ALIBABA CLOUD

阿里云

应用配置管理 ACM

SDK参考

文档版本：20200907

阿里云

法律声明

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

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
`Courier字体`	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.SDK简介

SDK是ACM提供给用户在应用运行时获取、发布、监听、删除配置的手段。同时，我们也开放了SDK底层使用的API协议，如SDK不能满足您的使用场景，您也可以基于API协议来实现应用运行时配置操作和监听的功能。

SDK 应用层 配置 监听

> 说明 此SDK用于在应用中操作和监听配置，如您需要在管控节点对应用配置进行管理，可考虑使用获取OpenAPI SDK。

ACM SDK支持的类型

ACM SDK目前支持以下类型：

- ACM Java Native SDK：支持ACM配置监听和变更的Java原生SDK。
- Spring Cloud ACM：支持Spring Cloud Config接口规范的Java SDK。后续将不再继续维护，推荐使用Nacos的 spring-cloud-starter-alibaba-nacos-config。
- ACM Node.js SDK：支持ACM配置监听和变更的Node.js原生SDK。
- ACM C++ SDK：支持ACM配置监听和变更的C++原生SDK。
- ACM Python（已开源）：支持ACM配置监听和变更的Python原生SDK。
- ACM PHP（已开源）：支持ACM配置监听的PHP原生SDK。
- Nacos Client：支持ACM配置监听和变更的Java原生SDK。

ACM SDK功能特性

ACM SDK 的功能特性如下：

功能	Java Native	Java Spring Cloud	Python	Node.JS	C++	PHP	Nacos SDK
获取特定配置	支持	支持	支持	支持	支持	支持	支持
监听特定配置	支持	支持	支持	支持	支持	不支持	支持
写入配置	支持	不支持	支持	不支持	不支持	支持	支持
支持固定租户下枚举配置	支持	不支持	支持	不支持	不支持	不支持	不支持
支持单一IP方式进行服务端连接	支持	不支持	支持	不支持	不支持	不支持	支持
支持多IP LB方式进行服务端连接	支持	支持	支持	支持	支持	支持	支持

功能	Java Native	Java Spring Cloud	Python	Node.JS	C++	PHP	Nacos SDK
支持HmacSHA1算法方式用户认证	支持	支持	支持	支持	支持	支持	支持
支持本地缓存备份	支持	支持	支持	支持	支持	不支持	支持
支持 ECS 实例 RAM 角色鉴权	支持	支持	支持	不支持	不支持	不支持	不支持
开源地址	计划中	计划中	acm-sdk-python	计划中	计划中	acm-sdk-php	Nacos

- 多IP LB方式是指ACM SDK基于地址服务器返回的多ACM服务端IP地址的LoadBalance功能，以提高性能和实现高可用。
- 本地缓存备份功能可以让ACM SDK在没有服务端连接情况下，通过读取上次获取配置后保存的本地缓存备份文件，来避免客户端应用宕机。
- `spring-cloud-starter-acm` 后续将不再继续维护，推荐使用Nacos的 spring-cloud-starter-alibaba-nacos-config。

地域和接入点列表

地域	地址服务器域名
公网	acm.aliyun.com
华东 1（杭州）	addr-hz-internal.edas.aliyun.com
华北 1（青岛）	addr-qd-internal.edas.aliyun.com
华东 2（上海）	addr-sh-internal.edas.aliyun.com
华北 2（北京）	addr-bj-internal.edas.aliyun.com
华南 1（深圳）	addr-sz-internal.edas.aliyun.com
中国（香港）	addr-hk-internal.edas.aliyuncs.com
新加坡（新加坡）	addr-singapore-internal.edas.aliyun.com
澳大利亚（悉尼）	addr-ap-southeast-2-internal.edas.aliyun.com
美国（硅谷）	addr-us-west-1-internal.acm.aliyun.com
美国（弗吉尼亚）	addr-us-east-1-internal.acm.aliyun.com

地域	地址服务器域名
华东 2（上海）金融云	addr-cn-shanghai-finance-1-internal.edas.aliyun.com

2.HTTP API

2.1. API 概览

本文介绍 ACM API 的概要信息，包括 API 列表、获取服务器 IP 的方法、通信协议、请求方法、公共参数、签名算法等。

配置管理 API

您可以使用本文档介绍的 API 对应用配置服务进行相关操作。请确保在使用这些接口前，您已充分了解 ACM 产品说明和使用协议。

API	描述
getConfig	获取 ACM 配置内容。
getAllConfigByTenant	获取指定命名空间内的 ACM 配置信息。
addListener	监听 ACM 配置的变更。
syncUpdateAll	发布 ACM 配置。
deleteAllDatums	删除 ACM 配置。

获取服务器 IP 列表

通过地域和接入点列表获取服务 IP 列表，以便通过服务 IP 发起请求。

```
http://${接入点}:8080/diamond-server/diamond
```

代码示例：

```
curl http://acm.aliyun.com:8080/diamond-server/diamond


# 返回
XX.XX.XX.144
```

通信协议

支持通过 HTTP 进行请求通信。

请求方法

支持 HTTP GET 或 POST 方法发送请求，GET 方式下请求参数需要包含在请求的 URL 中。

请求参数

每个请求都需要包含公共的鉴权、签名相关请求参数和相关操作所特有的请求参数。

公共参数

调用 ACM API 时都会用到的 Header 参数如下表所示。

参数	类型	是否必需	描述
Spas-AccessKey	String	是	在 ACM 控制台上的**命名空间详情**对话框内可获取 AccessKey。
timeStamp	String	是	以毫秒为单位的请求时间。
Spas-Signature	String	是	使用 SecretKey 对“Tenant+TimeStamp”签名，注意加号+也是签名的一部分。（ `SpasSigner.sign(Tenant+Group+TimeStamp, secretKey)` ），签名算法为 HmacSHA1。TimeStamp 签名的作用是防止重放攻击。该签名有效期为 60 秒。
Spas-SecurityToken	String	否	SecurityToken 需从 STS 临时凭证中获取。STS 临时凭证需从实例元数据 URL 中获取。详情请参考： • 借助于实例 RAM 角色访问其他云产品 • 通过 ECS 实例 RAM 角色访问 ACM
longPullingTimeout	String	是	长轮询等待 30 秒，此处填写 *30000*。

字符编码

请求及返回结果都使用 GBK 字符集进行编码。

签名机制

ACM 服务会对每个访问的请求进行身份验证，使用 HTTP 需要在请求中包含签名（Signature）信息。ACM 通过使用 AccessKey 和 SecretKey 进行对称加密的方法来验证请求的发送者身份。

AccessKey 和 SecretKey 由 ACM 颁发给访问者。其中 AccessKey 用于标识访问者的身份，SecretKey 是用于加密签名字符串和服务器端验证签名字符串的密钥，出于安全考虑，请务必严格保密。

签名算法

签名采用 HmacSHA1 算法。

- Java 签名算法参考

```
public static void main(String[] args) throws Exception {
String tenant= "tenant";
String group = "group";
String timeStamp = String.valueOf(System.currentTimeMillis());
String abc = HmacSHA1Encrypt(tenant+ "+" + group + "+" + timeStamp , "1234");
System.out.println(abc);
}
public static String HmacSHA1Encrypt(String encryptText, String encryptKey) throws Exception {
byte[] data = encryptKey.getBytes("UTF-8");
// 根据给定的字节数组构造一个密钥，第二参数指定一个密钥算法的名称
SecretKey secretKey = new SecretKeySpec(data, "HmacSHA1");
// 生成一个指定 Mac 算法 的 Mac 对象
Mac mac = Mac.getInstance("HmacSHA1");
// 用给定密钥初始化 Mac 对象
mac.init(secretKey);
byte[] text = encryptText.getBytes("UTF-8");
byte[] textFinal = mac.doFinal(text);
// 完成 Mac 操作, base64编码，将byte数组转换为字符串
return new String(Base64.encodeBase64(textFinal));
}
```

- Shell 签名算法

```
## config sign
timestamp=`echo $[$(date +%s%N)/1000000]`
signStr=$namespace+$group+$timestamp
signContent=`echo -n $signStr | openssl dgst -hmac $sk -sha1 -binary | base64`
echo $signContent
```

签名处理步骤

1. 使用请求参数构造规范的请求字符串（QueryParam）。
2. 使用上一步构造的规范字符串，按照以下规则构造用于计算签名的字符串。

   ```
   Signature=HMAC-SHA1(QueryParam)
   ```

   > **说明** 对于不同的请求，QueryParam 会不同。

3. 按照 RFC2104 的定义，使用上一步构造的用于签名的字符串来计算签名 HMAC 值。

   > **说明** 计算签名时使用的 Key 就是您持有的 AccessKeySecret（ASCII:38），使用的哈希算法是 SHA1。

4. 按照 Base64 编码规则将上一步算出的 HMAC 值编码成字符串，即得到签名值（Signature）。

5. 将上一步得到的签名值作为 Signature 参数添加到请求参数中，即完成对请求的签名处理。

示例代码

以下示例代码展示了如何以 Shell 构造 ACM 请求。

```
#!/bin/bash
## config param
dataId="com.alibaba.nacos.example.properties"
group="DEFAULT_GROUP"
namespace="04754ad1-4f67-4d67-b2bf-1f73a04a****"
accessKey="8c5cbb849ae04682ad9f455a96aa****"
secretKey="lwO5T7vfPJu27FclPa+/CyIG****"
endpoint="acm.aliyun.com"
## config param end
## get serverIp from address server
serverIp=`curl $endpoint:8080/diamond-server/diamond -s | awk '{a[NR]=$0}END{srand();i=int(rand()*N
R+1);print a[i]}'`
## config sign
timestamp=`echo $[$(date +%s%N)/1000000]`
signStr=$namespace+$group+$timestamp
signContent=`echo -n $signStr | openssl dgst -hmac $secretKey -sha1 -binary | base64`
## request to get a config
curl -H "Spas-AccessKey:"$accessKey -H "timeStamp:"$timestamp -H "Spas-Signature:"$signContent "
http://"$serverIp":8080/diamond-server/config.co?dataId="$dataId"&group="$group"&tenant="$name
space -v
```

限流机制

ACM 对访问频率采取限制，主要规则如下：

- 每个 IP 长连接数最多为 30 个。
- 每个 IP 每秒修改同一个配置不能超过 5 次。
- 每个 IP 每秒获取同一个配置不能超过 10 次。

2.2. getAllConfigByTenant

使用 getAllConfigByTenant 接口获取指定命名空间内的 ACM 配置信息。

请求类型

GET

请求 URL

/diamond-server/basestone.do?method=getAllConfigByTenant

请求参数

参数	类型	是否必需	描述
tenant	String	是	租户信息，对应 ACM 的命名空间 ID。
pageNo	Integer	是	分页页号
pageSize	Integer	是	分页大小

Header 参数

参数	类型	是否必需	描述
Spas-AccessKey	String	是	在 ACM 控制台上的**命名空间详情**对话框内可获取 AccessKey。
timeStamp	String	是	以毫秒为单位的请求时间。
Spas-Signature	String	是	使用 SecretKey 对“Tenant+TimeStamp”签名，注意加号+也是签名的一部分。（`SpasSigner.sign(Tenant+Group+TimeStamp, secretKey)`），签名算法为 HmacSHA1。TimeStamp 签名的作用是防止重放攻击。该签名有效期为 60 秒。
Spas-SecurityToken	String	否	SecurityToken 需从 STS 临时凭证中获取。STS 临时凭证需从实例元数据 URL 中获取。详情请参考： • 借助于实例 RAM 角色访问其他云产品 • 通过 ECS 实例 RAM 角色访问 ACM

返回参数

参数	类型	描述
totalCount	Integer	总配置数
pageNumber	Integer	分页页号
pagesAvailable	Integer	可用分页数

参数	类型	描述
pageItems	Array	配置信息
└appName	String	归属应用的名称
└dataId	String	配置的 ID
└group	String	配置的分组

错误码

错误码	错误信息	描述
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	客户端错误，未找到。
500	Internal Server Error	服务器内部错误

代码示例

- 请求示例（Shell）

```
#!/bin/bash
## config param
namespace="04754ad1-4f67-4d67-b2bf-1f73a04a****"
accessKey="8c5cbb849ae04682ad9f455a96aa****"
secretKey="lwO5T7vfPJu27FclPa+/CyIG****"
endpoint="acm.aliyun.com"
## config param end
## get serverIp from address server
serverIp=`curl $endpoint:8080/diamond-server/diamond -s | awk '{a[NR]=$0}END{srand();i=int(rand(
)*NR+1);print a[i]}'`
## config sign
timestamp=`echo $[$(date +%s%N)/1000000]`
signStr=$namespace+$timestamp
signContent=`echo -n $signStr | openssl dgst -hmac $secretKey -sha1 -binary | base64`
## request to get configs in a namespace
curl -H "Spas-AccessKey:"$accessKey -H "timeStamp:"$timestamp -H "Spas-Signature:"$signConten
t "http://"$serverIp":8080/diamond-server/basestone.do?method=getAllConfigByTenant&tenant="
$namespace"&pageNo="1"&pageSize="10 -v
```

> **说明** 由于 shell 脚本在 Windows 系统下编辑容易编码错误，建议在 Linux 系统里新建 shell 脚本文件，再将此段代码复制到文件中。

- 返回示例（JSON）

```
{
"totalCount":4,
"pageNumber":1,
"pagesAvailable":1,
"pageItems":[
{
"appName":"",
"dataId":"com.alibaba.nacos.example01.properties",
"group":"DEFAULT_GROUP"
},
{
"appName":"",
"dataId":"com.alibaba.nacos.example02.properties",
"group":"DEFAULT_GROUP"
},
{
"appName":"",
"dataId":"com.alibaba.nacos.example03.properties",
"group":"DEFAULT_GROUP"
},
{
"appName":"",
"dataId":"com.alibaba.nacos.example04.properties",
"group":"DEFAULT_GROUP"
}
]
}
```

更多信息

API 概览

2.3. getConfig

使用 getConfig 接口获取 ACM 配置。

请求类型

GET

请求 URL

/diamond-server/config.co

请求参数

参数	类型	是否必需	描述
tenant	String	是	租户信息，对应 ACM 的命名空间 ID。
dataId	String	是	配置的 ID
group	String	是	配置的分组

Header 参数

参数	类型	是否必需	描述
Spas-AccessKey	String	是	在 ACM 控制台上的**命名空间详情**对话框内可获取 AccessKey。
timeStamp	String	是	以毫秒为单位的请求时间。
Spas-Signature	String	是	使用 SecretKey 对“Tenant+TimeStamp”签名，注意加号+也是签名的一部分。（`SpasSigner.sign(Tenant+Group+TimeStamp, secretKey)`），签名算法为 HmacSHA1。TimeStamp 签名的作用是防止重放攻击。该签名有效期为 60 秒。
Spas-SecurityToken	String	否	SecurityToken 需从 STS 临时凭证中获取。STS 临时凭证需从实例元数据 URL 中获取。详情请参考： • 借助于实例 RAM 角色访问其他云产品 • 通过 ECS 实例 RAM 角色访问 ACM

返回参数

参数	类型	描述
-	String	配置内容

错误码

错误码	错误信息	描述
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	客户端错误，未找到。
500	Internal Server Error	服务器内部错误

代码示例

- 请求示例（Shell）

```
#!/bin/bash
## config param
dataId="com.alibaba.nacos.example.properties"
group="DEFAULT_GROUP"
namespace="04754ad1-4f67-4d67-b2bf-1f73a04a****"
accessKey="8c5cbb849ae04682ad9f455a96aa****"
secretKey="lwO5T7vfPJu27FclPa+/CyIG****"
endpoint="acm.aliyun.com"
## config param end
## get serverIp from address server
serverIp=`curl $endpoint:8080/diamond-server/diamond -s | awk '{a[NR]=$0}END{srand();i=int(rand()*NR+1);print a[i]}'`
## config sign
timestamp=`echo $[$(date +%s%N)/1000000]`
signStr=$namespace+$group+$timestamp
signContent=`echo -n $signStr | openssl dgst -hmac $secretKey -sha1 -binary | base64`
## request to get a config
curl -H "Spas-AccessKey:"$accessKey -H "timeStamp:"$timestamp -H "Spas-Signature:"$signContent "http://"$serverIp":8080/diamond-server/config.co?dataId="$dataId"&group="$group"&tenant="$namespace -v
```

> **说明** 由于 shell 脚本在 Windows 系统下编辑容易编码错误，建议在 Linux 系统里新建 shell 脚本文件，再将此段代码复制到文件中。

- 返回示例

```
connectTimeoutInMills=3000
```

更多信息

API 概览

2.4. addListener

使用 addListener 接口监听 ACM 配置的变更。

API 描述

addListener 接口可监听 ACM 上的配置，并实时感知配置变更。如果配置变更，则您可以用 getConfig 接口获取配置的最新值，并动态刷新本地缓存。

注册监听采用的是异步 Servlet 技术。注册监听的本质是将配置和配置值的 MD5 值与后台对比，如果 MD5 值不一致，则立即返回不一致的配置。如果 MD5 值一致，则等待 30 秒，且返回值为空。

请求类型

POST

请求 URL

/diamond-server/config.co

请求参数

参数	类型	是否必需	描述
Probe-Modify-Request	String	是	监听数据报文，格式为 `dataId^2group^2contentMD5^2tenant^1` 。 • 配置的多个字段间分隔符： `(Java)^2 = Character.toString((char) 2)` 或 `(Shell) ^2=%02` 。 • 配置间分隔符： `(Java) ^1 = Character.toString((char) 1)` 或 `(Shell) ^1=%01` • contentMD5：MD5(content)。第一次本地缓存为空，所以为空串。

Header 参数

参数	类型	是否必需	描述
Spas-AccessKey	String	是	在 ACM 控制台上的命名空间详情对话框内可获取 AccessKey。
timeStamp	String	是	以毫秒为单位的请求时间。
Spas-Signature	String	是	使用 SecretKey 对“Tenant+TimeStamp”签名，注意加号+也是签名的一部分。（ SpasSigner.sign(Tenant+Group+TimeStamp, secretKey) ），签名算法为 HmacSHA1。TimeStamp 签名的作用是防止重放攻击。该签名有效期为 60 秒。
Spas-SecurityToken	String	否	SecurityToken 需从 STS 临时凭证中获取。STS 临时凭证需从实例元数据 URL 中获取。详情请参考： • 借助于实例 RAM 角色访问其他云产品 • 通过 ECS 实例 RAM 角色访问 ACM

返回参数

参数	类型	描述
configInfo	String	已变更配置的 Data ID、Group、命名空间信息，格式为 dataId^2group^2tenant^1 • 配置的多个字段间分隔符： (Java)^2 = Character.toString((char) 2) 或 (Shell) ^2=%02 。 • 配置间分隔符： (Java) ^1 = Character.toString((char) 1) 或 (Shell) ^1=%01

错误码

错误码	错误信息	描述
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	客户端错误，未找到。
500	Internal Server Error	服务器内部错误

代码示例

- 请求示例（Shell）

```
#!/bin/bash
## config param
dataId="com.alibaba.nacos.example.properties"
group="DEFAULT_GROUP"
namespace="04754ad1-4f67-4d67-b2bf-1f73a04a****"
accessKey="8c5cbb849ae04682ad9f455a96aa****"
secretKey="lwO5T7vfPJu27FclPa+/CyIG****"
endpoint="acm.aliyun.com"
## config param end
## get serverIp from address server
serverIp=`curl $endpoint:8080/diamond-server/diamond -s | awk '{a[NR]=$0}END{srand();i=int(rand(
)*NR+1);print a[i]}'`
## config sign
timestamp=`echo $[$(date +%s%N)/1000000]`
signStr=$namespace+$group+$timestamp
signContent=`echo -n $signStr | openssl dgst -hmac $secretKey -sha1 -binary | base64`

echo "Listening..."
curl -X POST -H "Spas-AccessKey:"$accessKey -H "timeStamp:"$timestamp -H "Spas-Signature:"$sig
nContent -H "longPullingTimeout:30000" "http://"$serverIp":8080/diamond-server/config.co" -d "Pro
be-Modify-Request=com.alibaba.nacos.example.properties%02DEFAULT_GROUP%02f4da32aa4289aa
861974f949b639****%0204754ad1-4f67-4d67-b2bf-1f73a04a****%01" -v
```

> **说明** 由于 shell 脚本在 Windows 系统下编辑容易编码错误，建议在 Linux 系统里新建 shell 脚本文件，再将此段代码复制到文件中。

- 返回示例（Shell）

```
# 如果配置有变更
com.alibaba.nacos.example.properties%02DEFAULT_GROUP%02f4da32aa4289aa861974f949b639****%
0204754ad1-4f67-4d67-b2bf-1f73a04a****%01

# 如果配置无变更，则返回空串
```

更多信息

API 概览

2.5. syncUpdateAll

使用 syncUpdateAll 接口发布 ACM 配置。

请求类型

POST

请求 URL

/diamond-server/basestone.do

请求参数

参数	类型	是否必需	描述
tenant	String	是	租户信息，对应 ACM 的命名空间 ID。
dataId	String	是	配置的 ID
group	String	是	配置的分组
content	String	是	配置的内容

Header 参数

参数	类型	是否必需	描述
Spas-AccessKey	String	是	在 ACM 控制台上的命名空间详情对话框内可获取 AccessKey。
timeStamp	String	是	以毫秒为单位的请求时间。

参数	类型	是否必需	描述
Spas-Signature	String	是	使用 SecretKey 对“Tenant+TimeStamp”签名，注意加号+也是签名的一部分。（ SpasSigner.sign(Tenant+Group+TimeStamp, secretKey) ），签名算法为 HmacSHA1。TimeStamp 签名的作用是防止重放攻击。该签名有效期为 60 秒。
Spas-SecurityToken	String	否	SecurityToken 需从 STS 临时凭证中获取。STS 临时凭证需从实例元数据 URL 中获取。详情请参考： • 借助于实例 RAM 角色访问其他云产品 • 通过 ECS 实例 RAM 角色访问 ACM

返回参数

参数	类型	描述
-	Boolean	是否成功

错误码

错误码	错误信息	描述
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	客户端错误，未找到。
500	Internal Server Error	服务器内部错误

代码示例

- 请求示例（Shell）

```
#!/bin/bash
## config param
dataId="com.alibaba.nacos.example.properties2"
group="DEFAULT_GROUP"
namespace="04754ad1-4f67-4d67-b2bf-1f73a04a****"
accessKey="8c5cbb849ae04682ad9f455a96aa****"
secretKey="lwO5T7vfPJu27FclPa+/CyIG****"
endpoint="acm.aliyun.com"
## config param end
## get serverIp from address server
serverIp=`curl $endpoint:8080/diamond-server/diamond -s | awk '{a[NR]=$0}END{srand();i=int(rand(
)*NR+1);print a[i]}'`
## config sign
timestamp=`echo $[$(date +%s%N)/1000000]`
signStr=$namespace+$group+$timestamp
signContent=`echo -n $signStr | openssl dgst -hmac $secretKey -sha1 -binary | base64`
## request to publish a config
curl -X POST -H "Spas-AccessKey:"$accessKey -H "timeStamp:"$timestamp -H "Spas-Signature:"$sig
nContent "http://"$serverIp":8080/diamond-server/basestone.do?method=syncUpdateAll" -d "dataI
d="$dataId"&group="$group"&tenant="$namespace"&content="key1=value1 -v
```

> **说明** 由于 shell 脚本在 Windows 系统下编辑容易编码错误，建议在 Linux 系统里新建 shell 脚本文件，再将此段代码复制到文件中。

- 返回示例

```
True
```

更多信息

API 概览

2.6. deleteAllDatums

使用 deleteAllDatums 接口删除 ACM 配置。

请求类型

POST

请求 URL

/diamond-server/datum.do

请求参数

参数	类型	是否必需	描述
tenant	String	是	租户信息，对应 ACM 的命名空间 ID。
dataId	String	是	配置的 ID
group	String	是	配置的分组

Header 参数

返回参数

参数	类型	是否必需	描述
Spas-AccessKey	String	是	在 ACM 控制台上的**命名空间详情**对话框内可获取 AccessKey。
timeStamp	String	是	以毫秒为单位的请求时间。
Spas-Signature	String	是	使用 SecretKey 对“Tenant+TimeStamp”签名，注意加号+也是签名的一部分。（ `SpasSigner.sign(Tenant+Group+TimeStamp, secretKey)` ），签名算法为 HmacSHA1。TimeStamp 签名的作用是防止重放攻击。该签名有效期为 60 秒。

错误码

错误码	错误信息	描述
400	Bad Request	客户端请求中的语法错误
403	Forbidden	没有权限
404	Not Found	客户端错误，未找到。
500	Internal Server Error	服务器内部错误

代码示例

- 请求示例（Shell）

```
#!/bin/bash
## config param
dataId="com.alibaba.nacos.example.properties2"
group="DEFAULT_GROUP"
namespace="04754ad1-4f67-4d67-b2bf-1f73a04a****"
accessKey="8c5cbb849ae04682ad9f455a96aa****"
secretKey="lwO5T7vfPJu27FclPa+/CyIG****"
endpoint="acm.aliyun.com"
## config param end
## get serverIp from address server
serverIp=`curl $endpoint:8080/diamond-server/diamond -s | awk '{a[NR]=$0}END{srand();i=int(rand(
)*NR+1);print a[i]}'`
## config sign
timestamp=`echo $[$(date +%s%N)/1000000]`
signStr=$group+$timestamp
signContent=`echo -n $signStr | openssl dgst -hmac $secretKey -sha1 -binary | base64`
## request to delete a config
curl -X POST -H "Spas-AccessKey:"$accessKey -H "timeStamp:"$timestamp -H "Spas-Signature:"$sig
nContent "http://"$serverIp":8080/diamond-server/datum.do?method=deleteAllDatums" -d "dataId=
"$dataId"&group="$group -v
```

> **说明** 由于 shell 脚本在 Windows 系统下编辑容易编码错误，建议在 Linux 系统里新建 shell 脚本文件，再将此段代码复制到文件中。

- 返回示例

```
True
```

更多信息

API 概览

3.ACM Java SDK

3.1. ACM Java Native SDK

3.1.1. ACM Java Native SDK 概述

您可以使用 ACM Java Native SDK 来获取、监听、发布和删除配置。

添加依赖

在 *pom.xml* 文件中添加以下依赖，即可开始使用 ACM Java Native SDK。

```
<dependency>
<groupId>com.alibaba.edas.acm</groupId>
<artifactId>acm-sdk</artifactId>
<version>1.0.9</version>
</dependency>
<!-- 如果已有日志实现，则可去除以下依赖 -->
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-classic</artifactId>
<version>1.1.7</version>
</dependency>
```

> 注意 如需使用以 KMS AES-128 方式加密的配置，则依赖的 acm-sdk 版本不可低于 1.0.9。

示例代码

添加依赖后，即可在程序中使用 ACM Java Native SDK 提供的接口。

> 说明 请将代码中的 *$regionId*、*$endpoint*、*$namespace*、*$accessKey*、*$secretKey* 分别替换为 ACM 控制台上命名空间详情对话框内的地域 ID、End Point、命名空间 ID、AccessKey、SecretKey。

```
import java.util.Properties;
import com.alibaba.edas.acm.ConfigService;
import com.alibaba.edas.acm.exception.ConfigException;
import com.alibaba.edas.acm.listener.ConfigChangeListener;
import com.alibaba.edas.acm.listener.PropertiesListener;
// 示例代码，仅用于示例测试
public class ACMTest {
// 属性/开关
private static String config = "DefaultValue";
private static Properties acmProperties = new Properties();
```

```
public static void main(String[] args) {
try {
// 从控制台命名空间管理中拷贝对应值
Properties properties = new Properties();
properties.put("endpoint", "$endpoint");
properties.put("namespace", "$namespace");
// 通过 ECS 实例 RAM 角色访问 ACM
// properties.put("ramRoleName", "$ramRoleName");
properties.put("accessKey", "$accessKey");
properties.put("secretKey", "$secretKey");
// 如果是加密配置，则添加下面两行进行自动解密
//properties.put("openKMSFilter", true);
//properties.put("regionId", "$regionId");
ConfigService.init(properties);
// 主动获取配置
String content = ConfigService.getConfig("${dataId}", "${group}", 6000);
System.out.println(content);
// 初始化的时候，给配置添加监听，配置变更会回调通知
ConfigService.addListener("${dataId}", "${group}", new ConfigChangeListener() {
public void receiveConfigInfo(String configInfo) {
// 当配置更新后，通过该回调函数将最新值返回给用户。
// 注意回调函数中不要做阻塞操作，否则阻塞通知线程。
config = configInfo;
System.out.println(configInfo);
}
});
/**
* 如果配置值的内容为properties格式（key=value），可使用下面监听器。以便一个配置管理多个配置项
*/
/**
ConfigService.addListener("${dataId}", "${group}", new PropertiesListener() {
@Override
public void innerReceive(Properties properties) {
// TODO Auto-generated method stub
acmProperties = properties;
System.out.println(properties);
}
});
**/
} catch (ConfigException e) {
e.printStackTrace();
```

```
e.printStackTrace();
}
// 测试让主线程不退出，因为订阅配置是守护线程，主线程退出守护线程就会退出。 正式代码中无需下面代码
while (true) {
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
// 通过get接口把配置值暴露出去使用
public static String getConfig() {
return config;
}
// 通过get接口把配置值暴露出去使用
public static Object getPorpertiesValue(String key) {
if (acmProperties != null) {
return acmProperties.get(key);
}
return null;
}
}
```

传参方式

为了帮助您快速入门，以上示例代码中采用了以代码初始化参数的方式。但是，实际生产中可能有不同环境，例如不同的账号、地域或命名空间，而参数因环境而异，因此需要通过变量传入。为了方便配置入参和降低配置成本，ACM 提供了多种传参方式。

> **说明** 在 EDAS 环境下发布需要注意，EDAS 没有公网环境。

初始化参数	传参方法
endpoint	优先级由高到低： 1. JVM 参数：`-Daddress.server.domain=xxx` 2. 环境变量：`address_server_domain=xxx` 3. 代码设置：如以上示例代码

初始化参数	传参方法
namespace	优先级由高到低： 1. JVM 参数：`-Dtenant.id=xxx` 2. 代码设置：如以上示例代码
ramRoleName	优先级由高到低： 1. JVM 参数：`-Dram.role.name=xxx` 2. 代码设置：如以上示例代码 说明 ramRoleName 的鉴权优先级高于 accessKey/ secretKey。
accessKey/secretKey	优先级由高到低： 1. 文件方式：accessKey 和 secretKey 以 Properties 格式（满足 `public void java.io.Reader.Properties.load(Reader reader)` 方法）存储在 -Dspas.identity 指定文件中。如果不指定，则默认取 */home/admin/.spas_key/<appName>*文件（*<appName>* 以 -Dproject.name 指定） 2. 环境变量：`spas_accessKey=xxx spas_secretKey=xxx` 3. 代码设置：如以上示例代码

更多信息

- 获取配置
- 监听配置
- 发布配置
- 删除配置
- 通过ECS实例RAM角色访问ACM

3.1.2. 获取配置

用于从 ACM 获取配置内容。

描述

使用以下接口从 ACM 获取配置内容。

```
public static String getConfig(String dataId, String group, long timeoutMs) throws ConfigException
```

请求参数

参数	参数类型	描述
dataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。
group	String	配置分组，建议填写 `产品名:模块名`（如 `ACM:Test`）来保证唯一性。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。
timeout	String	读取配置超时时间，单位为 ms，推荐值为 3000。

返回值

参数类型	描述
String	配置值

请求示例

> **说明** 请将代码中的 *$regionId*、*$endpoint*、*$namespace*、*$accessKey*、*$secretKey* 分别替换为 ACM 控制台上命名空间详情对话框内的地域 ID、End Point、命名空间 ID、AccessKey、SecretKey。

```
try {
// 初始化配置服务
ConfigService.init("$endpoint", "$namespace", "$accessKey", "$secretKey");
// 主动获取配置
String content = ConfigService.getConfig("$dataId", "$group", 3000);
System.out.println(content);
} catch (ConfigException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

异常说明

读取配置超时或网络异常，抛出 `ConfigException` 异常。

限流机制

ACM 对访问频率采取限制，主要规则如下：

- 每个 IP 长连接数最多为 30 个。
- 每个 IP 每秒修改同一个配置不能超过 5 次。
- 每个 IP 每秒获取同一个配置不能超过 10 次。

更多信息

- ACM Java Native SDK 概述
- 监听配置
- 发布配置
- 删除配置

3.1.3. 监听配置

用于监听 ACM 配置的变更，以即时获取最新的配置内容。

描述

使用以下接口监听 ACM 配置的变更。

```
public static void addListener(String dataId, String group, ConfigChangeListenerAdapter listener)
```

请求参数

参数	参数类型	描述
dataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。
group	String	配置分组，建议填写 `产品名:模块名`（如 `ACM:Test`）来保证唯一性。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。
listener	ConfigChangeListener	监听器，配置变更进入监听器的回调函数。

返回值

参数类型	描述
String	配置值，初始化或者配置变更时通过回调函数返回该值。

请求示例

> 说明 请将代码中的 *$regionId*、*$endpoint*、*$namespace*、*$accessKey*、*$secretKey* 分别替换为 ACM 控制台上命名空间详情对话框内的地域 ID、End Point、命名空间 ID、AccessKey、SecretKey。

```
// 初始化配置服务
ConfigService.init("$endpoint", "$namespace", "$accessKey", "$secretKey");
// 初始化时给配置添加监听，配置变更会回调通知。
ConfigService.addListener("$dataId", "$group", new ConfigChangeListener() {
public void receiveConfigInfo(String configInfo) {
// 当配置更新后，通过该回调函数将最新值返回给用户
System.out.println(configInfo);
}
});
// 以下代码用于测试，作用是让主线程不退出。订阅配置是守护线程，如果主线程退出守护线程就会退出。
while (true) {
try {
Thread.sleep(1000);
} catch (InterruptedException e) {
e.printStackTrace();
}
}
```

更多信息

- ACM Java Native SDK 概述
- 获取配置
- 发布配置
- 删除配置

3.1.4. 发布配置

用于通过程序自动发布 ACM 配置，以自动化手段降低运维成本。

描述

使用以下接口将配置发布到 ACM。

> 注意 创建和更新配置时均使用此接口，若配置不存在则创建此配置，若配置已存在则更新此配置。

```
public static boolean publishConfig(String dataId, String group, String content) throws ConfigExceptio
n
```

请求参数

参数	参数类型	描述
dataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。
group	String	配置分组，建议填写 `产品名:模块名`（如 `ACM:Test`）来保证唯一性。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。
content	String	配置内容，不超过 100K 字节。

返回值

参数类型	描述
Boolean	是否发布成功

请求示例

> **说明** 请将代码中的 *$regionId*、*$endpoint*、*$namespace*、*$accessKey*、*$secretKey* 分别替换为 ACM 控制台上命名空间详情对话框内的地域 ID、End Point、命名空间 ID、AccessKey、SecretKey。

```
try {
// 初始化配置服务
ConfigService.init("$endpoint", "$namespace", "$accessKey", "$secretKey");
// 主动获取配置
boolean isPublishOk = ConfigService.publishConfig("$dataId", "$group", "$content");
System.out.println(isPublishOk);
} catch (ConfigException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

异常说明

读取配置超时或网络异常，抛出 `ConfigException` 异常。

限流机制

ACM 对访问频率采取限制，主要规则如下：

- 每个 IP 长连接数最多为 30 个。
- 每个 IP 每秒修改同一个配置不能超过 5 次。
- 每个 IP 每秒获取同一个配置不能超过 10 次。

更多信息

- ACM Java Native SDK 概述
- 获取配置
- 监听配置
- 删除配置

3.1.5. 删除配置

用于通过程序自动删除 ACM 配置，以自动化手段降低运维成本。

描述

使用以下接口将配置从 ACM 删除。

> **说明** 若配置存在则删除该配置，若配置不存在则返回成功消息。

```
public static boolean removeConfig(String dataId, String group) throws ConfigException
```

请求参数

参数	参数类型	描述

参数	参数类型	描述
dataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。
group	String	配置分组，建议填写 `产品名:模块名`（如 `ACM:Test`）来保证唯一性。只允许英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。

返回值

参数类型	描述
Boolean	是否删除成功

请求示例

> 说明 请将代码中的 *$regionId*、*$endpoint*、*$namespace*、*$accessKey*、*$secretKey* 分别替换为 ACM 控制台上命名空间详情对话框内的地域 ID、End Point、命名空间 ID、AccessKey、SecretKey。

```
try {
// 初始化配置服务，控制台通过示例代码自动获取下面参数
ConfigService.init("$endpoint", "$namespace", "$accessKey", "$secretKey");
// 主动获取配置
boolean isRemoveOk = ConfigService.removeConfig("$dataId", "$group");
System.out.println(isRemoveOk);
} catch (ConfigException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
```

异常说明

读取配置超时或网络异常，抛出 `ConfigException` 异常。

更多信息

3.2. Spring Cloud ACM

3.2.1. 环境准备

本文介绍了Spring Cloud ACM SDK的配置步骤。

Spring Cloud ACM SDK的使用步骤

1. 增加Maven依赖。

```
<dependency>
<groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-acm</artifactId>
<version>1.10.0</version>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

> 说明 请使用Spring Cloud Greenwich或Spring Boot 2.1.X.RELEASE。

2. 配置应用名和应用组。
   在Spring Boot应用中编辑 `application.properties` 文件，配置 `spring.application.group` 和 `spring.application.name` 。

```
spring.application.group=com.alibaba.cloud.acm
spring.application.name=sample-app
```

3. 配置ACM环境和认证信息。
   在Spring Boot应用中编辑 `application.properties` 文件，配置 `alibaba.acm.endpoint` 、 `alibaba.acm.namespace` 、 `alibaba.acm.accessKey` 和 `alibaba.acm.secretKey` 。

```
spring.application.group=com.alibaba.cloud.acm
spring.application.name=sample-app
alibaba.acm.endpoint=xxx

# 命名空间ID。
alibaba.acm.namespace=xxx

# 若通过ECS实例RAM角色访问ACM，需添加。
alibaba.acm.ramRoleName=xxx

# 注意accessKey、secretKey的大小写。
alibaba.acm.accessKey=xxx
alibaba.acm.secretKey=xxx

# 如果是加密配置，则添加下面两行进行自动解密。regionId可以通过命名空间详情中的区域ID获取。
# alibaba.acm.openKMSFilter=true
# alibaba.acm.regionId=xxx

# 如果Group不是DEFAULT_GROUP，则需设置alibaba.acm.group。
# alibaba.acm.group指配置的分组，注意与spring.application.group的区分
# alibaba.acm.group=xxx

# 如果file-extension不是properties，则需设置alibaba.acm.file-extension。
# 可选值包括properties、yaml、yml，默认为properties (版本1.0.8以上支持更改此配置)
# alibaba.acm.file-extension=properties
```

4. 在ACM控制台添加应用配置。
   前往ACM控制台，在相应的命名空间（Namespace）下新建配置。
   - Data ID按照以下约定格式编写：
     `${spring.application.group}:${spring.application.name}.{alibaba.acm.file-extension}`
     例如： `com.alibaba.cloud.acm:sample-app.properties`
   - 配置格式选择 `Properties` ，配置内容即为想要注入到应用中的具体Key-Value：

     ```
     user.id = 001
     user.name = juven2
     user.age = 88
     ```

5. 如果需要ACM加载多配置，则需要在Spring Boot应用中编辑 `application.properties` 文件，加上配置 `spring.profiles.active=dev,online` 。
   实际项目中，也可以在应用运行参数中加上 `-Dspring.profiles.active=dev,online` 。
   加上该配置后，ACM会加载三个配置：

```
${spring.application.group}:${spring.application.name}.{alibaba.acm.file-extension},
${spring.application.group}:${spring.application.name}-dev.{alibaba.acm.file-extension},
${spring.application.group}:${spring.application.name}-online.{alibaba.acm.file-extension}
```

6. 如果项目中需要用到公共配置的需求，可以利用ACM中的级联方式，将公共的配置写在一个特定格式的Data ID内，然后在应用中可按照一定的格式顺序加载配置项，假设 `spring.application.group` 为 `com.alibaba.acm.biz` ， `${spring.application.name}` 为 `demoapp` ， `alibaba.acm.file-extension` 为 `properties` （默认），此时ACM SDK加载配置的Data ID顺序如下所示。

```
#首先加载原有格式的配置
${spring.application.group}:${spring.application.name}.${alibaba.acm.file-extension}
com.alibaba.acm.biz:demoapp.properties

#随后根据Group按照“.”切分后，拼接application.properties依次加载，如：
com.alibaba.acm:application.properties
com.alibaba:application.properties
com:application.properties
```

备注

- `spring-cloud-starter-acm 1.0.7` 及更高版本已支持 `Spring Boot 2.x` 。
- `spring-cloud-starter-acm 1.0.8` 及更高版本已支持 `YAML` 格式。
- 推荐使用 `2.0.1.RELEASE` 及更高版本的 `Spring Boot 2.x` 。 `2.0.0.RELEASE` 版本有读取旧数据的Bug。
- 如需下载完整示例代码，请单击：spring-cloud-acm-sample.zip。

相关文档

通过ECS实例RAM角色访问ACM

3.2.2. 配置注入

使用 Spring MVC 注解注入配置，降低使用配置成本。
可以直接使用 `@Value` 注入配置：

```
@Component
class SampleRunner implements ApplicationRunner {

@Value("${user.id}")
String userId;

@Value("${user.name}")
String userName;

@Value("${user.age}")
int userAge;

@Override
public void run(ApplicationArguments args){
System.out.println(userId);
System.out.println(userName);
System.out.println(userAge);
}

}
```

说明 如果同时在 Spring Boot 应用的 `application.properties` 和 ACM 的 `${spring.application.group}:${spring.application.name}.properties` 中配置了同一个 key，ACM 中的 value 会覆盖应用默认的 value。

3.2.3. Spring Boot 集成

健康检查

Spring Cloud ACM 集成了 Spring Boot 的 Health Check。访问 health endpoint 可以看到 Spring Boot 应用是否正确连接了 ACM 服务器：

```
{
"status": "UP",
"acm": {
"status": "UP",
"dataIds": [
"com.alibaba.cloud.acm:sample-app.properties"
]
},
"diskSpace": {
"status": "UP",
"total": 1000240963584,
"free": 858827423744,
"threshold": 10485760
},
"refreshScope": {
"status": "UP"
}
}
```

@RefreshScope

Spring Cloud ACM 也支持 Spring Cloud 的 Refresh Scope 特性。

标记了 `@RefreshScope` 注解的 Bean 会自动监听 ACM 服务端的变更，并在运行时自动更新配置。代码示例如下：

```
@RestController
@RequestMapping("/sample")
@RefreshScope
class SampleController {

@Value("${user.name}")
String userName;

@RequestMapping("/acm")
public String simple() {
return "Hello Spring Cloud ACM!" + " Hello " + userName + "!";
}

}
```

使用了 `@ConfigurationProperties` 注解的 Bean 默认就是 Refresh Scope 的。

ACM Endpoint

Spring Cloud ACM 内置了一个名为 `acm` 的 endpoint，您可以通过访问 Spring Boot 应用 management.port （默认为 8080）下的 /acm 访问，如：http://localhost:8080/acm


```
{
"runtime": {
"sources": [
{
"dataId": "com.alibaba.cloud.acm:sample-app.properties",
"lastSynced": "2017-10-10 10:46:27"
}
],
"refreshHistory": [
{
"timestamp": "2017-10-10 10:46:24",
"dataId": "com.alibaba.cloud.acm:sample-app.properties",
"md5": "8692ae986ec7bc345b3f0f4de602ff13"
}
]
},
"config": {
"group": "DEFAULT_GROUP",
"timeOut": 3000,
"endpoint": "xxx",
"namespace": "xxx",
"accessKey": "xxx",
"secretKey": "xxx"
}
}
```


备注

- 使用 Spring Boot 2.x 时，ACM Endpoint 访问路径为 `/actuator/acm` 。
- 使用 Spring Boot 2.x 时，必须添加配置 `management.endpoints.web.exposure.include=*` 才能访问 ACM Endpoint。

3.3. Nacos SDK

3.3.1. Nacos Client

本文说明如何使用 Nacos Client SDK 管理 ACM 配置。

前提条件

登录 ACM 控制台，并创建一个示例配置。

- Data ID：*com.alibaba.nacos.example.properties*
- Group：不填写，即使用默认的 *DEFAULT_GROUP*。
- 配置格式：*Properties*
- 配置内容：*connectTimeoutInMills=3000*

操作步骤

1. 在 Maven 项目的 pom.xml 文件中添加以下配置来获取 Nacos Client SDK。

```
<dependency>
<groupId>com.alibaba.nacos</groupId>
<artifactId>nacos-client</artifactId>
<version>${latest.version}</version>
</dependency>
<!-- 有日志实现，下面可去掉 -->
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-classic</artifactId>
<version>1.2.3</version>
</dependency>
```

2. 在工程中添加以下代码进行配置管理。

> 说明 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上命名空间详情对话框内的 End Point、命名空间 ID、AccessKey、SecretKey。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
import com.alibaba.nacos.api.NacosFactory;
import com.alibaba.nacos.api.PropertyKeyConst;
import com.alibaba.nacos.api.config.ConfigService;
import com.alibaba.nacos.api.exception.NacosException;
import com.alibaba.nacos.client.config.listener.impl.PropertiesListener;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import java.util.Properties;
/**
* 示例代码，仅用于示例测试
*/
public class NacosExample {
private static final Logger LOGGER = LoggerFactory.getLogger(NacosExample.class);
public static void main(String[] args) throws NacosException {
Properties properties = new Properties();
```

```
// 指定配置的 DataID 和 Group
String dataId = "${dataId}";
String group = "${group}";
String content = "connectTimeoutInMills=5000";
// 从控制台命名空间管理的"命名空间详情"中拷贝 endpoint、namespace
properties.put(PropertyKeyConst.ENDPOINT, "${endpoint}");
properties.put(PropertyKeyConst.NAMESPACE, "${namespace}");
 // 通过 ECS 实例 RAM 角色访问 ACM
// properties.put("ramRoleName", "$ramRoleName");
properties.put(PropertyKeyConst.ACCESS_KEY, "${accessKey}");
properties.put(PropertyKeyConst.SECRET_KEY, "${secretKey}");
ConfigService configService = NacosFactory.createConfigService(properties);
// 发布配置
boolean publishConfig = configService.publishConfig(dataId, group, content);
LOGGER.info("publishConfig: {}", publishConfig);
wait2Sync();
// 查询配置
String config = configService.getConfig(dataId, group, 5000);
LOGGER.info("getConfig: {}", config);
// 监听配置
configService.addListener(dataId, group, new PropertiesListener() {
@Override
public void innerReceive(Properties properties) {
LOGGER.info("innerReceive: {}", properties);
}
});
// 更新配置
boolean updateConfig = configService.publishConfig(dataId, group, "connectTimeoutInMills=3000"
);
LOGGER.info("updateConfig: {}", updateConfig);
wait2Sync();
// 删除配置
boolean removeConfig = configService.removeConfig(dataId, group);
LOGGER.info("removeConfig: {}", removeConfig);
wait2Sync();
config = configService.getConfig(dataId, group, 5000);
LOGGER.info("getConfig: {}", config);
}
private static void wait2Sync() {
try {
Thread.sleep(3000);
```

```
Thread.sleep(3000);
} catch (InterruptedException e) {
// ignore
}
}
}
```

结果验证

1. 在本地启动项目，若 Console 打印出以下信息，则说明 SDK 可正常使用。

```
12:40:45.567 [main] INFO DemoConfig - getConfig: connectTimeoutInMills=3000
```

2. 在 ACM 控制台将示例配置 `com.alibaba.nacos.example.properties` 更改为以下内容并发布。

```
connectTimeoutInMills=6000
```

若 Console 打印出以下信息 ，则说明 SDK 的配置监听功能正常。

```
[com.alibaba.nacos.client.Worker.longPolling.acm.aliyun.com-********] INFO DemoConfig - innerRece
ive: {connectTimeoutInMills=6000}
```

传参方式

为帮助您快速入门，以上示例代码采用了以代码初始化参数的方式。但是实际生产中可能有不同环境（如不同的账号、地域或 等），参数因环境而异，因此需要通过变量传入。为方便配置入参、降低配置成本，ACM 针对不同的初始化参数分别提供了多种传参方式，详见下表。

> 说明 对于阿里云 EDAS 用户而言，EDAS 会在发布环节自动注入参数，用户无需采取任何操作。

初始化参数	传参方式
endpoint	代码设置：详见以上示例代码。
namespace	优先级由高到低： 1. JVM 参数：`-Dtenant.id=xxx` 。 2. 代码设置：详见以上示例代码。

初始化参数	传参方式
accessKey/secretKey	优先级由高到低： 1. 文件：accessKey 和 secretKey 以 Properties 格式（满足 `public void java.io.Reader.Properties.load(Reader reader)` 方法的要求）存储在 `-Dspas.identity` 指定的文件中。如果不指定，则默认取 `/home/admin/.spas_key/<应用名>` 文件，<应用名>以 `-Dproject.name` 指定。 2. 环境变量：`spas_accessKey=xxx spas_secretKey=xxx` 3. 代码设置：请参见以上示例代码。
ramRoleName	优先级由高到低： 1. JVM 参数：`-Dram.role.name=xxx` 2. 代码设置：请参见以上示例代码。 说明 ramRoleName 的鉴权优先级高于 accessKey/ secretKey。

更多信息

- 创建配置
- 管理配置

3.3.2. Nacos Spring

本文说明如何使用 Nacos Spring SDK 管理 ACM 配置。

前提条件

- 登录 ACM 控制台，并创建一个示例配置。
  - Data ID：*com.alibaba.nacos.example.properties*
  - Group：不填写，即使用默认的 *DEFAULT_GROUP*。
  - 配置格式：*Properties*
  - 配置内容：*connectTimeoutInMills=3000*
- （可选）下载样例工程。

操作步骤

1. 在 Maven 项目的 pom.xml 文件中增加以下配置来获取 Nacos Spring SDK。

```
<dependency>
<groupId>com.alibaba.nacos</groupId>
<artifactId>nacos-spring-context</artifactId>
<version>${latest.version}</version>
</dependency>
```

2. 使用 @EnableNacosConfig 注解启用 Nacos Spring 的配置管理服务。

> 说明 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上命名空间详情对话框内的 End Point、命名空间 ID、AccessKey、SecretKey。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
@Configuration
@EnableNacosConfig(globalProperties = @NacosProperties(
endpoint = "${endpoint}",
namespace = "${namespace}",
accessKey = "${accessKey}",
secretKey = "${secretKey}"
))
```

3. 使用 @NacosPropertySource 注解加载配置源，并开启自动更新。

> 说明 实际开发时，请将 `com.alibaba.nacos.example.properties` 替换成真实的 ACM 配置 Data ID。

```
@NacosPropertySource(dataId = "com.alibaba.nacos.example.properties", autoRefreshed = true)
public class NacosConfiguration {


}
```

4. 使用 `@NacosValue` 注解设置属性值。

```
@Controller
@RequestMapping("config")
public class ConfigController {

@NacosInjected
private ConfigService configService;

@NacosValue(value = "${connectTimeoutInMills:5000}", autoRefreshed = true)
private int connectTimeoutInMills;

@RequestMapping(value = "/get", method = GET)
@ResponseBody
public int get() {
return connectTimeoutInMills;
}
}
```

结果验证

1. 在本地启动项目，并运行以下命令：

   ```
   curl localhost:8080/config/get
   ```

   若返回以下信息，则说明 SDK 可正常使用。

   ```
   3000
   ```

2. 在 ACM 控制台将示例配置 `com.alibaba.nacos.example.properties` 更改为以下内容并发布。

   ```
   connectTimeoutInMills=6000
   ```

   若 Console 打印出更新的配置内容 ，则说明 SDK 的配置自动更新功能正常。

更多信息

- 样例工程
- 创建配置
- 管理配置
- Nacos Spring Project
- ACM 无缝支持 Spring 全栈
- 公开课视频：ACM 无缝支持 Spring 全栈
- Nacos Spring 0.3.1 版本支持 YAML、JSON、XML 等格式

3.3.3. Nacos Spring Boot

本文说明如何使用 Nacos Spring Boot SDK 管理 ACM 配置。

前提条件

- 登录 ACM 控制台，并创建一个示例配置。
  - Data ID：*com.alibaba.nacos.example.properties*
  - Group：不填写，即使用默认的 *DEFAULT_GROUP*。
  - 配置格式：*Properties*
  - 配置内容：*connectTimeoutInMills=3000*
- （可选）下载样例工程。

操作步骤

1. 在 Maven 项目的 pom.xml 文件中增加以下配置来获取 Starter。

```
<dependency>
<groupId>com.alibaba.boot</groupId>
<artifactId>nacos-config-spring-boot-starter</artifactId>
<version>${latest.version}</version>
</dependency>
```

> 说明 使用时请根据 Spring Boot 版本选择相应的 nacos-config-spring-boot-starter 版本：nacos-config-spring-boot-starter 版本 0.2.x.RELEASE 对应 Spring Boot 2.x 版本，版本 0.1.x.RELEASE 对应 Spring Boot 1.x 版本。

2. 在 *application.properties* 文件中配置连接信息。

> 说明 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上**命名空间详情**对话框内的 **End Point**、**命名空间 ID**、**AccessKey**、**SecretKey**。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
nacos.config.endpoint=${endpoint}
nacos.config.namespace=${namespace}
# 推荐使用 RAM 用户的 accessKey 和 secretKey
nacos.config.access-key=${accessKey}
nacos.config.secret-key=${secretKey}
```

3. 使用 @NacosPropertySource 注解加载配置源，并开启自动更新。

> 说明 实际开发时，请将 `com.alibaba.nacos.example.properties` 替换成真实的 ACM 配置 Data ID。如果 group 不是 DEFAULT_GROUP，需要在 `@NacosPropertySource` 注解中进行指定。

```
@SpringBootApplication
@NacosPropertySource(dataId = "com.alibaba.nacos.example.properties", autoRefreshed = true)
public class NacosConfigApplication {
public static void main(String[] args) {
SpringApplication.run(NacosConfigApplication.class, args);
}
}
```

4. 使用 `@NacosValue` 注解设置属性值。

```
@Controller
@RequestMapping("config")
public class ConfigController {

@NacosValue(value = "${connectTimeoutInMills:5000}", autoRefreshed = true)
private int connectTimeoutInMills;

@RequestMapping(value = "/get", method = GET)
@ResponseBody
public int get() {
return connectTimeoutInMills;
}
}
```

结果验证

1. 在本地启动项目，并运行以下命令：

```
curl localhost:8080/config/get
```

若返回以下信息，则说明 SDK 可正常使用。

```
3000
```

2. 在 ACM 控制台将示例配置 `com.alibaba.nacos.example.properties` 更改为以下内容并发布。

```
connectTimeoutInMills=6000
```

若 Console 打印出更新的配置内容 ，则说明 SDK 的配置自动更新功能正常。

更多信息

- 样例工程
- 创建配置
- 管理配置
- Nacos Config Spring Boot

- ACM 无缝支持 Spring 全栈
- 公开课视频：ACM 无缝支持 Spring 全栈
- Nacos Spring Boot 0.2.2 以及 0.1.2 版本支持 YAML、JSON、XML 等格式

3.3.4. Nacos Spring Cloud

在开发 Spring Cloud 应用时，您可以使用 Nacos （https://nacos.io）在本地实现 Spring Cloud 应用的配置管理。

前提条件

- 登录 ACM 控制台，并创建一个示例配置。
  - Data ID：*com.alibaba.nacos.example.properties*
  - Group：不填写，即使用默认的 *DEFAULT_GROUP*。
  - 配置格式：*Properties*
  - 配置内容：*connectTimeoutInMills=3000*
- （可选）下载样例工程。

操作步骤

1. 在 Maven 项目的 pom.xml 文件中增加以下配置来获取 Starter。

```
<dependency>
<groupId>com.alibaba.cloud</groupId>
<artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
<version>${latest.version}</version>
</dependency>
```

> 说明 使用时请根据 Spring Boot 版本选择相应的 spring-cloud-starter-alibaba-nacos-config 版本。

Spring Cloud Version	Spring Cloud Alibaba Version	Spring Boot Version
Spring Cloud Hoxton	2.2.0.RELEASE	2.2.X.RELEASE
Spring Cloud Greenwich	2.1.1.RELEASE	2.1.X.RELEASE
Spring Cloud Finchley	2.0.1.RELEASE	2.0.X.RELEASE
Spring Cloud Edgware	1.5.1.RELEASE	1.5.X.RELEASE

2. 在 *bootstrap.properties* 文件中配置连接信息。

> 说明 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上**命名空间详情**对话框内的 **End Point**、**命名空间 ID**、**AccessKey**、**SecretKey**。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
spring.cloud.nacos.config.endpoint=${endpoint}
spring.cloud.nacos.config.namespace=${namespace}
# 推荐使用 RAM 用户的 accessKey 和 secretKey
spring.cloud.nacos.config.access-key=${accessKey}
spring.cloud.nacos.config.secret-key=${secretKey}
# 配置的 Data ID 等于 ${spring.application.name}.${spring.cloud.nacos.config.file-extension}
# 指定配置的 Data ID 前缀，例如：
spring.application.name=com.alibaba.nacos.example
# 指定配置的 Data ID 后缀，支持 properties、yaml、yml，默认为 properties
spring.cloud.nacos.config.file-extension=properties
# 指定 ACM 配置的 Group，默认为 DEFAULT_GROUP
spring.cloud.nacos.config.group=DEFAULT_GROUP
```

> 说明 ACM 配置的 Data ID 的约定格式为 `${spring.application.name}.${spring.cloud.nacos.config.file-extension}` 。本例中对应的 ACM 配置的 Data ID 为 `com.alibaba.nacos.example.properties` 。

3. 使用 Spring 的注解 `@Value` 设置属性值，使用 Spring Cloud 原生注解 `@RefreshScope` 实现配置自动更新。

```
@RestController
@RequestMapping("/config")
@RefreshScope
public class ConfigController {

@Value("${connectTimeoutInMills:5000}")
private int connectTimeoutInMills;

public void setConnectTimeoutInMills(int connectTimeoutInMills) {
this.connectTimeoutInMills = connectTimeoutInMills;
}

@RequestMapping(value = "/get", method = GET)
@ResponseBody
public int get() {
return connectTimeoutInMills;
}
}
```

结果验证

1. 在本地启动项目，并运行以下命令：

```
curl localhost:8080/config/get
```

若返回以下信息，则说明 SDK 可正常使用。

```
3000
```

2. 在 ACM 控制台将示例配置 `com.alibaba.nacos.example.properties` 更改为以下内容并发布。

```
connectTimeoutInMills=6000
```

若 Console 打印出更新的配置内容 ，则说明 SDK 的配置自动更新功能正常。

更多信息

- 样例工程
- 创建配置
- 管理配置
- Spring Cloud Alibaba Nacos Config
- ACM 无缝支持 Spring 全栈
- 公开课视频：ACM 无缝支持 Spring 全栈

4.Nacos Go SDK

4.1. Nacos Go SDK 概述

您可以在 Go 程序中使用 Nacos Go SDK 管理 Nacos 配置，包括获取、监听、发布和删除配置。

准备工作

- 在本地安装 Go
- 获取 Nacos Go SDK

公共参数

参数	参数类型	描述
ConfigParam.DataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。
ConfigParam.Group	String	配置分组，建议填写 `产品名:模块名`（如 `ACM:Test`）来保证唯一性。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。

示例代码

`Go` 格式

> **说明** 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上命名空间详情对话框内的 End Point、命名空间 ID、AccessKey、SecretKey。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
package main

import (
"fmt"
"github.com/nacos-group/nacos-sdk-go/clients"
"github.com/nacos-group/nacos-sdk-go/common/constant"
"github.com/nacos-group/nacos-sdk-go/vo"
"time"
)
```

```
func main() {
// 从控制台命名空间管理的"命名空间详情"中拷贝 End Point、命名空间 ID
var endpoint = "${endpoint}"
var namespaceId = "${namespaceId}"

// 推荐使用 RAM 用户的 accessKey、secretKey
var accessKey = "${accessKey}"
var secretKey = "${secretKey}"


clientConfig := constant.ClientConfig{
//
Endpoint: endpoint + ":8080",
NamespaceId: namespaceId,
AccessKey: accessKey,
SecretKey: secretKey,
TimeoutMs: 5 * 1000,
ListenInterval: 30 * 1000,
}

configClient, err := clients.CreateConfigClient(map[string]interface{}{
"clientConfig": clientConfig,
})

if err != nil {
fmt.Println(err)
return
}

var dataId = "com.alibaba.nacos.example.properties"
var group = "DEFAULT_GROUP"

// 发布配置
success, err := configClient.PublishConfig(vo.ConfigParam{
DataId: dataId,
Group: group,
Content: "connectTimeoutInMills=3000"})

if success {
fmt.Println("Publish config successfully.")
```

```
fmt.Println("Publish config successfully.")
}

time.Sleep(3 * time.Second)

// 获取配置
content, err := configClient.GetConfig(vo.ConfigParam{
DataId: dataId,
Group: group})

fmt.Println("Get config:  " + content)

// 监听配置
configClient.ListenConfig(vo.ConfigParam{
DataId: dataId,
Group: group,
OnChange: func(namespace, group, dataId, data string) {
fmt.Println("ListenConfig group:" + group + ", dataId:" + dataId + ", data:" + data)
},
})

// 删除配置
success, err = configClient.DeleteConfig(vo.ConfigParam{
DataId: dataId,
Group: group})

if success {
fmt.Println("Delete config successfully.")
}

}
```

更多信息

- GetConfig
- ListenConfig
- PublishConfig
- DeleteConfig
- Nacos Go SDK

4.2. GetConfig

使用 GetConfig 从 Nacos 获取配置值。

描述

使用以下接口从 Nacos 获取配置值。

```
GetConfig(param vo.ConfigParam) (string, error)
```

请求参数

参数	参数类型	描述
ConfigParam.DataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。
ConfigParam.Group	String	配置分组，建议填写 `产品名:模块名`（如 `ACM:Test`）来保证唯一性。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。

返回参数

参数类型	描述
String	配置值

示例

`Go` 格式

> **说明** 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上**命名空间详情**对话框内的 **End Point**、**命名空间 ID**、**AccessKey**、**SecretKey**。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
package main

import (
"fmt"
"github.com/nacos-group/nacos-sdk-go/clients"
"github.com/nacos-group/nacos-sdk-go/common/constant"
"github.com/nacos-group/nacos-sdk-go/vo"
"time"
```

```
)

func main() {
// 从控制台命名空间管理的"命名空间详情"中拷贝 End Point、命名空间 ID
var endpoint = "${endpoint}"
var namespaceId = "${namespaceId}"

// 推荐使用 RAM 用户的 accessKey、secretKey
var accessKey = "${accessKey}"
var secretKey = "${secretKey}"

clientConfig := constant.ClientConfig{
//
Endpoint: endpoint + ":8080",
NamespaceId: namespaceId,
AccessKey: accessKey,
SecretKey: secretKey,
TimeoutMs: 5 * 1000,
ListenInterval: 30 * 1000,
}

configClient, err := clients.CreateConfigClient(map[string]interface{}{
"clientConfig": clientConfig,
})

if err != nil {
fmt.Println(err)
return
}

var dataId = "com.alibaba.nacos.example.properties"
var group = "DEFAULT_GROUP"

// 获取配置
content, err := configClient.GetConfig(vo.ConfigParam{
DataId: dataId,
Group: group})

fmt.Println("Get config: " + content)

}
```

4.3. ListenConfig

使用 ListenConfig 监听 Nacos 配置的变更，以即时获取最新的配置值。

描述

使用以下接口监听 Nacos 配置的变更。

```
ListenConfig(params vo.ConfigParam) (err error)
```

请求参数

参数	参数类型	描述
ConfigParam.DataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。
ConfigParam.Group	String	配置分组，建议填写 `产品名:模块名`（如 `ACM:Test`）来保证唯一性。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。
ConfigParam.OnChange	Function	配置内容变更后回调的函数。

示例

`Go` 格式

> **说明** 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上命名空间详情对话框内的 End Point、命名空间 ID、AccessKey、SecretKey。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
package main

import (
"fmt"
"github.com/nacos-group/nacos-sdk-go/clients"
"github.com/nacos-group/nacos-sdk-go/common/constant"
"github.com/nacos-group/nacos-sdk-go/vo"
```

```
"time"
)

func main() {
// 从控制台命名空间管理的"命名空间详情"中拷贝 End Point、命名空间 ID
var endpoint = "${endpoint}"
var namespaceId = "${namespaceId}"

// 推荐使用 RAM 用户的 accessKey、secretKey
var accessKey = "${accessKey}"
var secretKey = "${secretKey}"


clientConfig := constant.ClientConfig{
//
Endpoint: endpoint + ":8080",
NamespaceId: namespaceId,
AccessKey: accessKey,
SecretKey: secretKey,
TimeoutMs: 5 * 1000,
ListenInterval: 30 * 1000,
}

configClient, err := clients.CreateConfigClient(map[string]interface{}{
"clientConfig": clientConfig,
})

if err != nil {
fmt.Println(err)
return
}

var dataId = "com.alibaba.nacos.example.properties"
var group = "DEFAULT_GROUP"

// 监听配置
configClient.ListenConfig(vo.ConfigParam{
DataId: dataId,
Group: group,
OnChange: func(namespace, group, dataId, data string) {
fmt.Println("ListenConfig group:" + group + ", dataId:" + dataId + ", data:" + data)
```

```
},
})

}
```

4.4. PublishConfig

使用 PublishConfig 将配置自动发布到 Nacos，以自动化手段降低运维成本。

描述

使用以下接口将配置发布到 Nacos。

```
PublishConfig(param vo.ConfigParam) (bool, error)
```

请求参数

参数	参数类型	描述
ConfigParam.DataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。
ConfigParam.Group	String	配置分组，建议填写 `产品名:模块名`（如 `ACM:Test`）来保证唯一性。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。
ConfigParam.Content	String	配置内容，不超过 100KB。

返回参数

参数类型	描述
Boolean	是否发布成功

示例

`Go` 格式

> 说明 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上命名空间详情对话框内的 End Point、命名空间 ID、AccessKey、SecretKey。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
package main

import (
"fmt"
"github.com/nacos-group/nacos-sdk-go/clients"
"github.com/nacos-group/nacos-sdk-go/common/constant"
"github.com/nacos-group/nacos-sdk-go/vo"
"time"
)

func main() {
// 从控制台命名空间管理的"命名空间详情"中拷贝 End Point、命名空间 ID
var endpoint = "${endpoint}"
var namespaceId = "${namespaceId}"

// 推荐使用 RAM 用户的 accessKey、secretKey
var accessKey = "${accessKey}"
var secretKey = "${secretKey}"


clientConfig := constant.ClientConfig{
//
Endpoint: endpoint + ":8080",
NamespaceId: namespaceId,
AccessKey: accessKey,
SecretKey: secretKey,
TimeoutMs: 5 * 1000,
ListenInterval: 30 * 1000,
}

configClient, err := clients.CreateConfigClient(map[string]interface{}{
"clientConfig": clientConfig,
})

if err != nil {
fmt.Println(err)
return
```

```
return
}

var dataId = "com.alibaba.nacos.example.properties"
var group = "DEFAULT_GROUP"

// 发布配置
success, err := configClient.PublishConfig(vo.ConfigParam{
DataId: dataId,
Group: group,
Content: "connectTimeoutInMills=3000"})

if success {
fmt.Println("Config published successfully.")
}

time.Sleep(3 * time.Second)

}
```

4.5. DeleteConfig

使用 DeleteConfig 将配置从 Nacos 删除，以自动化手段降低运维成本。

描述

使用以下接口将配置从 Nacos 删除。

```
DeleteConfig(param vo.ConfigParam) (bool, error)
```

请求参数

参数	参数类型	描述
ConfigParam.DataId	String	配置 ID，采用类似 `package.class`（如 `com.taobao.tc.refund.log.level`）的命名规则保证全局唯一性。建议根据配置的业务含义来定义 *class* 部分。全部字符均为小写。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 256 字节。

参数	参数类型	描述
ConfigParam.Group	String	配置分组，建议填写 产品名:模块名 （如 ACM:Test ）来保证唯一性。只允许使用英文字符和 4 种特殊字符（“.”、“:”、“-”、“_”），不超过 128 字节。

返回参数

参数类型	描述
Boolean	是否删除成功

示例

Go 格式

> 说明 请将代码中的 *${endpoint}*、*${namespace}*、*${accessKey}*、*${secretKey}* 分别替换为 ACM 控制台上命名空间详情对话框内的 End Point、命名空间 ID、AccessKey、SecretKey。出于安全考虑，建议使用 RAM 用户的 AccessKey 和 SecretKey。

```
package main

import (
"fmt"
"github.com/nacos-group/nacos-sdk-go/clients"
"github.com/nacos-group/nacos-sdk-go/common/constant"
"github.com/nacos-group/nacos-sdk-go/vo"
"time"
)

func main() {
// 从控制台命名空间管理的"命名空间详情"中拷贝 End Point、命名空间 ID
var endpoint = "${endpoint}"
var namespaceId = "${namespaceId}"

// 推荐使用 RAM 用户的 accessKey、secretKey
var accessKey = "${accessKey}"
var secretKey = "${secretKey}"


clientConfig := constant.ClientConfig{
//
```

```
Endpoint: endpoint + ":8080",
NamespaceId: namespaceId,
AccessKey: accessKey,
SecretKey: secretKey,
TimeoutMs: 5 * 1000,
ListenInterval: 30 * 1000,
}

configClient, err := clients.CreateConfigClient(map[string]interface{}{
"clientConfig": clientConfig,
})

if err != nil {
fmt.Println(err)
return
}

var dataId = "com.alibaba.nacos.example.properties"
var group = "DEFAULT_GROUP"

// 删除配置
success, err = configClient.DeleteConfig(vo.ConfigParam{
DataId: dataId,
Group: group})

if success {
fmt.Println("Config deleted successfully.")
}

}
```

5.ACM Node.js SDK

安装 ACM Node.js 版客户端

ACM Node.js 版客户端主要用于帮助前端开发解决前后端独立发布，开发与运营独立问题，大幅提升开发效率。

输入以下命令安装客户端：

```
npm i acm-client --save
```

示例代码

在您的工程中运行以下代码进行配置管理。

```
const { ACMClient } = require('acm-client');
const co = require('co');
const acm = new ACMClient({
endpoint: 'acm.aliyun.com', // ACM 控制台查看
namespace: '***************', // ACM 控制台查看
accessKey: '***************', // ACM 控制台查看
secretKey: '***************', // ACM 控制台查看
requestTimeout: 6000, // 请求超时时间，默认 6s
});
// 主动拉取配置
co(function*() {
const content= yield acm.getConfig('test', 'DEFAULT_GROUP');
console.log('getConfig = ',content);
});
// 监听数据更新
acm.subscribe({
dataId: 'test',
group: 'DEFAULT_GROUP',
}, content => {
console.log(content);
});
```

Error Events 异常处理

```
acm.on('error', function (err) {
// 可以在这里统一进行日志的记录
// 如果不监听错误事件，所有的异常都将会打印到 stderr
});
```

接口说明

- 获取配置接口
  用于服务启动的时候从 ACM 获取配置。

```
function* getConfig(dataId, group)
```

- 请求参数

参数名	参数类型	描述
dataId	string	配置 ID，采用类似 package.class（如 com.taobao.tc.refund.log.level ）的命名规则保证全局唯一性，class 部分建议是配置的业务含义。全部字符小写。只允许英文字符和 4 种特殊字符（”.”、”:”、”-“、”_”），不超过 256 字节。
group	string	配置分组，建议填写**产品名：模块名**（如 ACM:Test）保证唯一性，只允许英文字符和 4 种特殊字符（”.”、”:”、”-“、”_”），不超过 128 字节。

- 返回值

参数类型	描述
string	配置值

- 监听配置接口
  如果希望 ACM 推送配置变更，可以使用 ACM 动态监听配置接口来实现。

```
function subscribe(info, listener)
```

- 请求参数

参数名	参数类型	描述
info	Object	info.dataId：配置 ID。 info.group：配置分组
listener	Function	监听器，配置变更进入监听器的回调函数。

- 取消配置监听接口
  用于服务启动的时候取消配置监听。

```
function unSubscribe(info, [listener])
```

- 请求参数

参数名	参数类型	描述
info	Object	info.dataId:配置 ID。 info.group:配置分组
listener	Function	回调函数（可选，不传就移除所有监听函数）。

Node.js 项目链接

https://www.npmjs.com/package/acm-client

问题反馈

- @hustxiaoc

6.ACM C++ SDK

本文介绍 ACM C++ SDK 的使用方式。ACM C++ SDK 只支持 Linux 平台。

安装 ACM C++ SDK

1. 下载 SDK 依赖包：ACM C++ SDK
2. 下载完成后进行解压，会有如下目录结构：
   - example/
   - include/
   - lib/

   上面的目录和文件的作用如下：
   - example：acm.cpp 用于演示 SDK 使用。Makefile 用于 example 的编译和管理。
   - include：您自己编写的程序需要 include 的头文件。
   - lib：分别是 64 的静态库和动态库。
3. 设置环境变量 LD_LIBRARY_PATH，指定 ACM 动态库搜索路径，即安装的路径。

   ```
   export LD_LIBRARY_PATH=/usr/lib64:/usr/lib:/lib:/lib64:../lib
   ```
4. 进入 example 目录，修改 acm.cpp 文件的起始参数和配置的 dataId、group。执行 make 命令编译。执行示例代码如下：

   ```
   cd example //进入example目录
   vim acm.cpp //修改acm.cpp文件
   make //编译示例代码
   ./acm //执行示例代码
   ```

示例代码

在您的工程中运行以下代码进行配置管理。

```
#include "ACM.h"
using namespace std;
using namespace acm;
//定义监听器
class MyListener : public ManagerListener
{
public:
MyListener(const std::string& data_id, const std::string& group):data_id_(data_id),group_(group){}
virtual ~MyListener()
{}
virtual void getExecutor()
{
printf("data_id:%s group:%s getExecutor\n",data_id_.c_str(), group_.c_str());
}
```

```
//配置变更时回调
virtual void receiveConfigInfo( std::string &configInfo)
{
printf("data_id:%s group:%s configInfo:\n%s\n", data_id_.c_str(), group_.c_str(), configInfo.c_str());
config_ = configInfo;
}
private:
std::string data_id_;
std::string group_;
std::string config_;
};
int main() {
ACM::init("acm.aliyun.com", // endpoint: ACM 控制台查看
"***************", // namespace: ACM 控制台查看
"***************", // accessKey: ACM 控制台查看
"***************"); // secretKey: ACM 控制台查看
//监听配置的dataId和group
std::string dataId = "${dataId}";
std::string group = "${group}";
std::string content;
//主动拉取配置
ACM::getConfig(dataId, group, 5000, content);
printf("get ok config %s\n", content.c_str());
MyListener* listener = new MyListener(dataId, group);
//监听配置变更
ACM::addListener(dataId, group, listener);
printf("add listener ok %s %s\n", dataId.c_str(), group.c_str());
do {
printf("input q to quit\n");
} while (getchar() != 'q');
return 0;
}
```

接口说明

- 初始化设置接口
  设置 endpoint，nameSpace，accessKey，secretKey。

```
static void init(const char* endpoint,
const char* nameSpace,
char* accessKey,
char* secretKey);
```

各参数说明如下：

参数名	参数类型	描述
endpoint	const char*	ACM 域名，控制台获得
nameSpace	const char*	命名空间，控制台获得
accessKey	char*	accessKey，控制台获得
secretKey	char*	secretKey，控制台获得

- 获取配置接口
  用于服务启动的时候从 ACM 获取配置。

```
static bool getConfig(const std::string &dataId,
const std::string &group,
int timeoutMs,
std::string &content);
```

参数说明见下表：

参数名	参数类型	描述
dataId	const string	配置 ID，采用类似 package.class（如 com.taobao.tc.refund.log.level ）的命名规则保证全局唯一性，class 部分建议是配置的业务含义。全部字符小写。只允许英文字符和 4 种特殊字符（”.”、”:”、”-“、”_”），不超过 256 字节。
group	const string	配置分组，建议填写产品名：模块名（如 ACM:Test）保证唯一性，只允许英文字符和 4 种特殊字符（”.”、”:”、”-“、”_”），不超过 128 字节。
timeoutMs	int	超时时间
content	string	返回的配置内容

- 监听配置接口
  如果希望 ACM 推送配置变更，可以使用 ACM 动态监听配置接口来实现。

```
static void addListener(const std::string &dataId,
const std::string &group,
ManagerListener* listener);
```

参数说明见下表：

参数名	参数类型	描述
dataId	const string	dataId
group	const string	group
listener	ManagerListener	监听器，配置变更会执行监听器的回调函数。

- 取消配置监听接口
  用于服务启动的时候取消配置监听。

```
static void removeListener(const std::string &dataId,
const std::string &group,
ManagerListener* listener);
```

参数说明见下表：

参数名	参数类型	描述
dataId	const string	dataId
group	const string	group
listener	ManagerListener	待取消的监听器

7.ACM PHP SDK

为方便 PHP 程序使用 ACM 管理应用配置，ACM 提供了 PHP SDK。
ACM PHP SDK 现已开源，使用方法请参考 Github。

8.ACM Python SDK

为方便 Python 程序使用 ACM 管理应用配置，ACM 提供了 Python SDK。
ACM Python SDK 现已开源，使用方法请参考 Github。