Apusic 金蝶天燕

生成时间：2026-05-19

用户手册

金蝶Apusic分布式配置中心V1.0.243

版权声明

本文档所涉及的软件著作权、版权等知识产权已依法进行了注册，由金蝶天燕云计算股份有限公司合法拥有。受《中华人民共和国著作权法》《计算机软件保护条例》《知识产权保护条例》和相关国际版权条约、法律、法规以及其它知识产权法律和条约的保护。未经授权许可，不得非法使用。

免责声明

本文档包含的版权信息由金蝶天燕云计算股份有限公司合法拥有，受法律的保护，金蝶天燕云计算股份有限公司对本文档可能涉及到的非金蝶天燕云计算股份有限公司的信息不承担任何责任。在法律允许的范围内，您可以查阅并仅能够在《中华人民共和国著作权法》规定的合法范围内复制和打印本文档。任何单位和个人未经金蝶天燕云计算股份有限公司书面授权许可，不得使用、修改、再发布本文档的任何部分和内容，否则将被视为侵权，金蝶天燕云计算股份有限公司有依法追究其责任的权利。

本文档如有更新，不另行通知。对本文档中的问题您可向金蝶天燕云计算股份有限公司告知或查询。未经本公司明确授予的任何权利均予保留。

商标声明

Apusic是深圳市金蝶天燕云计算股份有限公司向中华人民共和国国家商标局申请注册的注册商标，注册商标专用权由金蝶天燕合法拥有，受法律保护。未经金蝶天燕的书面许可，任何单位及个人不得以任何方式或理由对该商标的任何部分进行使用、复制、修改、传播、抄录或与其它产品捆绑使用销售。凡侵犯金蝶天燕商标权的，金蝶天燕将依法追究其法律责任。本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

目录

1 版本更新说明

本文档根据实际进行更新，最新版本包含历史修改记录。

日期	手册版本	适用产品	更新说明
2025年12月	V10243M01F03	金蝶Apusic分布式配置中心V1.0.243 for nacos	兼容GBase数据库、OceanBase数据库 支持只认证管控不认证应用 提供只提交配置而不发布的openAPI接口
2024年11月	V10243M01F02	金蝶Apusic分布式配置中心V1.0.243 for nacos	更新版本名称 添加云部署内容
2024年5月	V10230M01F01	金蝶Apusic分布式配置中心V1.0.230 for nacos	更新授权模块

2 简介

2.1 产品简介

金蝶Apusic分布式配置中心软件for nacos（Apusic Distributed Config Center，简称：ADCC for nacos） 是一款易于配置管理和服务管理平台，旨在帮助构建云原生应用。它提供了服务注册与发现功能，让服务提供者能便捷地将自身信息注册，服务消费者可轻松获取服务列表，保障服务间通信顺畅；配置管理功能支持集中化管理应用配置，实时推送配置变更，确保应用使用最新配置；还具备服务管理能力，涵盖服务健康检查、流量管理等，助力优化服务性能。ADCC for nacos支持多种主流语言客户端，拥有高可用集群架构，广泛应用于微服务架构、云原生等场景，大幅提升了应用开发与运维的效率。

2.2 范围和读者

本手册介绍ADCC for nacos产品相关的内容，主要适用于用户、实施人员，维护人员等。

2.3 约定与术语

一些约定的缩略词诠释：

- ADCC

金蝶Apusic分布式配置中心软件（Apusic Distributed Config Center）

- ADCC_HOME

金蝶Apusic分布式配置中心软件安装目录

3 功能清单

一级功能模块	二级功能模块	功能说明	备注
服务注册与发现			
	服务的动态注册与发现	支持服务的动态注册与发现，使服务消费者能够实时感知服务提供者的变化	
	服务健康检测机制	提供服务健康检查机制，确保注册的服务的可用性	
配置管理			
	集中储存	开发者将配置信息（如数据库连接、API密钥等）集中存储	
	实时更新和推送	支持配置的实时更新和推送，使应用能够无缝地获取最新的配置信息，无需重启	
	版本控制	提供配置版本控制和回滚功能，确保配置变更的安全性	
服务管理			
	服务管理	提供可视化的服务管理界面，方便开发者查看和管理服务实例、配置信息等	
	服务的上下线管理	支持服务的上下线管理，以及服务流量的动态调整	
元数据管理			
	元数据管理	存储和管理服务的元数据	
权限控制			
	权限管理	提供细粒度的权限控制机制	
监控与告警			
	监听功能	提供监听功能，实时监控服务的运行状态和性能指标	
	告警机制	提供告警机制，当服务出现异常或性能下降时，能够及时通知开发者	

4 部署架构及技术架构

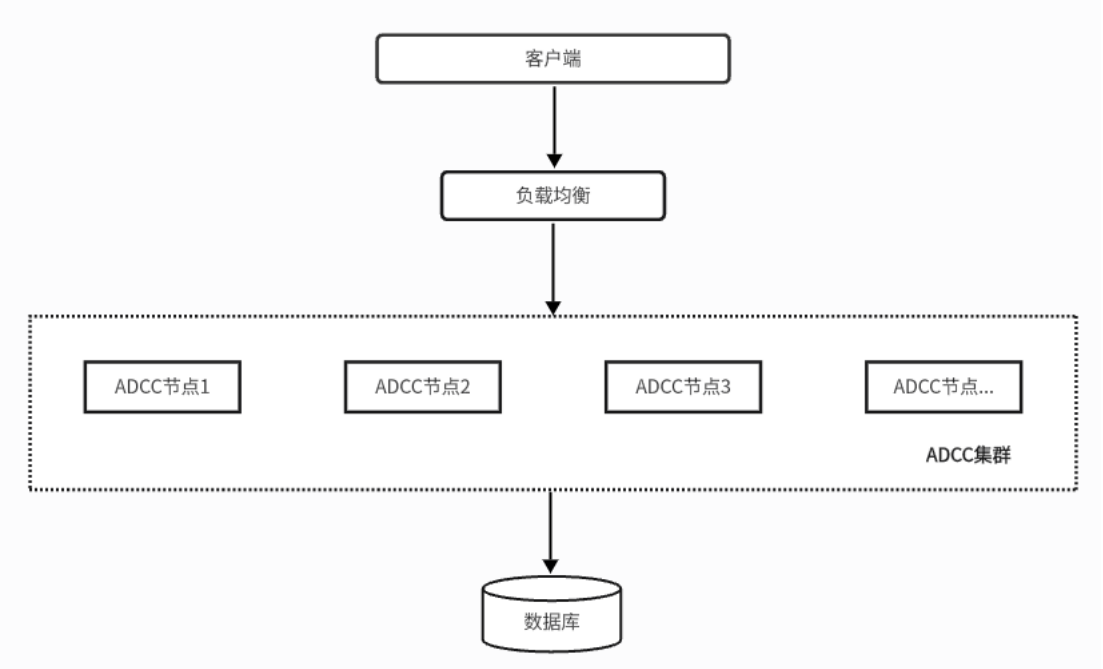
4.1 部署架构

ADCC for nacos支持单机部署和集群部署模式，通常情况下单机部署模式用于测试和单机试用，集群模式用于生成环境，确保高可用。

单机模式是ADCC最简单的部署方式。在这种模式下，所有的功能都集中在一个Server 实例上。从功能角度看，它既是服务注册中心，接收服务提供者发送的服务注册信息，包括服务的名称、IP 地址、端口号等关键数据，同时也是配置管理中心，用于存储和管理应用程序的配置信息。

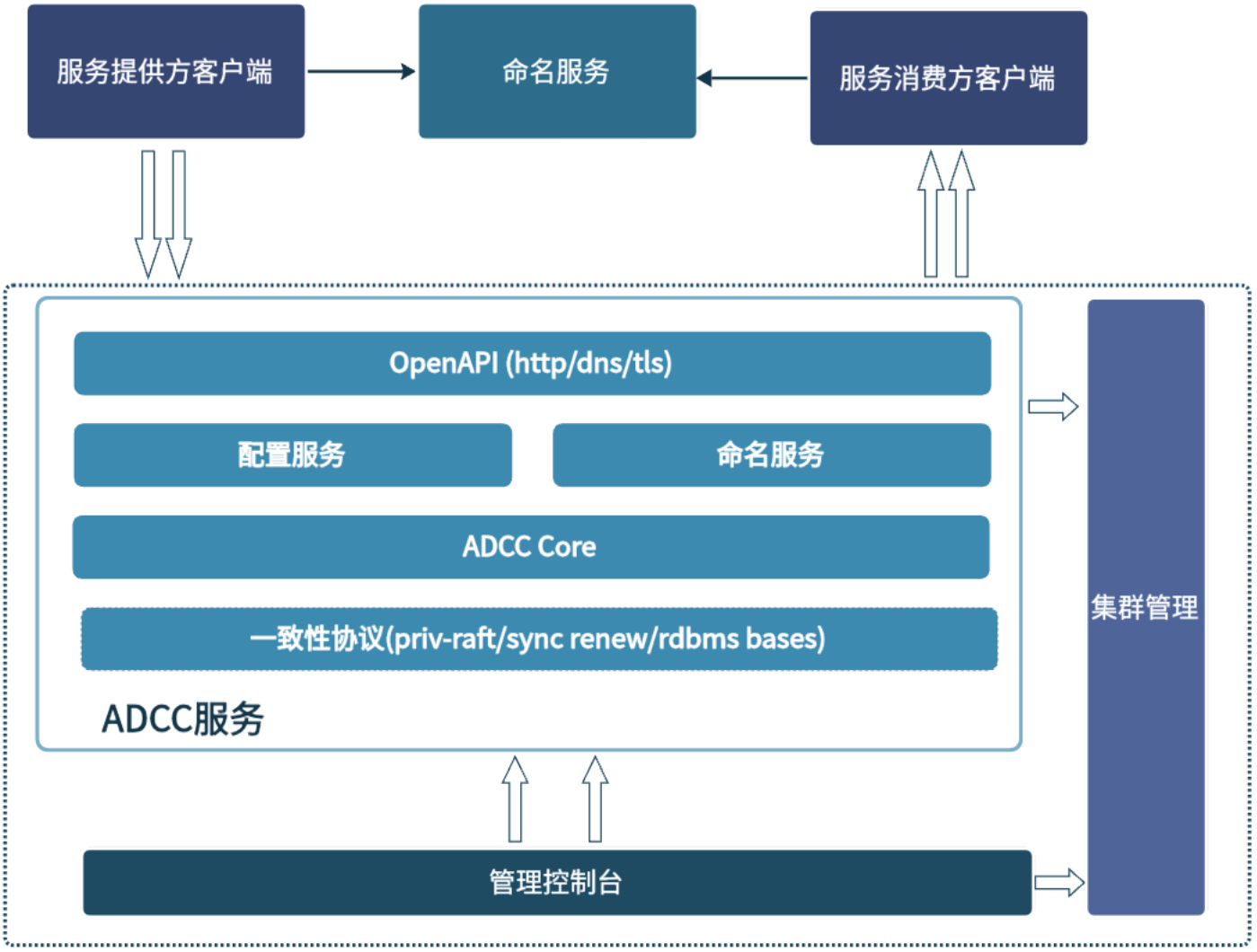
集群模式主要是为了提高ADCC 的可用性和可扩展性。在集群中有多个Server 节点，这些节点之间通过特定的通信协议相互连接。它们的角色是平等的，每个节点都具备完整的服务注册和配置管理功能。当服务提供者向集群注册服务时，请求可以被路由到任意一个 Server 节点；Server 节点之间将会进行信息同步，确保整个集群中的所有节点都能获取到最新的服务注册信息。对于服务消费者而言，无论它从集群中的哪个Server 节点获取服务列表，得到的都是完整的、经过同步的服务信息。

为了更好地存储和管理数据，建议使用外部数据库。外部数据库的使用可以提供更强大的存储能力和数据一致性保障。各个Server 节点会将服务注册信息和配置数据存储到外部数据库中，并且会从数据库中读取最新的数据来更新自身的状态，以保证整个集群的数据一致性和高可用性。



4.2 技术架构

ADCC for nacos 的技术架构是以服务注册发现和配置管理为核心功能，通过 HTTP 协议实现客户端与服务器、集群内服务器间通信，有单机和集群等存储方式，且能通过集群及跨区域多集群部署保障高可用性和扩展性的架构体系。


服务提供方客户端
命名服务
服务消费方客户端
OpenAPI (http/dns/tls)
配置服务
命名服务
ADCC Core
一致性协议(priv-raft/sync renew/rdbms bases)
ADCC服务
集群管理
管理控制台

5 系统环境要求

ADCC产品支持Windows Server、Linux（包括采用龙芯和飞腾等国产芯片的Linux服务器）、Unix等多个操作系统平台的安装部署。

5.1 环境要求

表格1-1 软件及操作系统环境要求

组件	要求
操作系统	Windows系列； Linux Red Hat 5.2或以上； 国产操作系统如银河麒麟系列、中标麒麟系列、普华、中科红旗、深度等
CPU	X86、鲲鹏、飞腾、龙芯、申威等
内存	4G或以上
硬盘	可用空间100G或以上
数据库	达梦、人大金仓、瀚高、南大通用、海扬、MySQL等
浏览器	FireFox 70及以上、Chrome 60及以上、IE 11及以上

6 安装前准备

6.1 安装JDK

ADCC运行环境依赖于JAVA运行环境，在运行ADCC之前，需要具备。

目前支持的JDK厂商版本包括：Oracle JDK 8+、Open JDK 8+、IBM JDK 8+、毕昇JDK 8+、ApusicJDK 8+。

注意：如果是ARM架构环境，建议使用毕昇JDK 8+。

6.2 产品安装包

ADCC产品安装包：`ADCC-V1.0.XXX-Nacos-XXXXXXXX.tar.gz` 或 `ADCC-V1.0.XXX-Nacos-XXXXXXXX.zip`，如 `ADCC-V1.0.243-Nacos-20241218.tar.gz`；安装包适用于windows/linux等操作系统。

ADCC产品安装包目录结构和端口使用情况如下：

表2-1 ADCC产品包主要目录结构说明

目录	内容
bin	ADCC的脚本目录
conf	ADCC的配置文件目录
target	ADCC的程序包放置目录

表2-2 ADCC端口使用说明

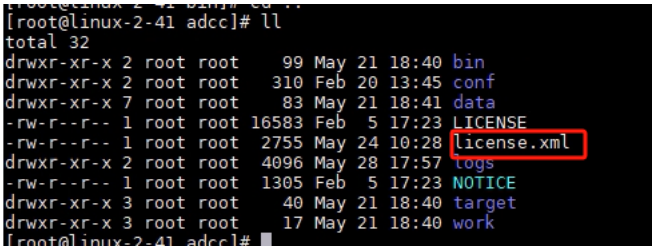
端口	说明
8848	主端口，客户端、控制台及OpenAPI所使用的HTTP端口
9848	客户端gRPC请求服务端端口，用于客户端向服务端发起连接和请求
9849	服务端gRPC请求服务端端口，用于服务间同步等
7848	Jraft请求服务端端口，用于处理服务端间的Raft相关请求

6.3 授权认证

ADCC for nacos启动需要依赖授权文件进行校验，一般情况下产品自带有效授权文件，如果授权文件将要过期或者已经过期，可联系金蝶天燕对接人员申领授权文件（license.xml），与此同时需要将产品特征码发给对接人员。

6.3.1 授权文件

获取到授权文件后，命名为 `license.xml`，放置ADCC安装目录下即可。



6.3.2 获取特征码

在 `${ADCC_HOME}/bin`，执行 `startup.sh -ac host`，host取值为ip地址或者网卡名称，类似如下：

```
./startup.sh -ac 172.20.140.17
```

打印特征码信息，类似如下，Auth Code=特征码内容：

```
ADCC JMX enabled by default
Auth Code=SZTY-777387783
```

获取特征码后再提供特征码申请授权文件。

6.3.3 集中授权

集中授权指连接授权中心，进行统一授权。需要先搭建金蝶Apusic授权中心，操作方式可参考《金蝶Apusic许可授权中心用户手册》，或联系金蝶天燕技术支持人员。

在系统环境配置环境变量，或在ADCC安装目录根目录 `${ADCC_HOME}` 下创建 `acls.properties` 文件，添加以下参数：

```
apusic_acls_enable=true
apusic_acls_authUrls=172.24.4.166:6886
apusic_acls_ns=apusic
apusic_acls_tenant=ApusicTest
```

连接参数说明：

参数名	参数值说明
apusic_acls_enable	是否开启授权中心认证，取值为true或false，为true则表示开启授权中心认证。没有该参数或该参数值为false，都表示没有开启授权中心认证；
apusic_acls_authUrls	授权中心的地址，可设置多个授权地址，格式为ip1:port1,ip2:port2，如果一个授权地址链接失败，会轮询其他的地址；如果开启授权中心认证，则为必填参数，其中端口为授权中心的https端口；
apusic_acls_ns	设置该实例所属的命名空间名称，可选参数；默认值为public，具体的命名空间可以在授权中心管理控制台-系统管理-授权管理查看。
apusic_acls_tenant	设置该实例所属的租户名称，可选参数。

ADCC启动时将会自动连接到Apusic授权中心。

6.4 安装数据库（单机模式下可选）

ADCC支持单机和集群部署模式。在单机模式下，ADCC提供内置数据源来实现数据的存储，可选择不使用外置数据源，但不方便观察数据存储的基本情况。

当前支持MySQL5.6.5+、达梦8、人大金仓V8、瀚高数据库安全版v4.5.8、Gbase 8s、OceanBase v4.x作为外置数据源来实现数据的存储。

6.5 负载均衡服务器(集群模式下要求)

集群模式通过配置负载均衡服务器作为统一的访问入口，将请求分摊到多个ADCC服务器上。

当前支持Nginx作为负载均衡服务器，版本要求：1.18.0+。使用nginx请求时，需要配置成TCP转发，不能配置http2转发，否则连接会被nginx断开。因此，在安装Nginx时需要启用stream模块，通过 `nginx -v` 查看是否包含 `--with-stream` 编译参数，包含则表示支持TCP转发。

```
# /usr/local/nginx/sbin/nginx -V
nginx version: nginx/1.18.0
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-44) (GCC)
configure arguments: --prefix=/usr/local/nginx --with-stream --with-pcre=/opt/nginx/pcre-8.34
```

6.6 Visual C++ 2015（Windows系列要求）

ADCC在Windows平台下使用时，需要依赖Microsoft Visual C++ 2015，要求版本在14.25.28508.3以上。

7 安装

7.1 单机部署模式

7.1.1 解压安装包

将安装包`ADCC-V1.0.243-Nacos-20241218.tar.gz`拷贝至指定目录，如`/opt`，执行解压命令，目录`/opt/adcc/`作为`${ADCC_HOME}`。

```
tar -zxvf ADCC-V1.0.243-Nacos-20241218.tar.gz
```

7.1.2 开启数据库配置（可选）

ADCC支持MySQL5.6.5+、达梦8、人大金仓V8、瀚高数据库安全版v4.5.8、Gbase、OceanBase作为外置数据源来实现数据的存储，可根据实际需要任选一种数据库类型进行配置，具体配置如下所示：

1）配置mysql数据库：

- 初始化数据库

  创建并初始化数据库adcc，数据库初始化文件：`${ADCC_HOME}/config/mysql-schema.sql`。具体操作如下

  ```
  #创建数据库adcc
  mysql -unext -pxxx  mysql -Ne "CREATE DATABASE adcc DEFAULT CHARSET utf8 COLLATE utf8_general_ci;"
  #初始化数据库adcc
  mysql -uroot -pxxx  adcc < ${ADCC_HOME}/config/mysql-schema.sql
  ```

- 开启数据源连接

  修改配置文件`${ADCC_HOME}/config/application.properties`。修改部分内容如下：

  ```
  spring.sql.init.platform=mysql

  ### Count of DB:
  db.num=1

  ### Connect URL of DB:
  db.url.0=jdbc:mysql://172.24.4.159:3306/adcc?
  characterEncoding=utf8&connectTimeout=20000&socketTimeout=60000&autoReconnect=true&useUnicode=true&us
  ### 指定数据库连接的用户名和密码
  db.user.0=xxxx
  db.password.0=xxxxxx
  ```

- 放置JDBC驱动包

  将mysql数据库的JDBC驱动包放置在`${ADCC_HOME}/target/ext`目录下。

2）配置达梦数据库：

- 初始化数据库

  创建并初始化数据库adcc，数据库初始化文件：`${ADCC_HOME}/config/dm-schema.sql`。

- 开启数据源连接

  修改配置文件`${ADCC_HOME}/config/application.properties`。修改部分内容如下：

```
spring.sql.init.platform=dm

### Count of DB:
db.num=1

#db.jdbcDriverName=xxx
db.jdbcDriverName=dm.jdbc.driver.DmDriver

### Connect URL of DB:
db.url.0=jdbc:dm://172.24.4.159:5236/ADCC
### 指定数据库连接的用户名和密码
db.user.0=xxxx
db.password.0=xxxxxx
```

- 放置JDBC驱动包

  将达梦数据库的JDBC驱动包放置在 `${ADCC_HOME}/target/ext` 目录下。

3）配置人大金仓数据库：

- 初始化数据库

  创建并初始化数据库adcc，数据库初始化文件： `${ADCC_HOME}/config/kingbase-schema.sql` 。

- 开启数据源连接

  修改配置文件 `${ADCC_HOME}/config/application.properties` 。修改部分内容如下：

```
spring.sql.init.platform=kingbase

### Count of DB:
db.num=1

#db.jdbcDriverName=xxx
db.jdbcDriverName=com.kingbase8.Driver

### Connect URL of DB:
db.url.0=jdbc:kingbase8://172.24.4.159:54321/adcc
### 指定数据库连接的用户名和密码
db.user.0=xxxx
db.password.0=xxxxxx
```

- 放置JDBC驱动包

  将人大金仓数据库的JDBC驱动包放置在 `${ADCC_HOME}/target/ext` 目录下。

4）配置瀚高数据库：

- 初始化数据库

  创建并初始化数据库adcc，数据库初始化文件： `${ADCC_HOME}/config/hgdb-schema.sql` 。

- 开启数据源连接

  修改配置文件 `${ADCC_HOME}/config/application.properties` 。修改部分内容如下：

```
spring.sql.init.platform=hgdb

### Count of DB:
db.num=1

#db.jdbcDriverName=xxx
db.jdbcDriverName=com.highgo.jdbc.Driver

### Connect URL of DB:
db.url.0=jdbc:highgo://172.24.4.159:5866/adcc
### 指定数据库连接的用户名和密码
db.user.0=xxxx
db.password.0=xxxxxx
```

- 放置JDBC驱动包

  将瀚高数据库的JDBC驱动包放置在 `${ADCC_HOME}/target/ext` 目录下。

5）配置gbase数据库：

- 初始化数据库

  创建数据库adcc，创建数据库时需加上 `dbcompatibility` 参数，且值不能为A。例如： `create database adcc dbcompatibility 'PG';`

  初始化数据库adcc，数据库初始化文件： `${ADCC_HOME}/config/gbase-schema.sql` 。

- 开启数据源连接

  修改配置文件 `${ADCC_HOME}/config/application.properties` 。修改部分内容如下：

```
spring.sql.init.platform=gbase

### Count of DB:
db.num=1

#db.jdbcDriverName=xxx gbase不同版本的驱动类不一样，根据实际情况填写
db.jdbcDriverName=com.gbase.Driver

### Connect URL of DB:
db.url.0=jdbc:gbase://172.24.1.1:15400/adcc
### 指定数据库连接的用户名和密码
db.user.0=xxxx
db.password.0=xxxxxx
```

- 放置JDBC驱动包

  将gbase数据库的JDBC驱动包放置在 `${ADCC_HOME}/target/ext` 目录下。

6）配置OceanBase数据库：

- 初始化数据库

  创建并初始化数据库adcc，数据库初始化文件： `${ADCC_HOME}/config/oceanbase-schema.sql` 。

- 开启数据源连接

  修改配置文件 `${ADCC_HOME}/config/application.properties` 。修改部分内容如下：

```
spring.sql.init.platform=oceanbase

### Count of DB:
db.num=1

#db.jdbcDriverName=xxx
db.jdbcDriverName=com.alipay.oceanbase.jdbc.Driver

### Connect URL of DB:
db.url.0=jdbc:oceanbase://172.24.1.1:2881/adcc?useUnicode=true&characterEncoding=utf-8
### 指定数据库连接的用户名和密码
db.user.0=xxxx
db.password.0=xxxxxx
```

- 放置JDBC驱动包

  将OceanBase数据库的JDBC驱动包放置在 `${ADCC_HOME}/target/ext` 目录下。

7.2 集群部署模式

集群部署模式下需要3个或3个以上ADCC节点、负载均衡服务器和数据库才能构成集群。

7.2.1 解压安装包

将安装包 `ADCC-V1.0.243-Nacos-20241218.tar.gz` 拷贝至指定目录，如 `/opt` ，执行解压命令，目录 `/opt/adcc/` 作为 `${ADCC_HOME}` 。

```
tar -zxvf ADCC-V1.0.243-Nacos-20241218.tar.gz
```

每个节点都需要安装ADCC。

7.2.2 开启数据库配置（可选）

每个ADCC节点服务器上，配置同一个数据库，可使用主备模式或高可用数据库。具体配置如“单机部署模式”的“开启数据库配置”所示。

7.2.3 开启集群配置

每个ADCC节点服务器的配置文件目录 `conf` 下，拷贝 `cluster.conf.example` 为 `cluster.conf` :

操作示例如下:

```
 cd $ADCC_HOME/config/
 cp cluster.conf.example cluster.conf
```

将各个节点的IP和端口填写到 `cluster.conf` ，每行配置成 `ip:port` 。（配置3个或3个以上节点）

```
# ip:port
172.24.4.152:8848
172.24.4.154:8848
172.24.4.163:8848
```

7.2.4 开启负载均衡配置

以Nginx为例，配置负载均衡。

修改配置文件， `${NGINX_HOME}/conf/nginx.conf` ，修改其内容如下所示:

```
worker_processes  1;
```

```
events {
    worker_connections  1024;
}

#http转发
http {
    include       mime.types;
    default_type  application/octet-stream;

    sendfile        on;

    keepalive_timeout  65;

    #adcc集群各节点ip:port
    upstream adcc-cluster {
        server 172.24.4.163:8848;
        server 172.24.4.152:8848;
        server 172.24.4.154:8848;
    }

    server {
        listen       8848;
        #负载均衡IP
        server_name  172.24.4.171;

        location / {
            #root   html;
            #index  index.html index.htm;
            proxy_pass http://adcc-cluster;
        }
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }

    }
}

#TCP转发
stream {
   upstream adcc-tcp{
        server 172.24.4.163:9848;
        server 172.24.4.152:9848;
        server 172.24.4.154:9848;
   }
   server {
        listen 9848;
        proxy_pass   adcc-tcp;
   }

}
```

8 启动

8.1 单机部署模式

以Linux为例，在 `${ADCC_HOME}/bin` 目录下，运行 `startup.sh -m standalone` 进行启动。

```
 cd ${ADCC_HOME}/bin
./startup.sh -m standalone
```

查看进程；或打开浏览器，输入 http://ip:port/adcc/index.html 进行访问；默认端口为8848（http）；如果访问成功，则说明ADCC安装成功。

8.2 集群部署模式

以Linux为例，在各个节点 `${${ADCC_HOME}}/bin` 目录下，运行 `startup.sh` 进行启动。

```
 cd ${ADCC_HOME}/bin
./startup.sh
```

启动负载均衡器Nginx。

查看进程；或打开浏览器，输入 http://ip:port/adcc/index.html 进行访问；其中ip为负载均衡器的IP，端口为负载均衡器的端口；如果访问成功，则说明ADCC安装成功。

9 停止

在 `${ADCC_HOME}/bin` 目录下，运行 `shutdown.sh` 停止ADCC。

```
cd ${ADCC_HOME}/bin
./shutdown.sh
```

10 卸载

删除ADCC安装部署目录。如安装部署目录`${ADCC_HOME}`为`/opt/adcc`，将该目录删除即可。

11 云部署

ADCC支持云部署。

以下说明是一些基本的使用，实际使用的时候应该根据情况进行调整。

11.1 build docker镜像

11.1.1 build adcc镜像

1. 联系销售人员获取adcc产品包。
2. 确定license授权方式，adcc支持两种license授权方式：
   - 根据ip授权：传统的授权方式，一个ip地址一个授权文件。从销售人员处获取该授权文件，授权文件名为 `license.xml` 。
   - 集中授权：不需要授权文件，但需要先搭建apusic授权中心，该中心的搭建可以咨询技术支持人员。然后修改Dockerfile，将 `APUSIC_ACLS_ENABLE` 设置为true，添加 `APUSIC_ACLS_URLS` 参数并指向授权中心地址，注释或者删掉 `ADD license.xml license.xml` 那一行。

```
    CLASSPATH=".:/home/apusic/adcc/conf:$CLASSPATH" \
    CLUSTER_CONF="/home/apusic/adcc/conf/cluster.conf" \
    FUNCTION_MODE="all" \
    JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk" \
    NACOS_USER="nacos" \
    JAVA="/usr/lib/jvm/java-1.8.0-openjdk/bin/java" \
    JVM_XMS="1g" \
    JVM_XMX="1g" \
    JVM_XMN="512m" \
    JVM_MS="128m" \
    JVM_MMS="320m" \
    ADCC_DEBUG="n" \
    ACCESSLOG_ENABLED="false" \
    TIME_ZONE="Asia/Shanghai" \
    APUSIC_ACLS_ENABLE="true"
    APUSIC_ACLS_URLS=授权中心地址

ARG ADCC_VERSION=1.0.230
ARG HOT_FIX_FLAG=""

WORKDIR $BASE_DIR

RUN curl http://mirrors.aliyun.com/repo/Centos-7.repo > /etc/yum.repos.d/CentOS-Base.repo
RUN yum clean all && yum makecache

RUN set -x \
    && yum update -y \
    && yum install -y java-1.8.0-openjdk java-1.8.0-openjdk-devel iputils nc vim libcurl \
    && yum clean all

COPY adcc-server-1.0.230.tar.gz /home/apusic
RUN tar -xzvf /home/apusic/adcc-server-1.0.230.tar.gz -C /home/apusic \
    && rm -rf /home/apusic/adcc-server-1.0.230.tar.gz \
    && ln -snf /usr/share/zoneinfo/$TIME_ZONE /etc/localtime && echo $TIME_ZONE > /etc/timezone

#use license.xml
#ADD license.xml license.xml
```

3. 添加数据驱动。adcc自带mysql数据库驱动，如果需要使用其他数据库，则需要将驱动拷贝到 `images/adcc` 目录下，并在Dockerfile中添加一条指令，例如：

```
ADD kingbase8-8.2.0.jar target/ext/kingbase8-8.2.0.jar
```

4. 创建Dockerfile。下例为在X86操作系统中构建的Dockerfile，安装包名称需要根据实际情况修改。

```
FROM harbor.apusic.com/public/centos:7.5


# set environment
ENV MODE="standalone" \
    PREFER_HOST_MODE="ip"\
    BASE_DIR="/home/apusic/adcc" \
    CLASSPATH=".:/home/apusic/adcc/conf:$CLASSPATH" \
    CLUSTER_CONF="/home/apusic/adcc/conf/cluster.conf" \
```

```
    FUNCTION_MODE="all" \
    JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk" \
    NACOS_USER="nacos" \
    JAVA="/usr/lib/jvm/java-1.8.0-openjdk/bin/java" \
    JVM_XMS="1g" \
    JVM_XMX="1g" \
    JVM_XMN="512m" \
    JVM_MS="128m" \
    JVM_MMS="320m" \
    ADCC_DEBUG="n" \
    ACCESSLOG_ENABLED="false" \
    TIME_ZONE="Asia/Shanghai" \
    APUSIC_ACLS_ENABLE="false"

ARG ADCC_VERSION=1.0.243
ARG HOT_FIX_FLAG=""

WORKDIR $BASE_DIR

RUN curl http://mirrors.aliyun.com/repo/Centos-7.repo > /etc/yum.repos.d/CentOS-Base.repo
RUN yum clean all && yum makecache

RUN set -x \
    && yum update -y \
    && yum install -y java-1.8.0-openjdk java-1.8.0-openjdk-devel iputils nc vim libcurl \
    && yum clean all

COPY adcc-server-1.0.tar.gz /home/apusic
RUN tar -xzvf /home/apusic/adcc-server-1.0.tar.gz -C /home/apusic \
    && rm -rf /home/apusic/adcc-server-1.0.tar.gz \
    && ln -snf /usr/share/zoneinfo/$TIME_ZONE /etc/localtime && echo $TIME_ZONE > /etc/timezone

#use license.xml
ADD license.xml license.xml

#add database driver
#ADD xxx.jar target/ext/xxx.jar

ADD application.properties conf/application.properties

# set startup log dir
RUN mkdir -p logs \
        && touch logs/start.out \
        && ln -sf /dev/stdout start.out \
        && ln -sf /dev/stderr start.out
RUN chmod +x bin/docker-startup.sh
RUN chmod -c 755 plugins/peer-finder/*.sh plugins/peer-finder/peer-finder
```

```
EXPOSE 8848
ENTRYPOINT ["bin/docker-startup.sh"]
```

5. 然后执行`docker build`命令：

```
docker build -t adcc/adcc-server:1.0.243 .
```

6. 构建成功后，通过`docker images`命令可以查看构建好的镜像。

7. 关于Dockerfile文件中环境变量的说明

   - `BASE_DIR`变量暂时不支持修改；
   - 其他变量除了license相关的外也没必要直接修改Dockerfile文件，而是在k8s或docker-compose的yaml文件中去指定，这样可以保证构建的镜像具备一定的通用性。

11.1.2 build mysql镜像

如果需要使用mysql数据库，可以进入`images/mysql`目录进行构建，并根据需要修改Dockerfile；也可以使用自己的mysql镜像，或者搭建外部mysql服务器。

11.2 配置参数

参数	描述	参考值
SPRING_DATASOURCE_PLATFORM	数据库类型	embedded（默认值，使用内置的derby），mysql，dm（达梦数据库），kingbase（人大金仓数据库），hgdb（瀚高数据库）
DB_URL	数据库连接URL	无
DB_USER	数据库连接用户名	无
DB_PASSWORD	数据库连接密码	无
DB_DRIVER_CLASSNAME	数据库连接驱动类	embedded和mysql数据库不需要填写，其他数据库需要填写
ACCESSLOG_ENABLED	访问日志	默认值false
ADCC_AUTH_ENABLE	是否开启认证	默认false
ADCC_AUTH_IDENTITY_KEY	认证identity key	ADCC_AUTH_ENABLE为true时需要
ADCC_AUTH_IDENTITY_VALUE	认证identity value	ADCC_AUTH_ENABLE为true时需要
ADCC_AUTH_TOKEN	认证 token	ADCC_AUTH_ENABLE为true时需要
MODE	adcc启动模式	支持standalone和cluster
ADCC_SERVERS	ADCC集群实例列表	adcc集群模式需要设置，格式为IP:端口，多个实例用空格分隔。使用k8s-operator创建集群时不要设置。
JVM_XMS、JVM_XMX、JVM_XMN、 JVM_MS、JVM_MMS	对应jvm相应参数	默认值分别为： 1g、1g、512m、128m、320m
APUSIC_ACLS_ENABLE	是否开启license集中授权	默认值false
APUSIC_ACLS_URLS	license集中授权服务器地址	APUSIC_ACLS_ENABLE为true时需要配置
APUSIC_ACLS_TENANT	租户	APUSIC_ACLS_ENABLE为true时才需要配置，默认public
APUSIC_ACLS_NS	命名空间	APUSIC_ACLS_ENABLE为true时才需要配置，默认public

上述参数有的是在启动脚本`docker-startup.sh`中定义的，有的是在adcc配置文件`conf/application.properties`中定义的。对于`conf/application.properties`中的相关参数，上述表格如果无法满足需求，可以自定义一份`application.properties`，替换`images/adcc`下的同名文件，然后再开始构建。

12 ADCC for nacos控制台

ADCC for nacos控制台主要旨在于增强对于服务列表，健康状态管理，服务治理，分布式配置管理等方面的管控能力，以便进一步帮助用户降低管理微服务应用架构的成本，将提供包括下列基本功能:

- 登录管理
- 服务管理
    - 服务列表及服务健康状态展示
    - 服务元数据存储及编辑
    - 服务流量权重的调整
    - 服务优雅上下线
- 配置管理
    - 多种配置格式编辑
    - 编辑DIFF
    - 示例代码
    - 推送状态查询
    - 配置版本及一键回滚
- 命名空间

12.1 登录管理

ADCC for nacos默认不开启鉴权，无需使用用户名和密码登录，即可访问控制台。如若开启鉴权，登录需要用户名密码。

12.2 服务管理

开发者或者运维人员往往需要在服务注册后，通过友好的界面来查看服务的注册情况，包括当前系统注册的所有服务和每个服务的详情。并在有权限控制的情况下，进行服务的一些配置的编辑操作。ADCC for nacos在这个版本开放的控制台的服务发现部分，主要就是提供用户一个基本的运维页面，能够查看、编辑当前注册的服务。

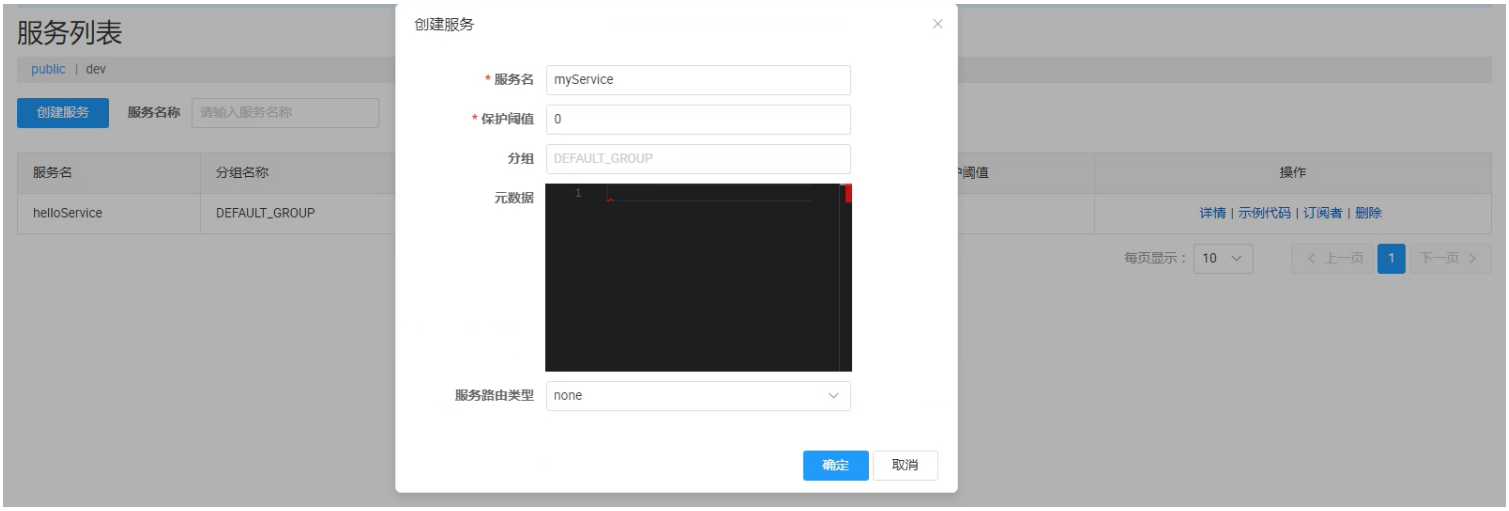
12.2.1 服务列表管理

服务列表帮助用户以统一的视图管理其所有的微服务以及服务健康状态。服务列表主要展示服务名、分组名称、集群数目、实例数、健康实例数、触发保护阈值和操作按钮。

12.2.2 创建服务

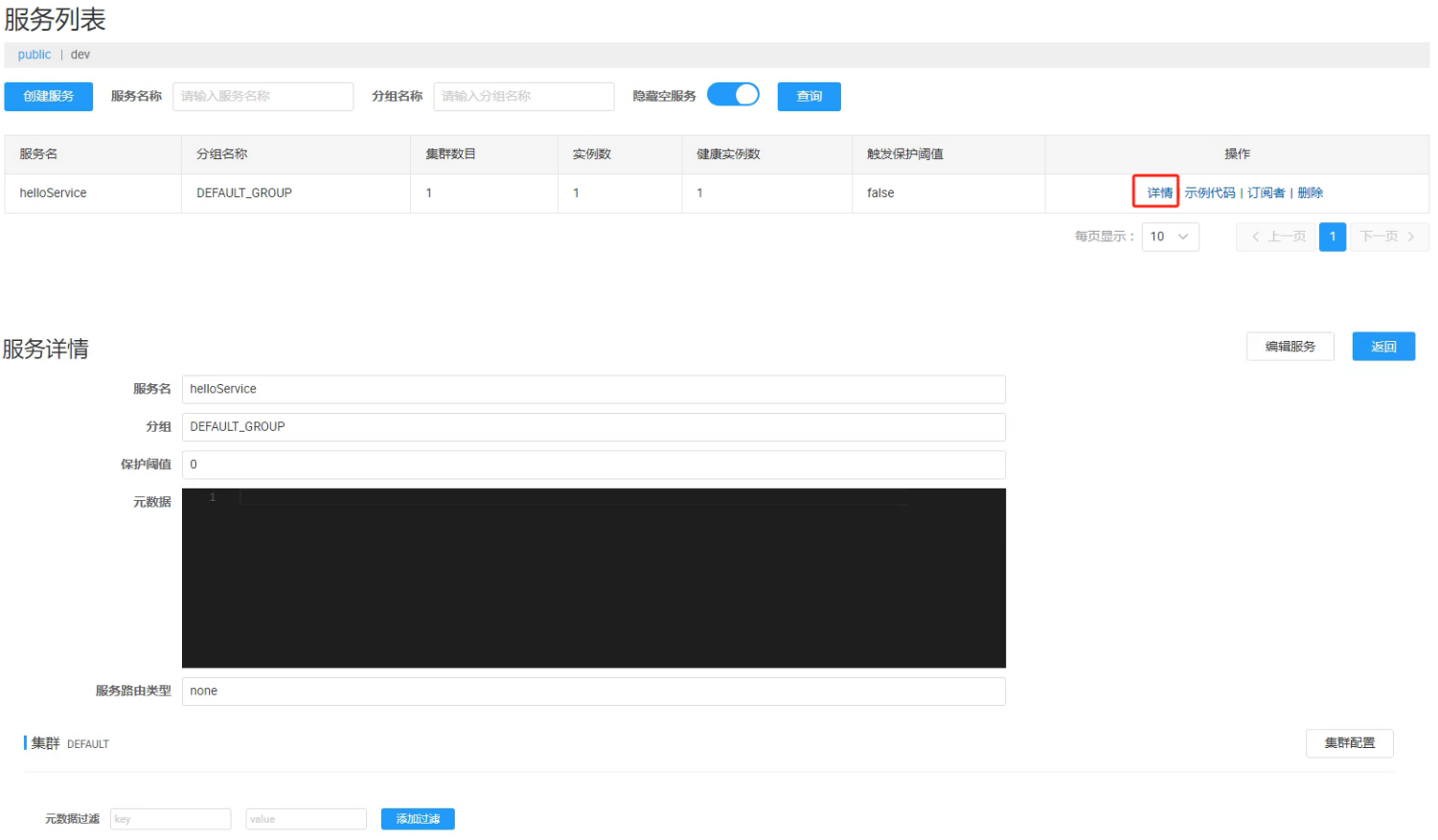
为方便操作，可通过服务列表页面创建服务。点击“创建服务”，进入创建服务页面。

配置项	说明	默认值
服务名	定义的服务名称，必填	
保护阈值	设置保护阈值，必填	
分组	设置该服务所属组	DEFAULT_GROUP
元数据	配置该服务的元数据	
服务路由类型	设置该服务的路由类型，默认可选项为none、label	none



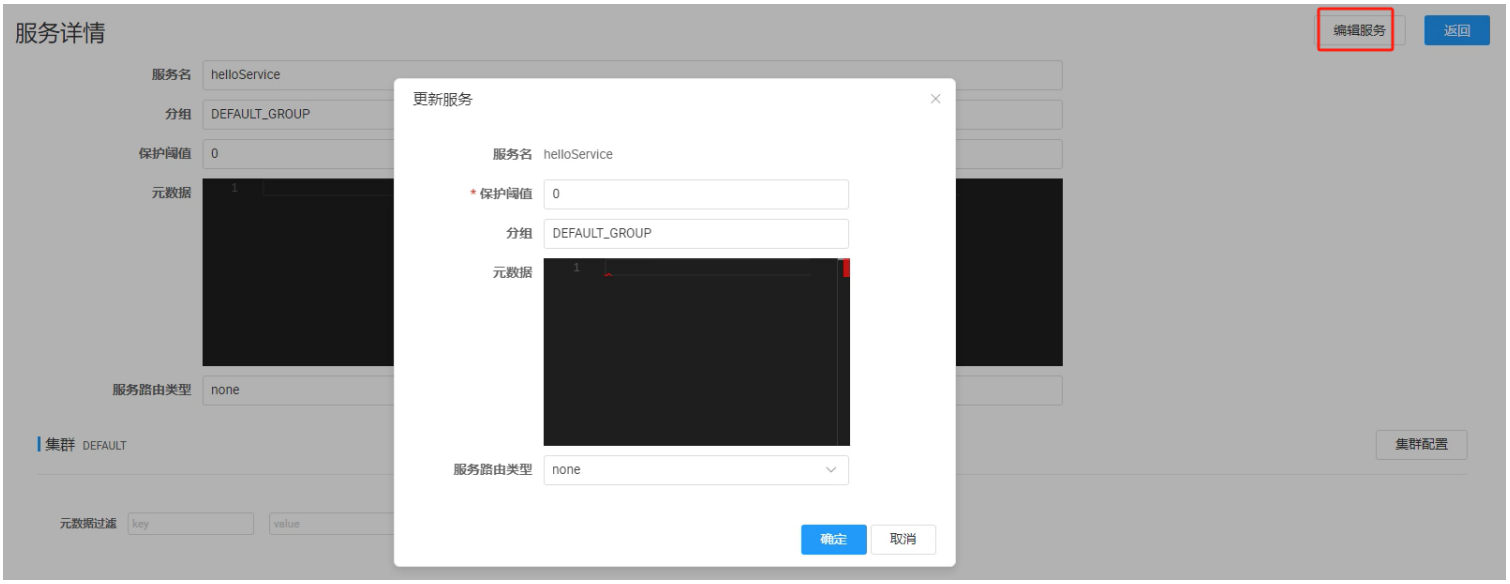
12.2.3 查看服务详情

在服务列表页面点击详情，可以看到服务的详情。可以查看服务、集群和实例的基本信息。



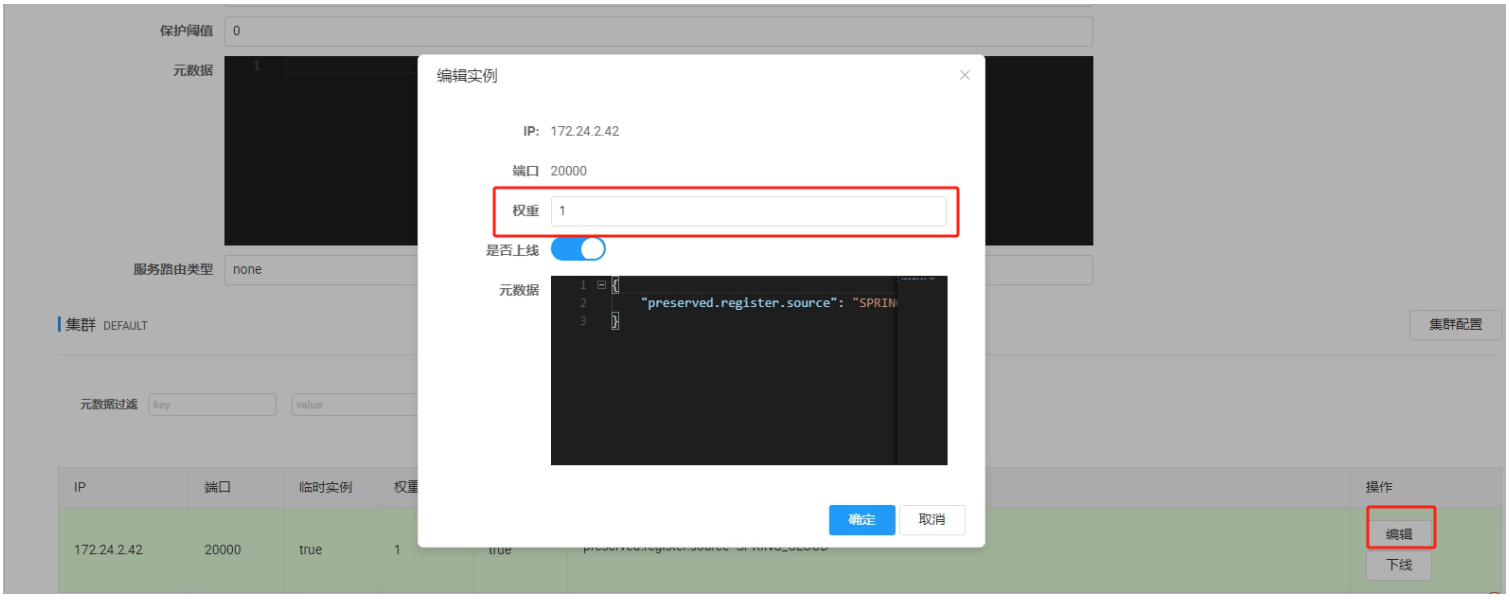
12.2.4 编辑服务信息

进入服务详情页面，点击"编辑服务"进入更新服务页面，更新服务信息。



12.2.5 服务流量权重支持及流量保护

ADCC for nacos 为用户提供了流量权重控制的能力，同时开放了服务流量的阈值保护，以帮助用户更好的保护服务服务提供者集群不被意外打垮。如下图所以，可以点击实例的编辑按钮，修改实例的权重。如果想增加实例的流量，可以将权重调大，如果不想实例接收流量，则可以将权重设为0。



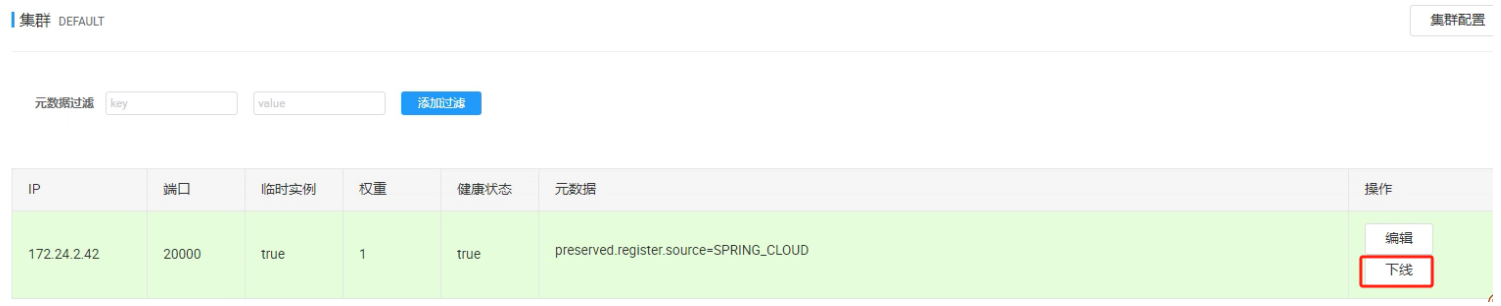
12.2.6 服务元数据管理

ADCC for nacos提供多个维度的服务元数据的暴露，帮助用户存储自定义的信息。这些信息都是以K-V的数据结构存储，在控制台上，会以k1=v1,k2=v2这样的格式展示。类似的，编辑元数据可以通过相同的格式进行。例如服务的元数据编辑，首先点击服务详情页右上角的"编辑服务"按钮，然后在元数据输入框输入：version=1.0,env=prod。

点击确认，就可以在服务详情页面，看到服务的元数据已经更新了。

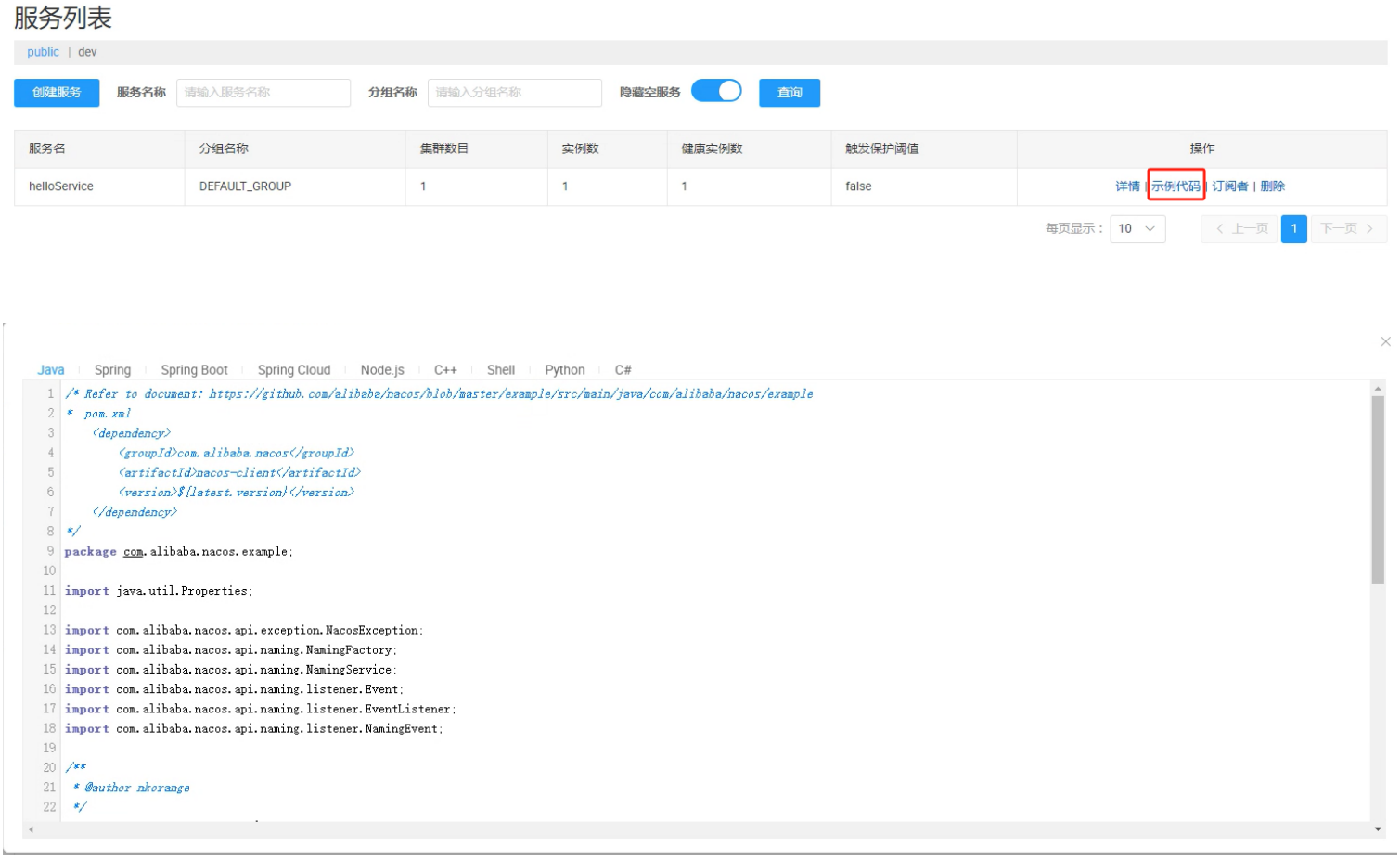
12.2.7 服务优雅上下线

ADCC for nacos还提供服务实例的上下线操作，在服务详情页面，可以点击实例的"上线"或者"下线"按钮，被下线的实例，将不会包含在健康的实例列表里。



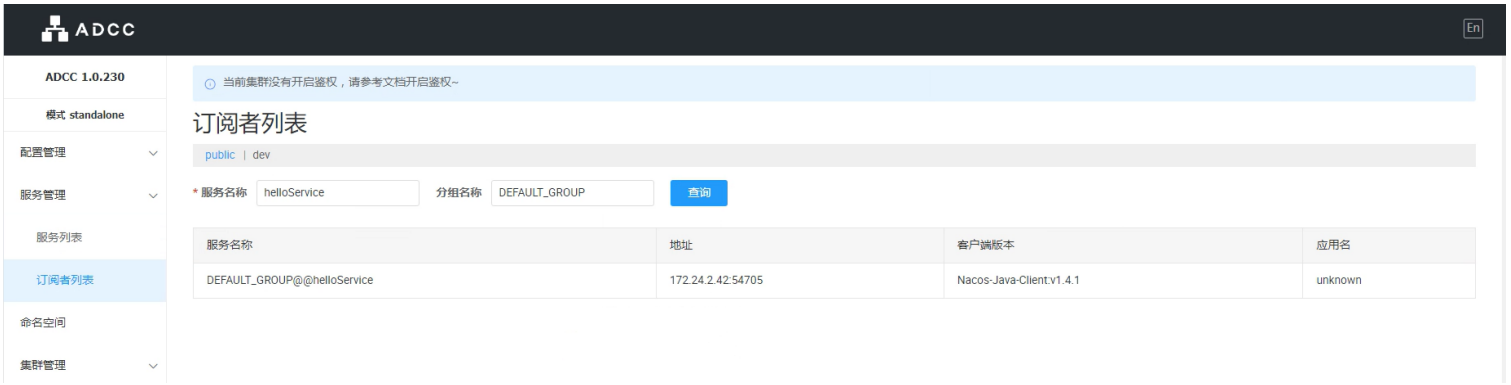
12.2.8 示例代码

ADCC for nacos提供示例代码能力，能够快速了解服务配置。



12.3 订阅者列表

当服务提供者注册实例到ADCC for nacos服务端后，服务消费者就需要订阅提供者服务来进行调用。在订阅者列表可以看到服务的订阅者信息。

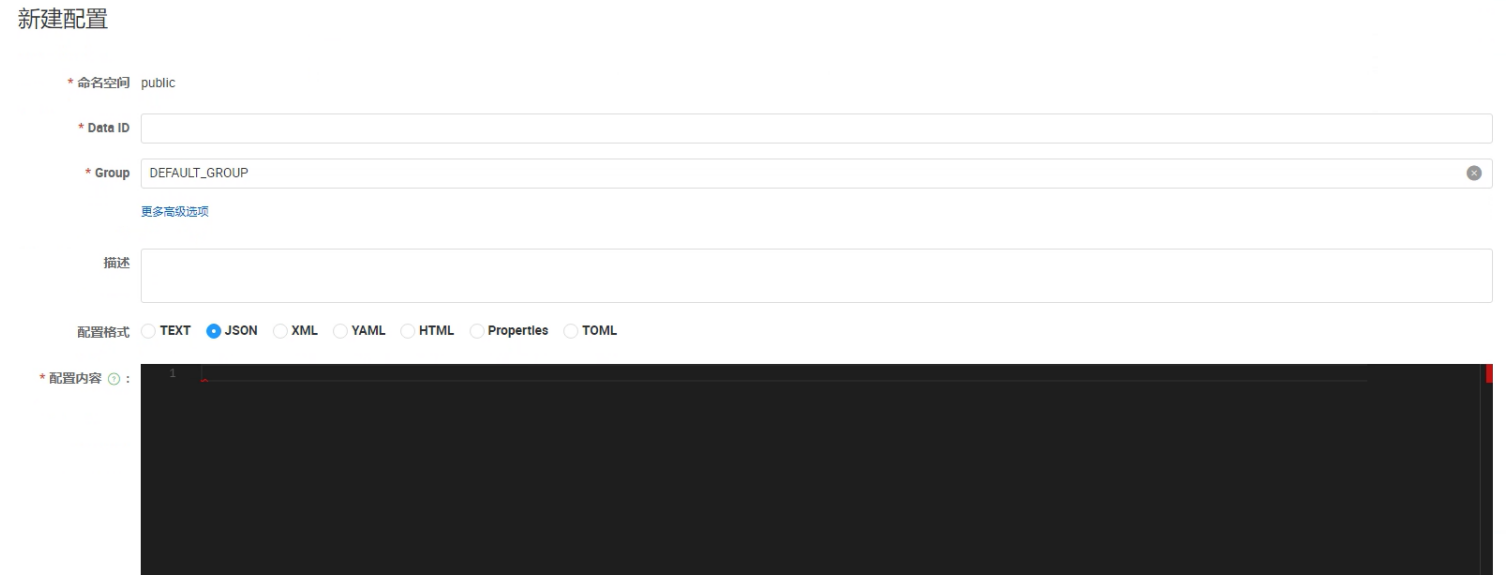


12.4 配置管理

ADCC for nacos支持基于Namespace和Group的配置分组管理，以便用户更灵活的根据自己的需要按照环境或者应用、模块等分组管理微服务以及Spring的大量配置，在配置管理中主要提供了配置历史版本、回滚、订阅者查询等核心管理能力。

12.4.1 创建配置

为方便操作，可通过配置列表页面创建配置。点击"创建配置"，进入创建配置页面。



12.4.2 多配置格式编辑器

ADCC for nacos支持 YAML、Properties、TEXT、JSON、XML、HTML 等常见配置格式在线编辑、语法高亮、格式校验，帮助用户高效编辑的同时大幅降低格式错误带来的风险。

ADCC for nacos支持配置标签的能力，帮助用户更好、更灵活的做到基于标签的配置分类及管理。同时支持用户对配置及其变更进行描述，方便多人或者跨团队协作管理配置。

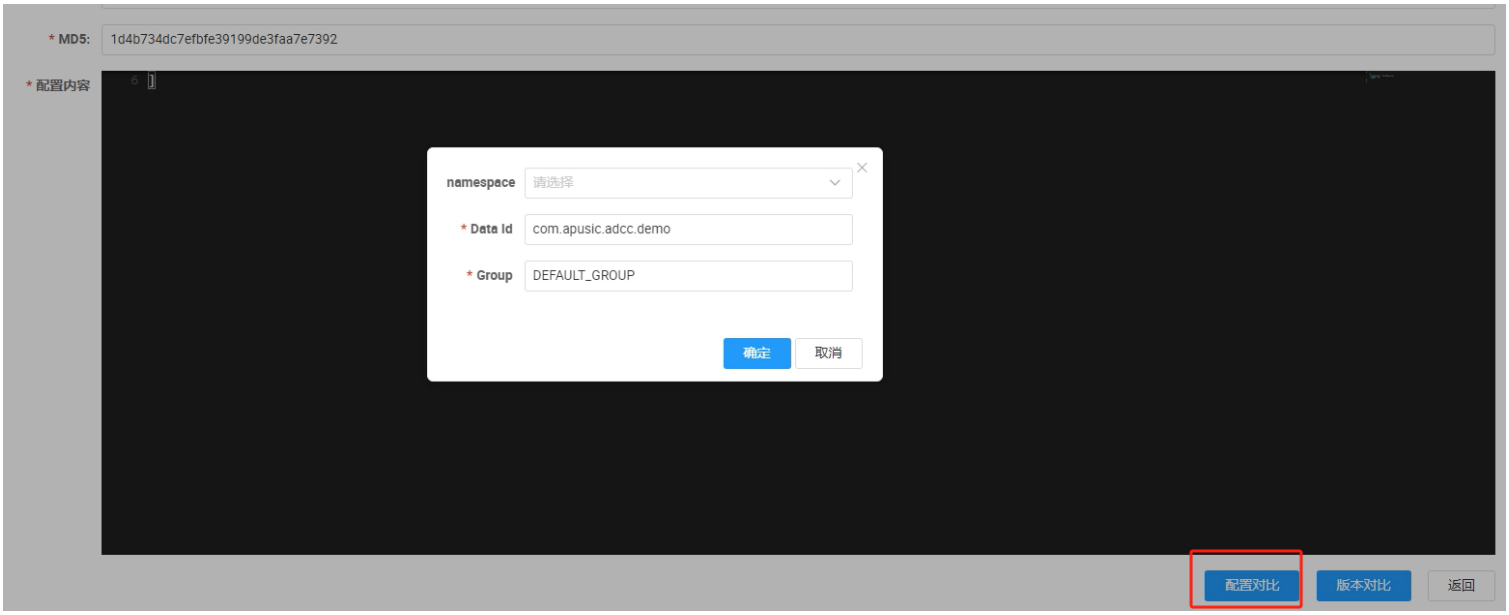
12.4.3 导入配置

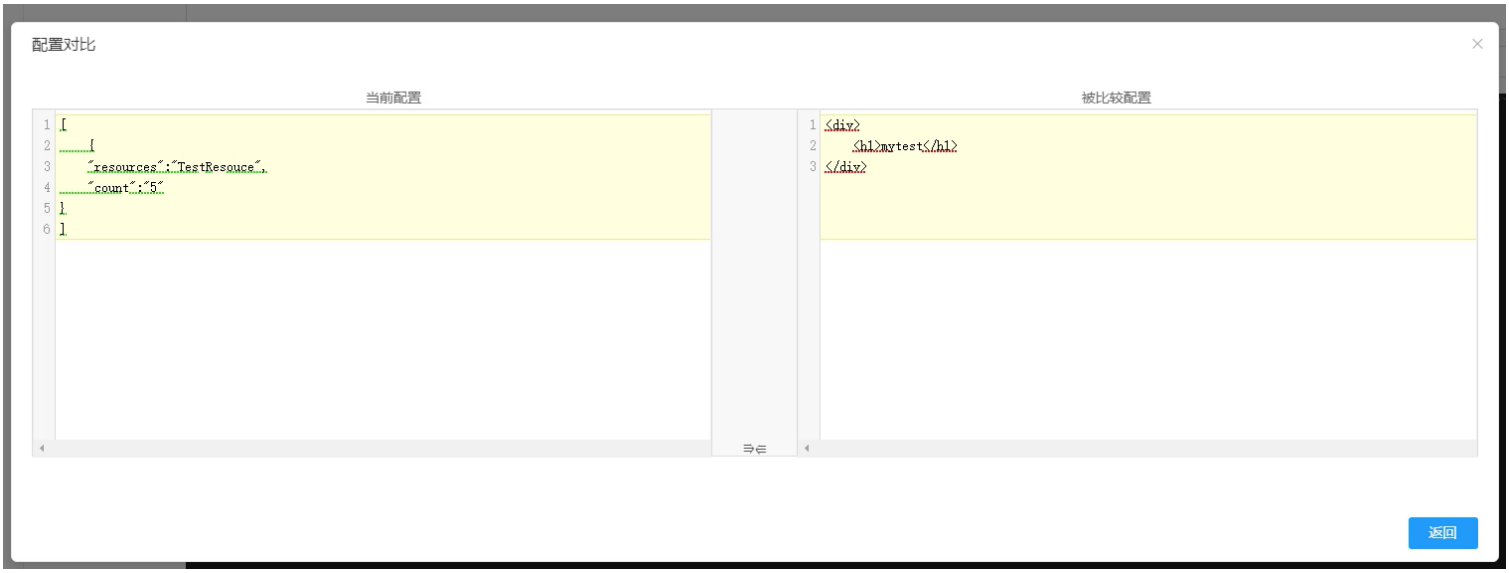
ADCC for nacos支持通过管控台导入配置信息。点击“导入配置”按钮，上传配置文件即可完成导入。

12.4.4 查看配置信息

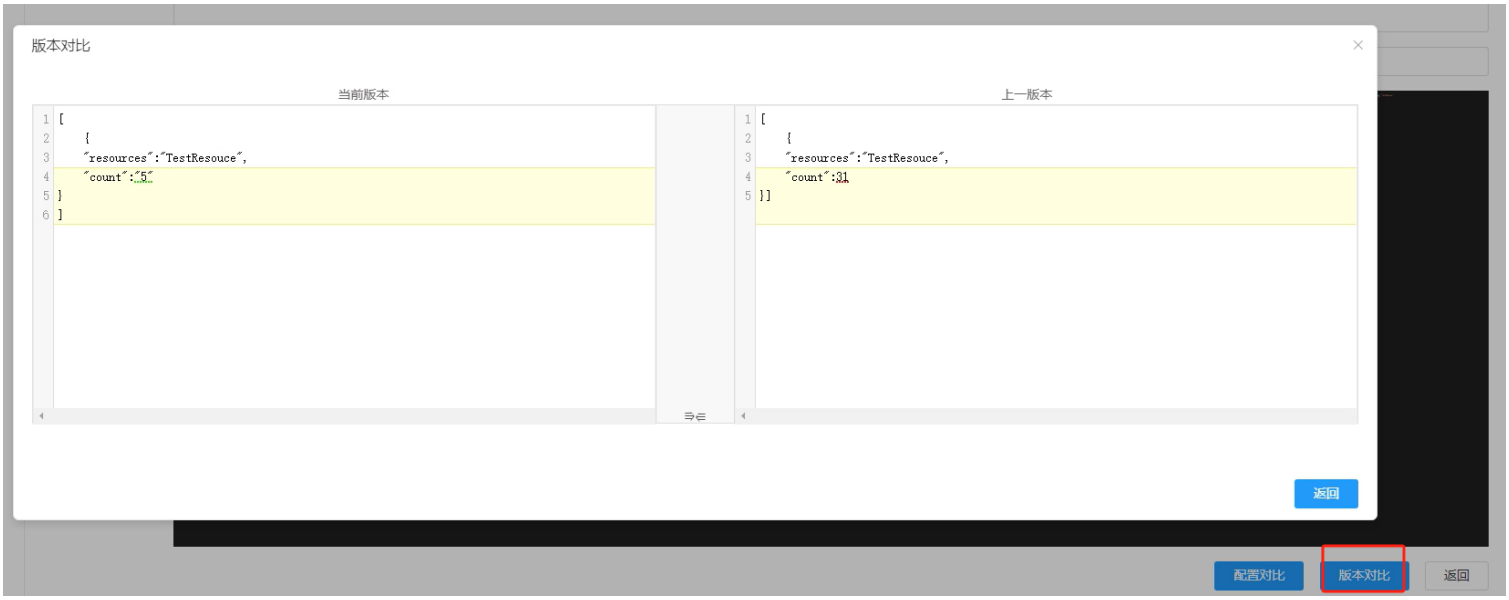
在配置列表页面，点击“详情”进入配置详情页面。

点击“配置对比”，可以与其他配置进行对比，查看差异。





点击“版本对比”，可以与其他版本进行对比，查看差异。

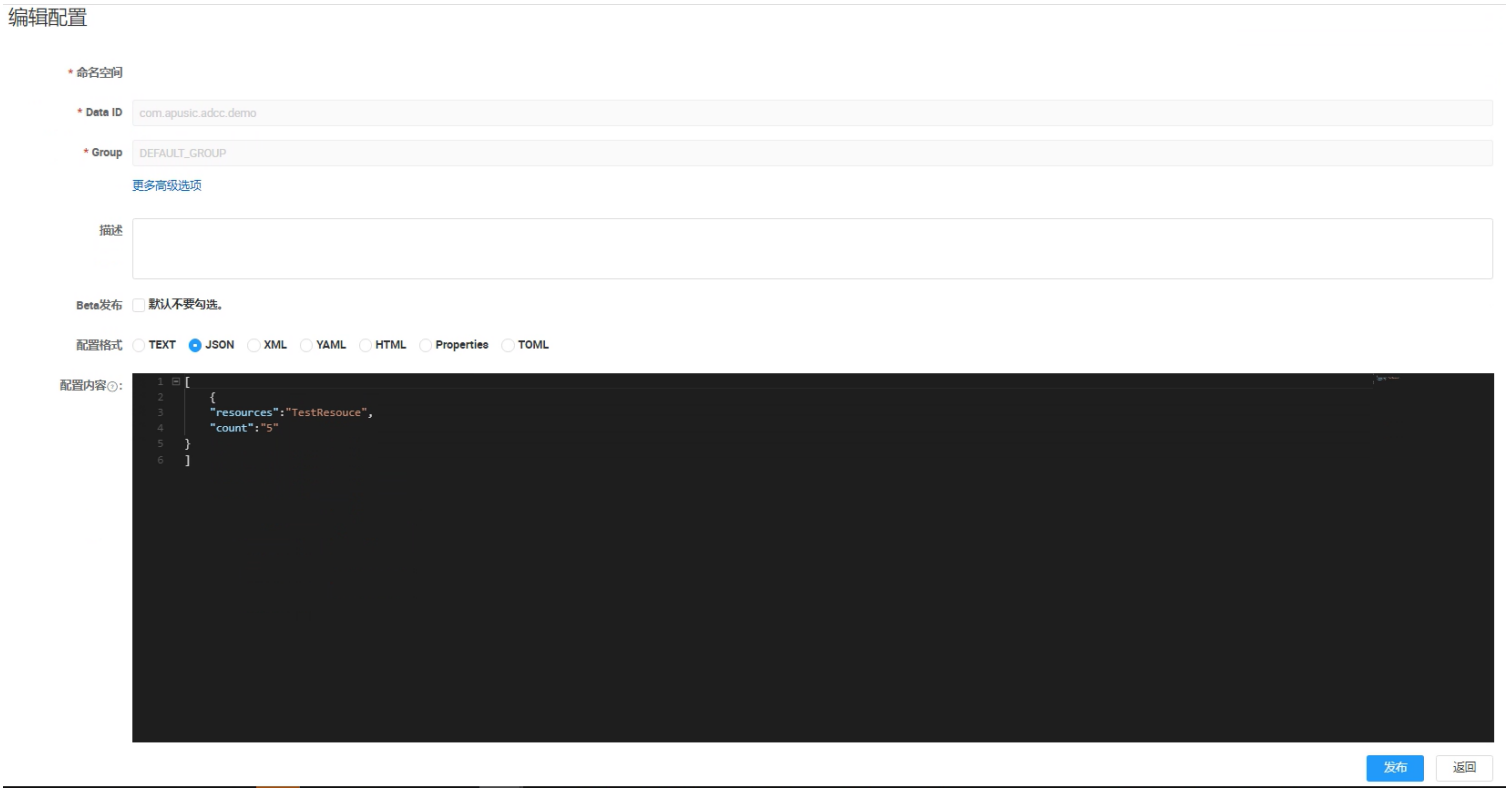


12.4.5 示例代码

ADCC for nacos提供示例代码能力，能够让新手快速使用客户端编程消费该配置，大幅降低新手使用门槛。

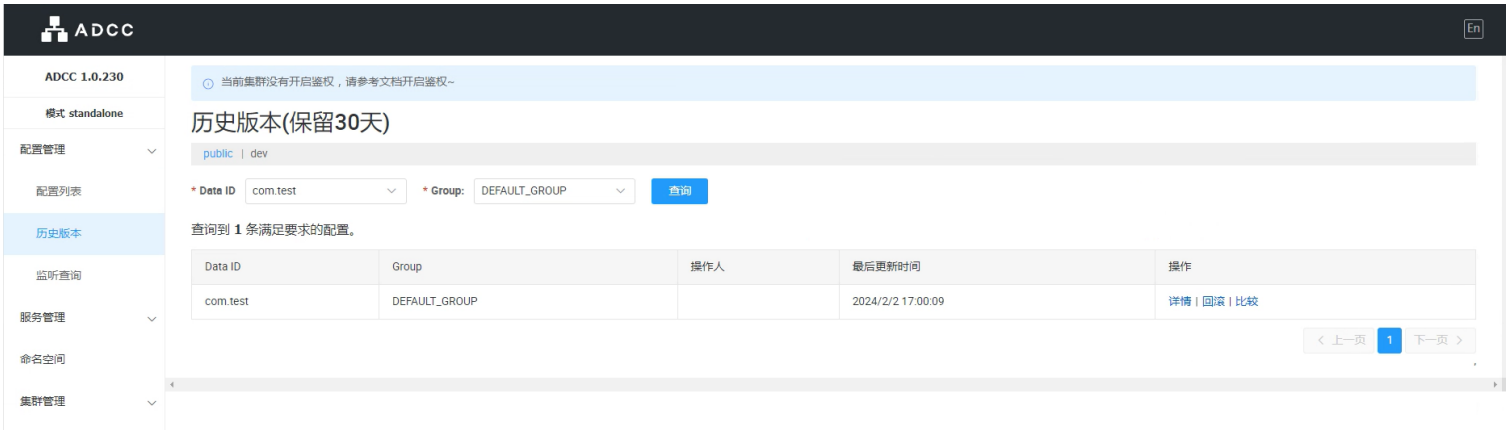
12.4.6 编辑配置

ADCC for nacos支持通过控制台编辑配置信息，点击操作中的"编辑"进入编辑页面。



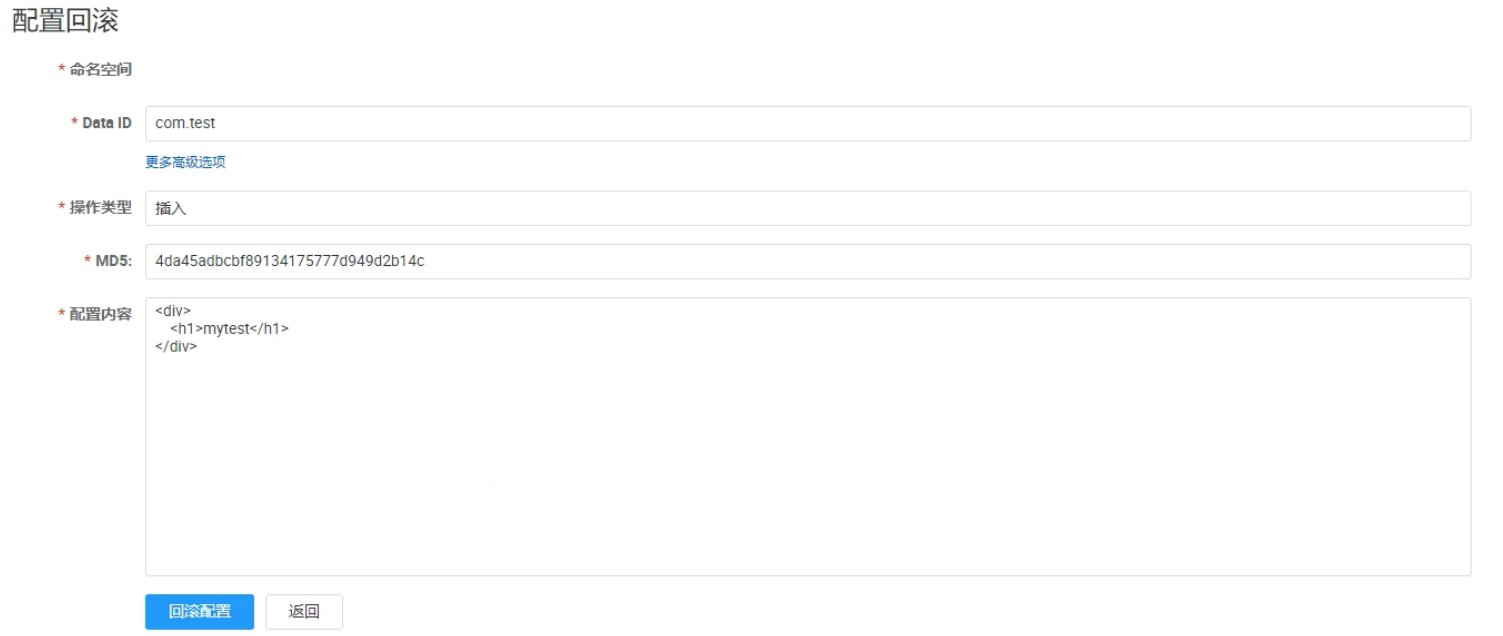
12.4.7 历史版本

ADCC for nacos通过提供配置版本管理及其一键回滚能力，帮助用户改错配置的时候能够快速恢复，降低微服务系统在配置管理上的一定会遇到的可用性风险。点击导航栏【历史版本】进入历史版本页面；或进入配置列表页面，点击操作中的更多，点击"历史版本"，可进入历史版本页面。



点击操作中的"详情"，可查看该版本的详细信息。

点击操作中的"回滚"，进入配置回滚页面。可回滚配置。

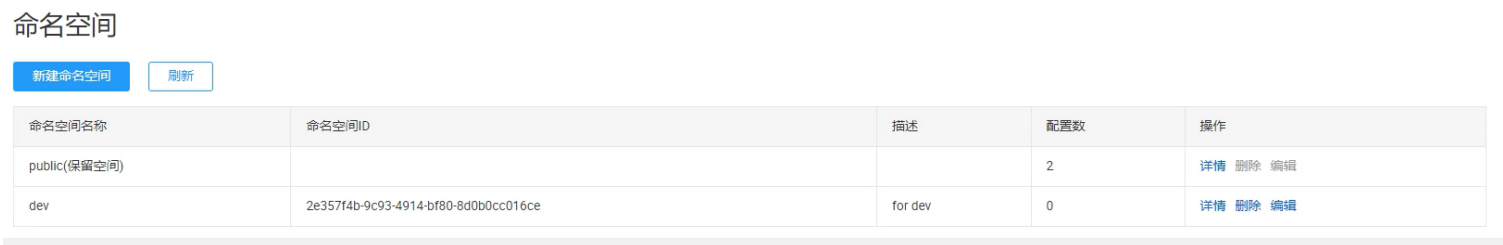


12.4.8 监听查询

ADCC for nacos提供配置订阅者即监听者查询能力，同时提供客户端当前配置的MD5校验值，以便帮助用户更好的检查配置变更是否推送到 Client 端。

12.5 命名空间管理

ADCC for nacos 基于Namespace 帮助用户逻辑隔离多个命名空间，这可以帮助用户更好的管理测试、预发、生产等多环境服务和配置，让每个环境的同一个配置（如数据库数据源）可以定义不同的值。



12.6 开启鉴权

ADCC for nacos默认不开启鉴权，若开启鉴权，需要在使用用户名和密码登录之后，才能正常使用ADCC for nacos。默认的用户名为adcc，密码为apusic@2024。

操作示例如下，修改配置文件 `${ADCC_HOME}/config/application.properties` 中的部分内容：

```
#兼容nacos鉴权
nacos.core.auth.system.type=nacos
#开启鉴权设置为true，关闭鉴权设置为false
nacos.core.auth.enabled=true
#只认证管控不认证应用
nacos.core.auth.consoleOnly=true

#设置apusic鉴权标签
nacos.core.auth.server.identity.key=apusic-key
nacos.core.auth.server.identity.value=apusic-value
#设置JWT令牌的密钥
nacos.core.auth.plugin.nacos.token.secret.key=U2VjcmV0S2V5YXB1c2ljMTIzNDU2NzhhcHVzaWMxMjM0NTY3ODkwMTIzNDU2Nz
```

【注意】鉴权开关是修改之后立马生效的，不需要重启ADCC for nacos服务端。

12.7 修改JWT令牌的密钥

ADCC for nacos提供生成JWT令牌的密钥的功能，进入ADCC for nacos的脚本目录${${ADCC_HOME}}/bin，运行 `startup.sh -b64` 生成密钥。

```
cd ${ADCC_HOME}/bin

#SecretKey012345678901234567890123456789012345678901234567890123456789可自定义，其为SecretKey开头的Base64编码字符串，且原始密钥长度不得低于32字符，生成的密钥可替换application.properties中配置的默认密钥
nacos.core.auth.plugin.nacos.token.secret.key

./startup.sh -b64 SecretKey012345678901234567890123456789012345678901234567890123456789
```

12.8 修改密码

ADCC for nacos除了在控制台中提供修改默认密码的功能，也支持通过命令行的方式将明文密码生成加密字符串，以便直接初始化密码。

进入ADCC for nacos的脚本目录${${ADCC_HOME}}/bin，运行startup.sh -pw将明文密码转换成加密密码。

```
cd ${ADCC_HOME}/bin

#passwordvalue为明文密码

./startup.sh -pw passwordvalue
```

在生成加密码后，可在启动初始化前修改${ADCC_HOME}/config/derby-schema.sql和derby-schema.sql下的sql语句，具体操作如下：

```
INSERT INTO users (username, password, enabled) VALUES ('adcc', '$2a$10$VGJqmj0Jo3SJX9z6fwVmNeFWYZi0yZF2j3FzKNez27Nlpj7kSY4Ou', TRUE);
INSERT INTO roles (username, role) VALUES ('adcc', 'ROLE_ADMIN');
```

其中adcc是用户名，横线的是密码，替换成生成的加密字符串。

13 JAVA的SDK

13.1 概述部分

13.1.1 SDK依赖

ADCC for nacos提供JAVA版本的客户端，支持使用nacos-licent相关依赖来访问服务端。

具体依赖示例如下所示：

```
<properties>
    <!-- 2.1.2版本以上支持纯净版客户端 -->
    <nacos.version>2.3.0</nacos.version>
</properties>

<dependencies>
    <dependency>
        <groupId>com.alibaba.nacos</groupId>
        <artifactId>nacos-client</artifactId>
        <version>${nacos.version}</version>
        <!-- 指定纯净版SDK -->
        <classifier>pure</classifier>
    </dependency>
    <!-- 使用纯净版时必须要引入同版本nacos-api和nacos-common，否则可能出现运行时找不到类的问题 -->
    <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>nacos-common</artifactId>
        <version>${nacos.version}</version>
    </dependency>
    <dependency>
        <groupId>${project.groupId}</groupId>
        <artifactId>nacos-api</artifactId>
        <version>${nacos.version}</version>
    </dependency>
</dependencies>
```

13.1.2 SDK鉴权

在构建"Properties"类时，需传入URL上下文路径"CONTEXT_PATH"、用户名"USERNAME"和密码"PASSWORD"。

```
    String serverAddr = "172.24.4.163:8848";
    String contextpath="/adcc";
    String username = "adcc";
    String password = "apusic@2024";
    String namespace = "";
    Properties properties = new Properties();
    properties.put(PropertyKeyConst.SERVER_ADDR, serverAddr);
    properties.put(PropertyKeyConst.USERNAME, username);
    properties.put(PropertyKeyConst.PASSWORD, password);
    properties.put(PropertyKeyConst.NAMESPACE, namespace);
    properties.put(PropertyKeyConst.CONTEXT_PATH, contextpath);
```

13.2 配置管理

13.2.1 获取配置

- 描述

用于服务启动的时候从 ADCC for nacos获取配置。

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

- 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（ADCC: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。
timeout	long	读取配置超时时间，单位 ms，推荐值 3000。

- 返回值

参数类型	描述
string	配置值

- 请求示例

```
try {
    String serverAddr = "{serverAddr}";
    String dataId = "{dataId}";
    String group = "{group}";
    Properties properties = new Properties();
    properties.put("serverAddr", serverAddr);
    ConfigService configService = NacosFactory.createConfigService(properties);
    String content = configService.getConfig(dataId, group, 5000);
    System.out.println(content);
} catch (NacosException 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 NacosException 异常。

13.2.2 监听配置

- 描述

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

```
public void addListener(String dataId, String group, Listener listener)
```

- 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 ADCC: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。

listener	Listener	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的时候通过回调函数返回该值。

- 请求示例

```
String serverAddr = "{serverAddr}";
String dataId = "{dataId}";
String group = "{group}";
Properties properties = new Properties();
properties.put("serverAddr", serverAddr);
ConfigService configService = NacosFactory.createConfigService(properties);
String content = configService.getConfig(dataId, group, 5000);
System.out.println(content);
configService.addListener(dataId, group, new Listener() {
    @Override
    public void receiveConfigInfo(String configInfo) {
        System.out.println("recieve1:" + configInfo);
    }
    @Override
    public Executor getExecutor() {
        return null;
    }
});

// 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();
    }
}
```

13.2.3 删除监听

- 描述

取消监听配置，取消监听后配置不会再推送。

```
public void removeListener(String dataId, String group, Listener listener)
```

- 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
listener	ConfigChangeListenerAdapter	监听器，配置变更进入监听器的回调函数。

- 使用示例

```
String serverAddr = "{serverAddr}";
String dataId = "{dataId}";
String group = "{group}";
Properties properties = new Properties();
properties.put("serverAddr", serverAddr);
ConfigService configService = NacosFactory.createConfigService(properties);
configService.removeListener(dataId, group, yourListener);
```

13.2.4 发布配置

- 描述

用于通过程序自动发布 ADCC for nacos 配置，以便通过自动化手段降低运维成本。

注意：创建和修改配置时使用的同一个发布接口，当配置不存在时会创建配置，当配置已存在时会更新配置。

```
public boolean publishConfig(String dataId, String group, String content) throws NacosException;
```

- 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。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 ADCC:`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。
type	string	@Since 1.4.1. 配置类型，见 `com.alibaba.nacos.api.config.ConfigType`，默认为TEXT

- 返回参数

参数类型	描述
boolean	是否发布成功

- 请求示例

```
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
    String serverAddr = "{serverAddr}";
    String dataId = "{dataId}";
    String group = "{group}";
    Properties properties = new Properties();
    properties.put("serverAddr", serverAddr);
    ConfigService configService = NacosFactory.createConfigService(properties);
    boolean isPublishOk = configService.publishConfig(dataId, group, "content");
    System.out.println(isPublishOk);
} catch (NacosException 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 NacosException 异常。

13.2.5 删除配置

- 描述

用于通过程序自动删除 ADCC for nacos 配置，以便通过自动化手段降低运维成本。

注意：当配置已存在时会删除该配置，当配置不存在时会直接返回成功消息。

```
public boolean removeConfig(String dataId, String group) throws NacosException
```

- 请求参数

参数名	参数类型	描述
dataId	string	配置 ID
group	string	配置分组

- 返回参数

参数类型	描述
boolean	是否删除成功

- 请求示例

```
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
    String serverAddr = "{serverAddr}";
    String dataId = "{dataId}";
    String group = "{group}";
    Properties properties = new Properties();
    properties.put("serverAddr", serverAddr);

    ConfigService configService = NacosFactory.createConfigService(properties);
    boolean isRemoveOk = configService.removeConfig(dataId, group);
    System.out.println(isRemoveOk);
} catch (NacosException 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 NacosException 异常。

13.3 服务发现

13.3.1 注册实例

- 描述注册一个实例到服务。

```
void registerInstance(String serviceName, String ip, int port) throws NacosException;

void registerInstance(String serviceName, String ip, int port, String clusterName) throws NacosException;

void registerInstance(String serviceName, Instance instance) throws NacosException;
```

- 请求参数

名称	类型	描述
serviceName	字符串	服务名

ip	字符串	服务实例IP
port	int	服务实例port
clusterName	字符串	集群名
instance	参见代码注释	实例属性

- 返回参数

无

- 请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
        naming.registerInstance("nacos.test.3", "172.24.4.163", 8848, "TEST1");

        Instance instance = new Instance();
        instance.setIp("172.24.4.163");
        instance.setPort(9100);
        instance.setHealthy(true);
        instance.setWeight(2.0);
        Map<String, String> instanceMeta = new HashMap<>();
        instanceMeta.put("site", "et2");
        instance.setMetadata(instanceMeta);

        Service service = new Service("nacos.test.4");
        service.setAppName("nacos-naming");
        service.setProtectThreshold(0.8F);
        service.setGroupName("CNCF");
        Map<String, String> serviceMeta = new HashMap<>();
        serviceMeta.put("symmetricCall", "true");
        service.setMetadata(serviceMeta);
        instance.setServiceName(service.getName());

        Cluster cluster = new Cluster();
        cluster.setName("TEST5");
        Map<String, String> clusterMeta = new HashMap<>();
        clusterMeta.put("xxx", "yyyy");
        cluster.setMetadata(clusterMeta);

        instance.setClusterName(cluster.getName());

        naming.registerInstance("nacos.test.4", instance);
```

13.3.2 注销实例

- 描述

删除服务下的一个实例。

```
void deregisterInstance(String serviceName, String ip, int port) throws NacosException;

void deregisterInstance(String serviceName, String ip, int port, String clusterName) throws
NacosException;
```

- 请求参数

名称	类型	描述
serviceName	字符串	服务名
ip	字符串	服务实例IP
port	int	服务实例port
clusterName	字符串	集群名

- 返回参数

无

- 请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
naming.deregisterInstance("nacos.test.3", "11.11.11.11", 8888, "DEFAULT");
```

13.3.3 获取全部实例

- 描述

获取服务下的所有实例。

```
List<Instance> getAllInstances(String serviceName) throws NacosException;

List<Instance> getAllInstances(String serviceName, List<String> clusters) throws NacosException;
```

- 请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表

- 返回参数

List 实例列表。

- 请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
System.out.println(naming.getAllInstances("nacos.test.3"));
```

13.3.4 获取健康或不健康实例列表

- 描述

根据条件获取过滤后的实例列表。

```
List<Instance> selectInstances(String serviceName, boolean healthy) throws NacosException;

List<Instance> selectInstances(String serviceName, List<String> clusters, boolean healthy) throws
NacosException;
```

- 请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表

healthy	boolean	是否健康

- 返回参数

List 实例列表。

- 请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
System.out.println(naming.selectInstances("nacos.test.3", true));
```

13.3.5 获取一个健康实例

- 描述

根据负载均衡算法随机获取一个健康实例。

```
Instance selectOneHealthyInstance(String serviceName) throws NacosException;

Instance selectOneHealthyInstance(String serviceName, List<String> clusters) throws NacosException;
```

- 请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表

- 返回参数

Instance 实例。

- 请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
System.out.println(naming.selectOneHealthyInstance("nacos.test.3"));
```

13.3.6 监听服务

- 描述

监听服务下的实例列表变化。

```
void subscribe(String serviceName, EventListener listener) throws NacosException;

void subscribe(String serviceName, List<String> clusters, EventListener listener) throws NacosException;
```

- 请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表
listener	EventListener	回调listener

- 返回参数

无

- 请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
naming.subscribe("nacos.test.3", event -> {
    if (event instanceof NamingEvent) {
        System.out.println(((NamingEvent) event).getServceName());
        System.out.println(((NamingEvent) event).getInstances());
    }
});
```

13.3.7 取消监听服务

- 描述

取消监听服务下的实例列表变化。

```
void unsubscribe(String serviceName, EventListener listener) throws NacosException;

void unsubscribe(String serviceName, List<String> clusters, EventListener listener) throws
NacosException;
```

- 请求参数

名称	类型	描述
serviceName	字符串	服务名
clusters	List	集群列表
listener	EventListener	回调listener

- 返回参数

无

- 请求示例

```
NamingService naming = NamingFactory.createNamingService(System.getProperty("serveAddr"));
naming.unsubscribe("nacos.test.3", event -> {});
```

14 Open-API指南

14.1 概述部分

14.1.1 API 统一返回体格式

所有接口请求的响应均为`json`类型的返回体，返回体具有相同的格式

```
{
  "code": 0,
  "message": "success",
  "data": {}
}
```

返回体中各字段的含义如下表所示

名称	类型	描述
code	int	错误码，0代表执行成功，非0代表执行失败的某一种情况
message	String	错误码提示信息，执行成功为"success"
data	任意类型	返回数据，执行失败时为详细出错信息

> *由于执行成功的情况下code字段与message字段相同，后续在介绍接口的返回结果时，只介绍返回数据的data字段*

14.1.2 API 错误码汇总

API接口返回体中的错误码及对应提示信息汇总见下表

错误码	提示信息	含义
0	success	成功执行
10000	parameter missing	参数缺失
10001	access denied	访问拒绝
10002	data access error	数据访问错误
20001	'tenant' parameter error	tenant参数错误
20002	parameter validate error	参数验证错误
20003	MediaType Error	请求的MediaType错误
20004	resource not found	资源未找到
20005	resource conflict	资源访问冲突
20006	config listener is null	监听配置为空
20007	config listener error	监听配置错误
20008	invalid dataId	无效的dataId（鉴权失败）
20009	parameter mismatch	请求参数不匹配
21000	service name error	serviceName服务名错误
21001	weight error	weight权重参数错误
21002	instance metadata error	实例metadata元数据错误
21003	instance not found	instance实例不存在
21004	instance error	instance实例信息错误

21005	service metadata error	服务metadata元数据错误
21006	selector error	访问策略selector错误
21007	service already exist	服务已存在
21008	service not exist	服务不存在
21009	service delete failure	存在服务实例，服务删除失败
21010	healthy param miss	healthy参数缺失
21011	health check still running	健康检查仍在运行
22000	illegal namespace	命名空间namespace不合法
22001	namespace not exist	命名空间不存在
22002	namespace already exist	命名空间已存在
23000	illegal state	状态state不合法
23001	node info error	节点信息错误
23002	node down failure	节点离线操作出错
...	...	...
30000	server error	其他内部错误

14.1.3 API 上下文路径

ADCC for nacos 的API上下文路径取决于配置文件 `${ADCC_HOME}/config/application.properties` 中 `server.servlet.contextPath` 的取值，默认值如下

```
server.servlet.contextPath=/adcc
```

举例说明，在获取ADCC for nacos配置的URL中，`{adcc_contextpath}` 为上下文路径的变量值：

```
http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/config?
dataId=adcc.example&group=DEFAULT_GROUP&namespaceId=public
```

当设置server.servlet.contextPath=/adcc时，{adcc_contextpath}的变量值应为adcc，则其实际请求URL为：

```
http://127.0.0.1:8848/adcc/v2/cs/config?dataId=adcc.example&group=DEFAULT_GROUP&namespaceId=public
```

当设置server.servlet.contextPath=/nacos时，{adcc_contextpath}的变量值应为nacos，则其实际请求URL为：

```
http://127.0.0.1:8848/nacos/v2/cs/config?dataId=adcc.example&group=DEFAULT_GROUP&namespaceId=public
```

当设置为{adcc_contextpath}/v2/cs/temp时，表示提交配置但没有即时发布。

```
http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/temp?
dataId=adcc.example&group=DEFAULT_GROUP&namespaceId=public
```

14.1.4 API 鉴权

首先需要使用用户名和密码登陆ADCC for nacos。

```
curl -X POST '127.0.0.1:8848/{adcc_contextpath}/v1/auth/login' -d 'username=adcc&password=apusic@2024'
```

若用户名和密码正确,返回信息如下:

```
{
    "accessToken":
"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZGNjIiwiZXhwIjoxNzA2MjY2Njg1fQ.O11QLNAEdb9etmGpTWfUhB_1NykercBT2Cf-
8SMhwxE",
    "tokenTtl": 18000,
    "globalAdmin": true,
    "username": "adcc"
}
```

接下来进行配置信息或服务信息时，应当使用该accessToken鉴权，在url后添加参数accessToken={accessToken}，其中{accessToken}为登录时返回的token信息，具体操作如下：

```
curl -X GET '127.0.0.1:8848/{adcc_contextpath}/v2/cs/configs?
accessToken=eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJuYWNvcyIsImV4cCI6MTYwNTYyMzkyM30.O-
s2yWfDSUZ7Svd3Vs7jy9tsfDNHs1SuebJB4KlNY8Q&dataId=adcc.example.1&group=test'
```

或者请求时带上用户名密码认证，具体操作如下：

```
curl -X GET '127.0.0.1:8848/{adcc_contextpath}/v2/cs/configs?
username=adcc&password=apusic@2024&dataId=adcc.example.1&group=test'
```

14.2 配置管理

14.2.1 获取配置

- 接口描述

获取指定配置

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/cs/config
```

- 请求参数

参数名	类型	必填	参数描述
namespaceId	String	否	命名空间，默认为public与 ''相同
group	String	是	配置分组名
dataId	String	是	配置名
tag	String	否	标签

- 返回数据

参数名	参数类型	描述
data	String	配置内容

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/config?
dataId=adcc.example&group=DEFAULT_GROUP&namespaceId=public'
```

- 返回示例

```
{
	"code": 0,
	"message": "success",
	"data": "contentTest"
}
```

14.2.2 发布配置

- 接口描述

发布指定配置

> *当配置已存在时，则对配置进行更新*

- 请求方式

```
POST
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/adcc/v2/cs/config
```

- 请求Body

参数名	类型	必填	参数描述
namespaceId	String	否	命名空间，默认为public与 ''相同
group	String	**是**	配置组名
dataId	String	**是**	配置名
content	String	**是**	配置内容
tag	String	否	标签
appName	String	否	应用名
srcUser	String	否	源用户
configTags	String	否	配置标签列表，可多个，逗号分隔
desc	String	否	配置描述
use	String	否	-
effect	String	否	-
type	String	否	配置类型
schema	String	否	-

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'dataId=adcc.example' \
  -d 'group=DEFAULT_GROUP' \
  -d 'namespaceId=public' \
```

```
  -d 'content=contentTest' \
  -X POST 'http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/config'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": true
}
```

14.2.3 删除配置

- 接口描述

删除指定配置

- 请求方式

```
DELETE
```

- 请求URL

```
/{adcc_contextpath}/v2/cs/config
```

- 请求参数

参数名	类型	必填	参数描述
namespaceId	String	否	命名空间，默认为public与''相同
group	String	是	配置分组名
dataId	String	是	配置名
tag	String	否	标签

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -X DELETE 'http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/config?
dataId=adcc.example&group=DEFAULT_GROUP&namespaceId=public'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": true
}
```

14.2.4 查询配置历史列表

- 接口描述

获取指定配置的历史版本列表

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/cs/history/list
```

- 请求参数

参数名	类型	必填	参数描述
namespaceId	String	否	命名空间，默认为public与 ''相同
group	String	是	配置分组名
dataId	String	是	配置名
pageNo	int	否	当前页，默认为1
pageSize	int	否	页条目数，默认为100，最大为500

- 返回数据

参数名	参数类型	描述说明
data	Object	分页查询结果
data.totalCount	int	总数
data.pageNumber	int	当前页
data.pagesAvailable	int	总页数
data.pageItems	Object[]	历史配置项列表，参见历史配置项信息

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/history/list?
dataId=adcc.example&group=DEFAULT_GROUP&namespaceId=public'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "totalCount": 1,
        "pageNumber": 1,
        "pagesAvailable": 1,
        "pageItems": [
            {
                "id": "203",
                "lastId": -1,
                "dataId": "adcc.example",
                "group": "DEFAULT_GROUP",
                "tenant": "",
                "appName": "",
```

```
                "md5": "9f67e6977b100e00cab385a75597db58",
                  "content": "contentTest",
                "srcIp": "0:0:0:0:0:0:0:1",
                "srcUser": null,
                "opType": "I",
                "createdTime": "2010-05-04T16:00:00.000+0000",
                "lastModifiedTime": "2020-12-05T01:48:03.380+0000"
            }
        ]
    }
}
```


14.2.5 查询具体版本的历史配置

- 接口描述

获取指定版本的历史配置

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/cs/history
```

- 请求参数

参数名	类型	必填	参数描述
namespaceId	String	否	命名空间，默认为public与''相同
group	String	是	配置分组名
dataId	String	是	配置名
nid	long	是	历史配置id

- 返回数据

参数名	参数类型	描述说明
data	Object	历史配置项
data.id	String	配置id
data.lastId	int	
data.dataId	String	配置名
data.group	String	配置分组
data.tenant	String	租户信息（命名空间）
data.appName	String	应用名
data.md5	String	配置内容的md5值
data.content	String	配置内容
data.srcIp	String	源ip
data.srcUser	String	源用户
data.opType	String	操作类型

data.createdTime	String	创建时间
data.lastModifiedTime	String	上次修改时间
data.encryptedDataKey	String	

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/history?
dataId=adcc.example&group=DEFAULT_GROUP&namespaceId=&nid=203'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "id": "203",
        "lastId": -1,
        "dataId": "adcc.example",
        "group": "DEFAULT_GROUP",
        "tenant": "",
        "appName": "",
        "md5": "9f67e6977b100e00cab385a75597db58",
          "content": "contentTest",
        "srcIp": "0:0:0:0:0:0:0:1",
        "srcUser": null,
        "opType": "I",
        "createdTime": "2010-05-04T16:00:00.000+0000",
        "lastModifiedTime": "2020-12-05T01:48:03.380+0000"
    }
}
```

14.2.6 查询配置上一版本信息

- 接口描述

获取指定配置的上一版本

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/cs/history/previous
```

- 请求参数

参数名	类型	必填	参数描述
namespaceId	String	否	命名空间，默认为public与 ''相同
group	String	**是**	配置分组名
dataId	String	**是**	配置名

id	long	是	配置id

- 返回数据

参数名	参数类型	描述说明
data	Object	历史配置项

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/history/previous?
id=309135486247505920&dataId=adcc.example&group=DEFAULT_GROUP&namespaceId='
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "id": "203",
        "lastId": -1,
        "dataId": "adcc.example",
        "group": "DEFAULT_GROUP",
        "tenant": "",
        "appName": "",
        "md5": "9f67e6977b100e00cab385a75597db58",
          "content": "contentTest",
        "srcIp": "0:0:0:0:0:0:0:1",
        "srcUser": null,
        "opType": "I",
        "createdTime": "2010-05-04T16:00:00.000+0000",
        "lastModifiedTime": "2020-12-05T01:48:03.380+0000"
    }
}
```

14.2.7 查询指定命名空间下的配置列表

- 接口描述

获取指定命名空间下的配置信息列表

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/cs/history/configs
```

- 请求参数

参数名	类型	必填	参数描述
namespaceId	String	是	命名空间

- 返回数据

参数名	参数类型	描述说明
data	Object[]	配置信息列表
data.id	String	配置id
data.dataId	String	配置名
data.group	String	配置分组
data.content	String	配置内容
data.md5	String	配置内容的md5值
data.encryptedDataKey	String	
data.tenant	String	租户信息（命名空间）
data.appName	String	应用名
data.type	String	配置文件类型
data.lastModified	long	上次修改时间

> *返回数据中的配置信息只有 `dataId`，`group`，`tenant`，`appName`，`type` 字段有效，其他字段为默认值*

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/cs/history/configs?namespaceId='
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": [
        {
            "id": "0",
            "dataId": "adcc.example",
            "group": "DEFAULT_GROUP",
            "content": null,
            "md5": null,
            "encryptedDataKey": null,
            "tenant": "",
            "appName": "",
            "type": "yaml",
            "lastModified": 0
        }
    ]
}
```

14.2.8 提交配置但暂不发布

- 接口描述

  支持提交配置，但暂时不发布。

- 请求URL

```
/{adcc_contextpath}/v2/cs/temp
```

- 示例

新增和修改:

```
curl -d 'dataId=nacos.example' -d 'group=DEFAULT_GROUP' -d 'namespaceId=public' -d
'content=contentTest' -X POST 'http://127.0.0.1:8848/adcc/v2/cs/temp'
```

查询:

```
curl  'http://127.0.0.1:8848/adcc/v2/cs/temp?
dataId=nacos.example&group=DEFAULT_GROUP&namespaceId=public'
```

删除:

```
curl -d 'dataId=nacos.example'   -d 'group=DEFAULT_GROUP'   -d 'namespaceId=public'   -X DELETE
'http://127.0.0.1:8848/adcc/v2/cs/temp'
```

发布:

```
curl -d 'dataId=nacos.example'   -d 'group=DEFAULT_GROUP'   -d 'namespaceId=public'    -X POST
'http://127.0.0.1:8848/adcc/v2/cs/temp/publish'
```

14.3 服务发现

14.3.1 注册实例

- 接口描述

注册一个实例

- 请求方式

```
POST
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/instance
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	**是**	服务名
ip	String	**是**	IP地址
port	int	**是**	端口号
clusterName	String	否	集群名称，默认为DEFAULT
healthy	boolean	否	是否只查找健康实例，默认为true
weight	double	否	实例权重，默认为1.0
enabled	boolean	否	是否可用，默认为true
metadata	JSON格式String	否	实例元数据

ephemeral	boolean	否	是否为临时实例

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'serviceName=test_service' \
  -d 'ip=127.0.0.1' \
  -d 'port=8090' \
  -d 'weight=0.9' \
  -d 'ephemeral=true' \
  -X POST 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/instance'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": true
}
```

14.3.2 注销实例

- 接口描述

注销指定实例

- 请求方式

```
DELETE
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/instance
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	是	服务名
ip	String	是	IP地址
port	int	是	端口号
clusterName	String	否	集群名称，默认为DEFAULT
healthy	boolean	否	是否只查找健康实例，默认为true
weight	double	否	实例权重，默认为1.0
enabled	boolean	否	是否可用，默认为true

metadata	JSON格式String	否	实例元数据
ephemeral	boolean	否	是否为临时实例

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'serviceName=test_service' \
  -d 'ip=127.0.0.1' \
  -d 'port=8090' \
  -d 'weight=0.9' \
  -d 'ephemeral=true' \
  -X DELETE 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/instance'
```

- 返回示例

```
{
       "code": 0,
       "message": "success",
       "data": true
}
```

14.3.3 更新实例

- 接口描述

修改实例信息

> *通过该接口更新的元数据拥有更高的优先级，且具有记忆能力；会在对应实例删除后，依旧存在一段时间，如果在此期间实例重新注册，该元数据依旧生效；您可以通过 `nacos.naming.clean.expired-metadata.expired-time` 及 `nacos.naming.clean.expired-metadata.interval` 对记忆时间进行修改*

- 请求方式

```
PUT
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/instance
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	**是**	服务名
ip	String	**是**	IP地址
port	int	**是**	端口号
clusterName	String	否	集群名称，默认为DEFAULT

healthy	boolean	否	是否只查找健康实例，默认为true
weight	double	否	实例权重，默认为1.0
enabled	boolean	否	是否可用，默认为true
metadata	JSON格式String	否	实例元数据
ephemeral	boolean	否	是否为临时实例

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'serviceName=test_service' \
  -d 'ip=127.0.0.1' \
  -d 'port=8090' \
  -d 'weight=0.9' \
  -d 'ephemeral=true' \
  -X PUT 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/instance'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": true
}
```

14.3.4 查询实例详情

- 接口描述

查询某个具体实例的详情信息

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/instance
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	**是**	服务名
clusterName	String	否	集群名称，默认为DEFAULT
ip	String	**是**	IP地址
port	int	**是**	端口号

- 返回数据

参数名	参数类型	描述说明
data	Object	实例详情信息
data.serviceName	String	服务名
data.ip	String	IP地址
data.port	int	端口号
data.clusterName	String	集群名称
data.weight	double	实例权重
data.healthy	boolean	是否健康
data.instanceId	String	实例id
data.metadata	map	实例元数据

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/instance?
namespaceId=public&groupName=&serviceName=test_service&ip=127.0.0.1&port=8090'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "serviceName": "DEFAULT_GROUP@@test_service",
        "ip": "127.0.0.1",
        "port": 8090,
        "clusterName": "DEFAULT",
        "weight": 1.0,
        "healthy": true,
        "instanceId": null,
        "metadata": {
            "value": "1"
        }
    }
}
```

14.3.5 查询指定服务的实例列表

- 接口描述

查询指定服务下的实例详情信息列表

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/instance/list
```

- 请求头

参数名	参数类型	是否必填	描述说明
User-Agent	String	否	用户代理，默认为空
Client-Version	String	否	客户端版本，默认为空

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	是	服务名
clusterName	String	否	集群名称，默认为DEFAULT
ip	String	否	IP地址，默认为空，表示不限制IP地址
port	int	否	端口号，默认为0，表示不限制端口号
healthyOnly	boolean	否	是否只获取健康实例，默认为false
app	String	否	应用名，默认为空

- 返回数据

参数名	参数类型	描述说明
data		指定服务的实例列表
data.name	String	分组名@@服务名
data.groupName	String	分组名
data.clusters	String	集群名
data.cacheMillis	int	缓存时间
data.hosts	Object[]	实例列表
data.hosts.ip	String	实例IP
data.hosts.port	int	实例端口号
data.hosts.weight	double	实例权重
data.hosts.healthy	boolean	实例是否健康
data.hosts.enabled	boolean	实例是否可用
data.hosts.ephemeral	boolean	是否为临时实例
data.hosts.clusterName	String	实例所在的集群名称
data.hosts.serviceName	String	服务名
data.hosts.metadata	map	实例元数据
data.hosts.instanceHeartBeatTimeOut	int	实例心跳超时时间
data.hosts.ipDeleteTimeout	int	实例删除超时时间
data.hosts.instanceHeartBeatInterval	int	实例心跳间隔
data.lastRefTime	int	上次刷新时间
data.checksum	int	校验码
data.allIPs	boolean	
data.reachProtectionThreshold	boolean	是否到达保护阈值

data.valid	boolean	是否有效

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/instance/list?
serviceName=test_service&ip=127.0.0.1'
```

- 返回示例


```
{
    "code": 0,
    "message": "success",
    "data": {
        "name": "DEFAULT_GROUP@@test_service",
        "groupName": "DEFAULT_GROUP",
        "clusters": "",
        "cacheMillis": 10000,
        "hosts": [
            {
                "ip": "127.0.0.1",
                "port": 8080,
                "weight": 1.0,
                "healthy": true,
                "enabled": true,
                "ephemeral": true,
                "clusterName": "DEFAULT",
                "serviceName": "DEFAULT_GROUP@@test_service",
                "metadata": {
                    "value": "1"
                },
                "instanceHeartBeatTimeOut": 15000,
                "ipDeleteTimeout": 30000,
                "instanceHeartBeatInterval": 5000
            }
        ],
        "lastRefTime": 1662554390814,
        "checksum": "",
        "allIPs": false,
        "reachProtectionThreshold": false,
        "valid": true
    }
}
```


14.3.6 批量更新实例元数据

- 接口描述

批量更新实例的元数据,

> *对应元数据的键不存在时，则添加对应元数据*

- 请求方式

```
PUT
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/instance/metadata/batch
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	**是**	服务名
consistencyType	String	否	持久化类型，默认为空
instances	JSON格式String	否	需要更新的实例列表，默认为空
metadata	JSON格式String	**是**	实例元数据

- 参数说明

> - *`consistencyType`：实例的持久化类型，当为`persist`，表示对持久化实例的元数据进行更新；否则表示对临时实例的元数据进行更新*
> - *`instances`：待更新的实例列表，`json`数组，通过`ip+port+ephemeral+cluster`定位到某一实例，为空则表示更新指定服务下所有实例的元数据*

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
  curl -d 'serviceName=test_service' \
  -d 'consistencyType=`persist`' \
  -d 'instances=[{"ip":"127.0.0.1","port": "8090","ephemeral":"false"},
{"ip":"2.2.2.2","port":"8080","ephemeral":"false"}]' \
  -d 'metadata={"age":"20","name":"test_update44"}' \
  -X PUT 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/instance/metadata/batch'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": true
}
```

14.3.7 批量删除实例元数据

- 接口描述

批量删除实例的元数据，

> *对应元数据的键不存在时，则不做操作*

- 请求方式

```
DELETE
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/instance/metadata/batch
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	**是**	服务名
consistencyType	String	否	持久化类型，默认为空
instances	JSON格式String	否	需要更新的实例列表，默认为空
metadata	JSON格式String	**是**	实例元数据

- 参数说明

> - *`consistencyType`：实例的持久化类型，当为`persist`，表示对持久化实例的元数据进行删除；否则表示对临时实例的元数据进行*
> - *`instances`：待更新的实例列表，`json`数组，通过`ip+port+ephemeral+cluster`定位到某一实例，为空则表示更新指定服务下所有实例的元数据*

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'serviceName=test_service' \
  -d 'ip=127.0.0.1' \
  -d 'port=8090' \
  -d 'weight=0.9' \
  -d 'ephemeral=false' \
  -X DELETE 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/instance'
```

- 返回示例

```
{
       "code": 0,
       "message": "success",
       "data": true
}
```

14.3.8 更新实例健康状态

- 接口描述

更新实例的健康状态,仅在集群的健康检查关闭时才生效,当集群配置了健康检查时,该接口会返回错误

- 请求方式

```
PUT
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/health/instance
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	**是**	服务名
clusterName	String	否	集群名，默认为DEFAULT
ip	String	**是**	IP地址
port	int	**是**	端口号
healthy	boolean	**是**	是否健康

- 返回数据

参数名	参数类型	描述
data	String	"ok"表示执行成功

- 请求示例

```
curl -d 'serviceName=test_service' \
  -d 'ip=127.0.0.1' \
  -d 'port=8090' \
  -d 'healthy=false' \
  -X PUT 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/health/instance'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": "ok"
}
```

14.3.9 创建服务

- 接口描述

创建一个服务

> *服务已存在时会创建失败*

- 请求方式

```
POST
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/service
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	是	服务名
metadata	JSON格式String	否	服务元数据，默认为空
ephemeral	boolean	否	是否为临时实例，默认为false
protectThreshold	float	否	保护阈值，默认为0
selector	JSON格式String	否	访问策略，默认为空

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'serviceName=adcc.test.service001' \
  -d 'ephemeral=true' \
  -d 'metadata={"k1":"v1"}' \
  -X POST 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/service'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": true
}
```

14.3.10 删除服务

- 接口描述

删除指定服务

> *服务不存在时会报错，且服务还存在实例时会删除失败*

- 请求方式

```
DELETE
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/service
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public

groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	**是**	服务名

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -X DELETE 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/service?
serviceName=adcc.test.service001'
```

- 返回示例

```
{
        "code": 0,
        "message": "success",
        "data": true
}
```

14.3.11 修改服务

- 接口描述

更新指定服务

> *服务不存在时会报错*

- 请求方式

```
POST
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/service
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	**是**	服务名
metadata	JSON格式String	否	服务元数据，默认为空
protectThreshold	float	否	保护阈值，默认为0
selector	JSON格式String	否	访问策略，默认为空

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'serviceName=adcc.test.service001' \
  -d 'metadata={"k1":"v2"}' \
  -X PUT 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/service'
```

- 返回示例

```
{
       "code": 0,
       "message": "success",
       "data": true
}
```

14.3.12 查询服务详情

- 接口描述

查询某个具体服务的详情信息

> *服务不存在时会报错*

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/service
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	是	服务名

- 返回数据

参数名	参数类型	描述说明
data		服务信息
data.namespace	String	命名空间
data.groupName	String	分组名
data.serviceName	String	服务名
data.clusterMap	map	集群信息
data.metadata	map	服务元数据
data.protectThreshold	float	保护阈值
data.selector	Object	访问策略
data.ephemeral	Boolean	是否为临时实例

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/service?
serviceName=adcc.test.service001'
```

- 返回示例


```
{
    "code": 0,
    "message": "success",
    "data": {
        "namespace": "public",
        "serviceName": "adcc.test.service001",
        "groupName": "DEFAULT_GROUP",
        "clusterMap": {},
        "metadata": {},
        "protectThreshold": 0,
        "selector": {
            "type": "none",
            "contextType": "NONE"
        },
        "ephemeral": false
    }
}
```


14.3.13 查询服务列表

- 接口描述

查询符合条件的服务列表

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/service/list
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
selector	JSON格式String	是	访问策略
pageNo	int	否	当前页，默认为1
pageSize	int	否	页条目数，默认为20，最大为500

- 返回数据

参数名	参数类型	描述说明
data		服务列表信息
data.count	String	服务数目

data.services	String[]	分页后的服务列表

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/service/list'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "count": 2,
        "services": [
            "adcc.test.service001",
            "adcc.test.service002"
        ]
    }
}
```

14.3.14 查询系统开关

- 接口描述

查询系统开关

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/operator/switches
```

- 返回数据

参数名	参数类型	描述说明
data	Object	系统开关信息

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/operator/switches'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "masters": null,
        "adWeightMap": {

        },
```

```
"defaultPushCacheMillis": 10000,
"clientBeatInterval": 5000,
"defaultCacheMillis": 3000,
"distroThreshold": 0.7,
"healthCheckEnabled": true,
"autoChangeHealthCheckEnabled": true,
"distroEnabled": true,
"enableStandalone": true,
"pushEnabled": true,
"checkTimes": 3,
"httpHealthParams": {
    "max": 5000,
    "min": 500,
    "factor": 0.85
},
"tcpHealthParams": {
    "max": 5000,
    "min": 1000,
    "factor": 0.75
},
"mysqlHealthParams": {
    "max": 3000,
    "min": 2000,
    "factor": 0.65
},
"incrementalList": [

],
"serverStatusSynchronizationPeriodMillis": 2000,
"serviceStatusSynchronizationPeriodMillis": 5000,
"disableAddIP": false,
"sendBeatOnly": false,
"lightBeatEnabled": true,
"limitedUrlMap": {

},
"distroServerExpiredMillis": 10000,
"pushGoVersion": "0.1.0",
"pushJavaVersion": "0.1.0",
"pushPythonVersion": "0.4.3",
"pushCVersion": "1.0.12",
"pushCSharpVersion": "0.9.0",
"enableAuthentication": false,
"overriddenServerStatus": null,
"defaultInstanceEphemeral": true,
"healthCheckWhiteList": [

],
```

```
        "checksum": null,
        "name": "00-00---000-NACOS_SWITCH_DOMAIN-000---00-00"
    }
}
```

14.3.15 修改系统开关

- 接口描述

修改系统开关

- 请求方式

```
PUT
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/operator/switches
```

- 请求Body

参数名	参数类型	是否必填	描述说明
entry	String	是	开关名
value	String	是	开关值
debug	boolean	否	是否只在本机生效,true表示本机生效,false表示集群生效

- 返回数据

参数名	参数类型	描述
data	String	“ok”表示执行成功

- 请求示例

```
curl -d 'entry=pushEnabled' \
  -d 'value=false' \
  -d 'debug=true' \
  -X PUT 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/operator/switches'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": "ok"
}
```

14.3.16 查询系统当前数据指标

- 接口描述

查询系统当前数据指标

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/operator/metrics
```

- 请求参数

参数名	参数类型	是否必填	描述说明
onlyStatus	boolean	否	只显示状态，默认为true

> *当 `onlyStatus` 设置为 `true` 时，只返回表示系统状态的字符串*

- 返回数据

参数名	参数类型	描述说明
data	Object	系统当前数据指标
data.status	String	系统状态
data.serviceCount	int	服务数量
data.instanceCount	int	实例数量
data.subscribeCount	int	订阅数量
data.raftNotifyTaskCount	int	Raft通知任务数量
data.responsibleServiceCount	int	
data.responsibleInstanceCount	int	
data.clientCount	int	客户端数量
data.connectionBasedClientCount	int	连接数量
data.ephemeralIpPortClientCount	int	临时客户端数量
data.persistentIpPortClientCount	int	持久客户端数量
data.responsibleClientCount	int	
data.cpu	float	cpu使用率
data.load	float	负载
data.mem	float	内存使用率

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/operator/metrics?onlyStatus=false'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "status": "UP",
        "serviceCount": 2,
        "instanceCount": 2,
        "subscribeCount": 2,
        "raftNotifyTaskCount": 0,
        "responsibleServiceCount": 0,
        "responsibleInstanceCount": 0,
```

```
        "clientCount": 2,
        "connectionBasedClientCount": 2,
        "ephemeralIpPortClientCount": 0,
        "persistentIpPortClientCount": 0,
        "responsibleClientCount": 2,
        "cpu": 0,
        "load": -1,
        "mem": 1
    }
}
```

14.3.17 查询客户端列表

- 接口描述

查询当前所有的客户端列表

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/client/list
```

- 返回数据

参数名	参数类型	描述说明
data	String[]	客户端id列表

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/client/list'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": [
        "127.0.0.1:8099#false",
        "127.0.0.1:8090#false",
        "127.0.0.1:8088#false",
        "1706235962402_172.24.6.163_55736",
        "172.24.4.42:80#false",
        "172.24.4.27:6868#false",
        "127.0.0.1:8848#false"
    ]
}
```

> *ADCC for nacos支持不同版本的nacos client，建立客户端的方式不同。*
>
> *对于 `1.x` 版本的 `nacos client` ，每个实例会建立两个基于 `ip+port` 的客户端，分别对应实例注册与服务订阅， `clientId` 格式为 `ip:port#ephemeral`*

> *对于 `2.x` 版本的 `nacos client`，每个实例会建立一个 `RPC` 连接，对应一个基于 `RPC` 连接的客户端，兼具注册与订阅功能， `clientId` 格式为 `time_ip_port`*

14.3.18 查询客户端信息

- 接口描述

查询指定客户端的详细信息

> *客户端不存在时会报错*

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/client
```

- 请求参数

参数名	参数类型	是否必填	描述说明
clientId	String	是	客户端id

- 返回数据

参数名	参数类型	描述说明
data	Object	客户端信息
data.clientId	String	客户端id
data.ephemeral	boolean	是否为临时实例
data.lastUpdatedTime	int	上次更新时间
data.clientType	String	客户端类型
data.clientIp	String	客户端IP
data.clientPort	String	客户端端口
data.connectType	String	连接类型
data.appName	String	应用名
data.Version	String	客户端版本

> *只有当 `clientType` 为 `connection` 时，会显示 `connectType`， `appName` 和 `appName` 字段*

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/client?
clientId=1706235962402_172.24.6.163_55736'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "clientId": "1706235962402_172.24.6.163_55736",
        "ephemeral": true,
        "lastUpdatedTime": 1706235962563,
```

```
        "clientType": "connection",
        "connectType": "GRPC",
        "appName": "unknown",
        "version": "Nacos-Java-Client:v2.3.0",
        "clientIp": "172.24.6.163",
        "clientPort": "55736"
    }
}
```

14.3.19 查询客户端的注册信息

- 接口描述

查询指定客户端的注册信息

> *客户端不存在时会报错*

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/client/publish/list
```

- 请求参数

参数名	参数类型	是否必填	描述说明
clientId	String	是	客户端id

- 返回数据

参数名	参数类型	描述说明
data	Object[]	客户端注册的服务列表
data.namespace	String	命名空间
data.group	String	分组名
data.serviceName	String	服务名
data.registeredInstance	Object	该服务下注册的实例
data.registeredInstance.ip	String	IP地址
data.registeredInstance.port	int	端口号
data.registeredInstance.cluster	String	集群名

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/client/publish/list?
clientId=1706235962402_172.24.6.163_55736'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": [
```

```
        {
            "namespace": "public",
            "group": "DEFAULT_GROUP",
            "serviceName": "test_service421",
            "registeredInstance": {
                "ip": "172.24.4.163",
                "port": 6888,
                "cluster": "DEFAULT"
            }
        }
    ]
}
```

14.3.20 查询客户端的订阅信息

- 接口描述

查询指定客户端的订阅信息

> *客户端不存在时会报错*

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/client/subscribe/list
```

- 请求参数

参数名	参数类型	是否必填	描述说明
clientId	String	是	客户端id

- 返回数据

参数名	参数类型	描述说明
data	Object[]	客户端订阅的服务列表
data.namespace	String	命名空间
data.group	String	分组名
data.serviceName	String	服务名
data.subscriberInfo	Object	订阅信息
data.subscriberInfo.app	String	应用
data.subscriberInfo.agent	String	客户端信息
data.subscriberInfo.addr	String	地址

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/client/subscribe/list?
clientId=1706235962402_172.24.6.163_55736'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": [
        {
            "namespace": "public",
            "group": "DEFAULT_GROUP",
            "serviceName": "test_service421",
            "subscriberInfo": {
                "app": "unknown",
                "agent": "Nacos-Java-Client:v2.3.0",
                "addr": "172.24.6.163"
            }
        }
    ]
}
```


14.3.21 查询注册指定服务的客户端信息

- 接口描述

查询注册指定服务的客户端信息

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/client/service/publisher/list
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	是	服务名
ephemeral	boolean	否	是否为临时实例
ip	String	否	IP地址，默认为空，不限制IP地址
port	int	否	端口号，默认为空，表示不限制端口号

- 返回数据

参数名	参数类型	描述说明
data		客户端列表
data.clientId	String	客户端id
data.ip	String	客户端IP
data.port	int	客户端端口

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/client/service/publisher/list?serviceName=test_service421&ip=&port='
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": [
        {
            "clientId": "1706235962402_172.24.6.163_55736",
            "ip": "10.128.164.35",
            "port": 9950
        }
    ]
}
```

14.3.22 查询订阅指定服务的客户端信息

- 接口描述

查询订阅指定服务的客户端信息

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/ns/client/service/subscriber/list
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	否	命名空间Id，默认为public
groupName	String	否	分组名，默认为DEFAULT_GROUP
serviceName	String	是	服务名
ephemeral	boolean	否	是否为临时实例
ip	String	否	IP地址，默认为空，不限制IP地址
port	int	否	端口号，默认为空，表示不限制端口号

- 返回数据

参数名	参数类型	描述说明
data		客户端列表
data.clientId	String	客户端id
data.ip	String	客户端IP
data.port	int	客户端端口

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/ns/client/service/subscriber/list?serviceName=nacos.test.1&ip=&port='
```

- 返回示例


```
{
    "code": 0,
    "message": "success",
    "data": [
        {
            "clientId": "1664527125645_127.0.0.1_4443",
            "ip": "10.128.164.35",
            "port": 0
        },
        {
            "clientId": "172.24.144.1:54126#true",
            "ip": "172.24.144.1",
            "port": 54126
        }
    ]
}
```


14.4 命名空间

14.4.1 查询命名空间列表

- 接口描述

查询当前所有的命名空间

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/console/namespace/list
```

- 返回数据

参数名	参数类型	描述说明
data	Object[]	命名空间列表
data.namespace	String	命名空间ID
data.namespaceShowName	String	命名空间名称
data.namespaceDesc	String	命名空间描述
data.quota	int	命名空间的容量
data.configCount	int	命名空间下的配置数量
data.type	int	命名空间类型

> *命名空间分为3种类型，`0` - 全局命名空间 `1` - 默认私有命名空间 `2` - 自定义命名空间*

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/console/namespace/list'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": [
        {
            "namespace": "",
            "namespaceShowName": "public",
            "namespaceDesc": null,
            "quota": 200,
            "configCount": 1,
            "type": 0
        }
    ]
}
```

14.4.2 查询具体命名空间的信息

- 接口描述

查询具体命名空间的信息

> *命名空间不存在时会报错*

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/console/namespace
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	是	命名空间Id

- 返回数据

参数名	参数类型	描述说明
data	Object	命名空间信息
data.namespace	String	命名空间ID
data.namespaceShowName	String	命名空间名称
data.namespaceDesc	String	命名空间描述
data.quota	int	命名空间的容量
data.configCount	int	命名空间下的配置数量
data.type	int	命名空间类型

> *命名空间分为3种类型，`0` - 全局命名空间 `1` - 默认私有命名空间 `2` - 自定义命名空间*

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/console/namespace?
namespaceId=test_namespace'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": {
        "namespace": "test_namespace",
        "namespaceShowName": "test",
        "namespaceDesc": null,
        "quota": 200,
        "configCount": 0,
        "type": 2
    }
}
```

14.4.3 创建命名空间

- 接口描述

创建一个命名空间

> *命名空间已存在时会报错*

- 请求方式

```
POST
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/console/namespace
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	是	命名空间Id
namespaceName	String	是	命名空间名称
namespaceDesc	String	否	命名空间描述

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'namespaceId=test_namespace' \
  -d 'namespaceName=test' \
```

```
  -X POST 'http://127.0.0.1:8848/{adcc_contextpath}/v2/console/namespace'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": true
}
```

14.4.4 编辑命名空间

- 接口描述

编辑命名空间信息

- 请求方式

```
PUT
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/console/namespace
```

- 请求Body

参数名	参数类型	是否必填	描述说明
namespaceId	String	是	命名空间Id
namespaceName	String	是	命名空间名称
namespaceDesc	String	否	命名空间描述

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'namespaceId=test_namespace' \
  -d 'namespaceName=test.adcc' \
  -X PUT 'http://127.0.0.1:8848/{adcc_contextpath}/v2/console/namespace'
```

- 返回示例

```
{
      "code": 0,
      "message": "success",
      "data": true
}
```

14.4.5 删除命名空间

- 接口描述

删除指定命名空间

- 请求方式

```
DELETE
```

- 请求URL

```
/{adcc_contextpath}/v2/console/namespace
```

- 请求参数

参数名	参数类型	是否必填	描述说明
namespaceId	String	是	命名空间Id

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'namespaceId=test_namespace' \
  -X DELETE 'http://127.0.0.1:8848/{adcc_contextpath}/v2/console/namespace'
```

- 返回示例

```
{
       "code": 0,
       "message": "success",
       "data": true
}
```

14.5 集群管理

14.5.1 查询当前节点信息

- 接口描述

查询当前ADCC for nacos节点信息

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/core/cluster/node/self
```

- 返回数据

参数名	参数类型	描述说明
data	Object	当前节点信息
data.ip	String	节点IP地址
data.port	int	节点端口

data.state	String	节点状态
data.extendInfo	Object	节点扩展信息
data.address	String	节点地址（IP:port）
data.failAccessCnt	int	失败访问次数
data.abilities	Object	

请求示例：

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/core/cluster/node/self'
```

返回示例：

```
{
  "code": 0,
  "message": "success",
  "data": {
      "ip": "172.24.4.152",
      "port": 8848,
      "state": "UP",
      "extendInfo": {
          "lastRefreshTime": 1705917920473,
          "raftMetaData": {
              "metaDataMap": {
                  "naming_instance_metadata": {
                      "leader": "172.24.4.152:7848",
                      "raftGroupMember": [
                          "172.24.4.163:7848",
                          "172.24.4.154:7848",
                          "172.24.4.152:7848"
                      ],
                      "term": 1
                  },
                  "naming_persistent_service": {
                      "leader": "172.24.4.152:7848",
                      "raftGroupMember": [
                          "172.24.4.163:7848",
                          "172.24.4.154:7848",
                          "172.24.4.152:7848"
                      ],
                      "term": 1
                  },
                  "naming_persistent_service_v2": {
                      "leader": "172.24.4.152:7848",
                      "raftGroupMember": [
                          "172.24.4.163:7848",
                          "172.24.4.154:7848",
                          "172.24.4.152:7848"
                      ],
                      "term": 1
                  },
                  "naming_service_metadata": {
                      "leader": "172.24.4.152:7848",
```

```
                    "raftGroupMember": [
                        "172.24.4.163:7848",
                        "172.24.4.154:7848",
                        "172.24.4.152:7848"
                    ],
                    "term": 1
                }
            }
        },
        "raftPort": "7848",
        "readyToUpgrade": true,
        "version": "1.0.230"
    },
    "address": "172.24.4.152:8848",
    "failAccessCnt": 0,
    "abilities": {
        "remoteAbility": {
            "supportRemoteConnection": true,
            "grpcReportEnabled": true
        },
        "configAbility": {
            "supportRemoteMetrics": false
        },
        "namingAbility": {
            "supportJraft": true
        }
    },
    "grpcReportEnabled": true
  }
}
```


14.5.2 查询集群节点列表

- 接口描述

查询集群节点列表

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/core/cluster/node/list
```

- 请求参数

参数名	参数类型	是否必填	描述说明
address	String	否	节点地址，默认为空
state	String	否	节点状态，默认为空

> *`address` 对应于需要查询的节点地址的前缀匹配条件，为空时不做限制*
>
> *`state` 对应节点状态的筛选条件，为空时不做限制*

- 返回数据

参数名	参数类型	描述说明

data	Object[]	节点列表，详情参见节点详情

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/core/cluster/node/list'
```

- 返回示例


```
{
    "code": 0,
    "message": "success",
    "data": [
        {
            "ip": "172.24.4.154",
            "port": 8848,
            "state": "UP",
            "extendInfo": {
                "lastRefreshTime": 1706087752497,
                "raftMetaData": {
                    "metaDataMap": {
                        "naming_instance_metadata": {
                            "leader": "172.24.4.152:7848",
                            "raftGroupMember": [
                                "172.24.4.163:7848",
                                "172.24.4.154:7848",
                                "172.24.4.152:7848"
                            ],
                            "term": 1
                        },
                        "naming_persistent_service": {
                            "leader": "172.24.4.152:7848",
                            "raftGroupMember": [
                                "172.24.4.163:7848",
                                "172.24.4.154:7848",
                                "172.24.4.152:7848"
                            ],
                            "term": 1
                        },
                        "naming_persistent_service_v2": {
                            "leader": "172.24.4.152:7848",
                            "raftGroupMember": [
                                "172.24.4.163:7848",
                                "172.24.4.154:7848",
                                "172.24.4.152:7848"
                            ],
                            "term": 1
                        },
                        "naming_service_metadata": {
```

```
                        "leader": "172.24.4.152:7848",
                        "raftGroupMember": [
                            "172.24.4.163:7848",
                            "172.24.4.154:7848",
                            "172.24.4.152:7848"
                        ],
                        "term": 1
                    }
                }
            },
            "raftPort": "7848",
            "readyToUpgrade": true,
            "version": "1.0.230"
        },
        "address": "172.24.4.154:8848",
        "failAccessCnt": 0,
        "abilities": {
            "remoteAbility": {
                "supportRemoteConnection": true,
                "grpcReportEnabled": true
            },
            "configAbility": {
                "supportRemoteMetrics": false
            },
            "namingAbility": {
                "supportJraft": true
            }
        },
        "grpcReportEnabled": true
    },
    {
        "ip": "172.24.4.152",
        "port": 8848,
        "state": "UP",
        "extendInfo": {
            "lastRefreshTime": 1706087752877,
            "raftMetaData": {
                "metaDataMap": {
                    "naming_instance_metadata": {
                        "leader": "172.24.4.152:7848",
                        "raftGroupMember": [
                            "172.24.4.163:7848",
                            "172.24.4.154:7848",
                            "172.24.4.152:7848"
                        ],
                        "term": 1
                    },
                    "naming_persistent_service": {
```

```
                    "leader": "172.24.4.152:7848",
                    "raftGroupMember": [
                        "172.24.4.163:7848",
                        "172.24.4.154:7848",
                        "172.24.4.152:7848"
                    ],
                    "term": 1
                },
                "naming_persistent_service_v2": {
                    "leader": "172.24.4.152:7848",
                    "raftGroupMember": [
                        "172.24.4.163:7848",
                        "172.24.4.154:7848",
                        "172.24.4.152:7848"
                    ],
                    "term": 1
                },
                "naming_service_metadata": {
                    "leader": "172.24.4.152:7848",
                    "raftGroupMember": [
                        "172.24.4.163:7848",
                        "172.24.4.154:7848",
                        "172.24.4.152:7848"
                    ],
                    "term": 1
                }
            }
        },
        "raftPort": "7848",
        "readyToUpgrade": true,
        "version": "1.0.230"
    },
    "address": "172.24.4.152:8848",
    "failAccessCnt": 0,
    "abilities": {
        "remoteAbility": {
            "supportRemoteConnection": true,
            "grpcReportEnabled": true
        },
        "configAbility": {
            "supportRemoteMetrics": false
        },
        "namingAbility": {
            "supportJraft": true
        }
    },
    "grpcReportEnabled": true
},
```

```
{
    "ip": "172.24.4.163",
    "port": 8848,
    "state": "UP",
    "extendInfo": {
        "lastRefreshTime": 1705989255350,
        "raftMetaData": {
            "metaDataMap": {
                "naming_instance_metadata": {
                    "leader": "172.24.4.152:7848",
                    "raftGroupMember": [
                        "172.24.4.163:7848",
                        "172.24.4.154:7848",
                        "172.24.4.152:7848"
                    ],
                    "term": 1
                },
                "naming_persistent_service": {
                    "leader": "172.24.4.152:7848",
                    "raftGroupMember": [
                        "172.24.4.163:7848",
                        "172.24.4.154:7848",
                        "172.24.4.152:7848"
                    ],
                    "term": 1
                },
                "naming_persistent_service_v2": {
                    "leader": "172.24.4.152:7848",
                    "raftGroupMember": [
                        "172.24.4.163:7848",
                        "172.24.4.154:7848",
                        "172.24.4.152:7848"
                    ],
                    "term": 1
                },
                "naming_service_metadata": {
                    "leader": "172.24.4.152:7848",
                    "raftGroupMember": [
                        "172.24.4.163:7848",
                        "172.24.4.154:7848",
                        "172.24.4.152:7848"
                    ],
                    "term": 1
                }
            }
        },
        "raftPort": "7848",
        "readyToUpgrade": true,
```

```
                "version": "1.0.230"
            },
            "address": "172.24.4.163:8848",
            "failAccessCnt": 0,
            "abilities": {
                "remoteAbility": {
                    "supportRemoteConnection": true,
                    "grpcReportEnabled": true
                },
                "configAbility": {
                    "supportRemoteMetrics": false
                },
                "namingAbility": {
                    "supportJraft": true
                }
            },
            "grpcReportEnabled": true
        }
    ]
}
```


14.5.3 查询当前节点健康状态

- 接口描述

查询当前ADCC for nacos节点健康状态

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/core/cluster/node/self/health
```

- 返回数据

参数名	参数类型	描述说明
`data`	`String`	当前节点健康状态

> *节点共有* `STARTING`, `UP`, `SUSPICIOUS`, `DOWN`, `ISOLATION` *五种状态*

- 请求示例

```
curl -X GET 'http://127.0.0.1:8848/{adcc_contextpath}/v2/core/cluster/node/self/health'
```

- 返回示例


```
{
    "code": 0,
    "message": "success",
```

```
    "data": "UP"
}
```

14.5.4 切换集群寻址模式

- 接口描述

切换集群寻址模式

- 请求方式

```
PUT
Content-Type:application/x-www-form-urlencoded
```

- 请求URL

```
/{adcc_contextpath}/v2/core/cluster/lookup
```

- 请求Body

参数名	参数类型	是否必填	描述说明
type	String	是	寻址模式

> *寻址模式有两种：`file`（文件配置）和`address-server`（地址服务器）*

- 返回数据

参数名	参数类型	描述
data	boolean	是否执行成功

- 请求示例

```
curl -d 'type=file' \
  -X PUT 'http://127.0.0.1:8848/{adcc_contextpath}/v2/core/cluster/lookup'
```

- 返回示例

```
{
    "code": 0,
    "message": "success",
    "data": true
}
```

14.6 连接负载管理

14.6.1 查询当前节点客户端连接列表

- 接口描述

查询当前ADCC for nacos节点上的客户端连接列表

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/core/loader/current
```

- 返回数据

参数名	参数类型	描述说明
traced	Boolean	是否监控
abilityTable	Map	能力表
metaInfo	Object	元信息
connected	Integer	是否连接
labels	Map	标签

- 请求示例

```
curl -X GET 'http://localhost:8848/{adcc_contextpath}/v2/core/loader/current'
```

- 返回示例

```
{
    "1705989257929_172.24.4.152_42806": {
        "traced": false,
        "abilityTable": {

        },
        "metaInfo": {
            "connectType": "GRPC",
            "clientIp": "172.24.4.152",
            "localPort": 9849,
            "version": "Nacos-Java-Client:v1.0.230",
            "connectionId": "1705989257929_172.24.4.152_42806",
            "createTime": "2024-01-23T13:54:17.944+08:00",
            "lastActiveTime": 1706088386309,
            "appName": "-",
            "tenant": null,
            "labels": {
                "source": "cluster",
                "tls.enable": "false"
            },
            "tag": null,
            "clusterSource": true,
            "sdkSource": false
        },
        "connected": true,
        "labels": {
            "source": "cluster",
            "tls.enable": "false"
        }
    },
    "1705989257993_172.24.4.154_36220": {
        "traced": false,
        "abilityTable": {
```

```
        },
        "metaInfo": {
            "connectType": "GRPC",
            "clientIp": "172.24.4.154",
            "localPort": 9849,
            "version": "Nacos-Java-Client:v1.0.230",
            "connectionId": "1705989257993_172.24.4.154_36220",
            "createTime": "2024-01-23T13:54:18.002+08:00",
            "lastActiveTime": 1706088389756,
            "appName": "-",
            "tenant": null,
            "labels": {
                "source": "cluster",
                "tls.enable": "false"
            },
            "tag": null,
            "clusterSource": true,
            "sdkSource": false
        },
        "connected": true,
        "labels": {
            "source": "cluster",
            "tls.enable": "false"
        }
    }
}
```


14.6.2 重新加载当前节点客户端连接数量

- 接口描述

重新加载当前 `ADCC` 节点的客户端连接数量

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/core/loader/current/reloadCurrent
```

- 请求参数

参数名	参数类型	是否必填	描述说明
`count`	`Integer`	是	连接数量
`redirectAddress`	`String`	否	重定向地址

- 返回数据

参数名	参数类型	描述
`data`	`String`	是否执行成功

- 请求示例

```
curl -X GET 'http://localhost:8848/{adcc_contextpath}/v2/core/loader/reloadCurrent?
count=1&redirectAddress=127.0.0.1:8848'
```

- 返回示例

```
success
```

14.6.3 智能平衡集群节点的连接数

- 接口描述

智能平衡 `ADCC` 集群中所有节点的客户端连接数

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/core/loader/current/smartReloadCluster
```

- 请求参数

参数名	参数类型	是否必填	描述说明
`loaderFactor`	`Float`	否	加载因子，默认加载因子0.1，每个节点的SDK数量（1-loaderFactor）* avg~（1+loaderFactor）* avg
`force`	`String`	否	是否强制

- 返回数据

参数名	参数类型	描述
`data`	`String`	是否执行成功

- 请求示例

```
curl -X GET 'http://localhost:8848/{adcc_contextpath}/v2/core/loader/smartReloadCluster?
loaderFactor=1'
```

- 返回示例

```
Ok
```

14.6.4 重置指定客户端的连接

- 接口描述

根据 `SDK` 连接 `ID` 发送连接重置请求

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/core/loader/current/reloadClient
```

- 请求参数

参数名	参数类型	是否必填	描述说明
connectionId	String	是	连接ID
redirectAddress	String	否	重置地址

- 返回数据

参数名	参数类型	描述
data	String	是否执行成功

- 请求示例

```
curl -X GET 'http://localhost:8848/{adcc_contextpath}/v2/core/loader/reloadClient?
connectionId=1&redirectAddress=127.0.0.1:8848'
```

- 返回示例

```
success
```

14.6.5 获取集群的 SDK 指标

- 接口描述

获取 ADCC 集群中所有的SDK指标

- 请求方式

```
GET
```

- 请求URL

```
/{adcc_contextpath}/v2/core/loader/cluster
```

- 返回数据

参数名	参数类型	描述
total	Integer	当前集群节点数
min	Integer	最小负载值
avg	Integer	平均负载值
max	Integer	最大负载值
memberCount	Integer	当前节点的成员数
metricsCount	Integer	负载信息数量
threshold	Float	负载阈值。阈值的计算公式为：平均负载值 * 1.1
detail	List	包含每个节点的详细负载信息
detail.address	String	节点地址
detail.metric	Map<String,String>	指标信息
completed	Boolean	表示是否已完成负载信息的收集，如果为 true，则表示所有节点的负载信息均已收集，否则为 false

- 请求示例

```
curl -X GET 'http://localhost:8848/{adcc_contextpath}/v2/core/loader/cluster'
```

- 返回示例


```
{
    "total": 1,
    "min": 0,
    "avg": 0,
    "max": 1,
    "memberCount": 3,
    "metricsCount": 3,
    "threshold": 0.0,
    "detail": [
        {
            "address": "172.24.4.152:8848",
            "metric": {
                "load": "0.03",
                "sdkConCount": "0",
                "cpu": "0.008567107",
                "conCount": "2"
            }
        },
        {
            "address": "172.24.4.154:8848",
            "metric": {
                "load": "0.0",
                "sdkConCount": "0",
                "cpu": "0.0026181098",
                "conCount": "2"
            }
        },
        {
            "address": "172.24.4.163:8848",
            "metric": {
                "load": "0.0",
                "sdkConCount": "1",
                "cpu": "0.0044228476",
                "conCount": "3"
            }
        }
    ],
    "completed": true
}
```

全国统一服务热线

4008-555-800

金蝶天燕云计算股份有限公司(简称“金蝶天燕云”)成立于2000年，前身为“金蝶中间件公司”，是金蝶集团旗下新一代软件基础云平台服务商，云计算国家标准制定企业，国家信创产业核心软件企业。金蝶天燕是国家863重点研发计划与核高基重大专项承接企业，也是“两网一站四库十二金”国家重点工程的基础平台提供商，产品广泛应用于政府、军工、金融、能源等关键行业，累计服务客户总数超过10万家。

Apusic 金蝶天燕

云计算国家标准制定企业

金蝶集团旗下基础软件企业

信息技术应用创新核心企业

官网：www.apusic.com