日志服务

API 参考

API 参考

概览

日志服务（Log Service，简称 LOG）是针对日志平台化服务。服务提供各种类型日志的实时收集、存储和分发。除此之外，LOG 有 ODPS Table 间同步服务，通过 LOG 可以将日志投递至 ODPS 做大数据分析。

除了通过 **管理控制台** 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。目前开放如下 API：

对象	方法	
Log（日志）	日志、日志组表示等基本概念	
Config （配置）	List、Create、 Delete、Get、Update	
	GetAppliedMachineGroups（查询应用到的机器组）	
MachineGroup （机器组）	List、Create、Delete、Get、Update	
	Apply/Remove（应用/删除配置）	
	GetAppliedConfigs（查询已应用配置列表）	
LogStore （日志库）	List、Create、 Delete、Get、Update	
	GetLogs（查询日志）、GetHistograms（查询日志分布）	
Shard (分区)	List、Split、Merge、Delete	
	PostLogStoreLogs（写入日志）	
	GetCursor（定位日志位置）	
	PullLogs（消费日志）	
Shipper（日志投递规则）	GetShipperStatus（查询日志投递任务状态）	
	RetryShipperTask（重试失败投递任务）	

通过 API 可以操作下列服务：

- 根据 配置，机器组 信息收集日志
- 创建 日志库、向日志库写入、读取日志
- 对不同用户进行 访问控制

说明：

- API 目前提供 Rest 风格。
- 为使用 API，需要知道 API访问地址。
- API 所有请求都需要做安全验证，请参考 请求签名 解释了具体的 API 请求签名机制及流程。
- Log Service 支持 RAM、STS，RAM 子用户使用 API，和一般云账号没有区别，使用子用户的 AK 签名即可。STS 临时身份除了临时 AK 外，还需要填写一个特殊的 HTTP header，详见 文档，这个 HTTP header 需要参与签名，详见 文档。

服务入口

公网服务入口

日志服务入口是访问一个项目（Project）及其内部日志数据的 URL。它和 Project 所在的阿里云区域（Region）及 Project 名称相关。目前，日志服务已经在多个阿里云 Region 下开通，在各 Region 内的公网服务入口如下：

地域	服务入口
华东 1 (杭州)	cn-hangzhou.log.aliyuncs.com
华东 2 (上海)	cn-shanghai.log.aliyuncs.com
华东 1 (杭州-金融云, 不支持公网访问)	cn-hangzhou-finance-intranet.log.aliyuncs.com
华北 1 (青岛)	cn-qingdao.log.aliyuncs.com
华北 2 (北京)	cn-beijing.log.aliyuncs.com
华南 1 (深圳)	cn-shenzhen.log.aliyuncs.com

当访问某个具体的 Project 时，需要根据 Project 名称及其所在 Region 组合出最终访问地址。具体格式如下：

```
<project_name>.<region_endpoint>
```

例如，Project 名为 big-game，所在区域为“华东 1 (杭州)”，则对应访问地址如下：

```
big-game.cn-hangzhou.log.aliyuncs.com
```

在创建日志服务项目时需要指定某个 Region。一旦在创建时指定了 Region，该设置就不可以更改，且无法跨区域迁移项目。创建 Project 之后，必须选择与其所在区域相匹配的根服务入口地址来组成该 Project 访问地址，用做 API 请求的服务入口。

经典网络服务入口

如果在阿里云的 ECS 机器内使用日志服务 API，还可以使用内网服务入口（使用内网服务入口访问日志服务不消耗 ECS 公网流量，可以节约宝贵的 ECS 公网带宽），各个 Region 的日志服务内网根服务入口如下：

地域	根服务入口
华东 1 (杭州)	cn-hangzhou-intranet.log.aliyuncs.com
华东 2 (上海)	cn-shanghai-intranet.log.aliyuncs.com
华东 1 (杭州-金融云)	cn-hangzhou-finance-intranet.log.aliyuncs.com
华北 1 (青岛)	cn-qingdao-intranet.log.aliyuncs.com
华北 2 (北京)	cn-beijing-intranet.log.aliyuncs.com
华南 1 (深圳)	cn-shenzhen-intranet.log.aliyuncs.com
华南 1 (深圳-金融云)	无经典网络虚拟机，统一使用 VPC 网络服务入口
香港	cn-hongkong-intranet.log.aliyuncs.com
美国西部 1 (硅谷)	us-west-1-intranet.log.aliyuncs.com

如上例，其内网访问地址如下：

```
big-game.cn-hangzhou-intranet.log.aliyuncs.com
```

VPC 网络服务入口

如果使用的 ECS 机器为 VPC 网络，可以使用日志服务 VPC 区域服务入口，各个 Region 的日志服务 VPC 根服务入口如下：

地域	根服务入口
华东 1 (杭州)	cn-hangzhou-vpc.log.aliyuncs.com
华东 2 (上海)	cn-shanghai-vpc.log.aliyuncs.com
华北 1 (青岛)	cn-qingdao-vpc.log.aliyuncs.com
华北 2 (北京)	cn-beijing-vpc.log.aliyuncs.com

华北 3 (张家口)	cn-zhangjiakou-vpc.log.aliyuncs.com
华南 1 (深圳)	cn-shenzhen-vpc.log.aliyuncs.com
华南 1 (深圳-金融云)	cn-shenzhen-finance-vpc.log.aliyuncs.com
香港	cn-hongkong-vpc.log.aliyuncs.com
亚太东北 1 (东京)	ap-northeast-1-vpc.log.aliyuncs.com
亚太东南 1 (新加坡)	ap-southeast-1-vpc.log.aliyuncs.com
亚太东南 2 (悉尼)	ap-southeast-2-vpc.log.aliyuncs.com
中东东部 1 (迪拜)	me-east-1-vpc.log.aliyuncs.com
美国西部 1 (硅谷)	us-west-1-vpc.log.aliyuncs.com
欧洲中部 1 (法兰克福)	eu-central-1-vpc.log.aliyuncs.com

如上例，其 VPC 网络访问地址如下：

```
big-game.cn-hangzhou-vpc.log.aliyuncs.com
```

目前，日志服务 API 服务在如上服务入口上仅支持 HTTP 协议。

访问密钥

阿里云访问秘钥是阿里云为用户使用 API（非控制台）来访问其云资源设计的“安全口令”。您可以用它来签名 API 请求内容以通过服务端的安全验证。

该访问秘钥成对（AccessKeyId 与 AccessKeySecret）生成和使用。每个阿里云用户可以创建多对访问秘钥，且可随时启用（Active）、禁用（Inactive）或者删除已经生成的访问秘钥对。

您可以通过阿里云控制台的 **秘钥管理页面** 创建、管理所有的访问秘钥对。由于访问秘钥是阿里云对 API 请求进行安全验证的关键因子，请妥善保管你的访问秘钥。如果某些秘钥对出现泄漏风险，建议及时删除该秘钥对并生成新的替代秘钥对。

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），其详细定义如下：

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，更多细节请参考 **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，则不需要提供该请求头。
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，则不需要提供该请求头。
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，则不需要提供该请求头。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：%a, %d %b %Y %H:%M:%S GMT (如：Mon, 3 Jan 2010 08:33:47 GMT)。
Host	字符串	HTTP 请求的完整 HOST 名字（不包括如 http:// 这样的协议头）。例如，big-game.cn-hangzhou.sls.aliyuncs.com。
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-compresstype	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（RFC 1951，使用 zlib 格式，参考 RFC 1950）。如果不压缩可以不提供该请求头。
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（该字段不参与签名）。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 hmac-sha1。
x-acs-security-token	字符串	使用 STS 临时身份发送数据。当使用 STS 临时身份时必填，其他情况不要填写。

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-compresstype 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。

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，其详细定义如下：

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，为大写字符串。
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 (如：Mon, 3 Jan 2010 08:33:47 GMT）。
x-log-requestid	字符串	服务端产生的标示该请求的唯一 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。

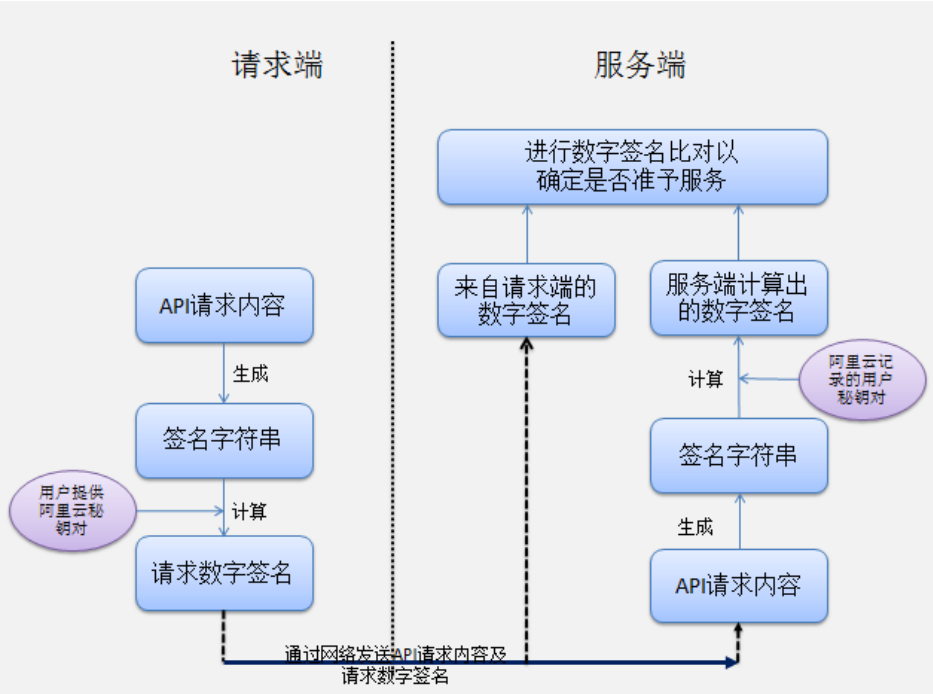
请求签名

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，该安全验证基于阿里云的访问秘钥，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. 请求端使用阿里云的访问秘钥对（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（**注意：**服务端会在后台取得该请求使用的用户访问秘钥对）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。

上面的整个流程也可以使用下图直观描述：



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秘钥对，在服务端即可通过该秘钥对确定用户身份，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，用户需要在客户端对其 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秘钥对中的 AccessKeyId，且与之对应的 AccessKeySecret 将用于 Signature 值的构造。下面将详细解释如何构造该 Signature 值。

第一步：准备合适的阿里云访问秘钥

如上所述，给 API 请求生成签名，需使用一对访问秘钥（AccessKeyId/AccessKeySecret)。您可以使用已经存在的访问秘钥对，也可以创建新的访问秘钥对，但需要保证使用的秘钥对处在“启用”状态。

第二步：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：

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（必须为大写字母串）	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 x-log 和 x-acs 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：

1. 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；
2. 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；
3. 删除请求头和内容之间分隔符两端出现的任何空格；
4. 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：

1. 将 CanonicalizedResource 设置为空字符串（” ”）；
2. 放入要访问的 LOG 资源,如 /logstores/logstorename（无 logstorename 则不填）；

3. 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"
```

第三步：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。其整个签名公式如下：

```
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 签名的访问秘钥对如下：

```
AccessKeyId = "bq2sjzesjmo86kq35behupbq"
AccessKeySecret = "4fdO2fTDDnZPU/L7CHNdemB2Nsk="
```

示例一：

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.cn-hangzhou-devcommon-intranet.sls.aliyuncs.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.cn-hangzhou-devcommon-intranet.sls.aliyuncs.com
x-log-apiversion: 0.6.0
x-log-signaturemethod: hmac-sha1
Authorization: LOG bq2sjzesjmo86kq35behupbq:jEYOTCJs2e88o+y5F4/S5IsnBJQ=
```

示例二：

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="10.230.201.117"
"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.cn-hangzhou-devcommon-intranet.sls.aliyuncs.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（请参考 ProtoBuffer格式 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-signaturemethod: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.cn-hangzhou-devcommon-intranet.sls.aliyuncs.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 bq2sjzesjmo86kq35behupbq:XWLGYHGg2F2hcfxWxMLiNkGki6g=

<日志内容序列化成 ProtoBuffer 格式的字节流>
```

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>
}
```

在所有服务端可能返回的错误信息中，一部分适用于多数 API，而另外一部分则为某些 API所独有。下表即为 API 响应中的通用错误码，它们会在多个 API 响应中出现。而每个 API 所独有的错误码会在该 API 参考中单独描述。

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。
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 heade.	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,	服务器正忙，请稍后再

		please try again later.	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。

日志库相关接口

CreateLogstore

在 project 下创建 logstore。

示例：

POST /logstores

请求语法

```
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>
}
```

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。
shardCount	integer	是	该 logstore 的 shard 数量

请求头

CreateLogstore 接口无特有请求头，关于 Log Service API 的公共请求头请参考 公共请求头。

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	LogstoreAlreadyExist	logstore {logstoreName} already exist
500	InternalServerError	Specified Server Error Message
400	LogstoreInfoInvalid	logstore info is invalid
400	ProjectQuotaExceed	Project Quota Exceed

细节描述

创建过程中 quota 如果非法，则会创建不成功。

示例

请求示例：

```
POST /logstores HTTP/1.1
Header :
{
x-log-apiversion=0.6.0,
Authorization=LOG 94to3z418yupi6ikawqqd370:8IwDTWugRK1AZAo0dWQYpffhy48=,
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
}
```

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
}
```

DeleteLogstore

删除 logstore，包括所有 shard 数据，以及索引等。

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

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。

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。

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} 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 94to3z418yupi6ikawqqd370:fPsNBIuJR1xvQZolwi8+Cw5R/fQ=,
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
Body:
{
Date=Wed, 11 Nov 2015 08:09:39 GMT,
Content-Length=0,
x-log-requestid=5642F7C399248C817B013A07,
Connection=close,
Server=nginx/1.6.1
}
```

UpdateLogstore

更新 logstore 的属性。目前只支持更新 TTL，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>
}
```

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1~365（额外需求请提交工单）
shardCount	integer	是	shard 个数，单位为个，范围为 1~10。

请求头

UpdateLogstore 接口无特有请求头。关于 Log Service API 的公共请求头，请参考 公共请求头。

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	Project {ProjectName} not exist
404	LogStoreNotExist	logstore {logstoreName} not exist

400	LogStoreAlreadyExist	logstore {logstoreName} already exist
500	InternalServerError	Specified Server Error Message
400	ParameterInvalid	invalid shard count,you can only increase the count

细节描述

shard 的数量目前只能增加不能减少。

示例

请求示例：

```
PUT /logstores/test-logstore HTTP/1.1
Header:
{
x-log-apiversion=0.6.0,
Authorization=LOG 94to3z418yupi6ikawqqd370:wFcl3ohVJupCi0ZFxRD0x4IA68A=,
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
}
```

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
}
```

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。

请求头

GetLogstore 接口无特有请求头。关于 Log Service API 的公共请求头，请参考 公共请求头。

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} not exist

404	LogstoreNotExist	logstore {logstoreName} 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 94to3z418yupi6ikawqqd370:6ga/Cvj51rFatX/DtTkcQB/CALk=,
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

}
```

ListLogstore

ListLogstores 接口列出指定 project 下的所有 logstore 的名称。

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
```

请求参数

参数名称	类型	是否必须	描述
offset(optional)	integer	否	返回记录的起始位置，默认值为 1。
size(optional)	integer	否	每页返回最大条目，默认 500（最大值）。
logstoreName	string	否	用于过滤的 logstore 名称（支持部分匹配）。

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。

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

ListLogStores 请求成功，其响应 Body 会包括指定 Project 下的所有 logstore 名称列表，具体格式如下：

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} 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 94to3z418yupi6ikawqqd370:we34Siz/SBVyVGMGmMDnvp0xSPo=,
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
}
```

ListShards

列出 logstore 下当前所有可用 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
```

请求参数

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

请求头

无特有请求头。关于 API 的公共请求头，请参考 **公共请求头**。

响应头

Content-Type: application/json

无特有响应头。关于 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
},
{
...
},
{
...
}
]
```

细节描述

N/A

特有错误码

除了返回 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} not exist
500	InternalServerError	Specified Server Error Message
400	LogStoreWithoutShard	logstore has no shard

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 94to3z418yupi6ikawqqd370:xEOsJ3xeivcRq0GbvACiO37jH0I="
}
```

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
}

]
```

SplitShard

分裂一个指定的 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

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

Content-Type: application/json

无特有响应头。关于 API 的公共响应头，请参考 公共响应头。

响应元素

3 个 shard 元素组成的数组，第一个 shard 为 split 之前的 shard，后两个为 split 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
}
]
```

细节描述

N/A

特有错误码

除了返回 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} not exist
400	ParameterInvalid	invalid shard id
400	ParameterInvalid	invalid mid hash
500	InternalServerError	Specified Server Error Message
400	LogStoreWithoutShard	logstore has no shard

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 94to3z418yupi6ikawqqd370:xEOsJ3xeidfdgRq0GbvACiO37jH0I="
}
```

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
}
]
```

MergeShards

合并两个相邻的 readwrite 状态的 shards。在参数中指定一个 shardid，服务端自动找相邻的下一个 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

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

Content-Type: application/json

无特有响应头。关于 API 的公共响应头，请参考 公共响应头。

响应元素

3 个 shard 元素组成的数组，第一个 shard 为 merge 之后的 shard，后两个为 merge 之前的 shard。

```
[
{
'shardID': 167,
'status': 'readwrite',
'inclusiveBeginKey': 'e000000000000000000000000000000',
'createTime': 1453953105,
'exclusiveEndKey': 'ffffffffffffffffffffffffffffffff'
},
{
'shardID': 30,
'status': 'readonly',
```

```
'inclusiveBeginKey': 'e00000000000000000000000000000000',
'createTime': 0,
'exclusiveEndKey':
'e70000000000000000000000000000000'
},
{
'shardID': 166,
'status': 'readonly',
'inclusiveBeginKey': 'e70000000000000000000000000000000',
'createTime': 1453953073,
'exclusiveEndKey': 'fffffffffffffffffffffffffffffffff'
}
]
```

细节描述

N/A

特有错误码

除了返回 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} 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

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 94to3z418yupi6ikawqqd370:xEOsJ3xeidfdgRq0GbvACiO37jH0I="
}
```

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': 'e00000000000000000000000000000000',
'createTime': 1453953105,
'exclusiveEndKey': 'ffffffffffffffffffffffffffffffff'
},
{
'shardID': 30,
'status': 'readonly',
'inclusiveBeginKey': 'e00000000000000000000000000000000',
'createTime': 0,
'exclusiveEndKey':
'e70000000000000000000000000000000'
},
{
'shardID': 166,
'status': 'readonly',
'inclusiveBeginKey': 'e70000000000000000000000000000000',
'createTime': 1453953073,
'exclusiveEndKey': 'ffffffffffffffffffffffffffffffff'
}
]
```

DeleteShard

删除一个 readonly 状态的 shard。

请求语法

```
DELETE /logstores/<logstorename>/shards/<shardid> 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

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

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

Content-Type: application/json

无特有响应头。关于 API 的公共响应头，请参考 公共响应头。

细节描述

N/A

特有错误码

除了返回 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} not exist
400	ParameterInvalid	invalid shard id
400	ParameterInvalid	only readonly shard id can be deleted
500	InternalServerError	Specified Server Error Message
400	LogStoreWithoutShard	logstore has no shard

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

示例

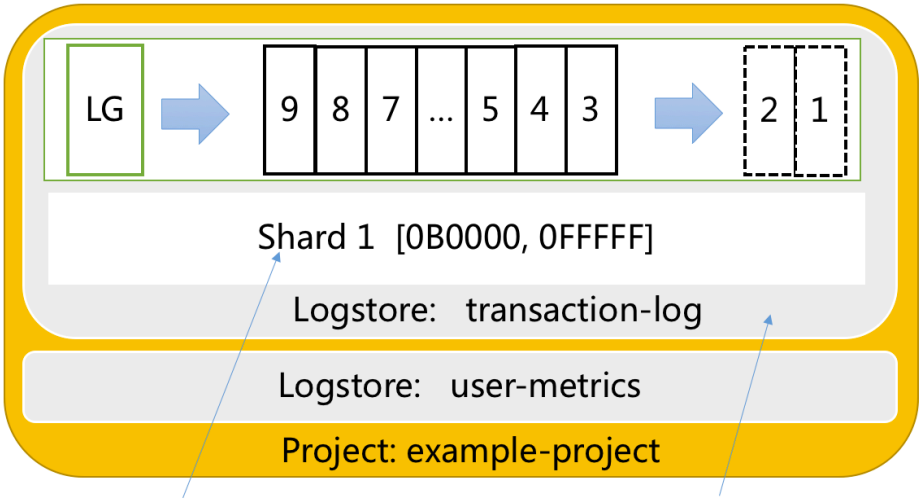
请求示例：

```
DELETE /logstores/logstorename/shards/30
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 94to3z418yupi6ikawqqd370:xEOsJ3xeidfdgRq0GbvACiO37jH0I="
}
```

GetCursor

GetCursor 根据时间获得游标（cursor），下图表示 project、logStore、shard 与 cursor 的关系：

- 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 /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
```

请求参数

参数名称	类型	是否必须	描述

shard	string	是	
type	string	是	此处为 cursor
from	string	是	时间点（UNIX下秒数），或 begin，end

logstore 生命周期：

logstore 生命周期由属性中 lifeCycle 字段指定。例如，当前时间为 2015-11-11 09:00:00，lifeCycle=24。则每个 shard 中可以消费的数据时间段为 [2015-11-10 09:00:00,2015-11-11 09:00:00)，这里的时间指的是 Server 端时间。

通过 from 关可以在 shard 中定位生命周期内的日志，假设 logstore 生命周期为 [begin_time,end_time)，from=from_time

```
 from_time <= begin_time or from_time == "begin" : 返回时间点为 begin_time 对应的 cursor 位置
from_time >= end_time or from_time == "end" : 返回当前时间点下，下一条将被写入的 cursor 位置(当前该 cursor 位置上无数据)
from_time > begin_time and from_time < end_time : 返回第一个服务端接收时间 >= from_time 的数据包对应的 cursor
```

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 API 的公共响应头，请参考 公共响应头。

响应元素

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

细节描述

N/A

错误码

除了返回 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} not exist
400	ParameterInvalid	Parameter From is not valid
400	ShardNotExist	Shard {ShardID} 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 94to3z418yupi6ikawqqd370:+vo0Td6PrN0CGoskJoOiAsnkXgA="
}
```

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=="
}
```

PullLogs

根据游标、数量获得日志。获得日志时必须指定 shard。如果在 storm 等情况下可以通过 LoghubClientLib 进行选举与协同消费。目前仅支持读取 [PB 格式 LogGroupList] 数据。

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

URL 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 0~1000

请求头

- Accept: application/x-protobuf
- Accept-Encoding: lz4 或 deflate 或 “”

关于 API 的公共请求头，请参考 **公共请求头**。

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

关于 API 的公共响应头，请参考 **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）。

细节描述

N/A

特有错误码

除了返回 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} not exist
400	ParameterInvalid	Parameter Cursor is not valid
400	ParameterInvalid	ParameterCount should be in [0-1000]
400	ShardNotExist	Shard {ShardID} 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 94to3z418yupi6ikawqqd370:WeMYZp6bH/SmWEgryMrLhbxK+7o=",
"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 loggrouplist 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。

PostLogStoreLogs

向指定的 logStore 写入日志数据。目前仅支持写入 PB 格式 LogGroup 日志数据。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。该方法对写入可用性较高（SLA: 99.95%），适合写入与消费数据与 shard 无关的场景，例如不保序。
- 根据 Key 路由 shard 模式（KeyHash）：写入时需要传递一个 Key，服务端自动根据 Key 选择当前符合该 Key 区间的 shard 写入。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，参见 shard 数据模型）。

请求语法

负载均衡写入模式

```
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>
```

根据 Key 路由 shard 模式

在 header 中增加 x-log-hashkey，用来判断落在哪个 shard 的 range 中。该参数为可选参数，不填情况下自动切换负载均衡写模式。

```
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-hashkey : 14d2f850ad6ea48e46e4547edbbb27e0
x-log-signaturemethod: hmac-sha1

<PB 格式日志压缩数据>
```

请求参数

名称	类型	必选	描述
logstorename	字符串	是	需要写入日志的 logstore 名称。

请求头

根据 Key 路由 shard 模式下需要增加 x-log-hashkey 请求头（参见上述示例）。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 API 的公共响应头，请参考 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数据量上限为 3MB 或者 4096 条。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（具体日志格式要求请参考 核心概念，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。
- 服务端会对每次 PutLogs 写入的日志数据时间戳做检查。目前只接受日志数据时间戳在服务端当前处理时间前后 [-7x24小时, +15分钟] 小时范围内的日志。如果日志数据内有任何一条日志的时间戳不在该时间范围内，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 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。

400	InvalidEncoding	Non-UTF8 charactors 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 3M and 4096 lines.	日志内容包含的日志必须小于 3MB 和 4096 条。
400	PostBodyUncompressError	Body is uncompressed fail.	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} not exist.	日志库（logstore）不存在。

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 94to3z418yupi6ikawqqd370:zLyKtgyGpwyv7ntXZs2dY2wWIg4="
}

<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"
```

```
}
```

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=success&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

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 success/fail/running 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
}
]
}
```

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：

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

tasks 内容：

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

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} not exist
404	LogStoreNotExist	logstore {logstoreName} not exist
400	ShipperNotExist	shipper {logstoreName} not exist
500	InternalServerError	internal server error
400	ParameterInvalid	start time must litter than end time
400	ParameterInvalid	only support query last 48 hours task status
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。

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 94to3z418yupi6ikawqqd370:wFcl3ohVJupCi0ZFxRD0x4IA68A=,
```

```
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
}
```

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
}
]
}
```

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

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

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} not exist
404	LogStoreNotExist	logstore {logstoreName} not exist
400	ShipperNotExist	shipper {logstoreName} not exist
500	InternalServerError	Specified Server Error Message
400	ParameterInvalid	only allow retry 10 task 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 94to3z418yupi6ikawqqd370:wFcl3ohVJupCi0ZFxRD0x4IA68A=,
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 :
["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
}
```

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

```
GET
/logstores/<logstorename>?type=histogram&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
```

请求参数

名称	类型	必选	描述
logstorename	字符串	是	需要查询日志的 logstore 名称。
type	字符串	是	查询 logstore 数据的类型。在 GetLogs 接口中该参数必须为 log。
from	整型	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	整型	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	字符串	否	查询日志主题。
query	字符串	否	查询表达式。关于查询表达式的详细语法，请参考 查询语法。
line	整型	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	整型	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	布尔型	否	是否按日志时间戳逆序返回日志。true 表示逆序，false 表示顺序

			，默认值为 false。

请求头

GetLogs 接口无特有请求头。关于日志服务 API 的公共请求头，请参考 **公共请求头**。

响应头

GetLogs 接口无特有响应头。关于日志服务 API 的公共响应头，请参考 **公共响应头**。

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的时候，该接口的响应结果可能并不完整，所以响应元素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：

名称	类型	描述
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。
count	整型	当前查询结果的日志总数。
logs	数组	当前查询结果的日志原始数据，具体结构见下面的描述。

上表中的 logs 数组中的每个元素结构如下：

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 计算起的秒数）。
__source__	字符串	日志的来源，由写入日志时指定。
[content]	Key-Value对	日志原始内容，以 Key-value 对的形式组织。

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。
- 如上所述，该接口一次调用必须要在限定时间内返回结果，每次查询只能扫描指定条数的日志量。如果一次请求需要处理的数据量非常大的时候，该请求会返回不完整的结果（并在返回结果中的 progress 成员标示是否完整）。如此同时，服务端会缓存 15 分钟内的查询结果。当查询请求的结果有部分被缓存命中，则服务端会在这次请求中继续扫描未被缓存命中的日志数据。为了减少您合并多次查询结果的工作量，服务端会把缓存命中的查询结果与本次查询新命中的结果合并返回给您。因此

，日志服务可以让您通过以相同参数反复调用该接口来获取最终完整结果。因为您的查询涉及的日志数据量变化非常大，日志服务 API 无法预测需要调用多少次该接口而获取完整结果。所以需要用户通过检查每次请求的返回结果中的progress 成员状态值来确定是否需要继续。需要注意的是，每次重复调用该接口都会重新消耗相同数量的查询 CU。

特有错误码

GetLogs 接口除了可能返回日志服务 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} 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 invalided	请求的查询字符串无效。
400	InvalidOffset	offset is invalided	请求的 offset 参数无效。
400	InvalidLine	line is invalided	请求的 line 参数无效。
400	InvalidReverse	Reverse value is invalid	Reverse 参数的值无效。

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

{
"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，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 progress 成员的状态为 Incomplete，则需要重复相同请求以获得完整结果。

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。

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延时为3秒。

请求语法

```
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
```

请求参数

名称	类型	必选	描述
logstorename	字符串	是	需要查询日志的 logstore 名称。
type	字符串	是	查询 logstore 数据的类型，在 GetHistograms 接口中该参数必须为 histogram。
from	整型	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	整型	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	字符串	否	查询日志主题。
query	字符串	否	查询表达式。关于查询表达式的详细语法，请参考 查询语法。

请求头

GetHistograms 接口无特有请求头。关于日志服务 API 的公共请求头，请参考 公共请求头。

响应头

GetHistograms 接口无特有响应头。关于日志服务 API 的公共响应头，请参考 公共响应头。

响应元素

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

查询的时间范围进行均匀分割成若干个（1 到 60 个不等）子时间区间，并返回每个子时间区间内命中的日志条数。由于日志服务需要在限定时间内返回结果以保证实时性，每次查询只能扫描指定条数的日志量。当需要查询日志数据量非常大时，该接口的响应结果可能并不完整。所以响应元素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：

名称	类型	描述
progress	字符串	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	整型	当前查询结果中所有日志总数。
histograms	数组	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	整型	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	整型	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	整型	当前查询结果在该子时间区间内命中的日志条数
progress	字符串	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。
- 如上所述，该接口一次调用必须要在限定时间内返回结果，每次查询只能扫描指定条数的日志量。如果一次请求需要处理的数据量非常大的时候，该请求会返回不完整的结果（并在返回结果中的 progress 成员标示是否完整）。如此同时，服务端会缓存 15 分钟内的查询结果。当查询请求的结果有部分被缓存命中，则服务端会在这次请求中继续扫描未被缓存命中的日志数据。为了减少您合并多次查询结果的工作量，服务端会把缓存命中的查询结果与本次查询新命中的结果合并返回给您。因此，日志服务可以让您通过以相同参数反复调用该接口来获取最终完整结果。因为您的查询涉及的日志数据量变化非常大，日志服务 API 无法预测需要调用多少次该接口而获取完整结果。所以需要您通过检查每次请求的返回结果中的 progress 成员状态值来确定是否需要继续。需要注意的是，每次重复

调用该接口都会重新消耗相同数量的查询 CU。

特有错误码

GetHistograms 接口除了可能返回日志服务 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} 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 invalided	请求的查询字符串无效。

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: <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
```

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

{
"progress": "Incomplete",
"count": 3,
"histograms": [
```

```
{
"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 （ 例如下响应 ）：

```
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

{
"progress": "Incomplete",
"count": 4,
"histograms": [
{
"from": 1409529600,
"to": 1409569200,
"count": 2,
"progress": "Complete"
},
{
"from": 1409569200,
"to": 1409608800,
"count": 2,
"progress": "complete"
}
]
}
```

Logtail机器组相关接口

CreateMachineGroup

您可以根据需求创建一组机器，用以日志收集下发配置。

示例：

POST /machinegroups

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：

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

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**。

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

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	GroupAlreadyExist	group {GroupName} already exist
400	InvalidParameter	invalid group resource json
500	InternalServerError	Internal server error

细节描述

无

示例

请求示例：

```
POST /machinegroups HTTP/1.1
Header :
{
"x-log-apiversion": "0.6.0",
"Authorization": "LOG 94to3z418yupi6ikawqqd370:aws39CB5OUyx39BjQ5bW3G/zBv4=",
"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"
}
```

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。

示例：

DELETE /machinegroups/{groupName}

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

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。

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

HTTP 状态码返回 200。

错误码

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

细节描述

无

示例

请求示例：

```
DELETE /machinegroups/test-machine-group-4 HTTP/1.1
Header :
{
"x-log-apiversion": "0.6.0",
"Authorization": "LOG 94to3z418yupi6ikawqqd370:JjQpxvfnkTYPsZIgicQ+IOkufI8=",
"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"
}
```

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。

示例：

PUT /machinegroups/{groupName}

```
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

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：

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

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**。

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

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} not exist
400	InvalidParameter	invalid group resource json
500	InternalServerError	internal server error

细节描述

无

示例

请求示例：

```
PUT /machinegroups/test-machine-group HTTP/1.1
Header :
{
"x-log-apiversion": "0.6.0",
"Authorization": "LOG 94to3z418yupi6ikawqqd370:ZJmBDS+LjRCzgSLuo21vFh6o7CE=",
"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"
}
```

ListMachineGroup

示例：

```
GET /machinegroups?offset=1&size=100
```

```
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
```

请求参数

URL 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）

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。

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

请求成功，其响应 Body 会包括指定 Project 下的所有 machinegroup 名称列表。具体格式如下：

名称	类型	描述
count	int	返回的 machinegroup 数目
total	int	返回 machinegroup 总数
machinegroups	json array	返回的 machinegroup 名称列表

```
{
"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，还可能返回如下特有错误码：

HTTP 状态码	ErrorCode	ErrorMessage
500	InternalServerError	internal server error

细节描述

无

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 94to3z418yupi6ikawqqd370:XN5helROz9QYV0FKhElSNuTBysA=",
"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
}
```

GetMachineGroup

查看具体的 machinegroup 信息。

示例：

```
GET /machinegroups/{groupName}
```

```
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

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**。

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

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

```
{
"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
}
```

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} not exist
500	InternalServerError	internal server error

细节描述

无

示例

请求示例：

```
GET /machinegroups/test-machine-group HTTP/1.1
Header :
{
"x-log-apiversion": "0.6.0",
"Authorization": "LOG 94to3z418yupi6ikawqqd370:CNQaXNeExV6S/nQZkP/R+baZPZc=",
"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
}
```

ApplyConfigToMachineGroup

将配置应用到机器组。

示例：

```
PUT /machinegroups/{GroupName}/configs/{ConfigName}
```

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

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 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 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} not exist
404	ConfigNotExist	config {ConfigName} 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 94to3z418yupi6ikawqqd370:skTdJCZXn8QPGNB2jL9k6u1xO1E="
}
```

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"
}
```

RemoveConfigFromMachineGroup

从机器组中删除配置。

示例：

DELETE /machinegroups/{GroupName}/configs/{ConfigName}

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

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 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 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} not exist
404	ConfigNotExist	config {ConfigName} 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 94to3z418yupi6ikawqqd370:t8v8y+zqOz3ZiqLDIkb6JQ8FUAU="
}
```

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"
}
```

ListMachines

获得 machinegroup 下属于用户并与 Server 端连接的机器状态信息。

示例：

```
GET /machinegroups/{groupName}/machines?offset=1&size=10
```

```
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
```

请求参数

URL 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）

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。

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

名称	类型	描述
count	int	返回的 machine 数目
total	int	machine 总数
machines	json array	返回的 machine 名称列表

machines 说明如下：

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

```
{
"count":10,
"total":100,
"machines":
[{
"ip" : "testip1",
"machine-uniqueid" : "testuuid1",
"userdefined-id" : "testuserdefinedid1",
"lastHeartbeatTime" : 1447182247
},
{
"ip" : "testip1",
"machine-uniqueid" : "testuuid2",
"userdefined-id" : "testuserdefinedid2",
"lastHeartbeatTime" : 1447182247
},
{
"ip" : "testip2",
"machine-uniqueid" : "testuuid",
"userdefined-id" : "testuserdefinedid"
"lastHeartbeatTime" : 1447182247
}]
}
```

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} not exist
500	InternalServerError	internal server error

细节描述

该接口只获取与 Server 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 94to3z418yupi6ikawqqd370:9yoK0iJPxr0RrWf/wW9NJYXu4zo=",
"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
}
```

GetAppliedConfigs

获得 machinegroup 上已经被应用的配置名称。

示例：

GET /machinegroups/{groupName}/configs

```
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

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。

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

请求成功，其响应 Body 会包括指定 machinegroup 下的所有 machine 列表，具体格式如下：

名称	类型	描述
count	整型	返回的 config 数目。
configs	字符串数组	返回的 config 名称列表。

```
{
"count":2,
"configs":
["config1","config2"]
}
```

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} not exist
500	InternalServerError	internal server error

细节描述

无

示例

请求示例：

```
GET /machinegroups/test-machine-group/configs HTTP/1.1
Header :
{
"x-log-apiversion": "0.6.0",
"Authorization": "LOG 94to3z418yupi6ikawqqd370:/Ntg290OaJ8JfInmhzYTG/GJwbE=",
"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
}
```

Logtail配置相关接口

CreateConfig

在 project 下创建日志配置。

示例：

POST /configs

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"
}
}
```

请求参数

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，现在只支持 file	
inputDetail	json	是	见下表格说明	
outputType	string	是	输出类型，现在只支持 LogService	
outputDetail	json	是	见下表格说明	
logSample	string	否	logtail 配置日志样例，最大支持 1000 字节	

inputDetail 内容：

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	和每个 filterKey 对应的正则表达式，filterRegex 的长度和 filterKey 的长度必须相同
topicFormat	string	否	topic 生成方式，支持四种类型：1）用于将日志文件路径的某部分作为 topic，如 /var/log/(.*).log；2）none，表示 topic 为空；3）default，表示将日志文件路径作为 topic；4）group_topic，表示将应用该配置的机器组 topic 属性作为 topic。
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
fileEncoding	string	否	支持两种类型：utf8、gbk

outputDetail 内容：

属性名称	**类型**	**必须**	**描述**

logstoreName	string	是	对应 logstore 名字

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 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 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	ConfigAlreadyExist	config {Configname} already exist
400	InvalidParameter	invalid config resource json
500	InternalServerError	internal server error

细节描述

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 94to3z418yupi6ikawqqd370:x/L1ymdn9wxe2zrwzcdSG82nXL0='
}
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+) (\\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'
}
```

ListConfig

列出 project 下所有配置信息，可以通过参数进行翻页。

示例：

```
GET /configs?offset=1&size=100
```

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
```

请求参数

URL 参数如下：

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

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 API 的公共响应头，请参考 公共响应头。

响应元素

返回值：Body 包含该 project 下所有 config 列表，具体格式如下：

名称	类型	描述
count	整型	返回的 config 数目
total	整型	在服务端 config 总数
configs	字符串数组	返回的 config 名称列表

错误码

除了返回 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	ConfigNotExist	config {Configname} not exist
500	InternalServerError	internal server error

细节描述

N/A

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 94to3z418yupi6ikawqqd370:teWnMylnM4Toohhp9dfBECrEgac="
}
```

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
}
```

GetAppliedMachineGroups

列出 config 应用的机器列表。

示例：

GET /configs/{configName}/machinegroups

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
```

请求参数

URL 参数：

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

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 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 machinegroup 下的所有 machine 列表，具体格式如下：

名称	类型	描述
count	整型	返回的 machineGroup 数目。
machinegroups	字符串数组	返回的 machineGroup 名称列表。

```
{
"count":2,
"machinegroups":
["group1","group2"]
}
```

错误码

除了返回 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} not exist
500	InternalServerError	internal server error

细节描述

无

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 94to3z418yupi6ikawqqd370:+6bo4MSUt/dyNa72kXeGCkVOi+4="
}
```

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"
]
}
```

GetConfig

获得一个配置的详细信息。

示例：

```
GET /configs/{configName}
```

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
```

请求参数

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

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 API 的公共响应头，请参考 公共响应头。

响应元素

属性名称	类型	描述
configName	string	日志配置名称，project 下唯一
inputType	string	输入类型，现在只支持 file
inputDetail	json	见下表格说明
outputType	string	输出类型，现在只支持 LogService
outputDetail	json	见下表格说明
createTime	Int	配置创建时间
lastModifyTime	Int	该资源服务端更新时间

inputDetail 内容：

属性名称	类型	描述
logType	string	日志类型，现在只支持 common_reg_log
logPath	string	日志所在的父目录，例如 /var/logs/
filePattern	string	日志文件的 Pattern，例如 access*.log
localStorage	boolean	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	日志时间格式, 如 %Y/%m/%d %H:%M:%S
logBeginRegex	string	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	日志对提取正则表达式
key	array	日志提取后所生成的 Key
filterKey	array	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	和每个 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	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	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	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 内容：

属性名称	类型	必须	描述
endpoint	string	否	project 所在的访问地址，不需要带 project 前缀，如果不填写，则使用默认为 EndPoint
logstoreName	string	是	对应 logstore 名称

错误码

除了返回 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	ConfigNotExist	Config {Configname} not exist
500	InternalServerError	Specified Server Error Message

细节描述

N/A

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 94to3z418yupi6ikawqqd370:yV5LsYLmn1UrAXvBg8CbZNZoiTk="
}
```

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
}
```


DeleteConfig

删除特定 config，如果 config 已被 **应用到机器组**，则 logtail 配置也会被删除。

DELETE /configs/{configName}

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
```

请求参数

URL参数:

参数名称	类型	是否必须	描述

ConfigName	String	是	日志配置名称

请求头

无特有请求头。关于 API 的公共请求头，请参考 **公共请求头**。

响应头

无特有响应头。关于 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 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	ConfigAlreadyExist	config {Configname} already 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 94to3z418yupi6ikawqqd370:utd/O1JNCYvcRGiSXHsjKhTzJDI="
}
```

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"
}
```

UpdateConfig

更新配置内容，如果配置被应用到机器组，对应机器也会同时更新。

示例：

PUT /configs/{configName}

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",
"logBeginReg": ".*",
"regex": "(\w+)(\s+)",
"key" :["key1", "key2"],
"filterKey":["key1"],
"filterRegex":["regex1"],
"topicFormat": "none"
},
"outputType": "LogService",
"outputDetail":
{
"logstoreName": "perfcounter"
}
}
```

请求参数

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，现在只支持 file	
inputDetail	json	是	见下表格说明	
outputType	string	是	输出类型，现在只支持 LogService	
outputDetail	json	是	见下表格说明	

inputDetail 内容：

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	和每个 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
fileEncoding	string	否	支持两种类型：utf8、gbk，默认值为utf8

outputDetail 内容：

属性名称	类型	必须	描述
logstoreName	string	是	对应 logstore 名称

请求头

无特有请求头。关于 API 的公共请求头，请参考 公共请求头。

响应头

无特有响应头。关于 API 的公共响应头，请参考 公共响应头。

响应元素

返回值：成功返回 200 状态码。

错误码

除了返回 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	ConfigNotExist	config {Configname} not exist
400	InvalidParameter	invalid config resource json
400	BadRequest	config resource configname not match request
500	InternalServerError	internal server error

细节描述

创建过程中遇到格式错误、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：

```
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 94to3z418yupi6ikawqqd370:GTPzFbLe8PZW0OFxTk/xMoCXA9E="
}
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"
}
```

RAM子用户访问

概述

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。

> **注意：**在了解如何使用 RAM 来授权和访问 Log Service 资源之前，请确保您已详细阅读了 RAM 产品文档 和 API 文档。

RAM 控制台中和日志服务相关的授权策略主要有三个：

AliyunLogFullAccess

给子用户授予该权限，那么子用户将对父账号拥有的日志服务的资源有完全的访问权限。授权策略描述如下：

```
 {
"Version": "1",
"Statement": [
{
"Action": "log:*",
```

```
"Resource": "*",
"Effect": "Allow"
}
]
}
```

AliyunLogReadOnlyAccess

给子用户授予该权限，那么子用户将对父账号拥有的日志服务的资源有只读的访问权限。授权策略描述如下：

```
 {
"Version": "1",
"Statement": [
{
"Action": [
"log:Get*",
"log:List*"
],
"Resource": "*",
"Effect": "Allow"
}
]
}
```

控制台查询指定日志库（logstore）数据

给子用户授予该权限，那么子用户在登录控制台后将对父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:ListProject", "log:ListLogStores"],
"Resource": ["acs:log:*:*:project/<指定的 proejct 名称>/*"],
"Effect": "Allow"
} ,
{
"Action": ["log:Get*"],
"Resource": ["acs:log:*:*:project/<指定的 proejct 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

资源列表

RAM 中可授权的 Log Service 资源类型

目前，可以在 RAM 中进行授权的资源类型及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/${logstoreName}/consumergroup/*
泛指模式	acs:log:${regionName}:${projectOwnerAliUid}:*
	acs:log:*:${projectOwnerAliUid}:*

注意：Log Service 中的资源有层级关系，project 是顶级资源，logstore、config、machinegroup 是 project 的子资源，相互之间是平级资源，shipper 和 consumergroup 是 logstore 的子资源。

其中：

- ${regionName} 为某个 region 的名称。
- ${projectOwnerAliUid} 为用户的阿里云账号 ID。
- ${projectName} 为 Log Service 项目的名称。
- ${logstoreName} 为 logstore 的名称。
- ${logtailconfig} 为 config 的名称。
- ${machineGroupName} 为 machinegroup 的名称。
- ${shipperName} 应为日志投递规则的名称。
- ${consumerGroupName} 应为协同消费组的名称。

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，每一个 Action 都和某一个或者两个 API 对应。

- log:GetLogStore
- log:ListLogStores
- log:CreateLogStore
- log:DeleteLogStore
- log:UpdateLogStore
- log:GetCursorOrData (GetCursor，PullLogs)
- log:ListShards
- log:PostLogStoreLogs
- log:CreateConfig
- log:UpdateConfig
- log:DeleteConfig
- log:GetConfig
- log:ListConfig
- log:CreateMachineGroup
- log:UpdateMachineGroup
- log:DeleteMachineGroup
- log:GetMachineGroup
- log:ListMachineGroup
- log:ListMachines
- log:ApplyConfigToGroup

- log:RemoveConfigFromGroup
- log:GetAppliedMachineGroups
- log:GetAppliedConfigs
- log:GetShipperStatus
- log:RetryShipperTask
- log:CreateConsumerGroup
- log:UpdateConsumerGroup
- log:DeleteConsumerGroup
- log:ListConsumerGroup
- log:ConsumerGroupUpdateCheckPoint
- log:ConsumerGroupHeartBeat
- log:GetConsumerGroupCheckPoint

鉴权规则

Log Service API 发生子账号访问主账号资源时的鉴权规则

当子账号通过 Log Service Open API 对主账号的资源进行访问时，Log Service 后台向 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。

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。具体地，各类 API 的鉴权规则见下表。

logstore

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/**${logtailConfigName}**
log:GetConfig	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logtailconfig/**${logtailConfigName}**
log:ListConfig	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logtailconfig/*

machinegroup

Actions	Resources
log:CreateMachineGroup	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/machinegroup/*
log:UpdateMachineGroup	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/machinegroup/**${machineGroupName}**
log:DeleteMachineGroup	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/machinegroup/**${machineGroupName}**

log:GetMachineGroup	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/machinegroup/**${machineGroupName}**
log:ListMachineGroup	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/machinegroup/*
log:ListMachines	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/machinegroup/**${machineGroupName}**

config 和 machinegroup 交互类 API

Actions	Resources
log:ApplyConfigToGroup	acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/logtailconfig/**${logtailConfigName}** acs:log:**${regionName}**:**${projectOwnerAliUid}**:project/**${projectName}**/machinegroup/**${machineGroupName}**
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}**

STS访问方式

借助 STS 实现跨账号访问日志资源资源

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。

如果需要授权其它云账号访问各类资源，需要通过 STS 获取临时 ak/token 调用具体操作。在阅读以下说明之前，请先阅读 STS产品文档。

假设用户 A 在日志服务创建了 project 和 logstore 等各类资源，用户 B 希望调用 API 接口进行访问，需要按照如下步骤进行操作。

用户 A 操作

创建角色

用户 A 通过 访问控制服务控制台 或者 API 创建角色信任用户 B 云账号，创建的角色详情如下：

```
{
"Statement": [
{
"Action": "sts:AssumeRole",
"Effect": "Allow",
"Principal": {
"RAM": [
"acs:ram::<云账号 B 的账号 ID>:root"
]
}
}
],
"Version": "1"
}
```

角色授权

角色创建完成后，用户 A 需要授权角色特定操作权限。

如果仅需要写数据，具体权限描述如下：

```
```
{
"Version": "1",
"Statement": [
{
"Action": "log:PostLogStoreLogs",
"Resource": "*",
"Effect": "Allow"
}
]
}
```
```

如果需要通过协同消费库拖取数据，具体权限描述如下：

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

```
"Statement": [
{
"Action": [
"log:GetCursorOrData",
"log:CreateConsumerGroup",
"log:ListConsumerGroup",
"log:ConsumerGroupUpdateCheckPoint",
"log:ConsumerGroupHeartBeat",
"log:GetConsumerGroupCheckPoint"
]
"Resource": "*",
"Effect": "Allow"
}
]
}
```

资源（Resource）设置说明：

- 以上两类资源都是授权指定用户的所有 project 和 logstore
- 如果授权指定 project：acs:log::*{projectOwnerAliUid}:project/*
- 如果授权指定 logstore：acs:log:*:{projectOwnerAliUid}:project/{projectName}/logstore/{logstoreName}/*

完整的资源说明请参考 日志服务 RAM 资源。

用户 B 操作

创建子账号并授权

登录访问控制服务控制台或者使用 API/SDK 创建子账号并且授予 AssumeRole 权限。

调用 STS 接口获取临时 ak/token

STS SDK 使用说明

调用日志服务接口

日志服务 SDk 使用说明

示例代码

样例代码以用户 B 通过 STS 向用户 A 的 project 写入数据为例（基于 JavaSDK）。

代码链接

公共资源说明

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。

地域（Region）

地域为阿里云的服务节点。您通过在不同的阿里云 Region 部署服务，让自己的服务距离客户更近，获得更低的访问延时及更好的用户体验。目前阿里云在全国各地拥有多个 Region。

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，具体如下表描述：

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 字节的 UTF-8 编码字符串

source	任意不超过 128 字节的 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 字节的 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

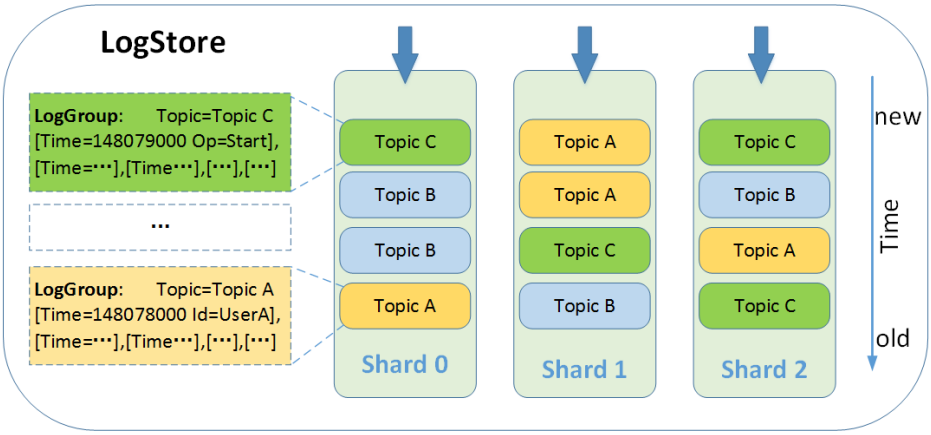
上表 content 中的 key 不可以使用如下关键字：__time__，__source__，__topic__，__partition_time__，_extract_others_，__extract_others__。

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。

注意：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。

下图描述了日志库、日志主题和日志之间的关系：



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 10.249.201.117，下面为其上的一条原始日志：

```
10.1.168.193 - - [01/Mar/2012:16:12:07 +0800] "GET /Send?AccessKeyId=8225105404 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：

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	“10.249.201.117”	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，例如下表：

key	value
ip	“10.1.168.193”
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

Logs

由多条日志产生的集合。

LogGroup

一组日志的集合。

LogGroupList

一组 LogGroup 集合，用于结果返回。

编码方式

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。

	含义	Content-Type
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;
}
```

注意：由于 PB 对 KeyValue 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。

数据编码方式

Protocol Buffer 是 Google 开发的用于结构化数据交换格式，被广泛用于 Google 内部及外部很多服务。目前，日志服务也使用 Protocol Buffer 格式作为标准的日志写入格式。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;
}
```

注意：

- 由于 PB 对 KeyValue 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 PostLogStoreLogs。

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，短横线（-）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 字节以内

完整资源示例：

```
{
  “logstoreName" : "access_log",
"ttl": 1,
"shardCount": 2,
"createTime":1439538649,
"lastModifyTime":1439538649
}
```

各参数含义：

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	日志数据服务单元
createTime（Output Only）	integer	否	该资源服务端创建时间（输出可见）
lastModifyTime（OutputOnly）	integer	否	该资源服务端更新时间（输出可见）

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，每个分区能承载一定量的

服务能力：

- 写入：5MB/s
- 读取：10MB/s

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：

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

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，短横线（-）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：

```
{
"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”：” sls”，
  “outputDetail”：
```

```
{
  “logstoreName”：” perfcounter”
},
  “createTime": 1400123456,
  “lastModifyTime": 1400123456
}
```

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 内容：

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	和每个 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 内容：

属性名称	类型	必须	描述
logstoreName	string	是	对应 logstore 名称

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示。

每台机器属性如下：

```
{
"ip" : "testip1",
"machine-uniqueid" : "testuuid1",
"userdefined-id" : "testuserdefinedid1",
```

```
"lastHeartbeatTime”:1397781420
}
```

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，短横线（-）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
}
```

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 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，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