【**Nacos** 配置文件中心】

1. **Nacos** 简介

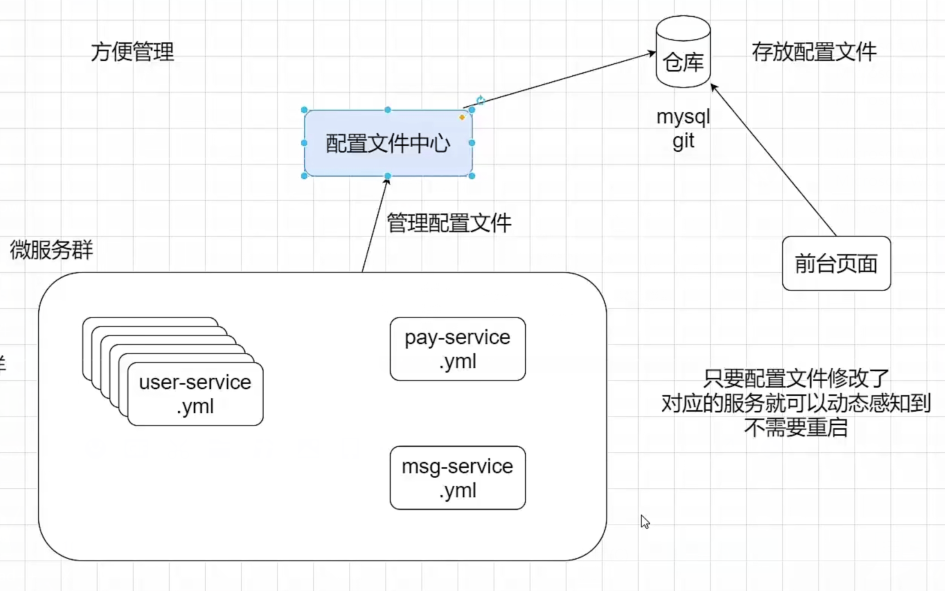


使用 Spring Cloud Alibaba Nacos Config，可基于 Spring Cloud 的编程模型快速接入 Nacos 配置管理功能。

上一节 Spring Cloud Alibaba Nacos 注册中心记录了 Nacos 作为注册中心的使用方式，这节继续记录下 Nacos 作为配置中心的使用方式。本节使用的 Spring Cloud 版本为 Hoxton.SR9 ， Spring Cloud Alibaba 版本为 2.2.6.RELEASE ， Spring Boot 版本为 2.3.2.RELEASE。

(如集群的，只有ip和port不一样，其他的都一样，那就可以把公共的抽取出来。

（若热部署，修改一点配置，服务无需重启就能更新



1. 创建项目 **config-client-a**

## 创建项目选择依赖

*<?*xml version="1.0" encoding="UTF-8"*?>*

<project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.3.2.RELEASE</version>

<relativePath/> *<!-- lookup parent from repository -->*

</parent>

<groupId>com.powernode</groupId>

<artifactId>01-config-client-a</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>config-client-a</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

<spring-cloud-alibaba.version>2.2.6.RELEASE</spring-cloud-alibaba.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

**<dependency>**

**<groupId>com.alibaba.cloud</groupId>**

**<artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>**

**</dependency>**

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>com.alibaba.cloud</groupId>

<artifactId>spring-cloud-alibaba-dependencies</artifactId>

<version>${spring-cloud-alibaba.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

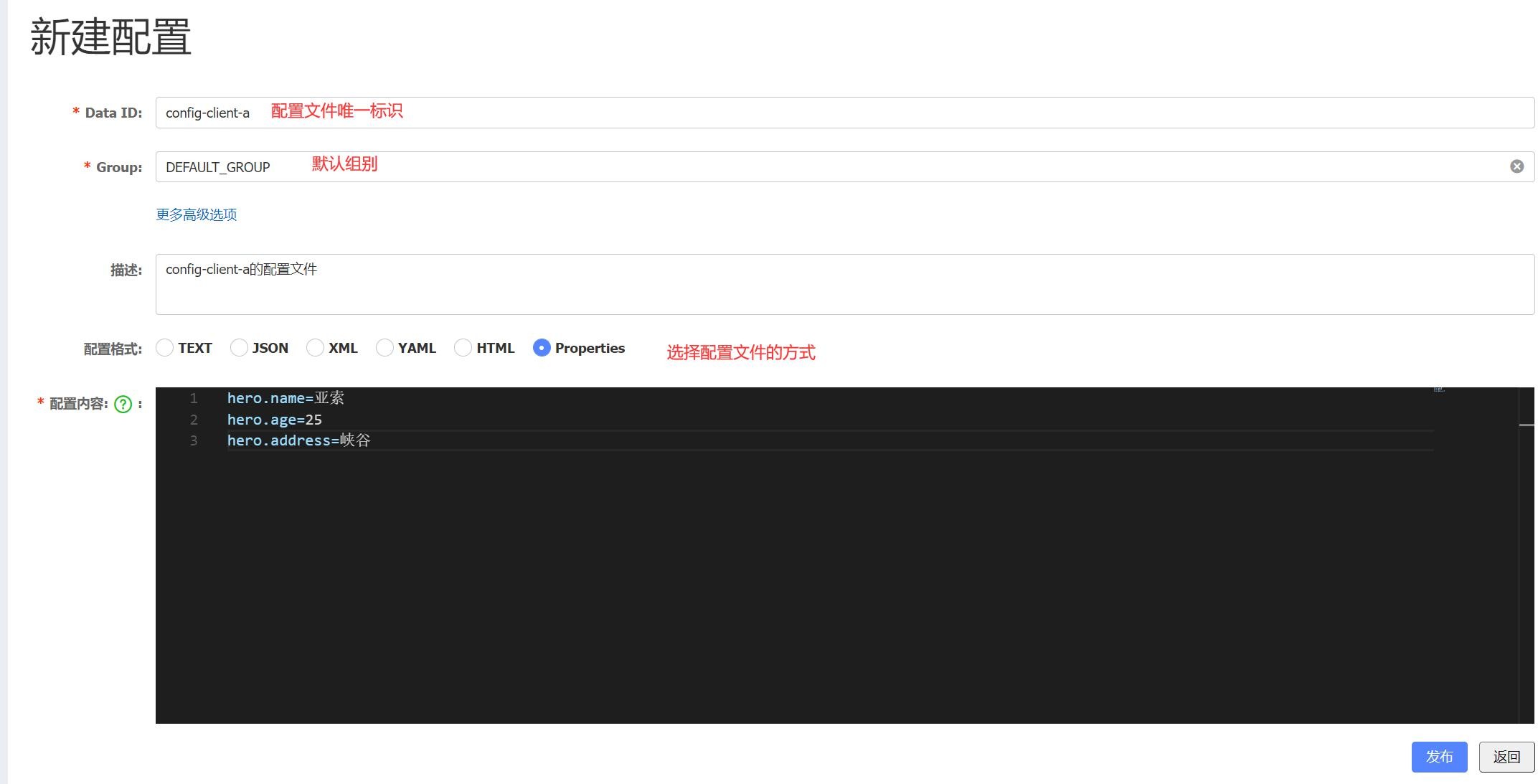
</build>

</project>

* 1. 在 **NacosServer** 里面添加一个配置文件

点击添加按钮：



填写具体信息：

发布配置文件：



返回查看配置文件发布成功了

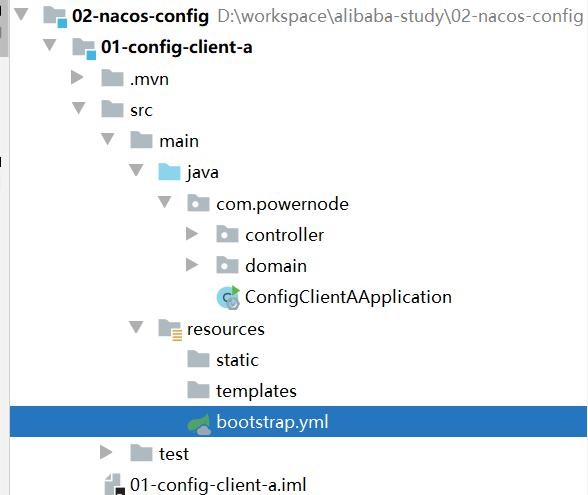


## 查看上一讲中创建的数据库中的信息



我们添加的配置文件就是插入到数据库里面的一条数据

* 1. **config-client-a** 项目中添加一个配置文件 **bootstrap.yml**

注意：不是 **application.yml** ，**bootstrap.yml** 比 **application** 有更高的优先级。

server:

port: 8080 spring:

application:

name: config-client-a

cloud:

nacos:

config: #指定配置中心的地址和配置中心使用的数据格式 server-addr: localhost:8848

file-extension: properties

* 1. **config-client-a** 中添加一个实体类 **Hero**

package com.bjpowernode.domain;

import lombok.AllArgsConstructor; import lombok.Data;

import lombok.NoArgsConstructor;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.cloud.context.config.annotation.RefreshScope; import org.springframework.stereotype.Component;

/\*\*

\* @Author 武汉动力节点

\*/ @Data

@AllArgsConstructor @NoArgsConstructor

@Component // 添加到IOC 中，一会在 controller 注入

@RefreshScope // 刷新的域，当配置文件修改后可以动态刷新

public class Hero {

@Value("${hero.name}") private String name; @Value("${hero.age}") private Integer age; @Value("${hero.address}") private String address;

}

* 1. **config-client-a** 中添加一个测试类 **Controller**

package com.bjpowernode.controller;

import com.bjpowernode.domain.Hero;

import org.springframework.beans.factory.annotation.Autowired; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RestController;

/\*\*

\* @Author 武汉动力节点

\*/ @RestController

public class HeroController {

/\*\*

* 注入hero

\*/ @Autowired

private Hero hero;

/\*\*

* 获取信息的接口

\*

* @return

\*/ @GetMapping("heroInfo") public String heroInfo() {

return hero.getName() + ":" + hero.getAge() + ":" + hero.getAddress();

}

}

## 启动测试

访问 http://localhost:8080/heroInfo

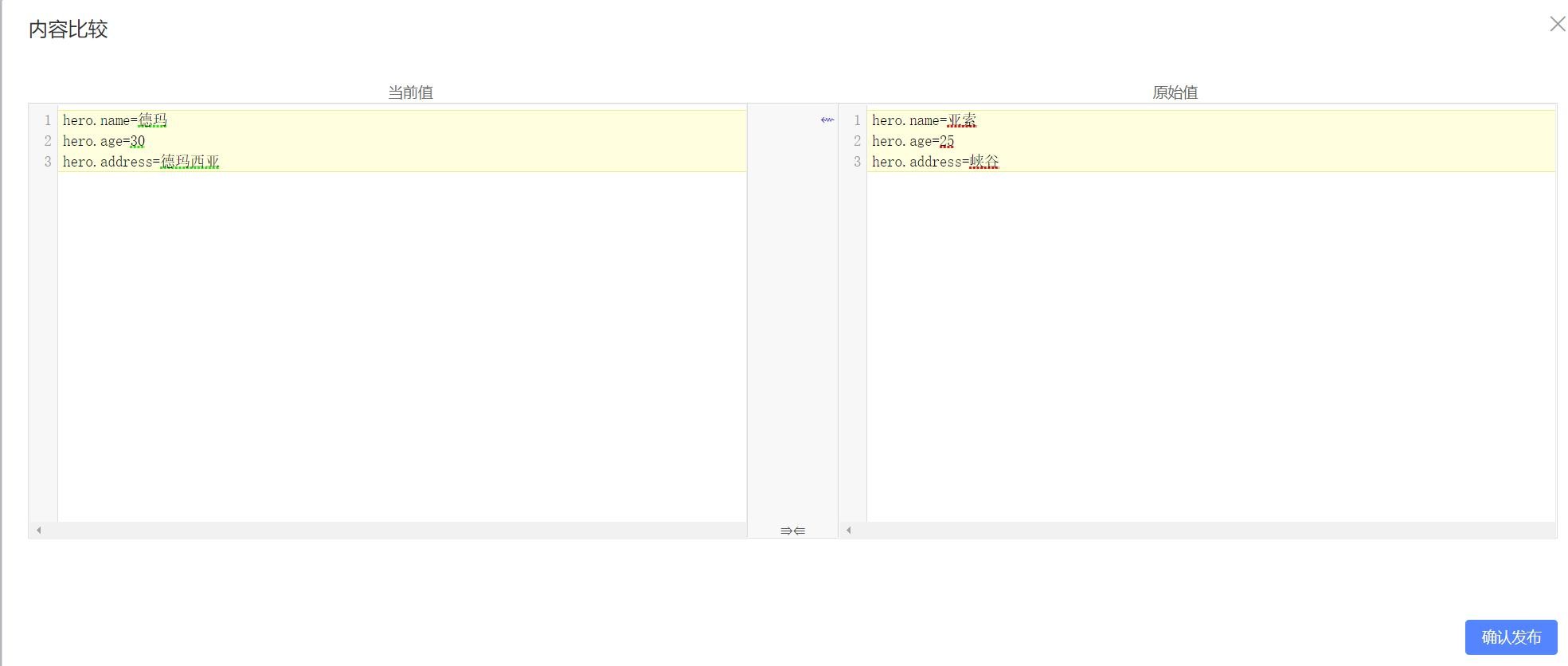
（这数据以字符串的形式放在了nacos数据库的config\_info表中了

（热更新@RefreshScope *// 给这个类上 添加一个刷新的作用域.热部署*

## 测试配置文件的动态刷新

修改配置文件



编辑后配置文件对比

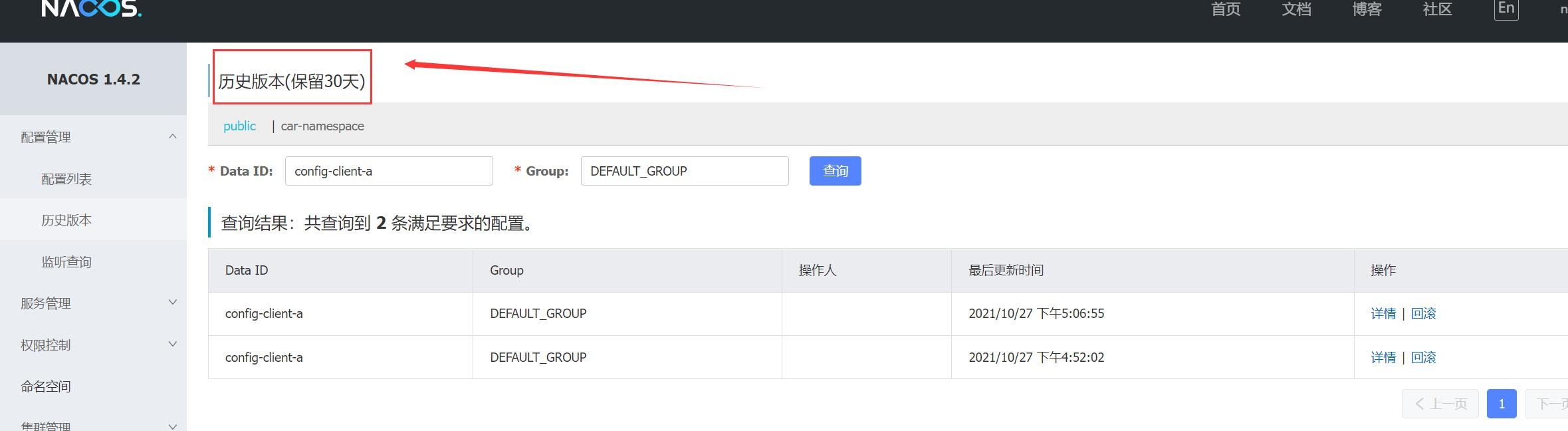
不需要重启，直接请求 http://localhost:8080/heroInfo 查看结果已经刷新了



## 配置文件的历史版本查询

（历史版本的信息放在了his\_config\_info表中



点击历史版本查看，注意历史版本只保留 30 天

## 配置文件的回滚

配置文件从上往下回滚，所以我们点击最上面的那一条

此时访问：http://localhost:8080/heroInfo



# 配置文件的读取方式【重点】

nacos 配置中心通过 namespace、dataId 和 group 来唯一确定一条配置。

* Namespace：即命名空间。默认的命名空间为 public，我们可以在 Nacos 控制台中新建命名空间；
* dataId：即配置文件名称
* Group ： 即 配 置 分 组 ， 默 认 为 DEFAULT\_GROUP ， 可 以 通 过

spring.cloud.nacos.config.group 配置。其中：dataId 是最关键的配置字段：格式如下：

**${prefix} - ${spring.profiles.active} . ${file-extension}**

说明：

* prefix 默 认 为 spring.application.name 的 值 ， 也 可 以 通 过 配 置 项

spring.cloud.nacos.config.prefix 来配置；

* spring.profiles.active 即 为 当 前 环 境 对 应 的 profile 。 注 意 ， 当 spring.profiles.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。这就是上面我们为什么能获得到配置的原因了。

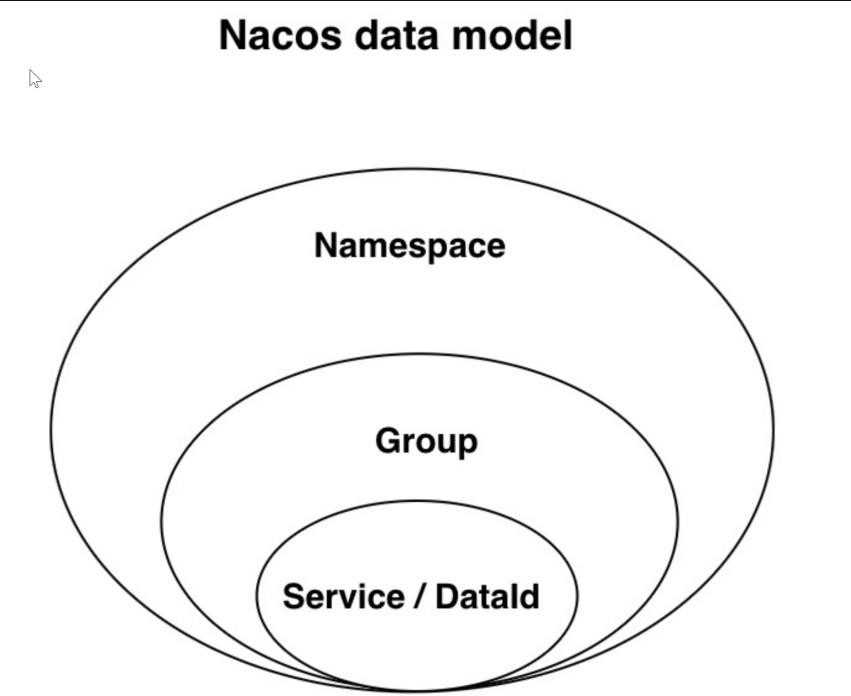
注意:在写 dataId 的时候一定要添加文件类型后缀

Eg: nacos-config-dev**.yml**

（通过这三个来指定，可以很方便切换配置文件

（

# 配置文件划分

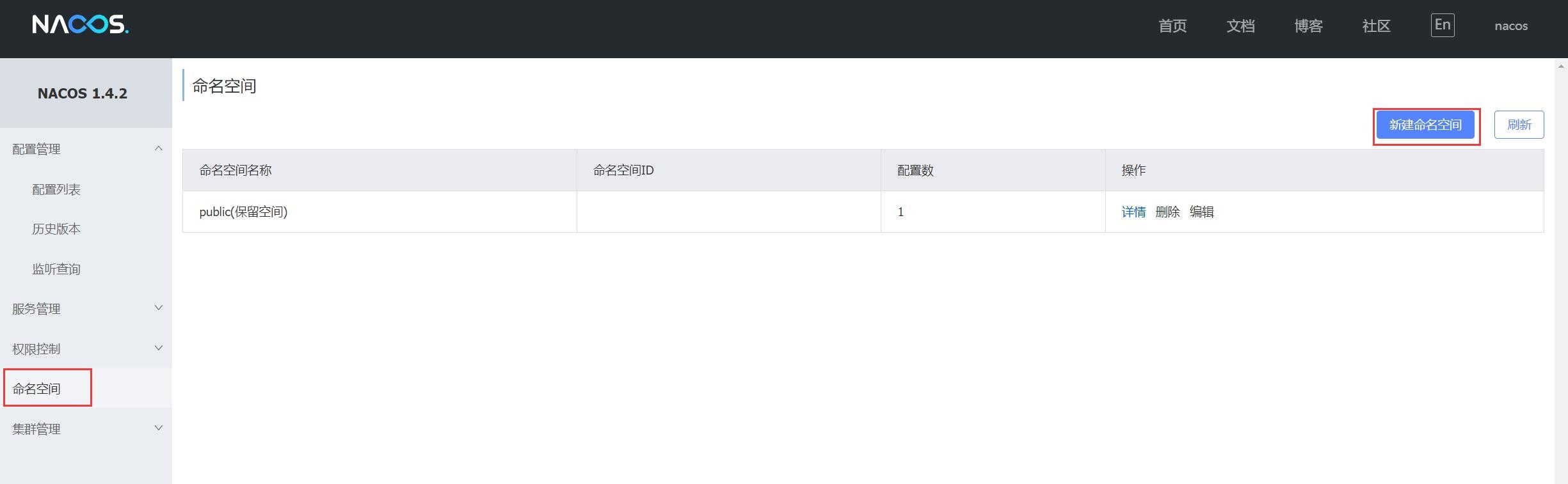


Nacos 配置中心的 namespace、dataId 和 group 可以方便灵活地划分配置。比如，我们现在有一个项目需要开发，项目名称为 bjpowernode，项目开发人员分为两个组：GROUP\_A 和

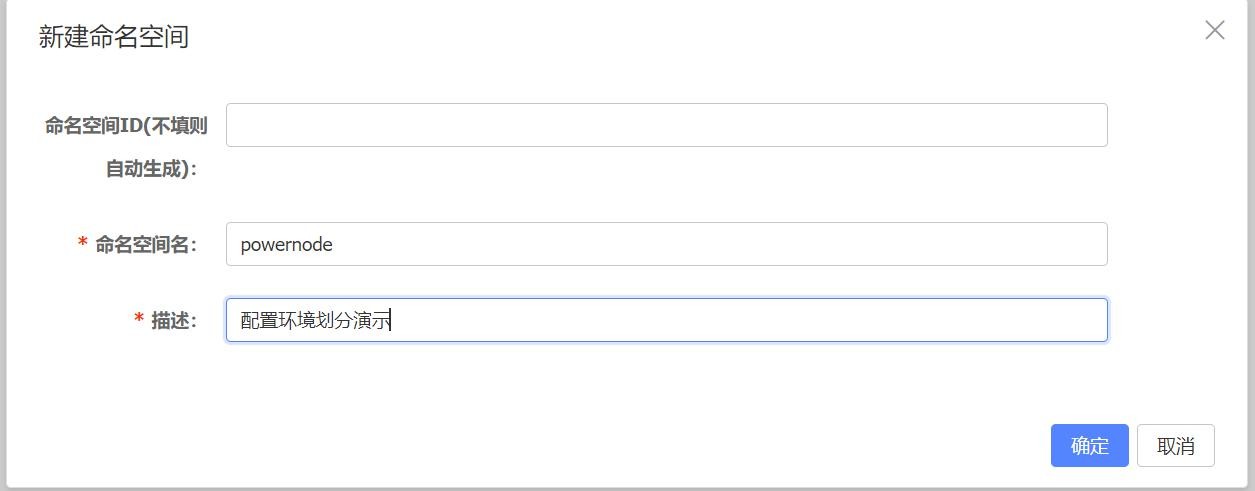
GROUP\_B，项目分为三个环境：开发环境 dev、测试环境 test 和生产环境 prod。 powernode->GRUOR\_A->dev

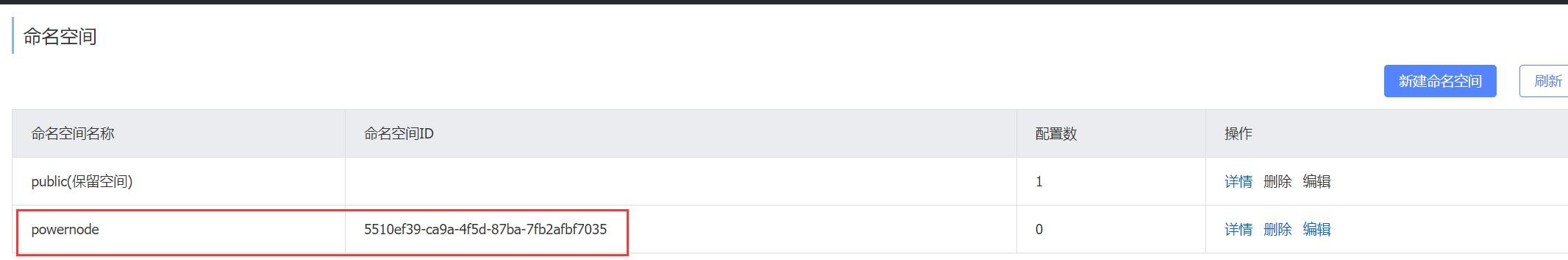
命名空间在服务列表和配置列表是共用的

* 1. 在 **Nacos** 中新建一个 **powernode** 的命名空间



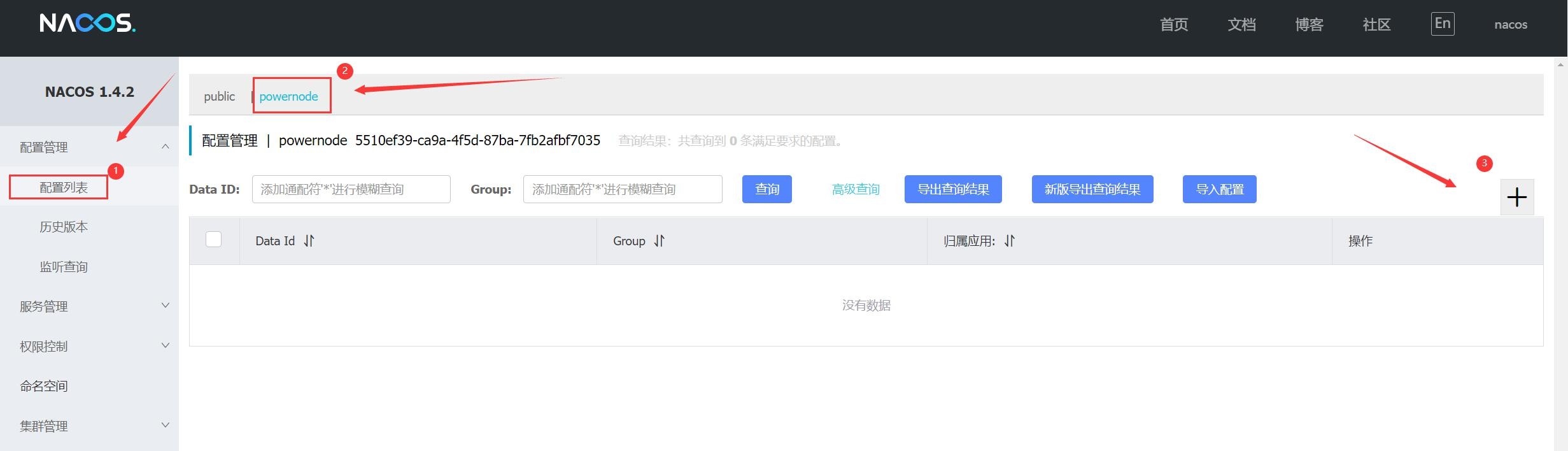
点击新添加一个命名空间



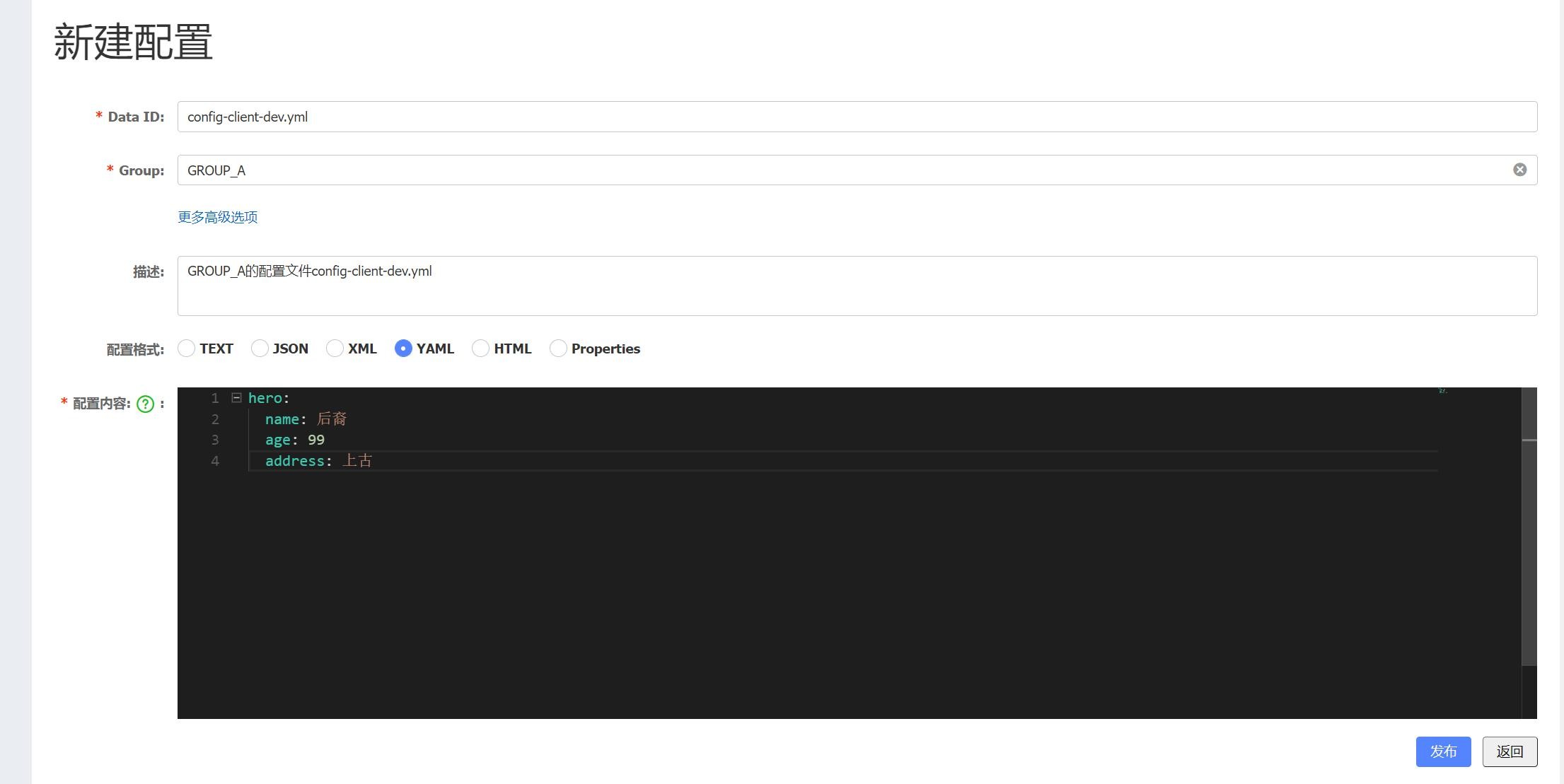
填写信息后确认

他给我们生成了一个 id，要记住这个 id：5510ef39-ca9a-4f5d-87ba-7fb2afbf7035

* 1. 在 **Nacos** 中新建一个配置文件 **config-client-dev.yml**

注意先选择 powernode 的命名空间，然后在选择添加配置文件

填写信息



点击完成

* 1. 修改 **config-client-a** 项目的配置文件

server:

port: 8080 spring:

application:

name: config-client-a cloud:

nacos:

config:

**server-addr: localhost:8848**

**namespace: 5510ef39-ca9a-4f5d-87ba-7fb2afbf7035 *#*** 命名空间注意使用 ***id***

**group: GROUP\_A *#*** 组别

**prefix: config-client *#*** 配置文件前缀，如果不写默认使用***${spring.application.name}***的值

**file-extension: yml *#*** 后缀文件格式

profiles:

active: dev *# spring* 的环境配置

* 1. 重启 **nacos-config-client** 项目测试

访问： http://localhost:8080/heroInfo

至此配置文件的信息已经获取成功

# 获取多配置文件

除了通过上面的方式指定一个唯一配置外，我们还可以同时获取多个配置文件的内容。提供这个功能 可以再次封装和抽象配置文件管理

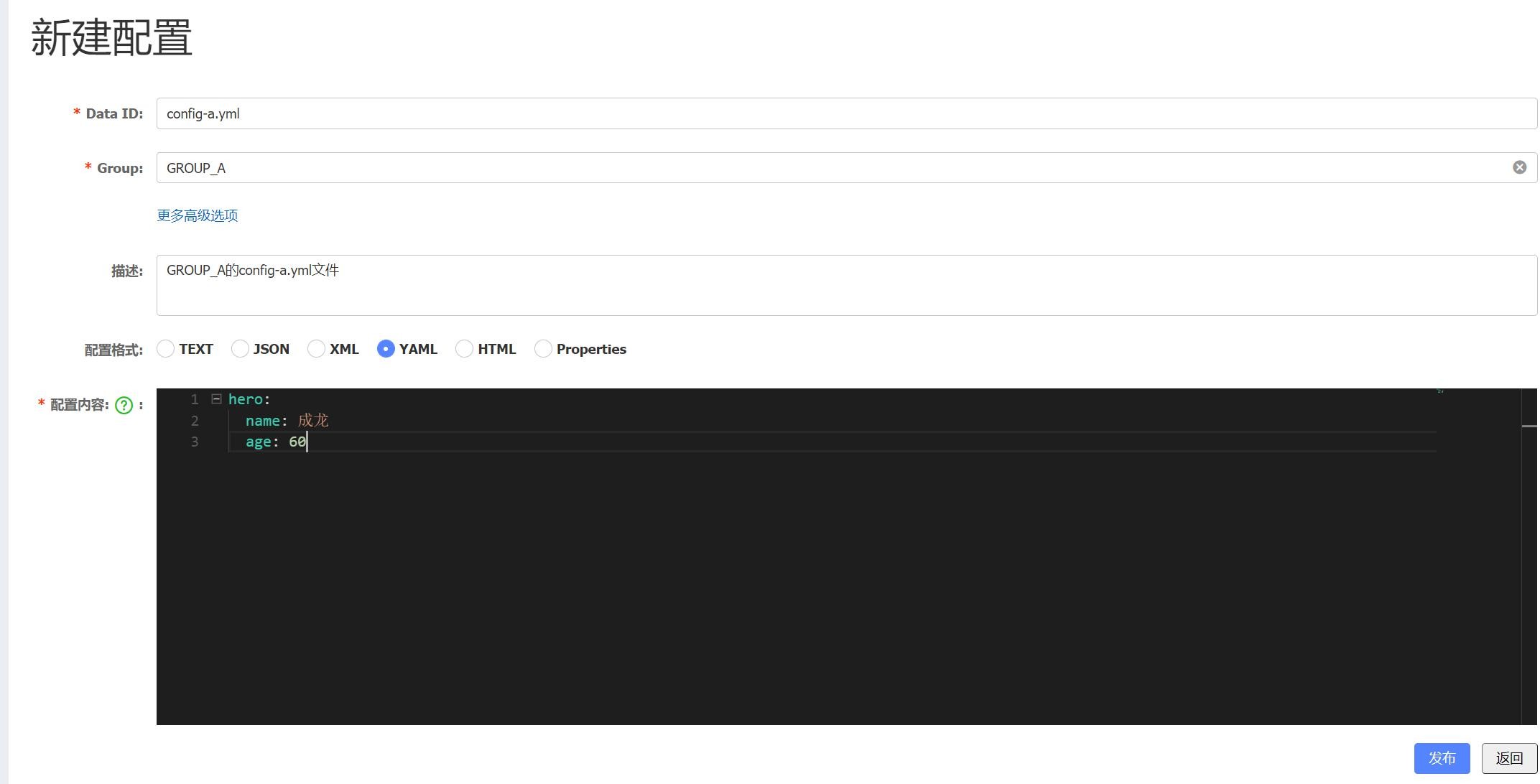
* 1. 在 **Nacos** 中新建两个配置文件

在 powernode 命名空间，继续点击添加两个配置文件

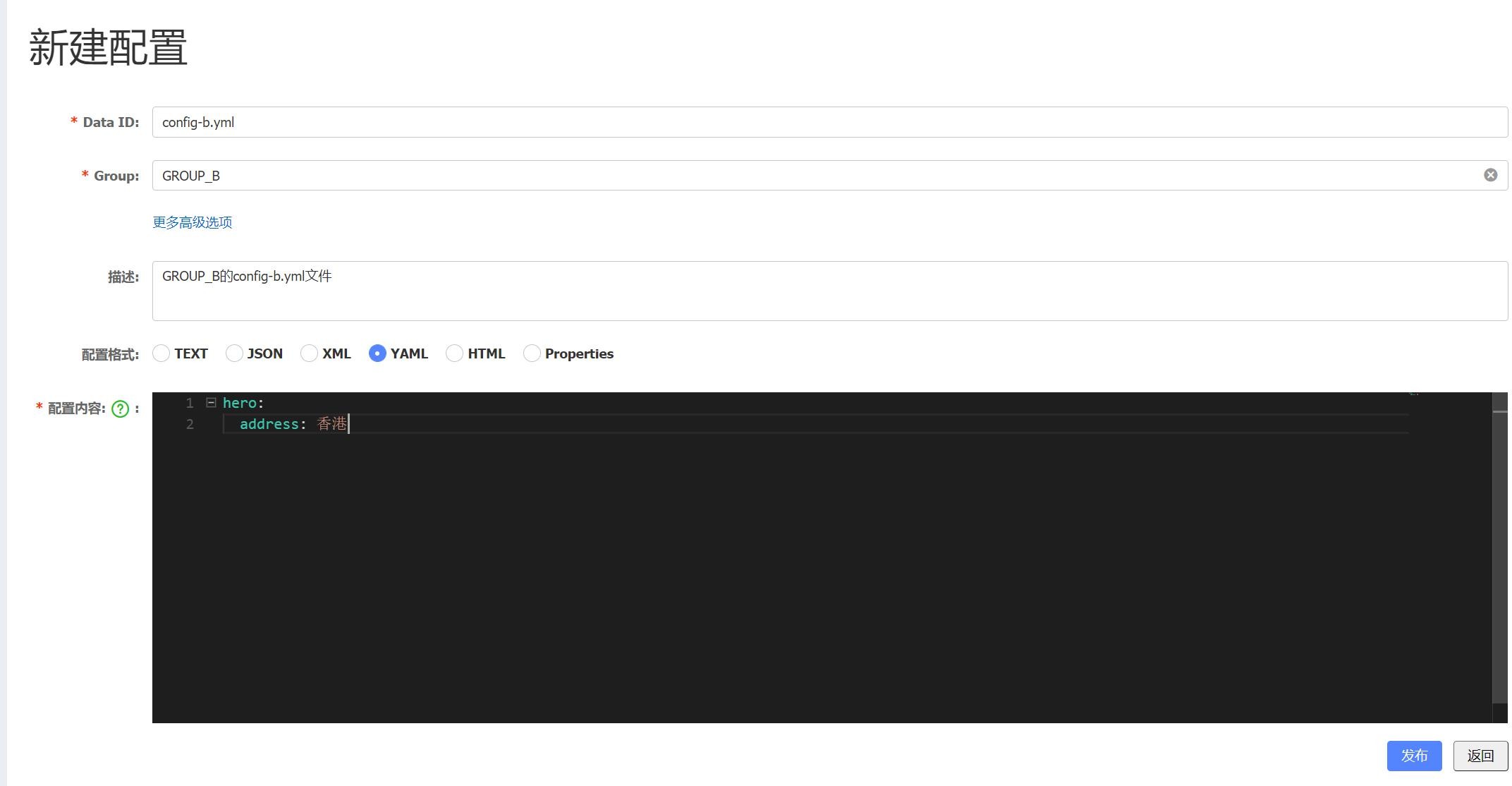


* 1. 添加一个配置文件 **config-a.yml**

注意这里只写了两个属性



* 1. 再添加一个配置文件 **config-b.yml**

注意这里只写了一个属性

* 1. 修改 **config-client-a** 项目的配置文件

server:

port: 8080 spring:

application:

name: config-client-a cloud:

nacos:

config:

server-addr: localhost:8848

namespace: 5510ef39-ca9a-4f5d-87ba-7fb2afbf7035 *#* 命名空间注意使用 *id*

extension-configs: *#* 配置多个配置文件数组形式

* data-id: config-a.yml *#* 配置*id*，必须要加文件后缀

group: GROUP\_A *#* 组别

refresh: true *#* 是否支持刷新

* data-id: config-b.yml group: GROUP\_B refresh: false

说明：

* spring.cloud.nacos.config.extension-configs[n].dataId ， 指定多个 配置的

dataId，必须包含文件格式，支持 properties、yaml 或 yml；

* spring.cloud.nacos.config.extension-configs[n].group，指定分组；
* spring.cloud.nacos.config.extension-configs[n].refresh，是否支持刷新。 上面的配置中，我们分别从 DEFAULT\_GROUP 中获取了 config-a.yml 和 config-b.yml 配置内容，并且 config-a.yml 支持刷新，config-b.yml 不支持刷新。

注意：

没有 **namespace** 的配置，言外之意就是 **Nacos** 目前还不支持多个配置指定不同的命名空间。

## 重启测试



1. **Spring Cloud Alibaba Nacos Config** 常用的配置

|  |  |  |  |
| --- | --- | --- | --- |
| 配置项 **key** 默认值 说明 | | | |
| 服务端地址 | spring.cloud.nacos.config.server-add r |  |  |
| DataId 前缀 | spring.cloud.nacos.config.prefix | spring.applicatio n.name |  |
| Group | spring.cloud.nacos.config.group | DEFAULT\_GROUP |  |
| dataID 后缀及内容文件格式 | spring.cloud.nacos.config.file-exten sion | properties | dataId 的后缀，同时也是配置内容的文件格式，目前只支持 properties |
| 配置内容的编码方式 | spring.cloud.nacos.config.encode | UTF-8 | 配置的编码 |
| 获取配置的超时时间 | spring.cloud.nacos.config.timeout | 3000 | 单位为 ms |
| 配置的命名空间 | spring.cloud.nacos.config.namespace |  | 常用场景之一是不同环境的 配置的区分隔离，例如开发测试环境和生产环境的资源隔  离等。 |
| AccessKey | spring.cloud.nacos.config.access-key |  |  |
| SecretKey | spring.cloud.nacos.config.secret-key |  |  |
| 相对路径 | spring.cloud.nacos.config.context-pa th |  | 服务端 API 的相对路径 |
| 接入点 | spring.cloud.nacos.config.endpoint |  | 地域的某个服务的入口域名，通过此域名可以动态地拿到  服务端地址 |
| 是否开启监听和自动刷新 | spring.cloud.nacos.config.refresh.en abled | true |  |

# 两端的配置文件写什么内容

本地的 bootstrap.yml(应用名称，配置文件中心（注册中心地址），读取的配置文件名称信息)

远端的配置文件(端口，数据源，redis，mq，mybatis，Swagger...)

为了方便去动态刷新和修改