阿里云
应用配置管理

SDK 参考

文档版本：20190816

法律声明

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

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

通用约定

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

目录

1 SDK 简介

ACM SDK 是 ACM 提供给用户的获取配置手段，用于获取、监听配置。

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 的功能特性如下：

功能 / 语言	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 / 语言	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	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。

2 ACM Java SDK

2.1 ACM Java Native SDK

2.1.1 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();
        }
        // 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:

EDAS 在发布环节自动为您注入，因此您无需采取任何操作。

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
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

- #unique_9
- #unique_10

- #unique_11
- #unique_12
- #unique_13

2.1.2 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。

更多信息

- ACM Java Native SDK 概述
- #unique_10
- #unique_11
- #unique_12

2.1.3 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, ConfigChan
geListenerAdapter 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 ConfigChan
geListener() {
    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
- #unique_9
- #unique_11
- #unique_12

2.1.4 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 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。
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。

更多信息

- ACM Java Native SDK 概述
- #unique_9
- #unique_10
- #unique_12

2.1.5 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
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

- ACM Java Native SDK 概述
- #unique_9
- #unique_10
- #unique_11

2.2 Spring Cloud ACM

2.2.1 环境准备

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>1.0.9</version>
 </dependency>
```

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

# 通过 ECS 实例 RAM 角色访问 ACM
# alibaba.acm.ramRoleName=xxx

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
# alibaba.acm.openKMSFilter=true
# regionId 可以通过命名空间详情中的区域 ID 获取
# alibaba.acm.regionId=xxx

# 如果 Group 不是 DEFAULT_GROUP，则需设置 alibaba.acm.group
# alibaba.acm.group=xxx

# 可选 properties、yaml、yml，默认为 properties（版本 1.0.8 以上支持）
#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
```

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

- #unique_20

2.2.2 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。

2.2.3 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,
    "endpoit": "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。

2.3 Nacos SDK

2.3.1 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.Properties
Listener;
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}");
        // 推荐使用 RAM 账号的 accessKey、secretKey,
        properties.put(PropertyKeyConst.ACCESS_KEY, "${accessKey}");
        properties.put(PropertyKeyConst.SECRET_KEY, "${secretKey}");
        ConfigService configService = NacosFactory.createConf
igService(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 Properties
Listener() {
            @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);
        } 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 - innerReceive: {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。

初始化参数	传参方式
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。
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：详见以上示例代码。

更多信息

- #unique_25
- #unique_26

2.3.2 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}", autoRefres
hed = 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
- #unique_25
- #unique_26
- 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

2.3.3 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。

```
@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}", autoRefres
hed = 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
- #unique_25
- #unique_26
- 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

2.3.4 Nacos Spring Cloud

本文说明如何使用 Nacos Spring Cloud 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>org.springframework.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-starter-alibaba-nacos-config 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 *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}
# ACM 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

- #unique_25
- #unique_26
- 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

3 Nacos Go SDK

3.1 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.")
    }

    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

- #unique_32
- #unique_33
- #unique_34
- #unique_35
- Nacos Go SDK

3.2 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)
```

```
}
```

3.3 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)
        },
    })
```

```
}
```

3.4 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
    }

    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)
```

```
}
```

3.5 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。
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.")
    }

}
```

4 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

参数名	参数类型	描述
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

5 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

6 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。

7 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。