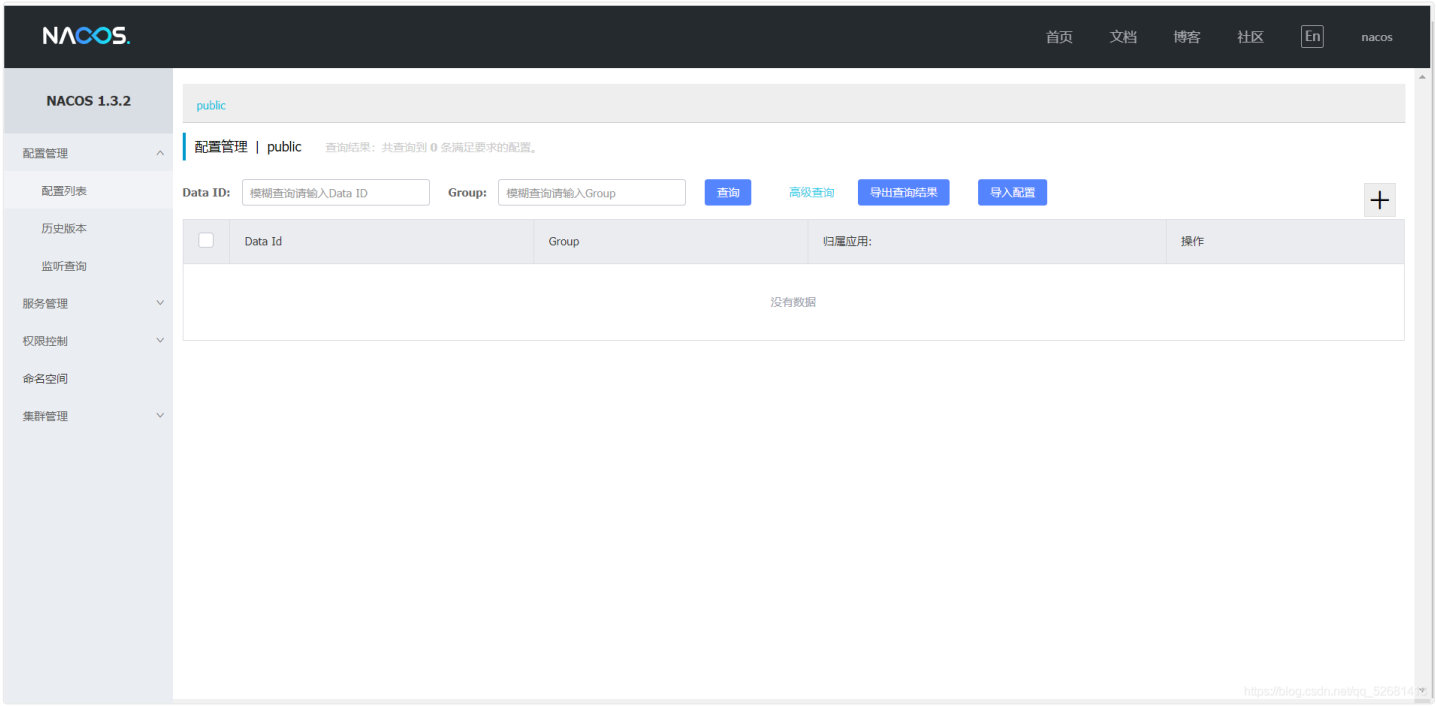
注册配置中心 | Nacos

Nacos 注册中心

Nacos是阿里巴巴的微服务组件，在SpringCloud项目中需要引入Alibaba依赖，内部集成了Ribbon。它既可以作为注册中心，也可以作为服务配置中心、可视化服务管理平台。



官网地址：https://nacos.io/zh-cn/docs/quick-start.html

搭建 Nacos 服务

Nacos官方提供了打包的Server，因此我们下载并启动即可，下载地址：https://github.com/alibaba/nacos/releases

单机启动：在nacos\bin目录下执行命令（1.4.1版本）。

```Bash
# windows下的启动，standalone代表着单机模式运行，非集群模式
startup.cmd -m standalone

#启动并挂后台，不加&时关闭控制台窗口后，服务关闭
startup.cmd -m standalone &

#windows下的停止
shutdown.cmd
```

启动后，直接访问管理界面：http://127.0.0.1:8848/nacos/，账号名/密码为 nacos/nacos。

服务注册、发现

使用HTTP请求的方式模拟服务注册于发现

- 注册接口（POST）：http://127.0.0.1:8848/nacos/v1/ns/instance

```JSON
{
  "serviceName":"服务名",
  "ip":"服务IP地址",
  "port":"服务端口号"
}
```

- 发现接口（GET）：http://127.0.0.1:8848/nacos/v1/ns/instance/list

```Bash
serviceName=服务名
```

项目中服务注册与发现

引入Nacos客户端依赖（内部集成了Ribbon）：

```XML
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```

配置Nacos server信息：

```YAML
# 应用服务 WEB 访问端口
server:
  port:8080

spring:
  application:
    name: nacos-discovery-provider-sample  # 应用名称
  cloud:
    nacos:                          # Nacos帮助文档: https://nacos.io/zh-cn/docs/concepts.html
      discovery:                    # Nacos 服务发现与注册配置 (nacos.discovery=false时不作为注册中心)
        username: nacos             # Nacos控制台登录信息
        password: nacos
        server-addr: 127.0.0.1:8848  # Nacos 服务器主机和端口
        namespace: public           # 注册到 nacos 的指定 namespace，默认为 public
      #config:
        #    enabled: false          # 不使用Nacos进行配置管理，默认true。
        #默认使用nacos管理配置，这里先考虑仅作为注册中心的情况
```

声明服务为Nacos客户端服务：配置好后启动服务，刷新控制台页面，服务注册了。

```Java
@EnableDiscoveryClient
@Configuration
public class NacosDiscoveryConfiguration {

}
```

服务高可用、Nacos 集群

Nacos服务搭建只需要下载并启动Nacos Server即可，在集群中也是如此，步骤如下：

- 准备多个Nacos server，并将端口区分开(不同主机就不用了)

- 所有 nacos server 都添加 nacos/conf/cluster.conf 文件：模板文件 cluster.conf.example

```
# 所有nacos server的 IP:端口，包括自身
127.0.0.1:9848
127.0.0.1:9858
127.0.0.1:9868
```

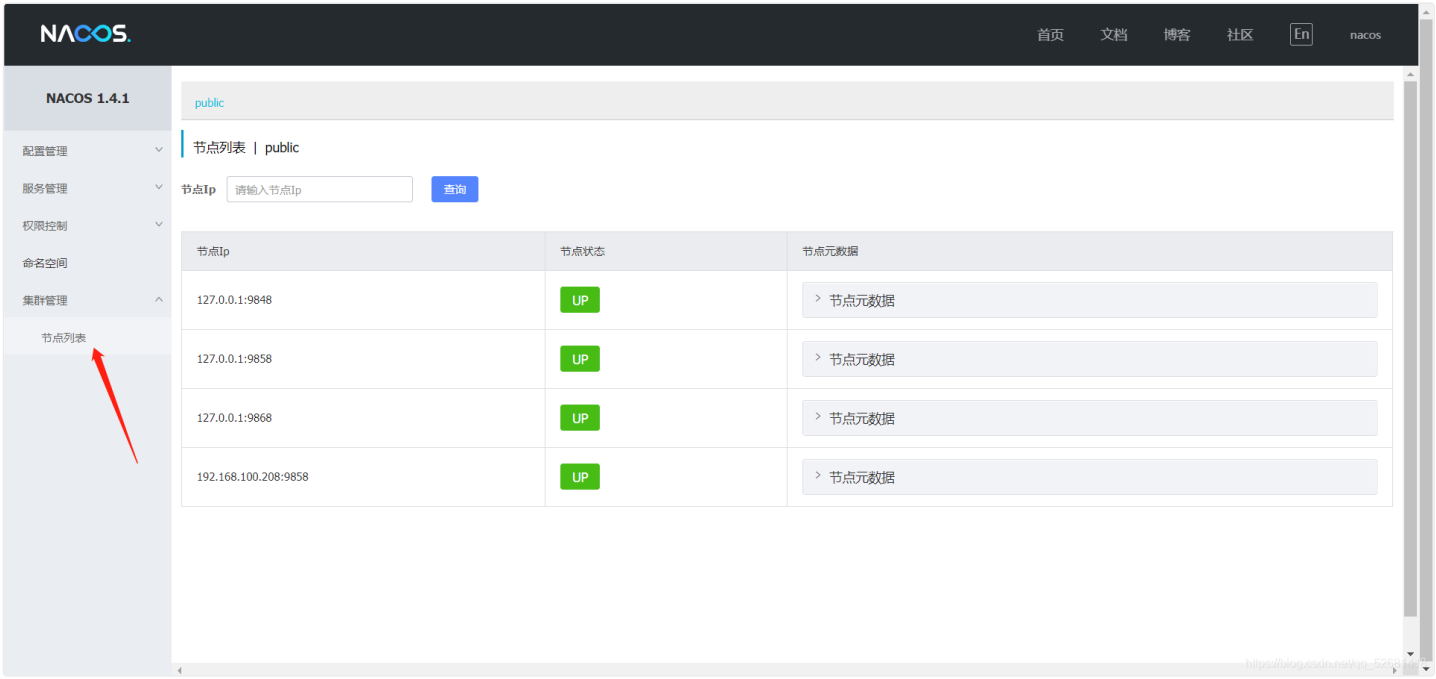
- 分别进入每个 server 的 bin 目录，使用集群模式启动 nacos

```
# 集群模式启动
startup.cmd -m cluster
```

注意：

1. cluster.conf 中不能使用 localhost 代替本机，否则报 tomcat 错误。
2. 集群必须使用 mysql 作为持久化。
3. 同一台主机，单机启动和集群启动不能共存，否则报 tomcat 错误。
4. 集群模式下，不可以手动模拟服务注册。

配置好之后，可以发现每个 nacos server 都能启动了，点击集群管理>节点列表就能看到其他 nacos server 了。



服务剔除机制、自我保护机制

服务每 5s 向注册中心发送心跳，证明自己还“活着”。

数据持久化到 MySQL

nacos 除了作为注册中心外，还能作为配置中心进行配置管理，在单机启动模式下，默认使用内嵌数据库，无需任何配置就能启动，不过有一定局限性。

将数据持久化到 mysql（版本必须 5.6.5+）：

- 将 nacos\conf\nacos-mysql.sql 文件执行到数据库中。
- 修改 nacos\conf\application.properties 配置文件，内容如下：

```
spring.datasource.platform=mysql  # 使用的数据库
db.num=1                          # 数据库数量

### 连接
db.url.0=jdbc:mysql://127.0.0.1:3306/nacos?characterEncoding=utf8&connectTimeout=1000&socketTimeout=3000&autoReconnect=true&useUnicode=true&
db.user.0=root
db.password.0=root
```

nacos支持多数据源，0表示第一个。配置完成后重启即可。

Nacos 配置中心

Nacos除了作为服务注册中心的同时，也可对配置服务的配置文件进行管理。不过我们可以通过如下方式排除其作为配置中心的功能：

```properties
spring.cloud.nacos.config.enabled = false
```

通过请求方式发布与获取配置：

- 发布配置：post，http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=nacos.cfg.dataId&group=test&content=HelloWorld
- 获取配置：get，http://127.0.0.1:8848/nacos/v1/cs/configs?dataId=nacos.cfg.dataId&group=test

搭建 Nacos 服务

请看Nacos注册中心，建议使用MySQL数据库作为持久化。

客户端配置

引入nacos config依赖：

```xml
<dependency>
  <groupId>com.alibaba.cloud</groupId>
  <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
```

创建一个bootstrap.yml(或properties)，并配置nacos config的配置：

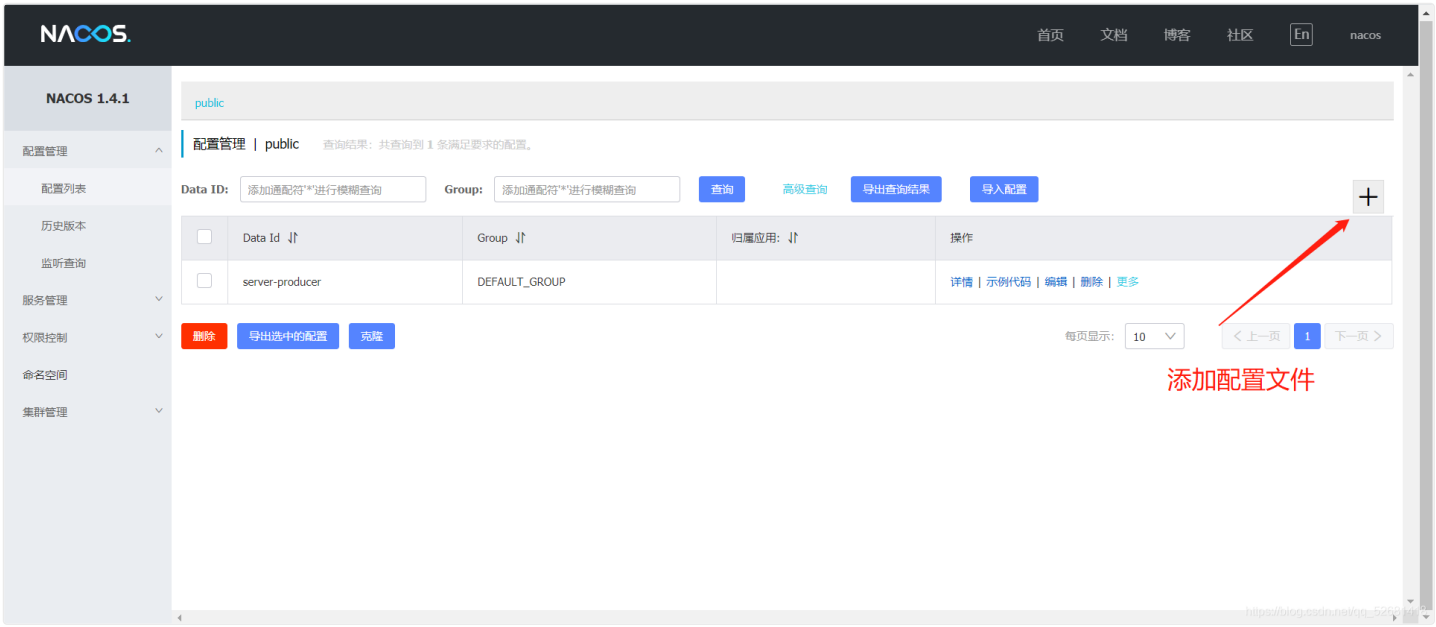
```yaml
spring:
 cloud:
   nacos:
     config:                        #nacos的信息，添加后可以省略注册中心的nacos此类信息
       server-addr: 127.0.0.1:8848
       username: nacos
       password: nacos
       contextPath: /nacos
       #--------------------------
       file-extension: yaml         #显示的声明dataId文件扩展名
```

application.yml(或properties)，除特殊配置外，只留下下面2个配置：

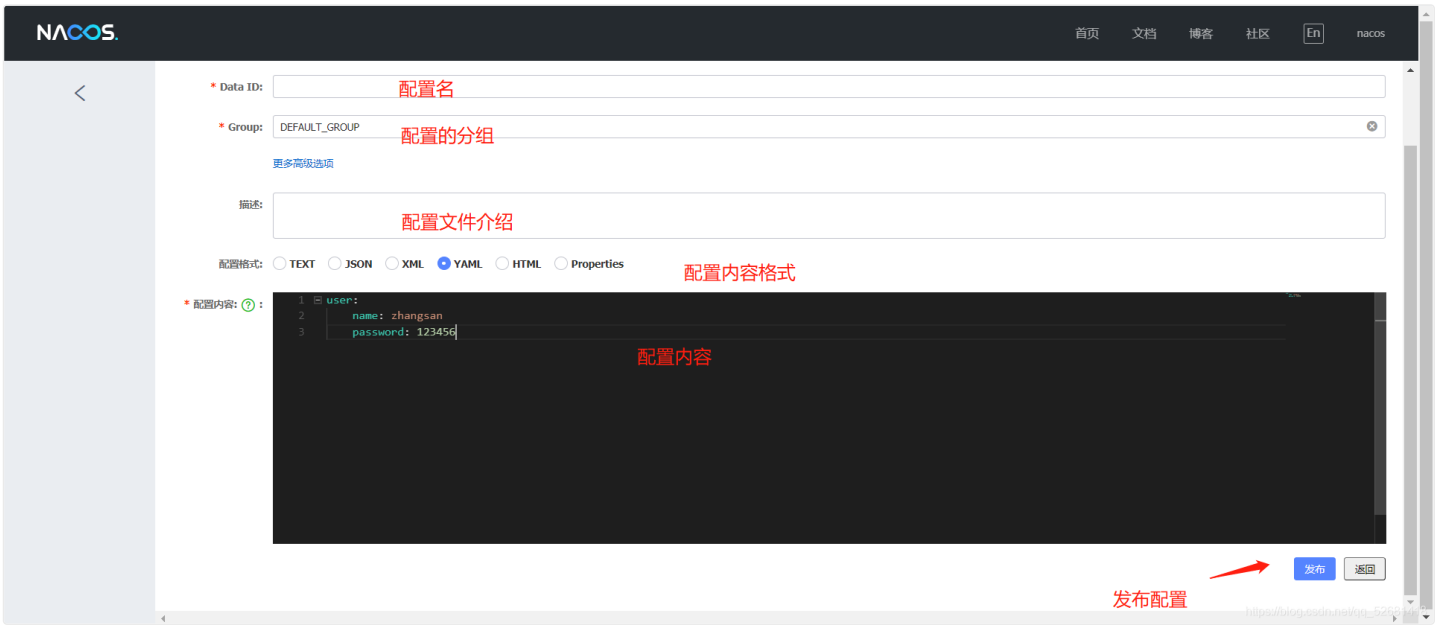
```yaml
server:
 port: 41150
spring:
 application:
   name: server-producer
```

- bootsteap.yml：这里只配置配置中心的信息，服务通过它去获取服务所需配置，而且这里的内容基本不变。
- application.yml：这里只配置端口+服务名，因为首次启动服务时，需要通过它注册达到nacos，然后才能获取配置。

在nacos配置中心添加配置文件：



配置名要和服务名一致，后缀可忽略：



仅第一次发布配置后，需要重启服务。

服务与配置文件的匹配部分机制：

- 文件名=服务名，建议显示指定配置文件后缀。
- 同名配置会互相覆盖，比如server、server.yml在指定yml格式后，两个文件会覆盖。

动态刷新

配置发生变化后，服务中绑定的属性值将刷新，会刷新的情况有3种：

- 类上加注解：@RefreshScope、类种非静态属性加@Value注解

```Java
@RestController
@RefreshScope
public class NacosConfigDemo {

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

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

    @PostConstruct
    public void init() {
        System.out.printf("[init] user name : %s , age : %d%n", userName, userAge);
    }
```

```
    @RequestMapping("/user")
    public String user() {
        return String.format("[HTTP] user name : %s , age : %d", userName, userAge);
    }

}
```

- 类上加注解：@RefreshScope、@ConfigurationProperties

Java

```
@RefreshScope
@ConfigurationProperties(prefix = "user")
public class User {

    private String name;

    private int age;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public String toString() {
        return "User{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }
}
```

- 使用 Nacos Config 监听来对 Bean 属性实施监听。

Java

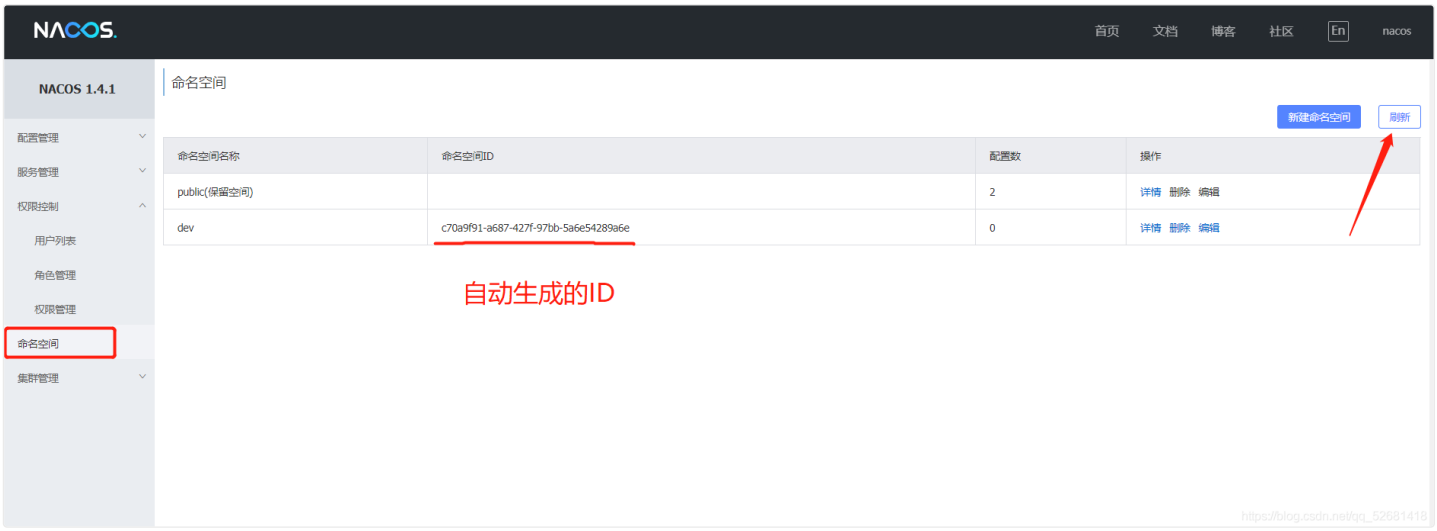
```
@RestController
@RefreshScope
@EnableConfigurationProperties(User.class)
public class NacosConfigDemo {

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

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

    @PostConstruct
    public void init() {
        System.out.printf("[init] user name : %s , age : %d%n", userName, userAge);
    }

    @PreDestroy
    public void destroy() {
        System.out.printf("[destroy] user name : %s , age : %d%n", userName, userAge);
    }

    @RequestMapping("/user")
    public String user() {
        return String.format("[HTTP] user name : %s , age : %d", userName, userAge);
    }

}
```

高级配置

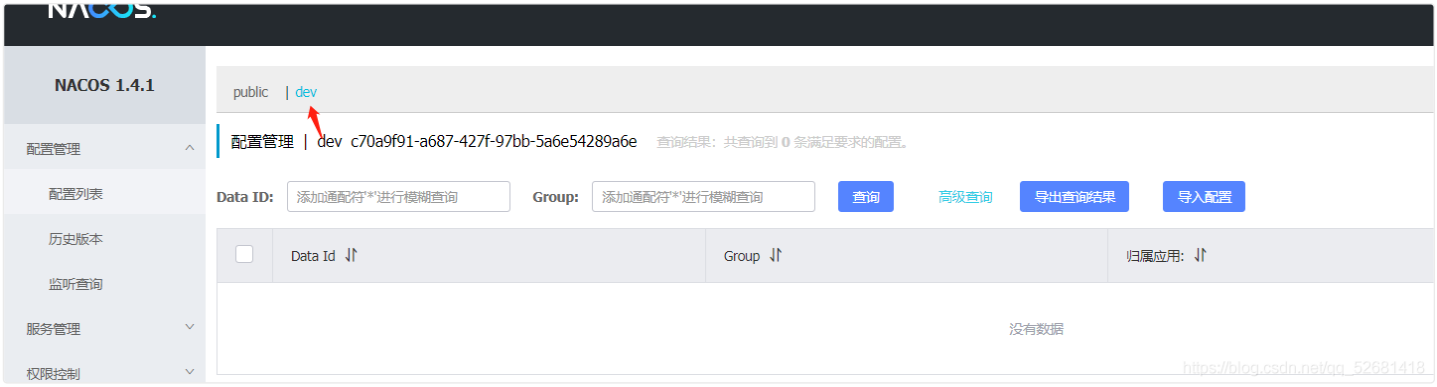
namespace 命名空间

nacos 可以实现配置文件之间隔离，不同的命名空间下，可以存在相同的 Group 或 Data ID 的配置，可以实现不同开发环境（开发、测试、生产等）配置文件的隔离。

创建命名空间：



切换命名空间：



创建后，在客户端 bootstrap.yml 添加如下配置进行绑定：

```yaml
spring:
  cloud:
    nacos:
      config:
        namespace: c70a9f91-a687-427f-97bb-5a6e54289a6e
```

Group 分组：在创建配置时可以指定分组，自定义分组在 bootstrap.yml 添加如下：

```yaml
spring:
  cloud:
    nacos:
      config:
        group: DEVELOP_GROUP
```

自定义配置名：三种情况完整配置案例如下

```yaml
spring:
  application:
    name: myserver
  cloud:
    nacos:
      config:
        server-addr: 127.0.0.1:8848
        extension-configs[0]:   # 1.在默认组,不支持动态刷新
          data-id: myconfig-1.yaml
        extension-configs[1]:  # 2.不在默认组，不支持动态刷新
          data-id: myconfig-2.yaml
          group: MY_GROUP
```

```
        extension-configs[2]:  # 3.不在默认的组，支持动态刷新
          data-id: myconfig-3.yaml
                  MY_GROUP
```