
400	BadRequest	config resource configname does not match with request
500	InternalServerError	internal server error

创建过程中遇到格式错误、必要参数遗失、quota超过限制等，则会创建失败。

示例

- 请求示例

```

PUT /configs/logtail-config-sample
Header :
{
  "Content-Length": 737,
  "Host": "ali-test-project.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com",
  "x-log-bodyrawsize": 737,
  "Content-MD5": "431263EB105D584A5555762A81E869C0",
  "x-log-signaturemethod": "hmac-sha1",
  "Date": "Mon, 09 Nov 2015 09:14:32 GMT",
  "x-log-apiversion": "0.6.0",
  "User-Agent": "log-python-sdk-v-0.6.0",
  "Content-Type": "application/json",
  "Authorization": "LOG <yourAccessKeyId>:<yourSignature>"
}
Body :
{
  "outputDetail": {
    "logstoreName": "sls-test-logstore"
  },
  "inputType": "file",
  "inputDetail": {
    "regex": "([\\d\\.]+) \\S+ \\S+ \\[(\\S+) \\S+\\] \\\"(\\w+) ([^\\\"]*)\\\" ([\\d\\.]+)
(\\d+) (\\d+) (\\d+|-) \\\"([^\"]*)\\\" \\\"([^\"]*)\\\".*",
    "filterKey": [],
    "logPath": "/var/log/nginx/",
    "logBeginRegex": "\\d+\\.\\d+\\.\\d+\\.\\d+ - .*",
    "logType": "common_reg_log",
    "topicFormat": "none",
    "localStorage": true,
    "key": [
      "ip",
      "time",
      "method",
      "url",
      "request_time",
      "request_length",
      "status",
      "length",
      "ref_url",
      "browser"
    ],
    "filePattern": "access*.log",
    "timeFormat": "%d/%b/%Y:%H:%M:%S",
    "filterRegex": []
  },
  "outputType": "LogService",
  "configName": "logtail-config-sample"
}

```

- 响应示例

```
{
  "date": "Mon, 09 Nov 2015 09:14:32 GMT",
  "connection": "close",
  "x-log-requestid": "564063F899248CAA2300B778",
  "content-length": "0",
  "server": "nginx/1.6.1"
}
```

3.10. 消费组接口

3.10.1. CreateConsumerGroup

调用CreateConsumerGroup接口在指定的Logstore上创建一个消费组。

请求语法

```
POST /logstores/<logstoreName>/consumergroups HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Content-MD5: F58544E4D022CC28A93D0B7CC208A5AA
Content-Length: <ContentLength>
{
  "consumerGroup": <consumerGroup>,
  "timeout": <timeout>,
  "order": <order>
}
```

请求元素

- 请求头
CreateConsumerGroup接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。
- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	Logstore名称。
consumerGroup	string	是	消费组名称，在Project下必须唯一。
timeout	integer	是	超时时间，在超时时间段内没有收到心跳，消费者将被删除。

属性名称	类型	是否必须	描述
order	bool	是	在单个shard中是否按顺序消费。 <ul style="list-style-type: none">◦ true：表示在单个shard中按顺序消费。shard分裂后，先消费原shard数据，然后并列消费两个新shard的数据。◦ false：表示不按顺序消费。

响应元素

- 响应头
CreateConsumerGroup接口无特有响应头，关于Log Service API的公共响应头，请参见公共响应头。
- 响应参数
HTTP状态码返回200。

错误码

除了返回Log Service API的通用错误码，还会返回该接口的特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
400	ConsumerGroupAlreadyExist	consumer group already exist
400	JsonInfoInvalid	consumerGroup or timeout is of error format
404	ProjectNotExist	The Project does not exist : {Project}
404	LogStoreNotExist	logstore {logstoreName} dose not exist
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
POST /logstores/my-logstore/consumergroups HTTP/1.1
Header:
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Fri, 04 May 2018 08:02:22 GMT
Content-Type: application/json
Content-MD5: F58544E4D022CC28A93D0B7CC208A5AA
Content-Length: 65
Connection: Keep-Alive
Body:
{
  "consumerGroup": "consumer-group-1",
  "timeout": 300,
  "order": true
}
```

- 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Fri, 04 May 2018 08:15:11 GMT
x-log-requestid: 5AEC168FA796F4195BF404CB
```

3.10.2. DeleteConsumerGroup

调用DeleteConsumerGroup接口删除一个指定的消费组。

请求语法

```
DELETE /logstores/<logstoreName>/consumergroups/<consumerGroupName> HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/x-protobuf
Content-MD5: F58544E4D022CC28A93D0B7CC208A5AA
```

请求元素

- 请求头

DeleteConsumerGroup接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	消费组所属Logstore的名称。
consumerGroup	string	是	待删除的消费组。

响应元素

- 响应头
DeleteConsumerGroup接口无特有响应头，关于Log Service API的公共响应头，请参见公共响应头。
- 响应参数
HTTP状态码返回200。

错误码

除了返回Log Service API的通用错误码，还会返回该接口的特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	The Project does not exist : {Project}
404	LogStoreNotExist	logstore {logstoreName} dose not exist
500	InternalServerError	Specified Server Error Message

细节描述

删除一个不存在的消费组时，返回成功。

示例

- 请求示例

```
DELETE /logstores/logstore-test/consumergroups/consumer-group-1 HTTP/1.1
Header:
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Fri, 04 May 2018 08:02:22 GMT
Content-Type: application/json
Content-MD5: F58544E4D022CC28A93D0B7CC208A5AA
Content-Length: 65
Connection: Keep-Alive
```
- 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Fri, 04 May 2018 08:15:11 GMT
x-log-requestid: 5AEC168FA796F4195BF404CB
```

3.10.3. GetCheckPoint

调用GetCheckPoint接口获取指定消费组消费某个或者所有Shard的checkpoint。

请求语法

```
GET /logstores/<logstoreName>/consumergroups/<consumerGroup>?shard=<shardId> HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Connection: Keep-Alive
```

请求元素

- 请求头

GetCheckPoint接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	消费组所属Logstore的名称。
consumerGroup	string	是	消费组名称。
shardId	integer	否	Shard ID。 <ul style="list-style-type: none">如果指定的Shard不存在则返回空列表。如果不指定Shard则返回所有Shard的checkpoint。

响应元素

- 响应头

GetCheckPoint接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

- 响应参数

GetCheckPoint请求成功，其响应Body会包含指定消费组消费的Shard的checkpoint，具体格式如下。

属性名称	类型	描述
shard	integer	Shard ID。
checkpoint	string	checkpoint的值。
updateTime	integer	checkpoint最后的更新时间。
consumer	string	Shard所属的消费者。

错误码

除了返回Log Service API的通用错误码，还会返回该接口的特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	The Project does not exist : {Project}
404	LogStoreNotExist	logstore {logstoreName} dose not exist
404	ConsumerGroupNotExist	consumer group not exist
500	InternalServerError	Specified Server Error Message

示例

● 请求示例

```
GET /logstores/my-logstore/consumergroups/consumer_group_test?shard=0
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Fri, 04 May 2018 09:26:53 GMT
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

● 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Type: application/json
Content-Length: 111
Connection: close
Access-Control-Allow-Origin: *
Date: Fri, 04 May 2018 09:26:53 GMT
x-log-requestid: 5AEC275D1FFC036AB254B20C
[
  {
    "shard": 0,
    "checkpoint": "MTUyNDElNTM3OTM3MzkwODQ5Ng==",
    "updateTime": 1524224984800922,
    "consumer": "consumer_1"
  }
]
```

3.10.4. HeartBeat

调用HeartBeat接口为指定消费者发送心跳到服务端。

请求语法

```
POST /logstores/<logstoreName>/consumergroups/<consumerGroup>?type=heartbeat&consumer=<consumer> HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Content-MD5: F58544E4D022CC28A93D0B7CC208A5AA
Content-Length: <ContentLength>
<shard ID list>
```

请求元素

- 请求头

HeartBeat接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	Logstore 的名称，在Project 下必须唯一。
consumerGroup	string	是	消费组名称，在Project 下必须唯一。

属性名称	类型	是否必须	描述
consumer	string	是	消费者。
{Shard ID List}	array	是	正在消费的Shard Id列表。

响应元素

- 响应头
Heart Beat接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。
- 响应参数
Heart Beat请求成功，HTTP状态码返回200，同时返回消费者负责消费的所有Shard ID列表。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
400	NotExistConsumerWithBody	non-exist consumer with non-empty body of heartbeat message
404	ProjectNotExist	The Project does not exist : {Project}
404	LogStoreNotExist	logstore {logstoreName} dose not exist
404	ConsumerGroupNotExist	consumer group not exist
500	InternalServerError	Specified Server Error Message

细节描述

只有消费者和服务端建立连接之后才能发送心跳。

示例

- 请求示例

```
POST /logstores/my-logstore/consumergroups/consumer_group_test?type=heartbeat&consumer=consumer_1 HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 06 May 2018 09:44:10 GMT
Content-Type: application/json
Content-MD5: 8D5162CA104FA7E79FE80FD92BB657FB
Content-Length: 3
Connection: Keep-Alive
[0]
```

- 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Type: application/json
Content-Length: 5
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 06 May 2018 09:44:11 GMT
x-log-requestid: 5AEECE6B1FFC0366B2553FB5
[0,1]
```

3.10.5. ListConsumerGroup

调用ListConsumerGroup接口查询指定Logstore的所有消费组。

请求语法

```
GET /logstores/<logstoreName>/consumergroups HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/x-protobuf
Connection: Keep-Alive
```

请求元素

- 请求头

ListConsumerGroup接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	Logstore 的名称。

响应元素

- 响应头
List ConsumerGroup接口无特有响应头，关于Log Service API的公共响应头，请参见公共响应头。
- 响应参数
List ConsumerGroup请求成功，其响应body包含从指定Logstore中读取日志数据的所有消费组，具体格式如下所示。

属性名称	类型	描述
consumerGroup	string	消费组名称。
timeout	integer	超时时间，在超时时间段内没有收到心跳，消费组将被删除。
order	bool	是否按顺序消费。

错误码

除了返回Log Service API的通用错误码，还会返回该接口的特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
404	ProjectNotExist	The Project does not exist : {Project}
404	LogStoreNotExist	logstore {logstoreName} dose not exist
500	InternalServerError	Specified Server Error Message

示例

- 请求示例

```
GET /logstores/my-logstore/consumergroups HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Fri, 04 May 2018 08:47:30 GMT
Content-Type: application/x-protobuf
Connection: Keep-Alive
```
- 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Type: application/json
Connection: close
Access-Control-Allow-Origin: *
Date: Fri, 04 May 2018 08:47:31 GMT
x-log-requestid: 5AEC1E23048191954B42EAB9
[
  {
    "name": "test-consumer-group",
    "timeout": 30,
    "order": true
  }
]
```

3.10.6. UpdateCheckpoint

调用UpdateCheckpoint接口更新指定Project和Logstore下的Consumer Group的某个Shard的checkpoint。

请求语法

```
POST /logstores/<logstoreName>/consumergroups/<consumerGroupName>?forceSuccess=<forceSuccess>&type=checkpoint&consumer=<consumer> HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Content-Length: <ContentLength>
Connection: Keep-Alive
{
  "shard": <shardID>,
  "checkpoint": <checkpoint>
}
```

请求元素

- 请求头

UpdateCheckpoint接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

- 请求参数

名称	类型	是否必须	描述
logstoreName	string	是	消费组所属Logstore 的名称。
consumerGroup	string	是	消费组名称。

名称	类型	是否必须	描述
forceSuccess	bool	否	是否强制更新。
consumer	string	否	消费者。
shard	integer	是	Shard ID。
checkpoint	string	是	checkpoint值。

响应元素

- 响应头
UpdateCheckPoint接口无特有响应头，关于Log Service API的公共响应头，请参见公共响应头。
- 响应参数
HTTP状态码返回200。

错误码

除了返回Log Service API的通用错误码，还会返回该接口的特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
400	InvalidShardCheckPoint	shard checkpoint not encoded by base64
404	ProjectNotExist	The Project does not exist : {Project}
404	LogStoreNotExist	logstore {logstoreName} dose not exist
404	ConsumerGroupNotExist	consumer group not exist
404	ShardNotExist	shard not exist
500	InternalServerError	Specified Server Error Message

细节描述

当不指定消费者时，必须指定forceSuccess为true才能更新checkpoint。

示例

- 请求示例

```
POST /logstores/logstore-4/consumergroups/consumer_group_test?forceSuccess=false&type=checkpoint&consumer=consumer_1 HTTP/1.1
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Sun, 06 May 2018 09:14:53 GMT
Content-Type: application/json
Content-MD5: 59457BAFB3F85A5117BDE243947583B0
Content-Length: 55
Connection: Keep-Alive
{
  "shard": 0,
  "checkpoint": "MTUyNDE1NTM3OTM3MzkwODQ5Ng=="
}
```

- 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Sun, 06 May 2018 09:14:54 GMT
x-log-requestid: 5AEEC78E1FFC0366B24F1C63
```

3.10.7. UpdateConsumerGroup

调用UpdateConsumerGroup接口修改指定消费组的属性。

请求语法

```
PUT /logstores/<logstoreName>/consumergroups/<consumerGroupName> HTTP/1.1
Authorization: <AuthorizationString>
x-log-bodyrawsize: 0
User-Agent: <UserAgent>
x-log-apiversion: 0.6.0
Host: <Project Endpoint>
x-log-signaturemethod: hmac-sha1
Date: <GMT Date>
Content-Type: application/json
Content-MD5: F58544E4D022CC28A93D0B7CC208A5AA
Content-Length: <ContentLength>
{
  "timeout": <timeout>,
  "order": <order>
}
```


请求元素

- 请求头

UpdateConsumerGroup接口无特有请求头，关于Log Service API的公共请求头，请参见[公共请求头](#)。

● 请求参数

属性名称	类型	是否必须	描述
logstoreName	string	是	消费组所属 Logstore 的名称。
consumerGroup	string	是	消费组名称。
timeout	integer	否	超时时间，在超时时间段内没有收到心跳，消费组将被删除。
order	bool	否	在单个shard中是否按顺序消费。 <ul style="list-style-type: none">◦ true：表示在单个shard中按顺序消费。shard分裂后，先消费原shard数据，然后并列消费两个新shard的数据。◦ false：表示不按顺序消费。

 说明 `timeout` 和 `order` 需要至少指定一个。

响应元素

● 响应头

UpdateConsumerGroup接口无特有响应头，关于Log Service API的公共响应头，请参见[公共响应头](#)。

● 响应参数

HTTP状态码返回200。

错误码

除了返回Log Service API的[通用错误码](#)，还会返回该接口的特有错误码。

HTTP状态码	ErrorCode	ErrorMessage
400	JsonInfoInvalid	timout is of error value type
404	ProjectNotExist	The Project does not exist : {Project}
404	ConsumerGroupNotExist	consumer group not exist
404	LogStoreNotExist	logstore {logstoreName} dose not exist
500	InternalServerError	Specified Server Error Message

示例

● 请求示例

```
PUT /logstores/logstore-test/consumergroups/consumer-group-1 HTTP/1.1
Header:
Authorization: LOG <yourAccessKeyId>:<yourSignature>
x-log-bodyrawsize: 0
User-Agent: sls-java-sdk-v-0.6.1
x-log-apiversion: 0.6.0
Host: my-project.cn-shanghai.log.aliyuncs.com
x-log-signaturemethod: hmac-sha1
Date: Fri, 04 May 2018 08:02:22 GMT
Content-Type: application/json
Content-MD5: F58544E4D022CC28A93D0B7CC208A5AA
Content-Length: 65
Connection: Keep-Alive
Body:
{
  "order": false,
  "timeout": 100
}
```

- 响应示例

```
HTTP/1.1 200
Server: nginx/1.12.1
Content-Length: 0
Connection: close
Access-Control-Allow-Origin: *
Date: Fri, 04 May 2018 08:15:11 GMT
x-log-requestid: 5AEC168FA796F4195BF404CB
```

3.11. RAM/STS

3.11.1. 概览

借助RAM实现子账号对主账号的 Log Service 资源的访问

您创建的Project、Logstore、Config、MachineGroup，都是您自己拥有的资源。默认情况下，您对自己的资源拥有完整的操作权限，可以使用本文档中列举的所有 API 对资源进行操作。

但在主子账号的场景下，子账号刚创建时是没有资格去操作主账号的资源的。需要通过RAM授权的方式，给予子账号操作主账号资源的权限。

RAM和日志服务相关的授权策略：

- 向指定日志库（Logstore）上传数据

给予子账号授予该权限，那么子账号将可以通过API/SDK直接向指定日志库上传数据，授权策略描述如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": [
        "log:Post*",
        "log:BatchPost*"
      ],
      "Resource": ["acs:log:*:*:project/<指定的 project 名称>/logstore/<指定的 logstore 名称>"],
      "Effect": "Allow"
    }
  ]
}
```

- 控制台查询指定日志库（Logstore）数据

给予账号授予该权限，那么子账号在登录控制台后将对主账号拥有指定日志库资源只读的访问权限（查询日志、拖取日志、查看日志库列表）。授权策略描述如下：

```
{
  "Version": "1",
  "Statement": [
    {
      "Action": ["log:List*"],
      "Resource": ["acs:log:*:*:project/<指定的 project 名称>/*"],
      "Effect": "Allow"
    },
    {
      "Action": ["log:Get*"],
      "Resource": ["acs:log:*:*:project/<指定的 project 名称>/logstore/<指定的 logstore 名称>"],
      "Effect": "Allow"
    }
  ]
}
```

如果您不需要跨账户进行 Log Service 资源的授权和访问，您可以跳过此章节。跳过这些部分并不影响您对文档中其余部分的理解和使用。

更多信息：

- [RAM 中可授权的 Log Service 资源类型](#)
- [RAM 中可对 Log Service 资源进行授权的 Action](#)
- [Log Service API 发生子账号访问主账号资源时的鉴权规则](#)

3.11.2. 资源列表

日志服务支持通过主账号为子账号授予相关资源权限，方便子账号对部分资源进行操作。本文档主要介绍为子账号授权的资源类型。

目前可以授予RAM用户的资源类型及其描述方式如下表所示。

资源类型	授权策略中的资源描述方式
Project/Logstore	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}
Project/Logstore/Shipper	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}/shipper/\${shipperName}
	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/*
Project/Config	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/\${logtailconfig}
	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/*
Project/MachineGroup	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/\${machineGroupName}
	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/*
Project/ConsumerGroup	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}/consumergroup/\${consumerGroupName}
	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}/consumergroup/*
Project/SavedSearch	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/savedsearch/\${savedSearchName}
	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/savedsearch/*
Project/Dashboard	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/dashboard/\${dashboardName}
	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/dashboard/*
Project/Alarm	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/alert/\${alarmName}
	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/alert/*
泛指模式	acs:log:\${regionName}:\${projectOwnerAliUid}:*
	acs:log:*:\${projectOwnerAliUid}:*

 **说明** Log Service中的资源具有层级关系。Project是根资源，Logstore、config、machinegroup是Project的子资源且相互之间平级，shipper和consumergroup是Logstore的子资源。

授权策略中各个参数的说明如下。

参数	说明
<code>\${regionName}</code>	某个region的名称。
<code>\${projectOwnerAliUid}</code>	阿里云账号ID。
<code>\${projectName}</code>	Project的名称。
<code>\${logstoreName}</code>	Logstore的名称。
<code>\${logtailconfig}</code>	Logtail配置的名称。
<code>\${machineGroupName}</code>	机器组的名称。
<code>\${shipperName}</code>	日志投递规则的名称。
<code>\${consumerGroupName}</code>	协同消费组的名称。
<code>\${savedSearchName}</code>	快速查询的名称。
<code>\${dashboardName}</code>	仪表盘的名称。
<code>\${alarmName}</code>	报警规则的名称。

3.11.3. 动作列表

本文档主要介绍RAM中对日志服务资源进行授权的动作列表。

RAM中可对Log Service资源进行授权的Action

在RAM中，可以对一个Log Service资源进行以下Action的授权。

动作（Action）	说明
<code>CreateProject</code>	创建Project。
<code>GetLogs</code>	查询指定Project下某个Logstore中的日志数据。
<code>GetHistograms</code>	查询指定的Project下某个Logstore中日志的分布情况。
<code>GetLogstore</code>	查看Logstore属性。
<code>ListLogstore</code>	列出指定Project下的所有Logstore的名称。
<code>CreateLogstore</code>	在Project下创建Logstore。
<code>DeleteLogstore</code>	删除Logstore，包括所有Shard数据以及索引等。
<code>UpdateLogstore</code>	更新Logstore的属性。

动作 (Action)	说明
log:GetCursorOrData (GetCursor , PullLogs)	根据时间获得游标 (cursor) ; 根据游标、数量获得日志。
ListShards	列出Logstore下当前所有可用Shard。
PostLog	向指定的Logstore写入日志数据。
CreateConfig	在Project下创建日志配置。
UpdateConfig	更新配置内容。
DeleteConfig	删除指定配置。
GetLogstore	获得一个配置的详细信息。
UpdateConfig	列出Project下所有配置信息, 可以通过参数进行翻页。
CreateMachineGroup	创建机器组, 用以指定需要收集日志的服务器。
UpdateMachineGroup	更新机器组信息。
DeleteMachineGroup	删除机器组。
GetMachineGroup	查看具体的机器组信息。
ListMachineGroup	查看机器组名称列表。
ListMachines	获得机器组下属于用户并与Server端连接的机器状态信息。
ApplyConfigToMachineGroup	将配置应用到机器组。
RemoveConfigFromMachineGroup	从机器组中删除配置。
GetAppliedConfigs	获得机器组上已经被应用的配置名称。
CreateConsumerGroup	在指定的Logstore上创建一个消费组。
UpdateConsumerGroup	修改指定消费组属性。
DeleteConsumerGroup	删除一个指定的消费组。
ListConsumerGroup	查询指定Logstore的所有消费组。
UpdateCheckPoint	更新指定消费组的某个Shard的checkpoint。
HeartBeat	为指定消费者发送心跳到服务端。
GetCheckPoint	获取指定消费组消费的某个或者所有Shard的checkpoint。
CreateIndex	为指定Logstore创建索引。

动作 (Action)	说明
DeleteIndex	删除指定Logstore的索引。
GetIndex	查询指定Logstore的索引。
UpdateIndex	更新指定Logstore的索引。
log:CreateSavedSearch	创建快速查询。
log:UpdateSavedSearch	更新快速查询。
log:GetSavedSearch	查看指定快速查询。
log>DeleteSavedSearch	删除快速查询。
log:ListSavedSearch	查看快速查询列表。
log:CreateDashboard	创建仪表盘。
log:UpdateDashboard	更新仪表盘。
log:GetDashboard	查看指定仪表盘。
log>DeleteDashboard	删除仪表盘。
log:ListDashboard	查看仪表盘列表。
log:CreateJob	创建任务。例如创建告警、订阅。
log:UpdateJob	更新任务。
log:CreateApp	日志服务APP（成本管家、日志审计）的创建权限。
log:UpdateApp	日志服务APP（成本管家、日志审计）的更新权限。
log:GetApp	日志服务APP（成本管家、日志审计）的查看权限。
log>DeleteApp	日志服务APP（成本管家、日志审计）的删除权限。

3.11.4. 鉴权规则

当子账号通过 Log Service OpenAPI 对主账号的资源进行访问时，Log Service 后台对 RAM 进行权限检查，以确保资源拥有者的确将相关资源的相关权限授予了调用者。本文档为您列举Log Service API发生子账号访问主账号资源时的鉴权规则。

Logstore

每个不同的 Log Service API 会根据涉及到的资源以及 API 的语义来确定需要检查哪些资源的权限。具体各类 API 的鉴权规则见下表。

Action	Resource
log:GetLogStore	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}
log:ListLogStores	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/*
log:CreateLogStore	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/*
log>DeleteLogStore	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}
log:UpdateLogStore	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}

loghub

数据写入以及消费类 API，其中获取数据游标 API GetCursor 以及获取数据 API GetLogs 共用同一个 Action（log:GetCursorOrData）。

Action	Resource
log:GetCursorOrData	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}
log:ListShards	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}
log:PostLogStoreLogs	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logstore/\${logstoreName}

config

Action	Resource
log:CreateConfig	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/*
log:UpdateConfig	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/\${logtailConfigName}
log>DeleteConfig	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/\${logtailConfigName}

Action	Resource
log:GetConfig	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/\${logtailConfigName}</code>
log:ListConfig	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/*</code>

machinegroup

Actions	Resources
log:CreateMachineGroup	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/*</code>
log:UpdateMachineGroup	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/\${machineGroupName}</code>
log:DeleteMachineGroup	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/\${machineGroupName}</code>
log:GetMachineGroup	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/\${machineGroupName}</code>
log:ListMachineGroup	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/*</code>
log:ListMachines	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/\${machineGroupName}</code>

config 和 machinegroup 交互类 API

Actions	Resources
log:ApplyConfigToGroup	<code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/\${logtailConfigName}</code> <code>acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/\${machineGroupName}</code>

Actions	Resources
log:RemoveConfigFromGroup	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/\${logtailConfigName} acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/\${machineGroupName}
log:GetAppliedMachineGroups	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/logtailconfig/\${logtailConfigName}
log:GetAppliedConfigs	acs:log:\${regionName}:\${projectOwnerAliUid}:project/\${projectName}/machinegroup/\${machineGroupName}

3.12. 公共资源说明

3.12.1. 数据加密

日志服务支持通过密钥管理服务KMS（Key Management Service）对数据进行加密存储，提供数据静态保护能力。本文主要介绍日志服务的数据加密机制以及使用KMS进行数据加密的操作步骤。

数据加密机制

日志服务通过KMS托管密钥，实现数据加密功能。数据加密机制如下：

- 日志服务支持的数据加密算法为AES算法和国密算法SM4。
- KMS生成和管理您的主密钥CMK（Customer Master Key），并保障密钥的安全性。
- 日志服务支持如下两种加密类型：
 - 通过日志服务的服务密钥加密：日志服务为每个Logstore生成独立的数据加密密钥，用于数据加密。该加密密钥永不过期。
 - 通过用户自带密钥（BYOK）加密：您可以在KMS控制台上创建主密钥CMK，并授权日志服务相应的权限。日志服务调用KMS接口时，使用该CMK创建用于数据加密的密钥。当您的主密钥CMK被删除或禁用后，BYOK密钥失效。

注意

- 仅调用CreateProject API创建Logstore时，可设置数据加密，即可配置加密算法和加密类型。配置后，不支持修改。
- 由KMS BYOK生成的主密钥（CMK）失效后，Logstore上的所有读写请求都会失败。
- 由日志服务的服务密钥生成的主密钥（CMK）不会进行轮转。

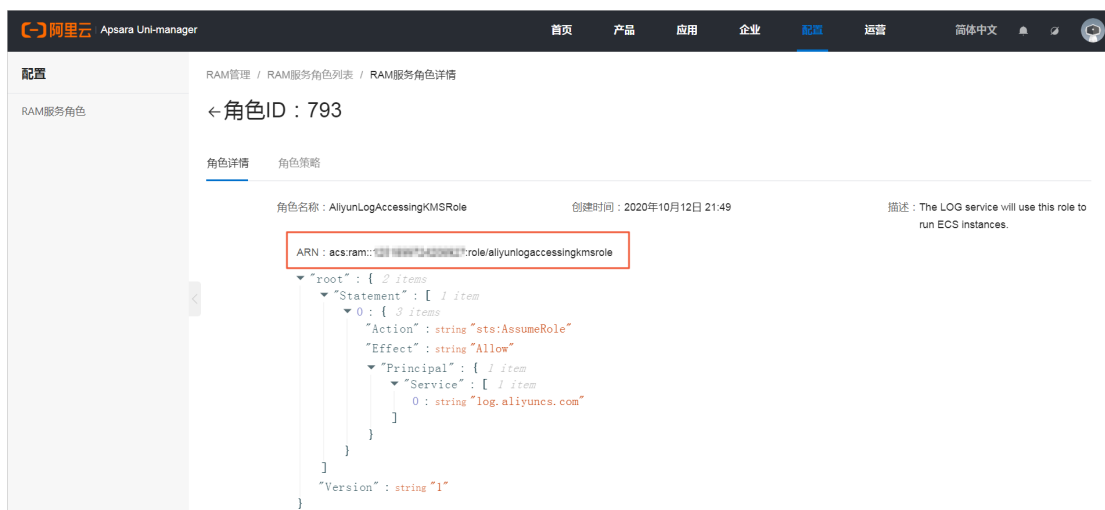
使用日志服务的服务密钥加密数据

调用CreateProject API创建Logstore，添加如下encrypt_conf节点完成加密设置。

```
encrypt_conf = {
    "enable": True,          # 配置为True，启用加密。
    "encrypt_type": "aes_gcm", # 加密算法，支持配置为sm4_gcm或aes_gcm。
}
```

使用BYOK加密数据

1. 在KMS控制台，创建BYOK密钥。
如何创建BYOK密钥，请参见[用户指南中密钥管理服务KMS > 创建密钥](#)章节中的描述。
2. 创建一级组织。
 - i. 登录Apsara Uni-manager控制台。如何登录，请参见[用户指南中Apsara Uni-manager用户指南 > 登录Apsara Uni-manager控制台](#)章节中的描述。
 - ii. 创建组织。如何创建，请参见[用户指南中Apsara Uni-manager用户指南 > 企业中心 > 组织管理 > 创建组织](#)章节中的描述。
3. 为一级组织创建服务角色。
 - i. 在Apsara Uni-manager控制台顶部菜单栏上，单击配置。
 - ii. 在RAM服务角色页面，单击创建RAM服务角色。
 - iii. 在组织下拉列表中，选择您已创建的组织，在服务下拉列表中，选择日志服务，然后单击确定。
 - iv. 在RAM服务角色列表中页面，找到服务角色AliyunLogAccessingKMSRole，单击详情，获取该服务角色的ARN。



4. 调用[CreateProject](#) API创建Logstore，添加如下encrypt_conf节点完成加密设置。

```
encrypt_conf = {
    "enable": True,          # 配置为True，启用加密。
    "encrypt_type": "aes_gcm", # 加密算法，支持配置为sm4_gcm或aes_gcm。
    "user_cmk_info": {
        "cmk_key_id": "",    # BYOK的主密钥ID，即步骤1中创建的密钥ID。
        "arn": "",          # RAM角色的ARN，即步骤3中创建的服务角色的ARN。
        "region_id": "",    # 主密钥所在的地域ID。
    }
}
```

3.12.2. 数据模型

为方便理解整个 Log Service 服务并顺利使用，这里首先介绍其中的几个基本概念。

项目（Project）

项目为 Log Service 中的基本管理单元，用于资源隔离和控制。用户可以通过项目来管理某一个应用的所有日志及相关的日志源。

日志库（Logstore）

日志库为 Log Service 中日志数据的收集、存储和消费单元。每个日志库隶属于一个项目，每个项目可以创建多个日志库。您可以根据实际需求为某一个项目生成多个日志库，其中常见的做法是为一个应用中的每类日志创建一个独立的日志库。例如，假如用户有一个 big-game 游戏应用，服务器上有三种日志：操作日志（operation_log）、应用程序日志（application_log）以及访问日志（access_log），您可以首先创建名为 big-game 的项目，然后在该项目下面为这三种日志创建三个日志库，分别用于它们的收集、存储和消费。

日志（Log）

日志为 Log Service 中处理的最小数据单元。Log Service 采用半结构数据模式定义一条日志，具体数据模型如下：

- 主题（Topic）：用户自定义字段，用以标记一批日志（例如：访问日志根据不同的站点进行标记）。默认该字段为空字符串（空字符串也为一个有效的主题）。
- 时间（Time）：日志中保留字段，用以表示日志产生的时间（精度为秒，从 1970-1-1 00:00:00 UTC 计算起的秒数），一般由日志中的时间直接生成。
- 内容（Content）：用以记录日志的具体内容。内容部分由一个或多个内容项组成，每一个内容项由 Key、Value 对组成。
- 来源（Source）：日志的来源地，例如产生该日志机器的 IP 地址。默认该字段为空。

与此同时，Log Service 对日志各字段的取值有不同要求，具体如下表所示。

日志字段


数据域	要求
time	整型，Unix 标准时间格式，最小单位为秒
topic	任意不超过 128 个字符的 UTF-8 编码字符串
source	任意不超过 128 个字符的 UTF-8 编码字符串
content	一个或多个 Key-Value 对。其中，Key 为仅包含字母、下划线、数字，不以数字开头，不超过 128 个字符的 UTF-8 编码字符串。Value 为不超过 1024*1024 字节的任意 UTF-8 编码字符串

② 说明 上表 content 中的 Key 不可以使用如下关键

字：`__time__`、`__source__`、`__topic__`、`__partition_time__`、`__extract_others__`。

日志主题（Topic）

一个日志库内的日志可以通过日志主题（Topic）来划分。用户可以在写入时指定日志主题。例如，一个平台用户可以使用用户编号作为日志主题写入日志。如果不需要划分一个日志库内的日志，让所有日志使用相同的日志主题即可。

 **说明** 空字符串是一个有效的日志主题（Topic），默认的日志主题都是空字符串。

实际使用场景中，日志的格式多样。为了帮助理解，以下以一条 Nginx 原始访问日志如何映射到 Log Service 日志数据模型为例说明。假设用户 Nginx 服务器的 IP 地址为 10.10.10.1，下面为其上的一条原始日志：

```
10.1.1.1 - - [01/Mar/2012:16:12:07 +0800] "GET /Send?AccessKeyId=<yourAccessKeyId> HTTP/1.1" 200 5 "-" "Mozilla/5.0 (X11; Linux i686 on x86_64; rv:10.0.2) Gecko/20100101 Firefox/10.0.2"
```

把该条原始日志映射到 Log Service 日志数据模型，如下表所示。

日志服务日志数据模型

数据域	内容	说明
topic	""	沿用默认值（空字符串）。
time	1330589527	日志产生的精确时间（精确到秒），从原始日志中的时间戳转换而来。
source	"10.10.10.1"	使用服务器 IP 地址作为日志源。
content	Key-Value 对	日志具体内容。

用户可以自己决定如何提取日志原始内容并组合成 Key-Value 对，如下表所示。

自定义Key-Value对

key	value
ip	"10.1.1.1"
method	"GET"
status	"200"
length	"5"
ref_url	"_ "
browser	"Mozilla/5.0 (X11; Linux i686 on x86_64; rv:10.0.2) Gecko/20100101 Firef

LogGroup

一组日志的集合。

Logs

由多条日志产生的集合。

LogGroupList

一组 LogGroup 集合，用于结果返回。

编码方式


如下表所示，系统目前支持以下内容编码方式（将来可扩展），Restful API 层通过 Content-Type 表示。

支持的编码方式

编码方式	Content-Type	说明
ProtoBuf	ProtoBuf 对数据模型进行编码	application/x-protobuf

以下 PB 定义了数据模型中的对象：

```
message Log
{
    required uint32 Time = 1; // UNIX Time Format
    message Content
    {
        required string Key = 1;
        required string Value = 2;
    }
    repeated Content Contents = 2;
}
message LogGroup
{
    repeated Log Logs = 1;
    optional string Reserved = 2; // reserved fields
    optional string Topic = 3;
    optional string Source = 4;
}
message LogGroupList
{
    repeated LogGroup logGroupList = 1;
}
```

 **说明** 在使用 Protocol Buffer 时要保证 Key-Value 对的唯一性，否则会出现行为未定义的错误。

3.12.3. 数据编码方式

日志服务支持使用 Protocol Buffer 格式作为标准的日志写入格式。

Protocol Buffer 用于结构化数据交换格式，当用户需要写入日志时，需要把原始日志数据序列化成如下格式的 Protocol Buffer 数据流，然后才能通过 API 写入服务端：

```
message Log
{
    required uint32 time = 1; // UNIX Time Format
    message Content
    {
        required string key = 1;
        required string value = 2;
    }
    repeated Content contents= 2;
}
message LogGroup
{
    repeated Log logs= 1;
    optional string reserved =2; // 内部字段，不需要填写
    optional string topic = 3;
    optional string source = 4;
}
message LogGroupList
{
    repeated LogGroup logGroupList = 1;
}
```

说明

- 由于 PB 对 Key-Value 对不要求唯一性，因此需要避免出现该情况，否则行为为未定义。
- 关于 Protocol Buffer 格式的更多信息请参考其 [Github 首页](#)。
- 关于日志服务写入日志的 API 的详细描述，请参考 [PostLog](#)。

3.12.4. 日志库

数据存储单元叫做日志库（logstore），每个 Project 默认可以创建 10 个日志库（Logstore）。Logstore 名称在 Project 下具备唯一性。Logstore 是所有日志数据的入口，可以对日志库进行读写操作。

Logstore 命名规范：

- 只能包括小写字母、数字、连字符（-）和下划线（_）
- 必须以小写字母或者数字开头和结尾
- 长度必须在 2~63 字符以内

完整资源示例：

```
{
  "logstoreName" : "access_log",
  "ttl": 1,
  "shardCount": 2,
  "createTime":1439538649,
  "lastModifyTime":1439538649
}
```

各参数含义如下表所示。

参数含义

参数名称	类型	必须	描述
logstoreName	string	是	Logstore 的名称，在该 Project 下唯一
ttl	integer	是	日志数据生命周期（TTL），单位为天，最小为 1 天
shardCount	integer	是	Logstore 中的分区数
createTime	integer	否	该资源服务端创建时间（输出可见）
lastModifyTime	integer	否	该资源服务端更新时间（输出可见）

3.12.5. 分区

分区（shard）是每个 Logstore 下读写基本单元，每个 Logstore 会指定分区数目。

在向 Shard 读写数据过程中，读必须指定对应的 Shard，而写的过程中可以使用 Load-Balance 方式。Load-Balance 会根据后台系统负载，自动均衡，保证写入高可用性。

完整资源示例如下表所示。

完整资源示例

参数名称	类型	必须	描述
shardID	int	是	Logstore 下 Shard 的唯一 ID，由系统自动生成，类型为整型。

3.12.6. Logtail 配置

Logtail 配置叫做 config，每个 Project 默认可以创建 100 个配置（config）。Config 名称在 Project 下具备唯一性。

您可以通过 config 指定日志收集的位置、方式和参数。

config 命名规范：

- 只能包括小写字母、数字、连字符（-）和下划线（_）
- 必须以小写字母或者数字开头和结尾
- 长度必须在 2~128 字符以内

完整资源示例：

```
{
  "configName": "testcategory1",
  "inputType": "file",
  "inputDetail": {
    "logType": "common_req_log",
    "logPath": "/var/log/httpd/",
    "filePattern": "access.log",
    "localStorage": true,
    "timeFormat": "%Y/%m/%d %H:%M:%S",
    "logBeginRegex": ".*",
    "regex": "(\\w+) (\\s+)",
    "key": ["key1", "key2"],
    "filterKey": ["key1"],
    "filterRegex": ["regex1"],
    "topicFormat": "none"
  },
  "outputType": "sls",
  "outputDetail": {
    {
      "logstoreName": "perfcounter"
    },
  },
  "createTime": 1400123456,
  "lastModifyTime": 1400123456
}
```

配置参数及描述如下表所示。

配置详情

属性名称	类型	是否必须	描述
configName	string	是	日志配置名称， Project 下唯一
inputType	string	是	输入类型，默认为 file
inputDetail	json	是	见下表格说明
outputType	string	是	输出类型，目前只支持 LogService
outputDetail	string	是	见下表格说明
createTime(output-only)	integer	否	配置创建时间
lastModifyTime(output-only)	integer	否	该资源服务端更新时间

inputDetail 类型及描述如下表所示。

inputDetail详情

属性名称	类型	必须	描述
logType	string	是	日志类型，现在只支持 common_reg_log。
logPath	string	是	日志所在的父目录，例如 /var/logs/ 。
filePattern	string	是	日志文件的 Pattern，例如 access*.log 。
localStorage	boolean	是	是否打开本地缓存，在服务端之间链路断开的情况下，本地可以缓存 1GB 日志。
timeFormat	string	是	日志时间格式，如 %Y/%m/%d %H:%M:%S 。
logBeginRegex	string	是	日志首行特征（正则表达式），由于匹配多行日志组成一条 log 的情况。
regex	string	是	日志对提取正则表达式。
key	array	是	日志提取后所生成的 Key。
filterKey	array	是	用于过滤日志所用到的 Key，只有 Key 的值满足对应 filterRegex 列中设定的正则表达式日志才是符合要求的。
filterRegex	array	是	每个 filterKey 对应的正正则表达式，filterRegex 的长度和 filterKey 的长度必须相同。
topicFormat	string	否	用于将日志文件路径的某部分作为 topic，如 /var/log/(.*) .log ，默认为 none，表示 topic 为空。
preserve	boolean	否	true 代表监控目录永不超时，false 代表监控目录 30 分钟超时，默认值为 true。
preserveDepth	integer	否	当设置 preserve 为 false 时，指定监控不超时目录的深度，最大深度支持 3。

outputDetail 类型及描述如下表所示。

outputDetail详情

属性名称	类型	必须	描述
logstoreName	string	是	对应 Logstore 名称

3.12.7. Logtail 机器组

机器：当机器安装 Logtail 并正常启动后，会根据 Logtail 配置中的用户信息自动关联到当前用户。目前 machine 有三种标示的方式，分别为：

- IP: hostname 对应 IP 地址。最容易理解，但在 VPC 等环境下可能会有重复。
- UUID (machine-uniqueid): DMI 设备中 UUID，参见 [RFC4122](#)。
- Userdefined-id: 用户在 Logtail 目录下自定义该机器标识。

每台机器属性如下：

```
{
  "ip" : "testip1",
  "machine-uniqueid" : "testuuid1",
  "userdefined-id" : "testuserdefinedid1",
  "lastHeartbeatTime":1397781420
}
```

机器属性参数说明如下表所示。

参数说明

参数名称	类型	描述
ip	string	机器 hostname 对应 IP 地址
uuid	string	机器标示的唯一主键，由 Logtail 上传
userdefined-id	string	用户自定义机器标示，由 Logtail 上传
lastHeartbeatTime(output-only)	integer	机器的最后心跳时间（从 epoch 时间开始的秒数）

machinegroup

project 下当前用户拥有的机器分组。机器分组可以通过两种方式来标示（IP 与 userdefined）。IP 较为容易辨识，userdefined 可以解决 VPC 下 IP 相同的问题，用户可以选任意一种方式进行机器标识。

machinegroup 命名规范：

- 只能包括小写字母、数字、连字符（-）和下划线（_）
- 必须以小写字母或者数字开头和结尾
- 长度必须在 2~128 字节以内

完整资源示例

```
{
  "groupName" : "testgroup",
  "groupType" : "",
  "groupAttribute" : {
    "externalName" : "testgroup",
    "groupTopic": "testgrouptopic"
  },
  "machineIdentifyType": "ip",
  "machineList" : [
    "ip1",
    "ip2"
    ...
  ],
  "createTime": 1431705075,
  "lastModifyTime" : 1431705075
}
```

machinegroup参数如下表所示。

machinegroup参数说明

属性名称	类型	必须	描述
groupName	string	是	机器分组名称， Project 下唯一
groupType	string	否	机器分组类型，默认为空
machineIdentifyType	string	是	机器标识类型，分为 IP 和 userdefined 两种
groupAttribute	object	是	机器分组的属性，默认为空
machineList	array	是	具体的机器标识，可以是 IP 或 userdefined-id
createTime(output-only)	int	否	该资源创建时间
lastModifyTime(output-only)	int	否	该资源服务端更新时间

groupAttribute 类型及描述如下表所示。

groupAttribute 详情

属性名称	类型	是否必须	描述
groupTopic	string	否	机器分组的 Topic，默认为空
externalName	string	否	机器分组所依赖的外部管理标识，默认为空