# 构建环境

## 环境要求

JDK: 1.8

Maven: 3.6.3

MySql: 8.0

Redis: 4.0

Nacos: 1.3.0

Sentinel: 1.7.2

Seata: 1.2.0

Zipkin-server：V2.21.4

## 开发工具

### IntelliJ IDEA

### Idea插件

Lombok(必装)，Maven Helper，Free mybatis

## 环境搭建

1）：maven <http://maven.apache.org/download.cgi>

2）：redis

Win版：<https://github.com/ServiceStack/redis-windows/tree/master/downloads>

Linux：<https://redis.io/download>

3）：nacos

下载：<https://github.com/alibaba/nacos/releases>

文档：https://nacos.io/zh-cn/docs/quick-start.html

4）：sentinel

下载：<https://github.com/alibaba/Sentinel/releases>

文档：https://github.com/alibaba/Sentinel/wiki/%E4%BB%8B%E7%BB%8D

5）：seata

下载：<http://seata.io/zh-cn/blog/download.html>

文档：http://seata.io/zh-cn/docs/user/quickstart.html

6）：zipkin-server

下载：<https://github.com/openzipkin/zipkin/releases>

文档：<https://github.com/openzipkin/zipkin/tree/master/zipkin-server>

# 项目搭建

## 获取项目

<https://github.com/phh2008/start-spring-cloud>

## 项目结构

Start-spring-cloud

├── cloud-core -- 框架模块

├ ├── core-authc – 认证与授权校验配置

├ ├── core-boot -- springboot通用配置

├ ├── core-elasticsearch -- ES封装

├ ├── core-common -- 各种baseEntity,mapper,service,result,annotation

├ └── core-tool -- 工具类

├── cloud-gateway -- 网关

├── cloud-devops -- 运维中心(各种 admin 控制台)

├ ├── code-generator -- 代码生成器

├── cloud-module -- 业务模块

├ ├── cloud-auth -- 认证服务(对外提供登录认证API)

├ ├── cloud-sys -- 系统服务(比如用户、菜单、角色、权限、字典等基础服务)

├ ├── cloud-demo -- 示例服务

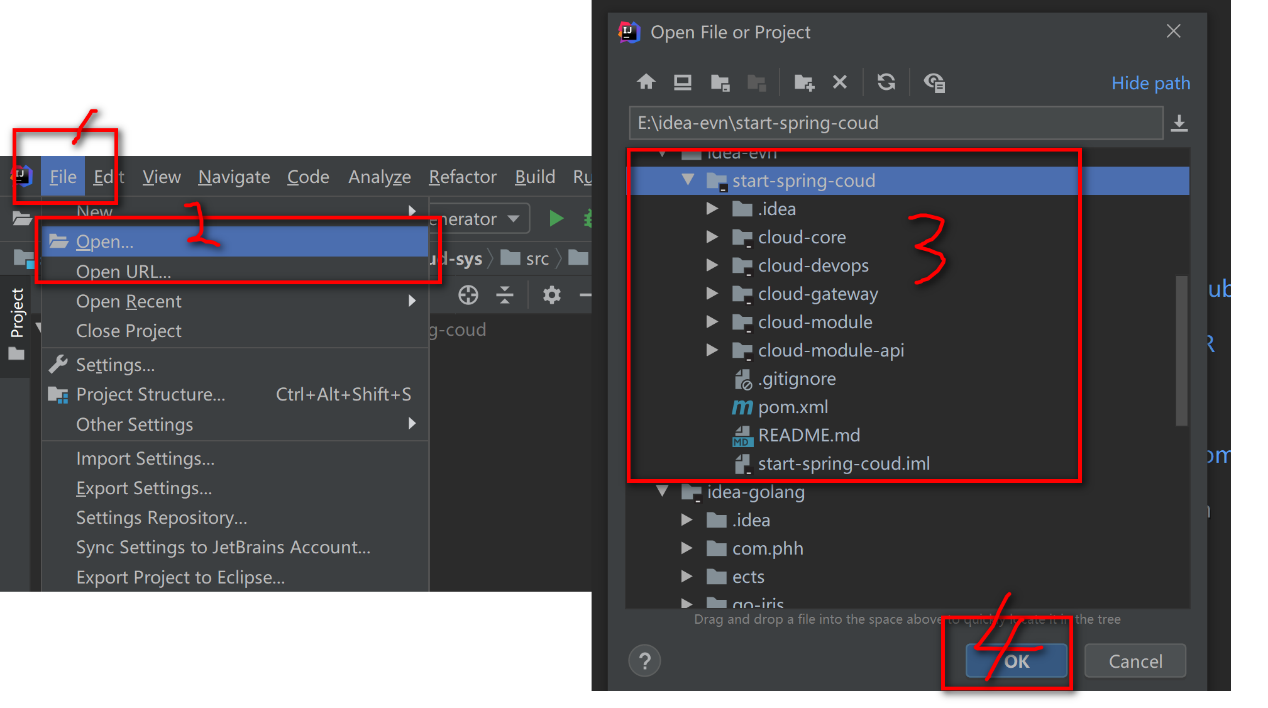
├── cloud-module-api -- 业务模块api(vo、dto、feign等)

└── └── cloud-sys-api -- 系统api

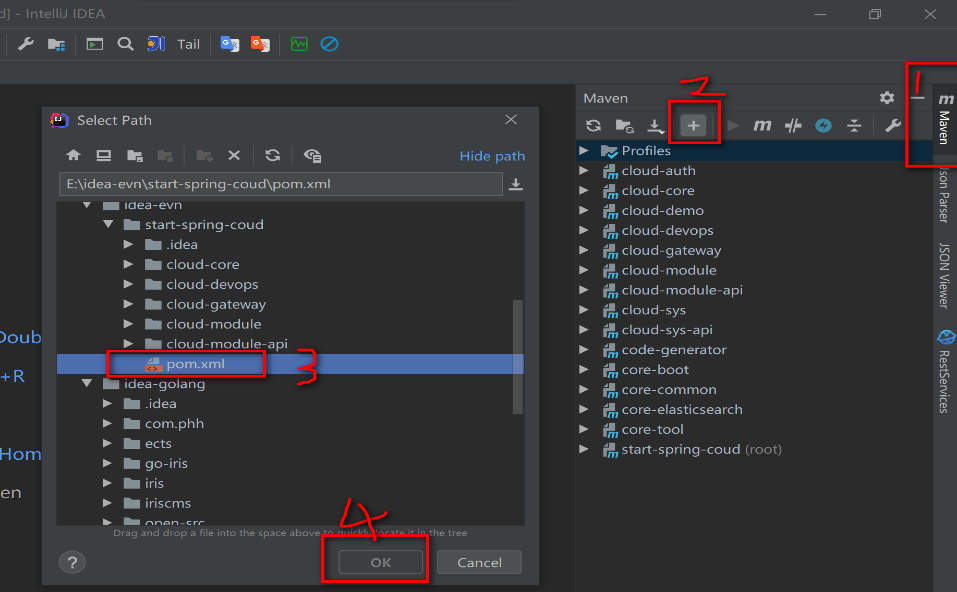
## 导入项目

使用idea可以有多种导入已有项目方式

导入一：



导入二：



以上两种方式都可以把新的项目导入到idea中

## 运行项目

Idea集成了spring boot 面板，导入项目后可看到springboot项目的启动入口类，可以直接运行这些服务

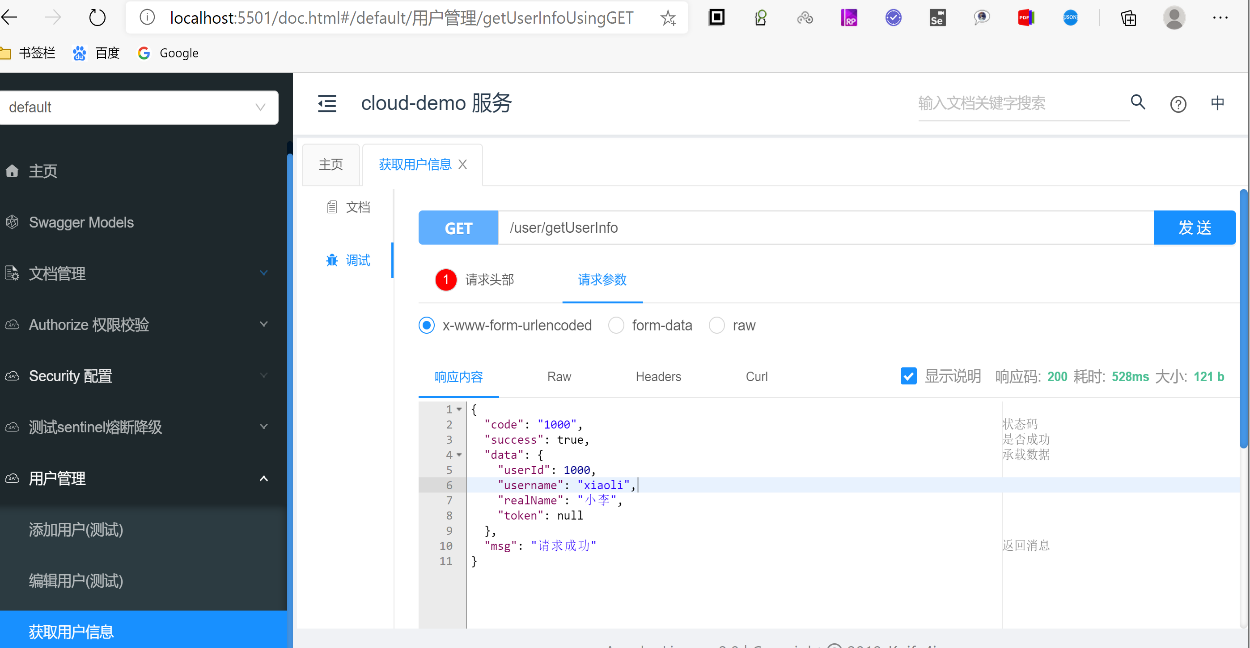
如果没有这个面板，可以展开项目找到相应的main方法类，启动项目即可。

启动这些项目前提是：必需启动好nacos，mysql，并修改好相应的链接配置

## 测试项目

项目启动后，可请求/doc.html查看swagger文档

比如：<http://localhost:5501/doc.html>



不仅可以查看到所有API信息，还可以直接在界面上调试我们的接口

# 项目配置

## bootstrap.yml配置

bootstrap.yml程序引导时加载的配置，要早于application.yml配置，示例中我们的配置如下：

**spring**:

**application**:

**name**: cloud-demo

**profiles**:

**active**: @profileActive@

**main**:

**allow-bean-definition-overriding**: **true**

**cloud**:

**nacos**:

**discovery**:

**server-addr**: 127.0.0.1:8848

**config**:

**server-addr**: 127.0.0.1:8848

**group**: CN

**file-extension**: yaml

**namespace**: start-cloud

参数说明：

spring.application.name每个微服务的名称都应是独有的

spring.profiles.active 结合maven实现编译打包多环境参数配置

spring.cloud.nacos.discovery配置nacos注册中心

spring.cloud.nacos.config配置nacos配置中心

## application.yml配置

application.yml 应用程序特有配置，后续用来配置各微服务模块中使用的公共配置

双击查看以下配置示例：



启用nacos配置中心后，这些配置以后就可配置到nacos中去了，后续会说明怎么用nacos配置application.yml，当然application.yml与nacos可以同时存在使用

# 开发新服务

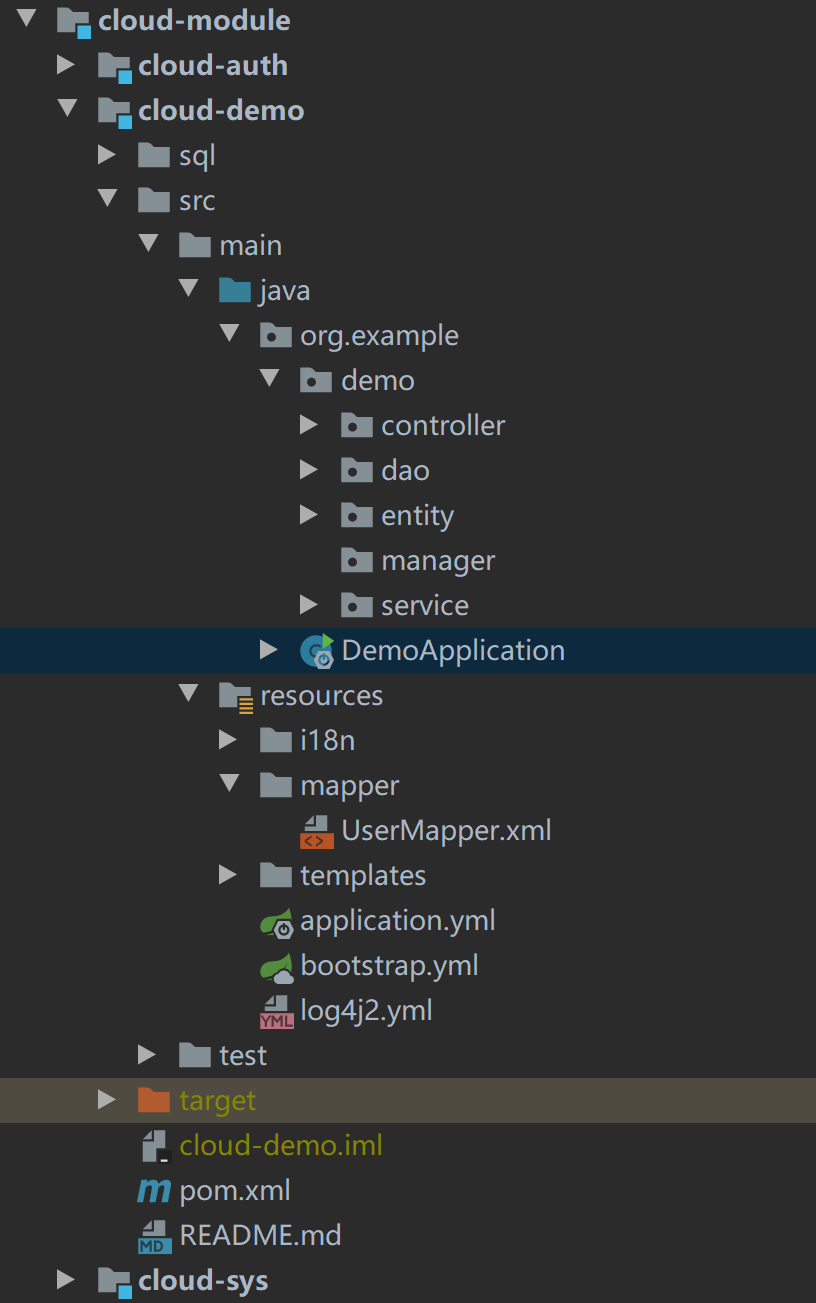
## 新建微服务工程

我们需要构建一个新的业务模块服务项目，做为一个独立的微服务项目，参照cloud-demo项目，按照我们的项目结构规范在cloud-module项目下新建一个maven子项目，pom.xml和springboot的配置都可从cloud-demo中复制出来，修改下服务名称、端口

等配置项

## 编写API

### 标准的项目内部结构



### 启动入口

@Slf4j

@EnableFeignClients

@EnableDiscoveryClient

@EnableScheduling

@SpringBootApplication

public class DemoApplication **{**

public static void main**(**String**[]** args**)** **{**

SpringApplication**.**run**(**DemoApplication**.**class**,** args**);**

log**.**info**(**"Demo服务启动成功！"**);**

**}**

**}**

### Controller控制器

@Api**(**value **=** "用户管理"**,** tags **=** "用户管理"**)**

@RestController

@RequestMapping**(**"/demo/user"**)**

public class UserController **extends** BaseController **{**

@Autowired

private IUserService userService**;**

@WithoutAuthentication

@ApiOperation**(**value **=** "用户例表"**,** httpMethod **=** "GET"**,** response **=** Result**.**class**)**

@GetMapping**(**"/list"**)**

public Result**<**List**<**User**>>** list**()** **{**

List**<**User**>** list **=** userService**.**list**();**

**return** Result**.**ok**(**list**);**

**}**

**}**

## 认证注解使用

@WithoutAuthentication 上例中使用到此注解，表示不需要登录校验，此注解可用在方法上，也可用在类上面对这类所有请求都会起作用。此注解是基于spring拦截器实现，具体实现方法在cloud-core/core-authc项目中的配置， UserAuthInterceptor.java、SecurityConfiguration.java

不仅可以通过@ WithoutAuthentication注解控制，还可以在配置文件中设置要排除认证校验的url，如下示例用法：

**start:**

**auth**:

# 认证拦截器要排除的 url配置

**excludePatterns**:

- /test/\*\*

- /static/\*\*

- /public/\*\*

- /userPubKey

- /swagger-ui.html

## 权限注解使用

@Authorize用来增加方法级别的权限控制，基于AOP来实现权限校验，具体实现类在cloud-core/core-authc项目中的aspect包AuthorizeAspect.java类。支持的表达式有以下几种，也支持spring EL表达式方便扩展。

permitAll() 可通过，只要用户登录

denyAll() 不可通过，除非是超级管理员 admin

hasRole('xx') 是否有此角色

hasAnyRole('xx', 'cc') 是否有其中任一角色

hasPermit('xx') 是否有此权限

hasAnyPermit('xx') 是否有其中任一权限

@Log**(**"测试接口:用户add"**)**

**@Authorize("hasPermit('sys:user:add')")**

@GetMapping**(**"/add"**)**

public Result add**()** **{**

//do something

**return** Result**.**ok**();**

**}**

## 微服务调用

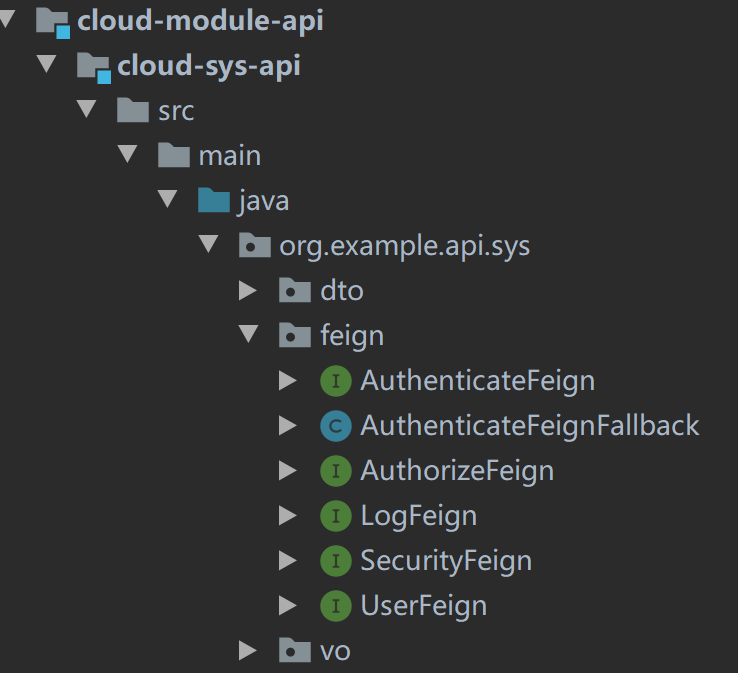
微服务之间使用OpenFeign来实现服务调用，OpenFeign的声明式方式定义Web服务客户端，其次还更进一步，通过集成Ribbon实现负载均衡的HTTP客户端。OpenFeign自身提供的标注（annotation）之外，还可以使用JAX-RS标注，或者Spring MVC标注。

启用OpenFeign只需引入spring-cloud-starter-openfeign依赖，在启动类上添加@EnableFeignClients即可。

服务调用实现示例：

1）：客户端定义服务接口

通常服务接口可能会被其它各服务调用，所以我们一般就定义在一个公共的API项目中，如下图实现：



@FeignClient**(**name **=** "cloud-sys"**)**

public interface UserFeign **{**

/\*\*

\* 获取用户信息

\*

\* **@return**

\*/

@GetMapping**(**"/user/getUserInfo"**)**

Result**<**UserInfoVO**>** getUserInfo**();**

**}**

2）：使用OpenFeign声明的服务接口

@Autowired

private UserFeign userFeign**;**

@GetMapping**(**"/getInfo"**)**

public Result**<**UserInfoVO**>** getInfo**()** **{**

**return** userFeign**.**getUserInfo**();**

**}**

# 组件使用

## 系统日志

### 日志文件

项目整合log4j2用来记录日志，每个项目的resources目录下都有一个log4j2.yml配置文件，可配置日志格式、文件目录、日志级别控制等

双击打开此文件



### 日志切面

项目cloud-core/core-boot中有配置LogAspect.java切面，配合@Log注解在相应的controller方法上使用此注解即可记录请求的日志，具体日志持久化需要项目中实现，框架暂时只打印相关日志信息。

## 代码生成

1）：代码生成项目 cloud-core/code-generator

2）：配置修改：可以修改类org.example.generator.CodeGenerator数据库驱动、连接信息、author、包名、entity、mapper、service、controller父类、模版等

3）：启动test/org.example.generator.CodeGenerator#generator，默认大部分参数可以控制台可输入

注意事项：使用mysql5与mysql8时记得要更改pom.xml中jdbc驱动对应的版本

## 文档生成

1）：配置

案例中整合了springfox-swagger2，相关配置类cloud-core/core-boot/org.example.core.boot.config.SwaggerConfiguration.java

在各微服务中配置以下内容即可:

**swagger**:

**enable**: **true #启用开关**

**basePackages: org.example #扫描的包，多个用豆号分隔**

**api**:

**title**: 'cloud-demo 服务' #标题设置

**version**: 'V1.0.0' #版本号设置

2）：注解使用，在代码中使用swagger提供的注解

**@Api(tags = "用户管理")**

@RestController

@RequestMapping**(**"/demo/user"**)**

public class UserController **extends** BaseController **{**

@Autowired

private IUserService userService**;**

**@ApiOperation(value = "用户例表", httpMethod = "GET", response = Result.class)**

@GetMapping**(**"/list"**)**

public Result**<**List**<**User**>>** list**()** **{**

List**<**User**>** list **=** userService**.**list**();**

**return** Result**.**ok**(**list**);**

**}**

**}**

更多注解介绍：



## Nacos注册中心、配置中心

### 官网文档

<https://nacos.io/zh-cn/docs/what-is-nacos.html>

### 快速启动nacos（单机）

1. 下载安装包

<https://github.com/alibaba/nacos/releases/download/1.3.0/nacos-server-1.3.0.zip>

1. 配置数据源（持久化）

默认数据保存到内存，可以设置为外部mysql存储nacos的数据，需要配置nacos\conf\application.properties

然后执行sql脚本生成需要用到的数据表

1. 启动服务

sh startup.sh

### 用作配置中心

**①：dataId、group、namespace说明**

Nacos 通过dataId、group、namespace来隔离分组配置，也就是这三个组合唯一

根据官方说明：nacos spring cloud中的dataId的配置格式如下：

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

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.profile.active 即为当前环境对应的 profile。 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-exetension 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。目前只支持 properties 和 yaml 类型。

示例：**cloud-demo-dev.yaml**

**②：配置bootstrap.yml**

示例：

**spring**:

**application**:

**name**: cloud-demo

**profiles**:

**active**: @profileActive@

**cloud**:

**nacos**:

**discovery**: #注册中心

**server-addr**: 127.0.0.1:8848

**config**: #配置中心

**server-addr**: 127.0.0.1:8848

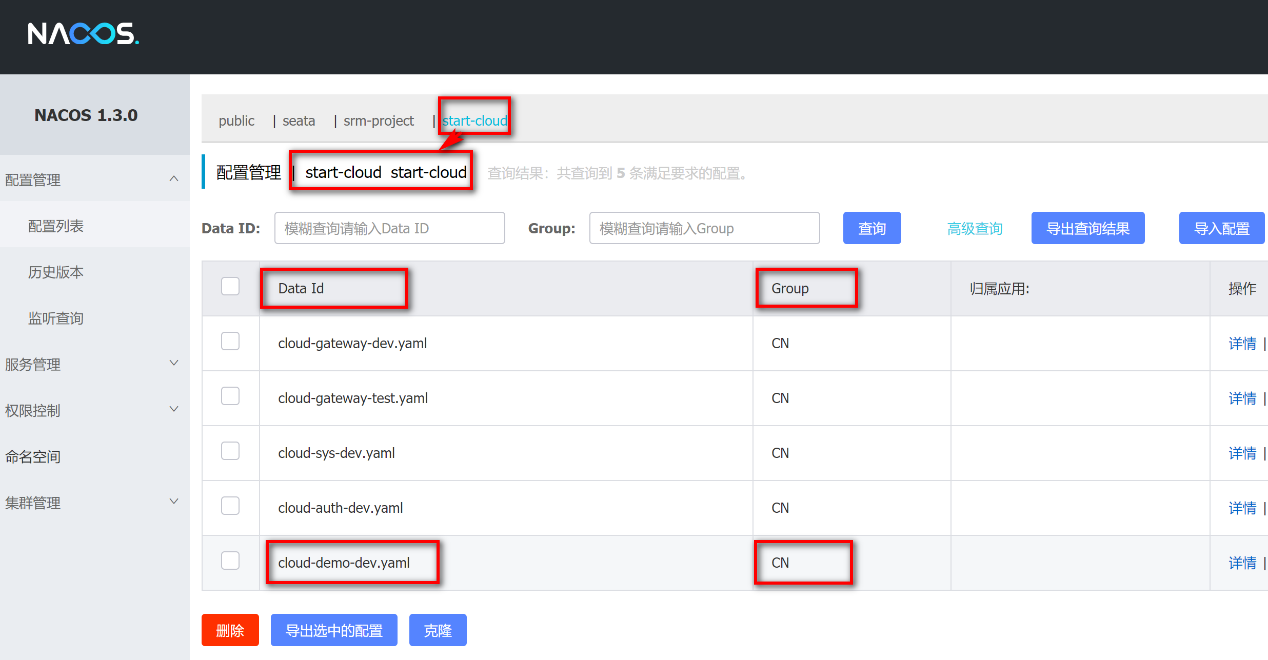
**group**: CN

**file-extension**: yaml

**namespace**: start-cloud

注意：如果不是默认namespace，group，就一定要配置这两值，否则会找不到配置

Nacos 控制台配置示例截图：

上图配置示例说明：

namespace：start-cloud

group：CN

data Id：cloud-demo-dev.yaml

在bootstrap.yml中配置应用名称为cloud-demo，profile为dev，所以dataId的格式为cloud-demo-dev.yaml

### 用作服务注册中心

案例配置：

**spring**:

**application**:

**name**: cloud-demo

**profiles**:

**active**: @profileActive@

**cloud**:

**nacos**:

**discovery: #注册中心**

**server-addr: 127.0.0.1:8848**

**config**: #配置中心

**server-addr**: 127.0.0.1:8848

**group**: CN

**file-extension**: yaml

**namespace**: start-cloud

服务提供者与服务消费者配置的nacos地址都是一样的

## Gateway网关配置

参考blog：<https://www.jianshu.com/p/6ff196940b67>

**Route 路由**：gateway的基本构建模块。它由ID、目标URI、断言集合和过滤器集合组成。如果聚合断言结果为真，则匹配到该路由。

**Predicate 断言**：这是一个Java 8 Function Predicate。输入类型是 Spring Framework ServerWebExchange。这允许开发人员可以匹配来自HTTP请求的任何内容，例如Header或参数。

**Filter 过滤器**：这些是使用特定工厂构建的 Spring Framework GatewayFilter实例。所以可以在返回请求之前或之后修改请求和响应的内容。

本项目示例：

**spring**:

**cloud**:

**gateway**:

**routes**:

**- id**: cloud-sys

**uri**: lb://cloud-sys

**order**: 1000

**predicates**:

- Path=/api/cloud/sys/\*\*

**filters**:

- StripPrefix=3

**- id**: cloud-auth

**uri**: lb://cloud-auth

**order**: 1000

**predicates**:

- Path=/api/cloud/auth/\*\*

**filters**:

- StripPrefix=3

## Sentinel熔断限流

### 介绍

主页：

<https://github.com/alibaba/Sentinel/wiki/%E7%86%94%E6%96%AD%E9%99%8D%E7%BA%A7>

对调用链路中不稳定的资源进行熔断降级也是保障高可用的重要措施。由于调用关系的复杂性，如果**调用链路中的某个资源不稳定，最终会导致请求发生堆积**。Sentinel 熔断降级会在调用链路中某个资源出现不稳定状态时（例如调用超时或异常比例升高），对这个资源的调用进行限制，让请求快速失败，避免影响到其它的资源而导致级联错误。当资源被降级后，在接下来的降级时间窗口之内，对该资源的调用都自动熔断（默认行为是抛出 DegradeException）

### Maven 坐标

在项目中添加以下maven坐标

<dependency>

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

<artifactId>**spring-cloud-starter-alibaba-sentinel**</artifactId>

</dependency>

### Openfeign 支持sentinel

首先需要添加openfeign相关的依赖

<dependency>

<groupId>**org.springframework.cloud**</groupId>

<artifactId>**spring-cloud-starter-openfeign**</artifactId>

</dependency>

然后在application.yml配置中开启sentinel

**feign**:

**sentinel**:

**enabled**: **true**

@FeignClient简单示例：

@FeignClient**(**name **=** "service-provider"**,** fallback **=** EchoServiceFallback**.**class**)**

public interface EchoService **{**

@RequestMapping**(**value **=** "/echo/{str}"**,** method **=** RequestMethod**.**GET**)**

String echo**(**@PathVariable**(**"str"**)** String str**);**

**}**

class EchoServiceFallback **implements** EchoService **{**

@Override

public String echo**(**@PathVariable**(**"str"**)** String str**)** **{**

**return** "echo fallback"**;**

**}**

**}**

### sentinel控制台

下载控制台jar或源码自行打包 <https://github.com/alibaba/Sentinel/releases>

项目application.yml配置接入控制台：

**spring**:

**cloud**:

**sentinel**:

**transport**:

**port**: 8719

**dashboard**: localhost:8080

启动控制台：

java **-**Dserver.port**=**8080 **-**Dcsp.sentinel.api.port**=**8719 **-**Dcsp.sentinel.dashboard.server**=**localhost**:**8080 **-**Dproject.name**=**sentinel-dashboard **-**jar sentinel-dashboard-1.7.2.jar

启动完成后，即可进入控制台操作流控等操作，详细操查看官网wiki

### sentinel流控信息持久化到nacos

sentinel默认的配置信息是保存到内存的，所有应用重启后之前配置的流控信息就会清空，所以需要持久化，sentine支持多种持久化到方式，这里介绍持久化到nacos。如果在控制台直接配置流控后需要持久化到nacos，还需要修改sentinel-dashboard源码结合推送模式即可在控制修改流控保存到nacos，并推送到客户端。需要做以下操作：

1）：修改sentinel-dashboard源码，本项目中已提供修改后的源码

2）：在微服务中添加sentinel-datasource-nacos 依赖:

<dependency>

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

<artifactId>**sentinel-datasource-nacos**</artifactId>

</dependency>

3）：在微服务application.yml中配置nacos数据源，如下示例：

**spring**:

**cloud**:

# sentinel 控制台地址配置

**sentinel**:

**enabled**: **true**

**transport**:

**port**: 8719

**dashboard**: localhost:8080

**datasource**:

# 名称随意

**flow**:

**nacos**:

**server-addr**: localhost:8848

**dataId**: ${spring.application.name}-flow-rules

**groupId**: SENTINEL\_GROUP

# 规则类型，取值见：

# org.springframework.cloud.alibaba.sentinel.datasource.RuleType

**rule-type**: flow

**degrade**:

**nacos**:

**server-addr**: localhost:8848

**dataId**: ${spring.application.name}-degrade-rules

**groupId**: SENTINEL\_GROUP

**rule-type**: degrade

**system**:

**nacos**:

**server-addr**: localhost:8848

**dataId**: ${spring.application.name}-system-rules

**groupId**: SENTINEL\_GROUP

**rule-type**: system

**authority**:

**nacos**:

**server-addr**: localhost:8848

**dataId**: ${spring.application.name}-authority-rules

**groupId**: SENTINEL\_GROUP

**rule-type**: authority

**param-flow**:

**nacos**:

**server-addr**: localhost:8848

**dataId**: ${spring.application.name}-param-rules

**groupId**: SENTINEL\_GROUP

**rule-type**: param-flow

经测试，修改后的源码对流控规则、降级规则、系统规则这种规则在控制台修改后可以持久化到nacos并推送到客户端，微服务重启后可也以从nacos加载规则。**热点规则与授权规则测试可持久到nacos，但是客户端获取的规则为空。后续处理。**

## 负载均衡配置

## Seata 分布式事务

## 调用链路追踪

# Maven 多环境打包