

# 微服务项目-京锋购 01 - Spring Cloud Alibaba

## Nacos

---

京锋购

Spring Cloud Alibaba Nacos

Nacos

面试题

什么是 Nacos

为什么是Nacos

可以干什么

Nacos快速开始

启动 Nacos server

启动 nacos-provider

查看 nacos 服务列表

启动 nacos-consumer

查看 nacos 服务列表

使用 Feign 调用服务

Nacos 配置中心

使用 Nacos 创建统一配置

新建配置

配置内容

从配置中心读取配置

名称空间切换环境

回滚配置

加载多配置文件

配置的分组

## 京锋购

京东 X 砺锋 = 京锋购商城

## 京锋购 - 微服务架构图



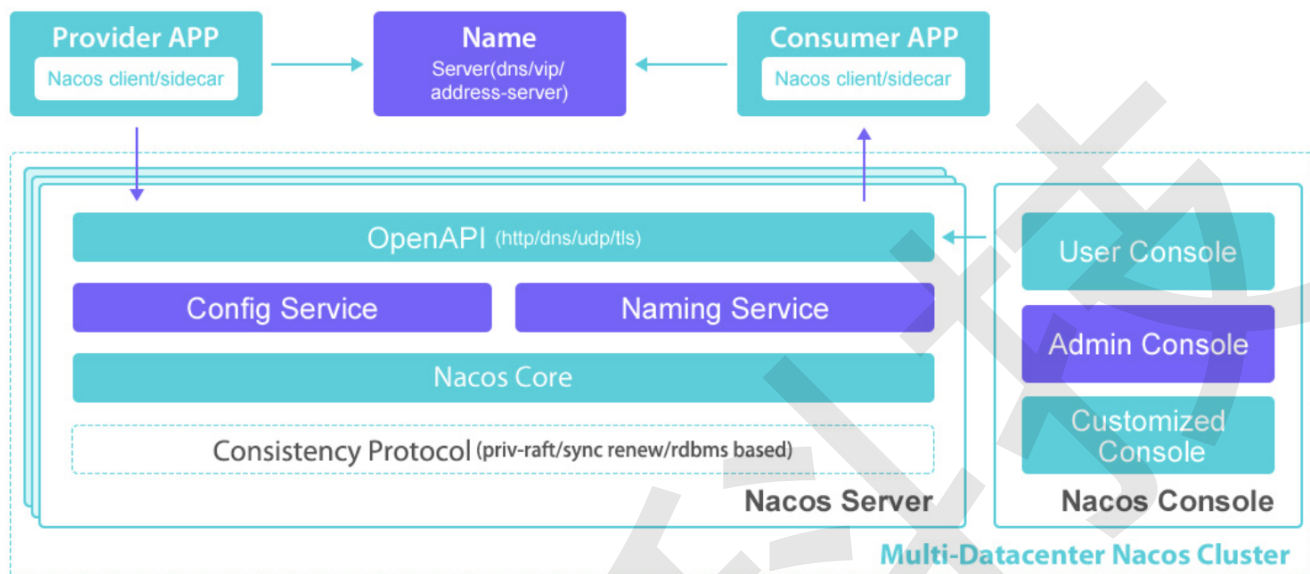
## Spring Cloud Alibaba Nacos

# NACOS.

动态服务发现、配置管理和服务管理平台。

# Nacos 架构

## 基本架构及概念



官方地址：<https://nacos.io>

github地址：<https://github.com/alibaba/nacos>

## Nacos

### 面试题

微服务间远程交互的过程？

1. 先去注册中心查询服务的服务器地址
2. 调用方给对方发送http请求

### 什么是 Nacos

Nacos 是阿里巴巴推出的一个开源项目，这是一个更易于构建云原生应用的动态服务发现、配置管理和服务平台。

Nacos 致力于帮助您发现、配置和管理微服务。

Nacos 提供了一组简单易用的特性集，帮助您快速实现动态服务发现、服务配置、服务元数据及流量管理。

Nacos 帮助您更敏捷和容易地构建、交付和管理微服务平台。

Nacos 是构建以“服务”为中心的现代应用架构（例如微服务范式、云原生范式）的服务基础设施。

## 为什么是Nacos

常见的注册中心：

1. Eureka（原生，2.0遇到性能瓶颈，停止维护）
2. Zookeeper（支持，专业的独立产品。例如：dubbo）
3. Consul（原生，GO语言开发）
4. Nacos

相对于 Spring Cloud Eureka 来说，Nacos 更强大。

Nacos = Spring Cloud Eureka + Spring Cloud Config.

Nacos 可以与 Spring, Spring Boot, Spring Cloud 集成，并能代替 Spring Cloud Eureka, Spring Cloud Config。

- 通过 Nacos Server 和 `spring-cloud-starter-alibaba-nacos-config` 实现配置的动态变更。
- 通过 Nacos Server 和 `spring-cloud-starter-alibaba-nacos-discovery` 实现服务的注册与发现。

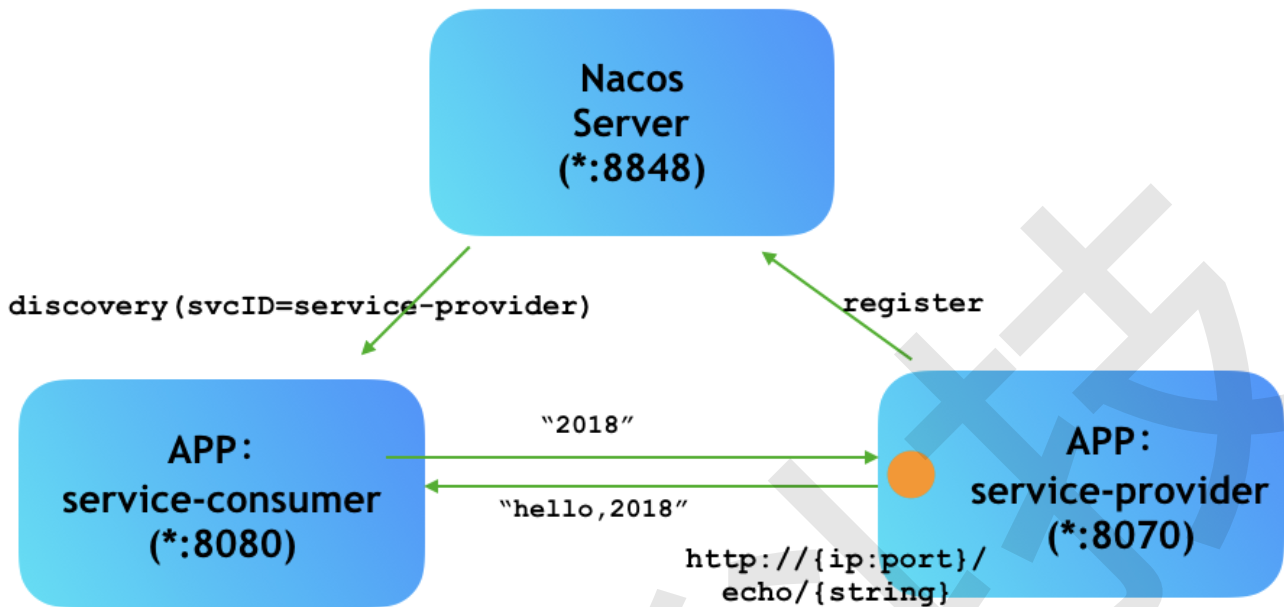
## 可以干什么

Nacos是以服务为主要服务对象的中间件，Nacos支持所有主流的服务发现、配置和管理。

Nacos主要提供以下四大功能：

1. 服务发现和服务健康监测
2. 动态配置服务
3. 动态DNS服务
4. 服务及其元数据管理

# Nacos快速开始



把服务注册到 Nacos 主要分3步：

1. 添加依赖
2. 在application.properties中配置nacos的服务名及服务地址
3. 在引导类（NacosConsumerApplication.java）中添加@EnableDiscoveryClient注解

官网已经非常详细，自行参考官网的文档进行操作。

## 启动 Nacos server

下载后解压，执行脚本即可启动服务端：

```

+ ~/nacos-1.4.2/bin ls
derby.log shutdown.cmd startup.cmd work
logs shutdown.sh startup.sh
+ ~/nacos-1.4.2/bin sh startup.sh -m standalone
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java -Xms512m -Xmx512m -Xmn256m -Dnacos.standalone=true -Dnacos.member.list= -Xlog:gc*:file=/Users/szw/nacos-1.4.2/logs/nacos_gc.log:time,tags:filecount=10,filesize=102400 -Dloader.path=/Users/szw/nacos-1.4.2/plugins/health,/Users/szw/nacos-1.4.2/plugins/cmdb -Dnacos.home=/Users/szw/nacos-1.4.2 -jar /Users/szw/nacos-1.4.2/target/nacos-server.jar --spring.config.additional-location=file:/Users/szw/nacos-1.4.2/conf/ --logging.config=/Users/szw/nacos-1.4.2/conf/nacos-logback.xml --server.max-http-header-size=524288
nacos is starting with standalone
nacos is starting, you can check the /Users/szw/nacos-1.4.2/logs/start.out
+ ~/nacos-1.4.2/bin tail -f /Users/szw/nacos-1.4.2/logs/start.out
2021-08-19 10:38:48,593 INFO Creating filter chain: any request, [org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter@a6e4897, org.springframework.security.web.context.SecurityContextPersistenceFilter@511f5b1d, org.springframework.security.web.header.HeaderWriterFilter@474179fa, org.springframework.security.web.csrf.CsrfFilter@7bc58891, org.springframework.security.web.authentication.logout.LogoutFilter@7ccfdaef, org.springframework.security.web.savedrequest.RequestCacheAwareFilter@7a687d8d, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@7ba93755, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@32f5ecc4, org.springframework.security.web.session.SessionManagementFilter@ca8ef3a, org.springframework.security.web.access.ExceptionTranslationFilter@2abc8034]
2021-08-19 10:38:48,686 INFO Initializing ExecutorService 'taskScheduler'
2021-08-19 10:38:48,704 INFO Exposing 2 endpoint(s) beneath base path '/actuator'
2021-08-19 10:38:48,805 INFO Tomcat started on port(s): 8848 (http) with context path '/nacos'
2021-08-19 10:38:48,809 INFO Nacos started successfully in stand alone mode. use embedded storage
2021-08-19 10:39:03,295 INFO Initializing Servlet 'dispatcherServlet'

```

## 启动 nacos-provider

```

2021-08-19 10:42:20.527 INFO 61207 --- [main] o.s.s.c.ThreadPoolTaskScheduler : Initializing ExecutorService
'Nacos-Watch-Task-Scheduler'
2021-08-19 10:42:20.888 INFO 61207 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8070 (http) with
context path ''
2021-08-19 10:42:21.155 INFO 61207 --- [main] c.a.c.n.registry.NacosServiceRegistry : nacos registry, DEFAULT_GROUP nacos-provider
192.168.2.96:8070 register finished
2021-08-19 10:42:21.165 INFO 61207 --- [main] c.r.n.NacosProviderApplication : Started NacosProviderApplication in 2.812 seconds
(JVM running for 4.195)

```

## 查看 nacos 服务列表

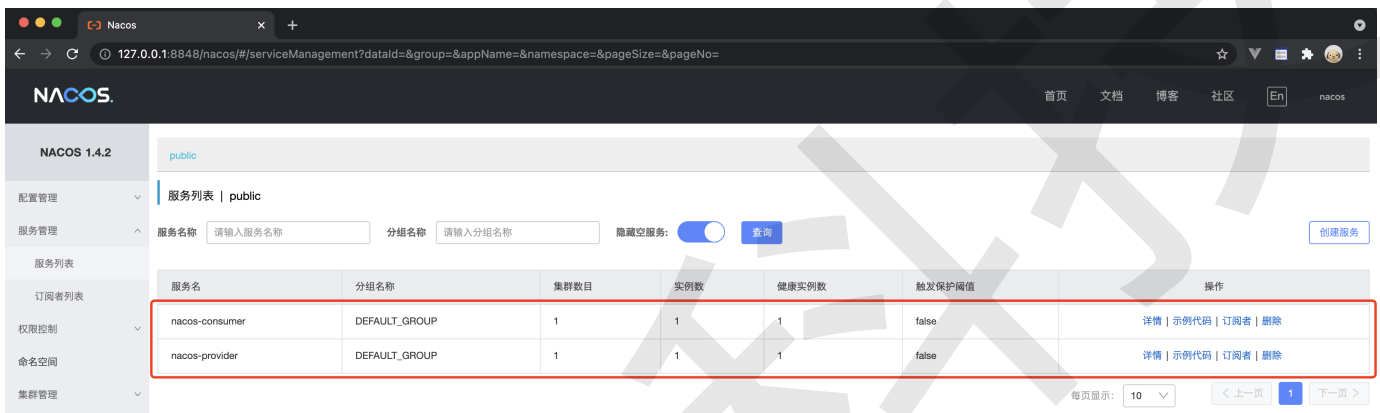
The screenshot shows the Nacos web interface at 127.0.0.1:8848. The 'public' namespace is selected. The 'Service List' tab is active, displaying a table of services. The table has columns for Service Name, Group Name, Cluster Count, Instance Count, Healthy Instance Count, Trigger Protection Threshold, and Actions. One service, 'nacos-provider', is listed with a group name of 'DEFAULT\_GROUP', 1 cluster, 1 instance, and 1 healthy instance. The trigger protection threshold is 'false'. The actions column includes links for 'Details', 'View Code', 'Subscribe', and 'Delete'.

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	操作
nacos-provider	DEFAULT_GROUP	1	1	1	false	<a href="#">详情</a>   <a href="#">示例代码</a>   <a href="#">订阅者</a>   <a href="#">删除</a>

## 启动 nacos-consumer

```
Run: NacosProviderApplication x NacosConsumerApplication x
2021-08-19 11:34:00.274 INFO 71520 --- [main] o.s.s.concurrent.InheritableTaskExecutor : Initializing ExecutorService
'applicationTaskExecutor'
2021-08-19 11:34:00.375 INFO 71520 --- [main] o.s.s.c.ThreadPoolTaskScheduler : Initializing ExecutorService
'Nacos-Watch-Task-Scheduler'
2021-08-19 11:34:00.699 INFO 71520 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 18072 (http) with
context path ''
2021-08-19 11:34:00.710 INFO 71520 --- [main] c.a.c.n.registry.NacosServiceRegistry : nacos registry, DEFAULT_GROUP nacos-consumer
192.168.2.96:18072 register finished
2021-08-19 11:34:00.719 INFO 71520 --- [main] c.r.n.NacosConsumerApplication : Started NacosConsumerApplication in 1.612 seconds
(JVM running for 2.232)
```

## 查看 nacos 服务列表



## 使用 Feign 调用服务

以前我们用 Dubbo 来进行远程调用，现在我们改用 Feign。

首先添加依赖：

```
XML 复制代码
1 <dependency>
2   <groupId>org.springframework.cloud</groupId>
3   <artifactId>spring-cloud-starter-openfeign</artifactId>
4 </dependency>
```

在 NacosConsumerApplication 类上添加 @EnableFeignClients 注解：

```
1 package com.rushuni.nacosconsumer;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
6 import org.springframework.cloud.openfeign.EnableFeignClients;
7
8 /**
9  * @author rushuni
10  * @date 2021/8/19
11  */
12 @SpringBootApplication
13 @EnableDiscoveryClient
14 @EnableFeignClients
15 public class NacosConsumerApplication {
16
17     public static void main(String[] args) {
18         SpringApplication.run(NacosConsumerApplication.class, args);
19     }
20
21 }
```

## 编写 FeignClient

```
1 package com.rushuni.nacosconsumer.feign;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.cloud.openfeign.FeignClient;
5 import org.springframework.web.bind.annotation.RequestMapping;
6
7 /**
8  * @author rushuni
9  * @date 2021/08/19
10  */
11 @FeignClient("nacos-provider")
12 public interface ProviderFeign {
13
14     /**
15      * 用于测试 Feign 远程调用
16      * @return
17      */
18     @RequestMapping("hello")
19     String hello();
20
21 }
```



```
1 package com.rushuni.nacosconsumer.controller;
2
3 import com.rushuni.nacosconsumer.feign.ProviderFeign;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.RestController;
7
8 /**
9  * @author rushuni
10  * @date 2021/08/19
11  */
12 @RestController
13 public class ConsumerController {
14
15     @Autowired
16     private ProviderFeign providerFeign;
17
18     @GetMapping("hi")
19     public String hi() {
20         // return "hi provider!";
21         return this.providerFeign.hello();
22     }
23 }
```

## Nacos 配置中心

在系统开发过程中，开发者通常会将一些需要变更的参数、变量等从代码中分离出来独立管理，以独立的配置文件的形式存在。

这样做的目的是让静态的系统工件或者交付物（如 WAR，JAR 包等）更好地和实际的物理运行环境进行适配。

配置管理一般包含在系统部署的过程中，由系统管理员或者运维人员完成。

配置变更是调整系统运行时的行为的有效手段。

如果微服务架构中没有使用统一配置中心时，所存在的问题：

- 配置文件分散在各个子项目中，不方便维护
- 配置内容安全与权限
- 更新配置后，项目需要重启

使用 Nacos 配置中心：

- 系统配置的集中管理（编辑、存储、分发）
- 动态更新不重启
- 回滚配置（变更管理、历史版本管理、变更审计）等所有与配置相关的活动。

现在改造生产者中的动态配置项，由配置中心统一管理。

## 使用 Nacos 创建统一配置

### 新建配置

dataId 的格式说明：

```
1 ${prefix}-${spring.profile.active}.${file-extension}
```

Shell

复制代码

- prefix 默认为所属工程配置 spring.application.name 的值（即：nacos-provider）
- 也可以通过配置项 spring.cloud.nacos.config.prefix 来配置。
- spring.profile.active 即为当前环境对应的 profile，详情可以参考 [Spring Boot文档] (<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-profiles.html#boot-features-profiles>)。
- 注意：当 spring.profile.active 为空时，对应的连接符 - 也将不存在，dataId 的拼接格式变成 \${prefix}.\${file-extension}
- file-extension 为配置内容的数据格式，可以通过配置项 spring.cloud.nacos.config.file-extension 来配置。

总结：配置所属工程的 spring.application.name 的值 + "." + properties/yml

### 配置内容

对项目中的易变的内容进行配置。例如变量：myName

我们创建的 nacos-provider 工程的 spring.application.name=nacos-provider，没有配置 spring.profiles.active。所以这里的数据Id填写的是 nacos-provider.yml

# 新建配置

\* Data ID:

\* Group:

[更多高级选项](#)

描述:

配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

\* 配置内容: (?) :

```
1  # 定义的参数
2  myName: nacos-provider
```

## 从配置中心读取配置

从配置中心读取配置，分以下3步：

1. 引入依赖

在生产者中引入依赖：

XML | 复制代码

```
1 <dependency>
2   <groupId>com.alibaba.cloud</groupId>
3   <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
4 </dependency>
```

2. 在 bootstrap.yml 中配置 Nacos server 的地址和应用名

```

1  spring:
2    cloud:
3      nacos:
4        config:
5          server-addr: 127.0.0.1:8848
6    application:
7      # 该配置影响统一配置中心中的dataId, 之前已经配置过
8      name: nacos-provider

```

说明：之所以需要配置 `spring.application.name`，是因为它是构成 Nacos 配置管理 `dataId` 字段的一部分。

另外，在 `springboot` 工程中，`bootstrap.properties(yml)` 的加载优先级更高。

### 3. 通过 Spring Cloud 原生注解 `@RefreshScope` 实现配置自动更新：

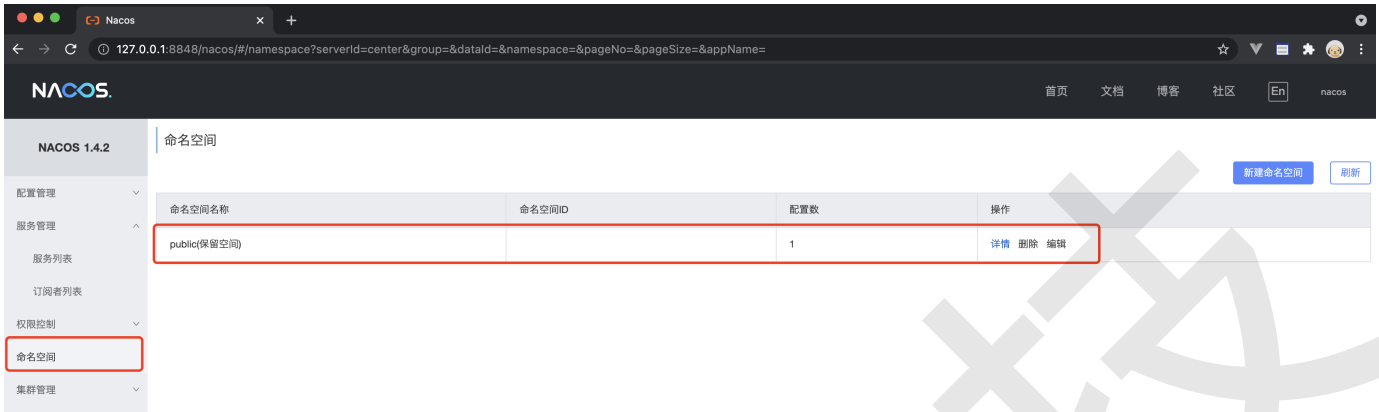
```

1  package com.rushuni.nacosprovider.controller;
2
3  import org.springframework.beans.factory.annotation.Value;
4  import org.springframework.cloud.context.config.annotation.RefreshScope;
5  import org.springframework.web.bind.annotation.GetMapping;
6  import org.springframework.web.bind.annotation.RestController;
7
8  /**
9   * @author rushuni
10  * @date 2021/08/19
11  */
12  @RestController
13  @RefreshScope
14  public class ProviderController {
15
16      @Value("${myName}")
17      private String name;
18
19      @GetMapping("hello")
20      public String hello(){
21          return "hello " + name;
22      }
23  }

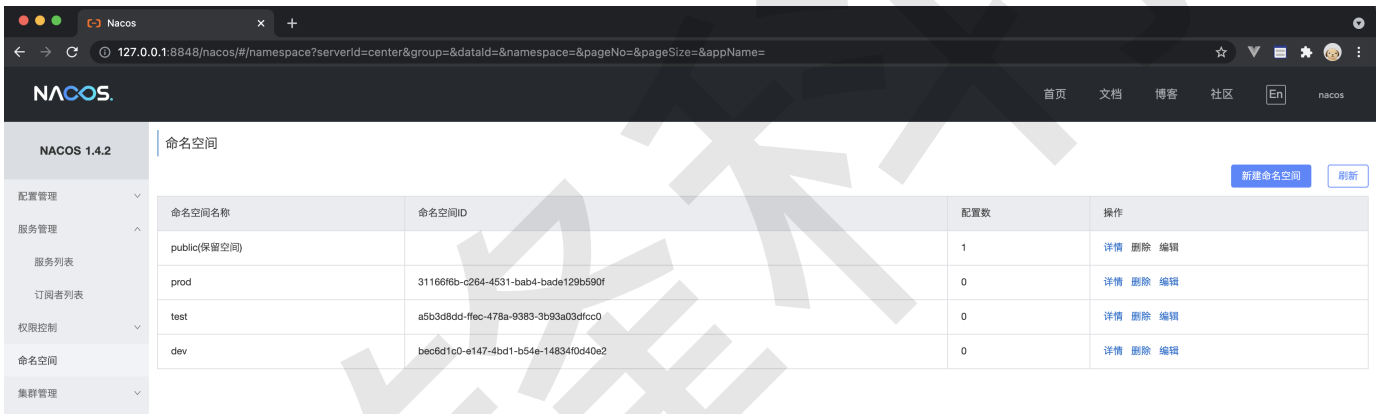
```

## 名称空间切换环境

在实际开发中，通常有多套不同的环境（默认只有public），那么这个时候可以根据指定的环境来创建不同的 namespace，例如常见的开发、测试和生产三个不同的环境。那么使用一套 nacos 集群可以分别建以下三个不同的 namespace，以此来实现多环境的隔离。

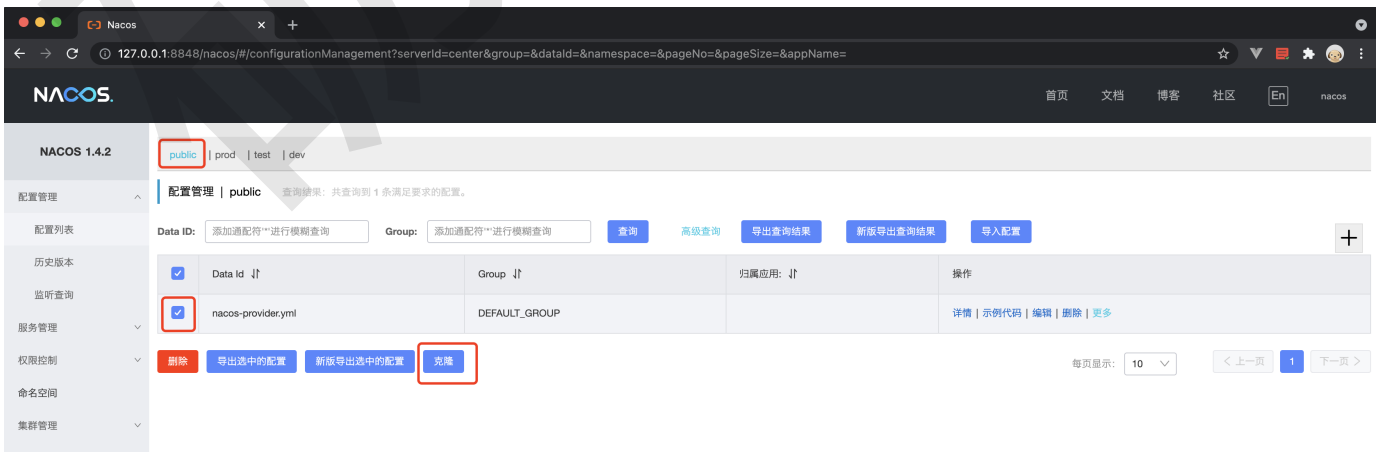


新增命名空间

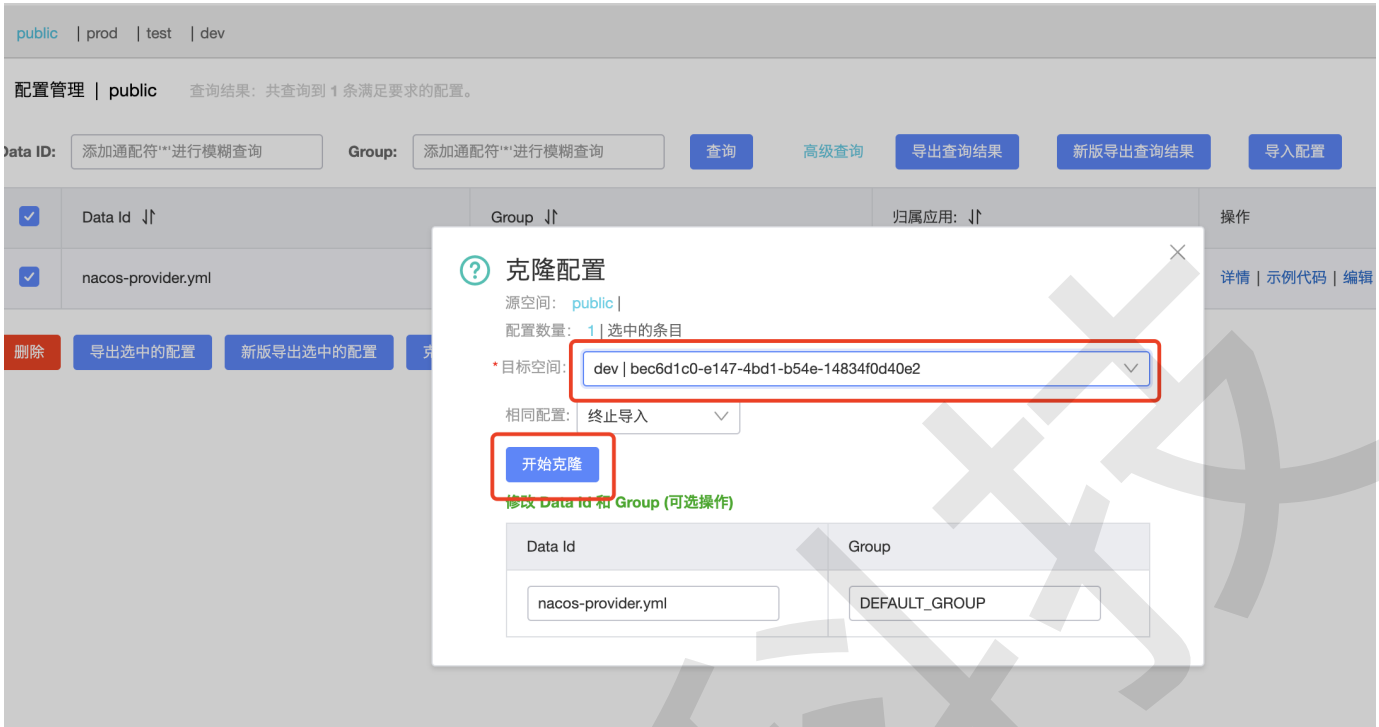


此时给不同的命名空间添加配置有两种方式：

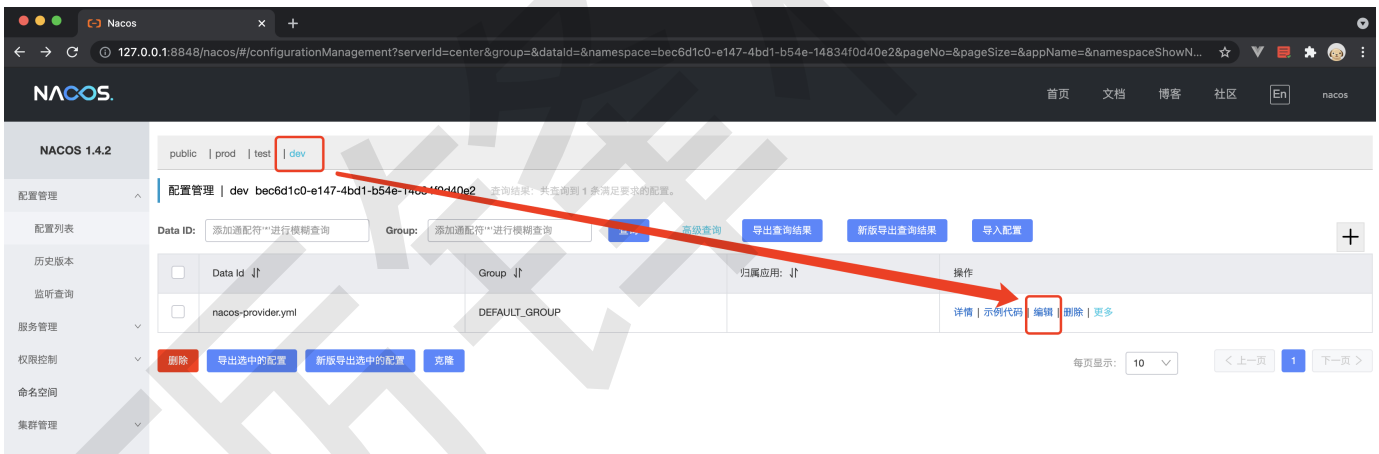
- 1. 切换到其中一个名称空间中，添加一个新的配置文件（缺点：每个环境都要重复配置类似的项目）。
- 2. 直接通过 clone 方式添加配置，并修改即可（推荐）。



接着选择克隆到的命名空间：



修改一下内容已作区分：



## 编辑配置

\* Data ID:

\* Group:

[更多高级选项](#)

描述:

Beta发布: ☐ 默认不要勾选。


配置格式: ☐ TEXT ☐ JSON ☐ XML ☒ YAML ☐ HTML ☐ Properties

配置内容 :

```
1 # 定义的参数
2 myName: nacos-provider - from dev
```

在服务提供方 nacos-provider 中切换命名空间，修改 bootstrap.yml 添加如下配置：

YAML

 复制代码

```
1 spring:
2   cloud:
3     nacos:
4       config:
5         server-addr: 127.0.0.1:8848
6         namespace: bec6d1c0-e147-4bd1-b54e-14834f0d40e2
```

namespace的值：

public | prod | test | dev

配置管理 | dev 

bec6d1c0-e147-4bd1-b54e-14834f0d40e2

 查询结果: 共查询到 1 条满足要求的配置。

Data ID:

Group:

查询

<input type="checkbox"/>	Data Id ⬆️⬆️	Group ⬆️⬆️
<input type="checkbox"/>	nacos-provider.yml	DEFAULT_GROUP

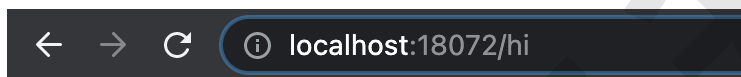
删除

导出选中的配置

新版导出选中的配置

克隆

重启服务提供方服务，在浏览器中访问测试：



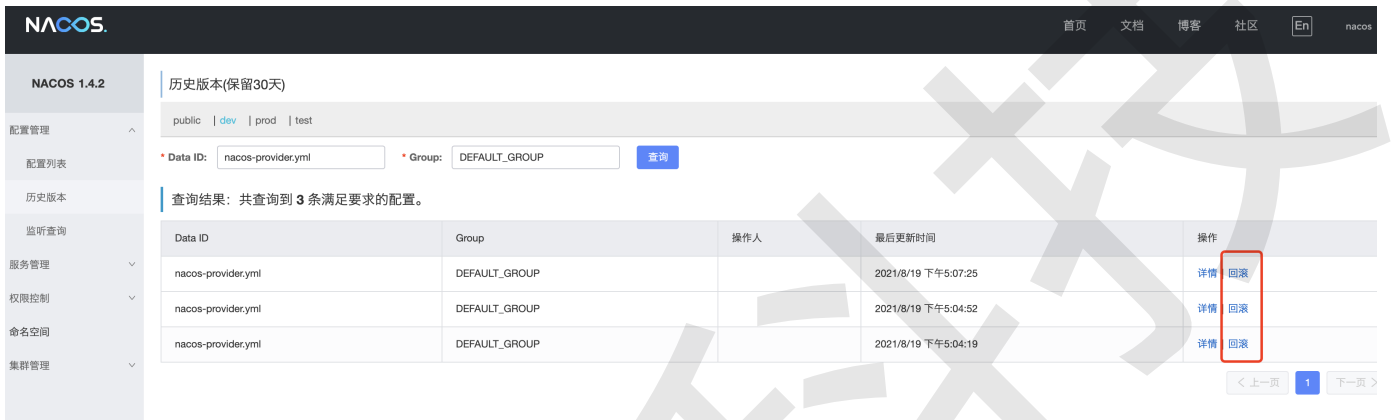
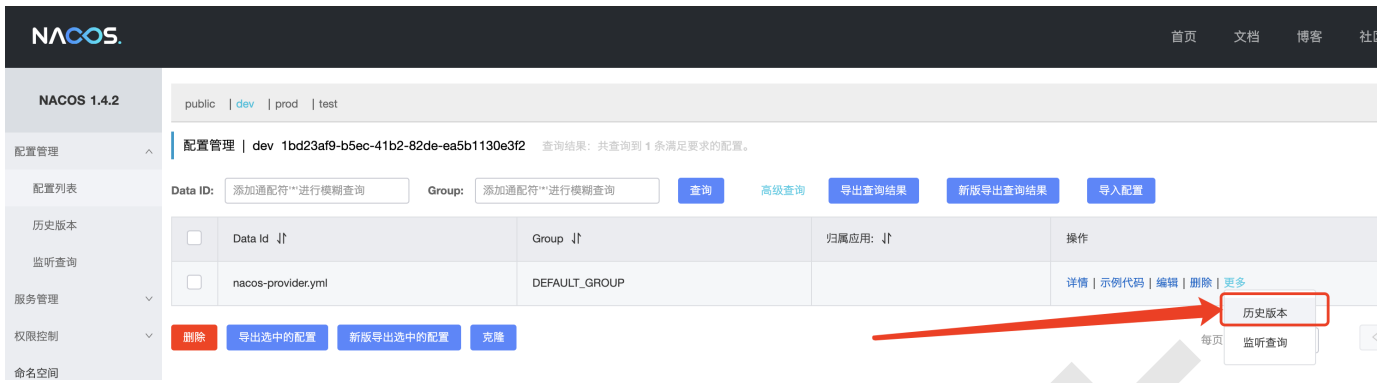
hello nacos-provider-from-dev

## 回滚配置

回滚配置只需要两步：

1. 查看历史版本
2. 回滚到某个历史版本





## 加载多配置文件

Nacos 可以加载多个配置文件。假如现在 dev 名称空间下有三个配置文件：nacos-provider.yml、redis.yml、jdbc.yml



nacos-provider.yml 默认加载，怎么加载另外两个配置文件？

在 bootstrap.yml 文件中添加如下配置：

```
1 extension-configs:  
2   - data-id: jdbc.yml  
3     refresh: true  
4   - data-id: redis.yml  
5     refresh: true
```

## 配置的分组

在实际开发中，除了不同的环境外。不同的微服务或者业务功能，可能有不同的 redis 及 mysql 数据库。

区分不同的环境我们使用名称空间（namespace），区分不同的微服务或功能，使用分组（group）。

当然，也可以反过来使用，名称空间和分组只是为了更好的区分配置，提供的两个维度而已。

新增一个 redis.properties，所属分组为 provider：

配置管理 | dev 1bd23af9-b5ec-41b2-82de-ea5b1130e3f2 查询结果: 共查询到 4 条满足要求的配置。

Data ID:

添加通配符<sup>1\*</sup>进行模糊查询

Group:

添加通配符<sup>1\*</sup>进行模糊查询

查询

<input type="checkbox"/>	Data Id ⌵⌴	Group ⌵⌴
<input type="checkbox"/>	nacos-provider.yml	DEFAULT_GROUP
<input type="checkbox"/>	jdbc.yml	DEFAULT_GROUP
<input type="checkbox"/>	redis.yml	DEFAULT_GROUP
<input type="checkbox"/>	redis.yml	provider

删除

导出选中的配置

新版导出选中的配置

克隆

现在开发环境中有两个 redis.yml 配置文件，一个是默认分组（DEFAULT\_GROUP），一个是 provider 组

默认情况下从 DEFAULT\_GROUP 分组中读取 redis.yml，如果要切换到 provider 分组下的 redis.yml，需要添加如下配置：

```
1 extension-configs:
2   - data-id: redis.yml
3     refresh: true
4     group: provider
```

YAML | 复制代码

缺点：将来每个分组下会有太多的配置文件，不利于维护。

所以，一种最佳实践是命名空间区分业务功能，分组区分环境。

研銳科技