

- 一、Spring Cloud Alibaba简介
 - 1.1 什么是Spring Cloud Alibaba?
 - 1.2 主要模块
 - 1.3 模块对应组件
 - 1、阿里开源组件
 - 2、阿里商业化组件
 - 3、集成 Spring Cloud 组件
 - 1.4 与Spring版本对应关系
- 二、注册中心+配置中心Nacos
 - 2.1 什么是Nacos?
 - Nacos版本介绍
 - 2.2 Nacos的安装启动
 - 1. 环境要求
 - 2. 下载地址
 - 3. 启动Nacos服务
 - 4. 访问Nacos服务
 - 2.3 注册中心案例
 - 1、目标：
 - 2、实现步骤：
 - 第一步：搭建提供者和消费者服务
 - 第二步：服务注册到Nacos注册中心
 - 3、实现过程：
 - 第一步：搭建提供者和消费者服务
 - 第二步：服务注册到Nacos注册中心
 - 查看服务详情
 - 2.4 配置中心案例
 - 1、目标：
 - 2、实现步骤：
 - 3、实现过程：
 - 扩展1：配置自动刷新
 - 扩展2：多环境配置使用
 - 1. 方式一：文件名称命名
 - 2. 方式二：Group分组

- 三、流量防卫兵Sentinel
 - 3.1 Sentinel简介
 - 1、主要特征
 - 3、Sentinel 分为两个部分:
 - 4、那些公司在用?
 - 3.2 Sentinel控制台安装
 - 1、下载jar
 - 2、启动:
 - 3、访问测试:
 - 3.3 服务集成Sentinel
 - 3.4 演示：服务降级和限流

一、Spring Cloud Alibaba简介

1.1 什么是Spring Cloud Alibaba?

Spring Cloud Alibaba 是阿里提供的微服务开发一站式解决方案，是阿里巴巴开源中间件与 Spring Cloud 体系的融合。马老师左手双十一，右手阿里开源组件，不仅占据了程序员的购物车，还要攻占大家的开发工具。

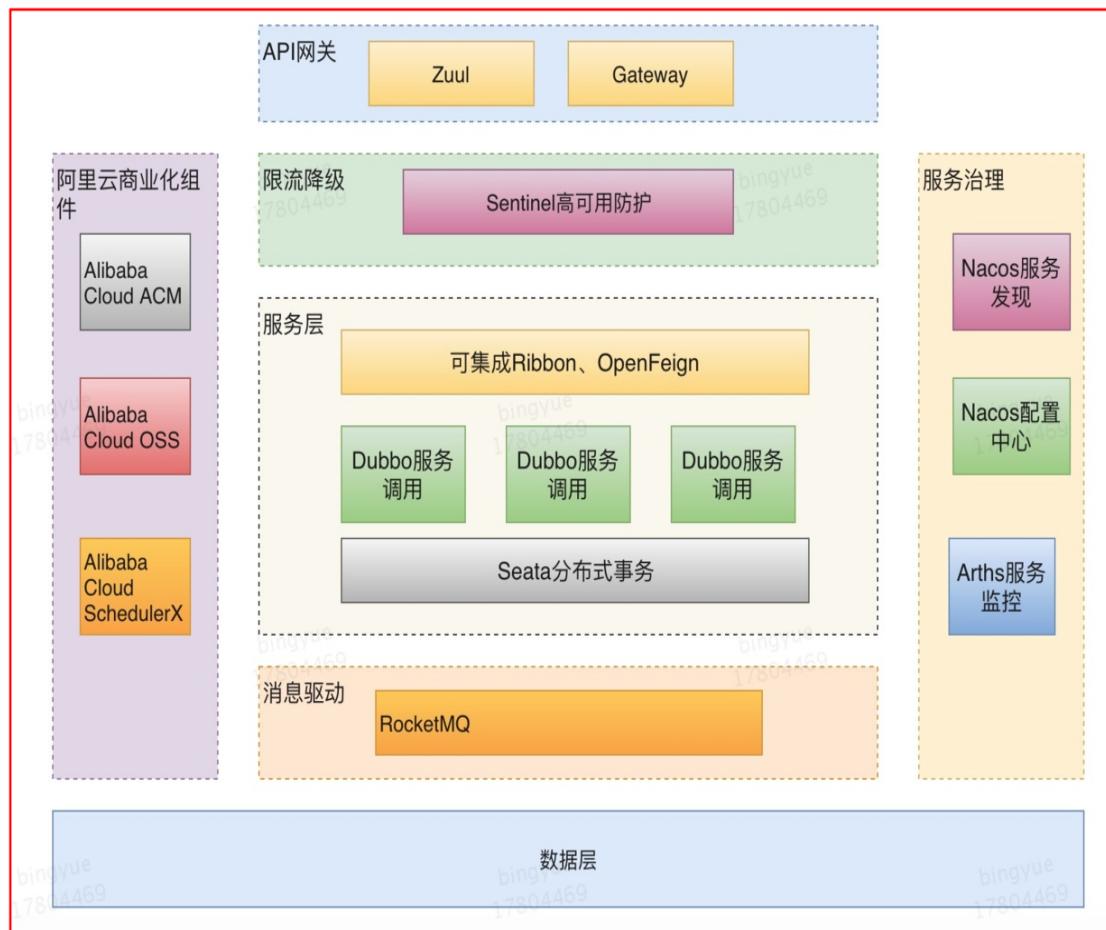
同 Spring Cloud 一样，Spring Cloud Alibaba 也是一套微服务解决方案，包含开发分布式应用微服务的必需组件，方便开发者通过 Spring Cloud 编程模型轻松使用这些组件来开发分布式系统。

依托 Spring Cloud Alibaba，需要添加一些注解和少量配置，就可以将 Spring Cloud 应用接入阿里分布式应用解决方案，通过阿里中间件来迅速搭建分布式应用系统。

官方文档地址：<https://github.com/alibaba/spring-cloud-alibaba/blob/master/README-zh.md>

1.2 主要模块

- **服务注册和发现**: 实例可以在Nacos中注册，客户可以使用Spring管理的bean发现实例。支持通过Spring Cloud Netflix的客户端负载均衡器Ribbon。
- **流量控制和服务降级**: 使用Sentinel进行流量控制，断路和系统自适应保护。
- **分布式配置中心**: 使用Nacos作为数据存储
- **消息总线**: 使用Spring Cloud Bus RocketMQ链接分布式系统的节点
- **Dubbo RPC**: 通过Dubbo RPC扩展Spring Cloud服务到服务调用的通信协议
- **分布式事务**: 支持高性能且易于使用的Seata分布式事务解决方案
- **阿里云对象存储**: OSS的Spring资源抽象。阿里云对象存储服务（OSS）是一种加密，安全，经济高效且易于使用的对象存储服务，可让您在云中存储，备份和存档大量数据
- **阿里云短信服务**: 覆盖全球的短信服务，友好、高效、智能的互联化通讯能力，帮助企业迅速搭建客户触达通道。



1.3 模块对应组件

1、阿里开源组件

- Nacos：一个更易于构建云原生应用的动态服务发现、配置管理和服务管理平台。
- Sentinel：把流量作为切入点，从流量控制、熔断降级、系统负载保护等多个维度保护服务的稳定性。
- RocketMQ：开源的分布式消息系统，基于高可用分布式集群技术，提供低延时的、高可靠的消息发布与订阅服务。
- Dubbo：这个就不用多说了，在国内应用非常广泛的一款高性能 Java RPC 框架。
- Seata：阿里巴巴开源产品，一个易于使用的高性能微服务分布式事务解决方案。
- Arthas：开源的Java动态追踪工具，基于字节码增强技术，功能非常强大。

2、阿里商业化组件

作为一家商业公司，阿里巴巴推出 Spring Cloud Alibaba，很大程度上希望通过抢占开发者生态，来帮助推广自家的云产品。所以在开源社区，夹带了不少私货，这部分组件整体易用性和稳定性还是很高的。

- Alibaba Cloud ACM：一款在分布式架构环境中对应用配置进行集中管理和推送的应用配置中心产品。
- Alibaba Cloud OSS：阿里云对象存储服务（Object Storage Service，简称 OSS），是阿里云提供的云存储服务。
- Alibaba Cloud SchedulerX：阿里中间件团队开发的一款分布式任务调度产品，提供秒级、精准的定时（基于 Cron 表达式）任务调度服务。
- Alibaba Cloud SMS：覆盖全球的短信服务，友好、高效、智能的互联化通讯能力，帮助企业迅速搭建客户触达通道。

3、集成 Spring Cloud 组件

Spring Cloud Alibaba 作为整套的微服务解决组件，只依靠目前阿里的开源组件是不够的，更多的是集成当前的社区组件，所以 Spring Cloud Alibaba 可以集成 Zuul，OpenFeign 等网关，也支持 Spring Cloud Stream 消息组件。

1.4 与 Spring 版本对应关系

Spring Cloud Version	Spring Cloud Alibaba Version	Spring Boot Version
Spring Cloud Greenwich	2.1.x.RELEASE	2.1.x.RELEASE
Spring Cloud Finchley	2.0.x.RELEASE	2.0.x.RELEASE
Spring Cloud Edgware	1.5.x.RELEASE	1.5.x.RELEASE

我们统一使用 2.1版本的与前面的springcloud对应

二、注册中心+配置中心Nacos

2.1 什么是Nacos?

Nacos诞生于2019年~



一句话概括：它既是 注册中心 也是 配置中心，也就是类似Eureka+config的微服务管理平台！

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。

使用 Nacos 简化服务发现、配置管理、服务治理及管理的解决方案，让微服务的发现、管理、共享、组合更加容易。

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

Nacos版本介绍

目前最新版本为：nacos-server-1.1.4（最新稳定版，2019年10月24日发布）

1.2.0-beta.1	... on 17 Jan	fd5f0fe	zip	tar.gz	Notes	Downloads	Verified
1.2.0-beta.0	... on 17 Jan	fd5f0fe	zip	tar.gz	Notes	Downloads	Verified
1.1.4	... on 24 Oct 2019	4d68565	zip	tar.gz	Notes	Downloads	Verified
1.1.3	... on 6 Aug 2019	6ca2ee0	zip	tar.gz	Notes	Downloads	Verified
1.1.0	... on 6 Jul 2019	a15fc9f	zip	tar.gz	Notes	Downloads	Verified
1.0.1	... on 12 Jun 2019	0f72a34	zip	tar.gz	Notes	Downloads	Verified
1.0.0	... on 10 Apr 2019	97ddd39	zip	tar.gz	Notes	Downloads	Verified

2.2 Nacos的安装启动

1. 环境要求

Nacos 依赖 Java 环境来运行。请确保是在以下版本环境中安装使用：

1. 支持 Linux/Unix/Mac/Windows, 64 位操作系统
2. 支持JDK 1.8及以上，64 位版本；
3. Maven必须是3.2.x及以上版本；

2. 下载地址

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

[1.1.4
Oct 24th, 2019](#)

 nkorange released this on 24 Oct 2019 · 171 commits to develop since this release

[Compare](#)

#182 Health check field and protection threshold are confused
#1409 Integrate with Istio as a service discovery backend
#1507 It is recommended that the shutdown.sh script close only the nodes in the current directory, not all nodes.
#1665 DataSync always not executed after network partition
#1671 'Client-Version' header not set in http request from nacos client
#1759 The number of new configuration words is too long. Tip 400 is not friendly enough.
#1825 In modifying service instance page.Error information not clear
#1874 Repeat defined class NacoException
#1906 Set a new thread pool here, but do not open the Notifier task, whether to add executor.execute(new Notifier());

[▼ Assets 4](#)

 nacos-server-1.1.4.tar.gz	Linux发行包	49.7 MB
 nacos-server-1.1.4.zip	Windows发行包	49.7 MB
 Source code (zip)		
 Source code (tar.gz)		

发行包见资料，已经下载好了。

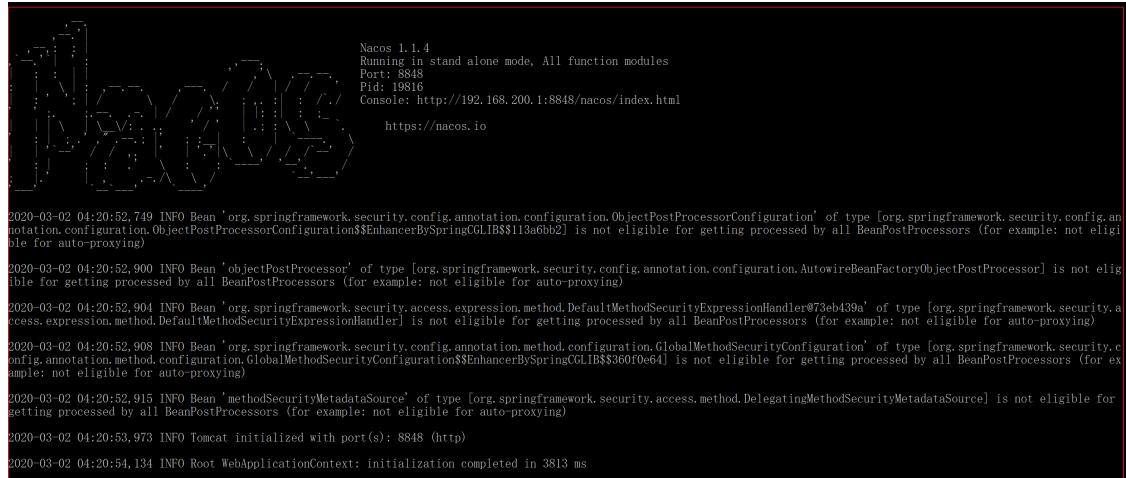
apollo-build-scripts-master.zip	2020/2/19 15:36	压缩(zipped)文件夹	171,132 KB
nacos-server-1.1.4.zip	2020/2/19 15:36	压缩(zipped)文件夹	50,898 KB
sentinel-dashboard-1.6.3.jar	2020/2/19 15:36	Executable Jar File	20,633 KB

3. 启动Nacos服务

启动命令：或者双击startup.cmd运行文件，（如果cmd窗口闪一下就没有了，请检查JAVA_HOME）。

```
startup.cmd
```

启动成功如下图：



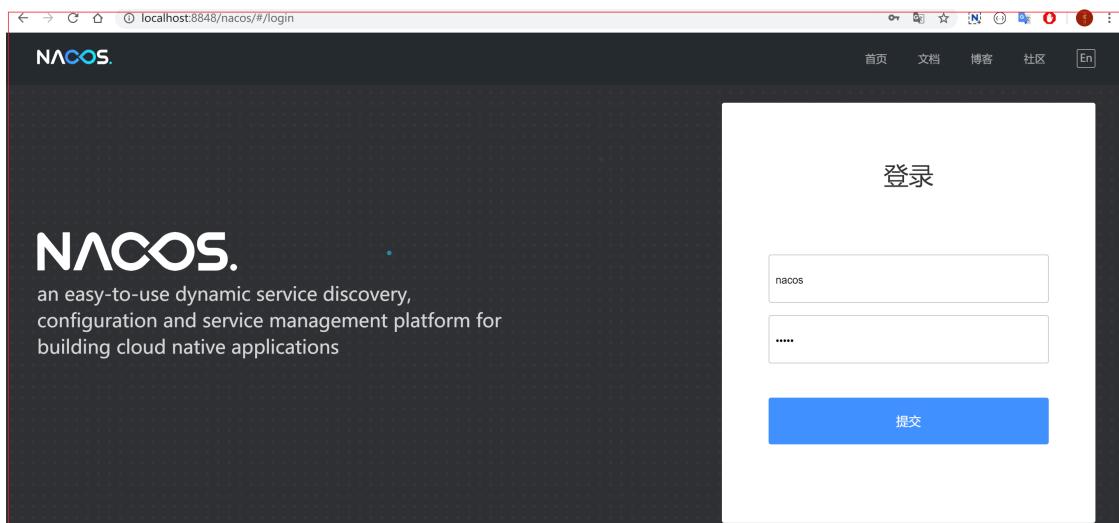
```
Nacos 1.1.4
Running in stand alone mode, All function modules
Port: 8848
Pid: 19816
Console: http://192.168.200.1:8848/nacos/index.html
https://nacos.io

2020-03-02 04:20:52,749 INFO Bean 'org.springframework.security.config.annotation.configuration.ObjectPostProcessorConfiguration' of type [org.springframework.security.config.annotation.configuration.ObjectPostProcessorConfiguration$EnhancerBySpringCGLIB$113a6bb2] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2020-03-02 04:20:52,900 INFO Bean 'objectPostProcessor' of type [org.springframework.security.config.annotation.configuration.AutowireBeanFactoryObjectPostProcessor] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2020-03-02 04:20:52,904 INFO Bean 'org.springframework.security.access.expression.method.DefaultMethodSecurityHandler@73eb439a' of type [org.springframework.security.access.expression.method.DefaultMethodSecurityHandler] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2020-03-02 04:20:52,908 INFO Bean 'org.springframework.security.config.annotation.method.configuration.GlobalMethodSecurityConfiguration' of type [org.springframework.security.config.annotation.method.configuration.GlobalMethodSecurityConfiguration$EnhancerBySpringCGLIB$360f0e64] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2020-03-02 04:20:52,915 INFO Bean 'methodSecurityMetadataSource' of type [org.springframework.security.access.method.DelegatingMethodSecurityMetadataSource] is not eligible for getting processed by all BeanPostProcessors (for example: not eligible for auto-proxying)
2020-03-02 04:20:53,973 INFO Tomcat initialized with port(s): 8848 (http)
2020-03-02 04:20:54,134 INFO Root WebApplicationContext: initialization completed in 3813 ms
```

4. 访问Nacos服务

浏览器访问<http://localhost:8848/nacos/>

输入默认用户名：nacos 密码：nacos



登陆成功界面如下

A screenshot of the Nacos configuration management interface. The URL is localhost:8848/nacos/#/configurationManagement?dataId=&group=&appName=&namespace=. The left sidebar shows sections like Configuration Management, Service Management, and Cluster Management. The main panel is titled "public" and shows a table of configuration items. The table has columns for Data ID, Group, Belong Application, and Operation. Items listed include user-service.properties, user-service.yml, user-service-dev.yml, user-service-pro.properties, and provider-service.properties. Each item has a "更多" (More) link next to its operation buttons.

关闭命令：或者双击shutdown.cmd运行文件。

```
shutdown.cmd
```

2.3 注册中心案例

1、目标：

搭建提供者服务和消费者服务，并注册到注册中心Nacos。

2、实现步骤：

第一步：搭建提供者和消费者服务

使用SpringBoot创建项目，搭建提供者服务(user-service)和消费者服务(consumer-service)。**第1天微服务业务场景搭建，基本一致，【可省略，资料中有搭建好的环境包：spring-cloud-alibaba.zip】**

1. 创建项目user-service。勾选web、Mybatis、数据库驱动的starter
 - ii. 执行创建数据库表，sql脚本
 - iii. 搭建三层架构Mapper、Service、Controller，创建根据id查询用户接口
 - iv. 配置user-service服务
 - v. 启动2个端口9091、9092服务，并测试服务user-service
2. 创建项目consumer-service。勾选web的starter
 - vii. 配置RestTemplate对象，注入Spring容器
 - viii. 搭建service层和controller层，通过RestTemplate对象发送请求访问提供者接口
 - ix. 测试消费者服务consumer-service

第二步：服务注册到Nacos注册中心

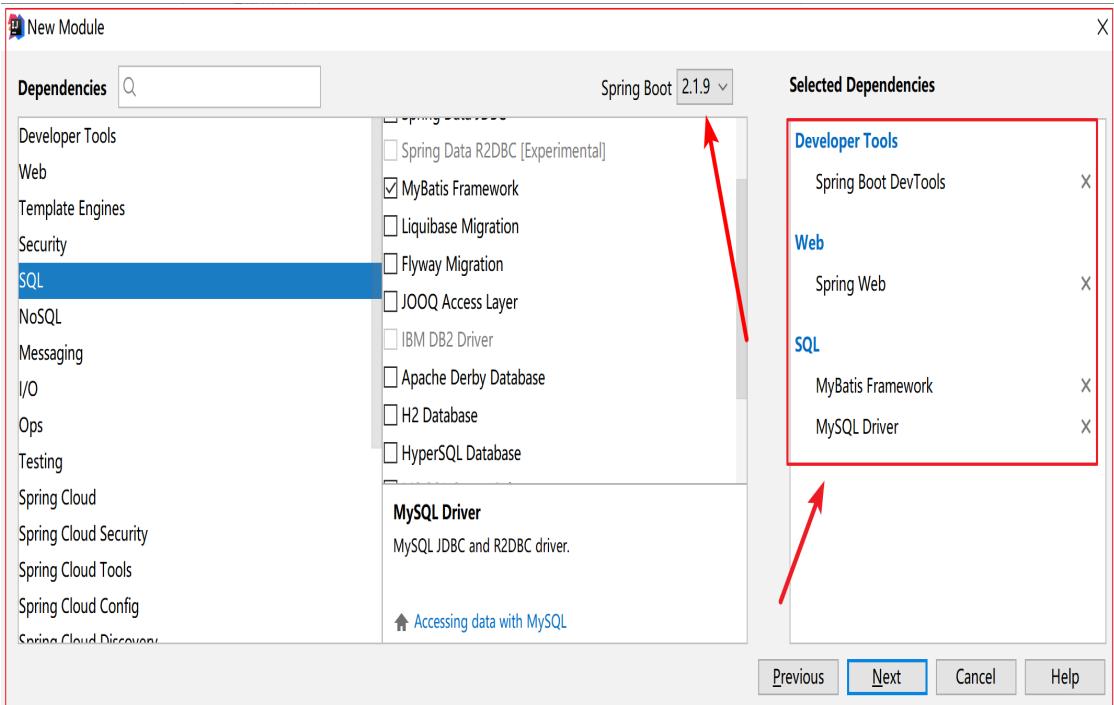
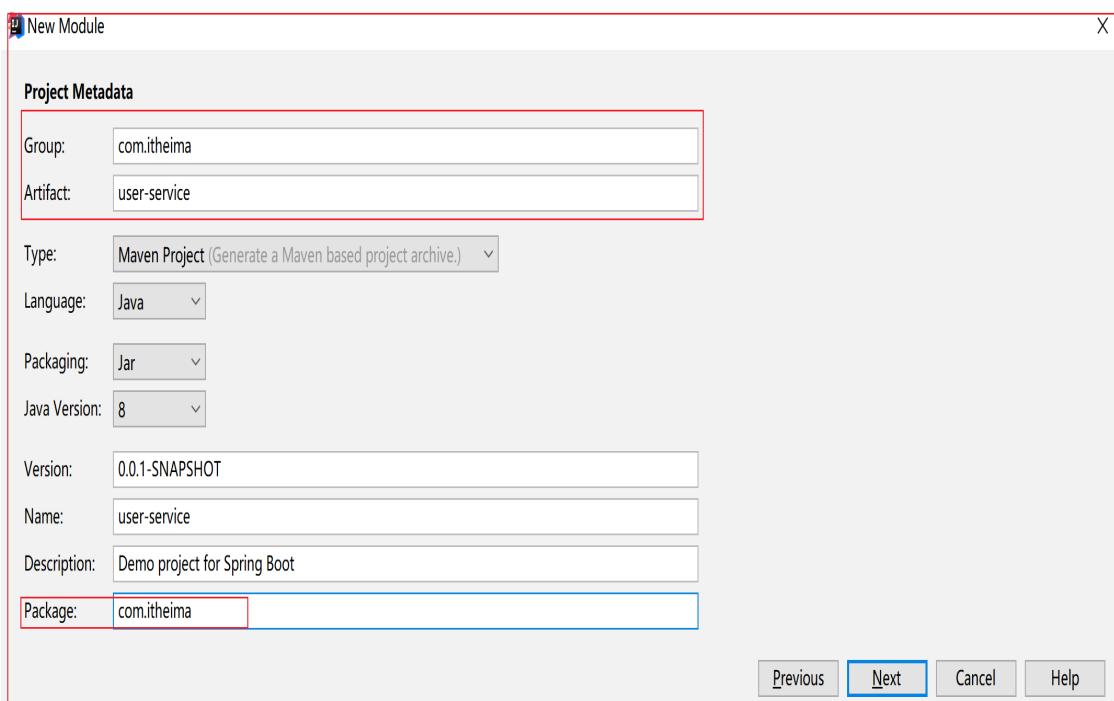
提供者服务，注册到注册中心Nacos。

消费者服务，注册到注册中心Nacos。

1. 在pom.xml中，添加SpringCloud和Nacos相关依赖
2. 配置Nacos注册中心地址：8848端口
3. 启动引导类中添加 @EnableDiscoveryClient 注解
4. 配置开启负载均衡，加@LoadBanlancer注解。使用ribbon负载均衡器去访问的用户服务
5. 将访问地址改为服务名称
6. 查看nacos管控台服务列表，测试服务负载均衡访问效果

3、实现过程：

第一步：搭建提供者和消费者服务



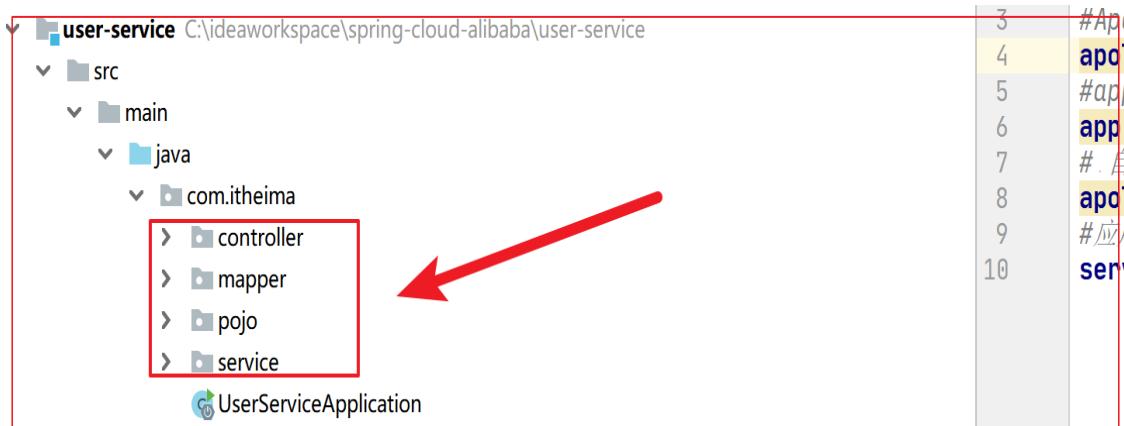
2. 执行创建数据库表，sql脚本

```

-- 如果表存在，则删除
DROP TABLE IF EXISTS `nacos_user`;
-- 创建表语句
CREATE TABLE `nacos_user` (
`id` int(11) NOT NULL AUTO_INCREMENT,
`username` varchar(50) DEFAULT NULL,
`password` varchar(50) DEFAULT NULL,
`name` varchar(50) DEFAULT NULL,
`port` varchar(50) DEFAULT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
-- 新增3条数据
INSERT INTO `nacos_user` VALUES ('1', 'wuson', '123456', '武松','');
INSERT INTO `nacos_user` VALUES ('2', 'wudalang', '654321', '武大
郎','');
INSERT INTO `nacos_user` VALUES ('3', 'panjinlian', '123456', '潘金
莲','');
SELECT * FROM nacos_user;

```

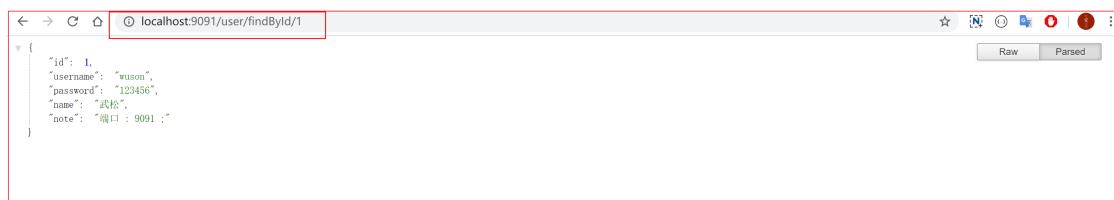
3. 搭建三层架构Mapper、Service、Controller，创建根据id查询用户接口



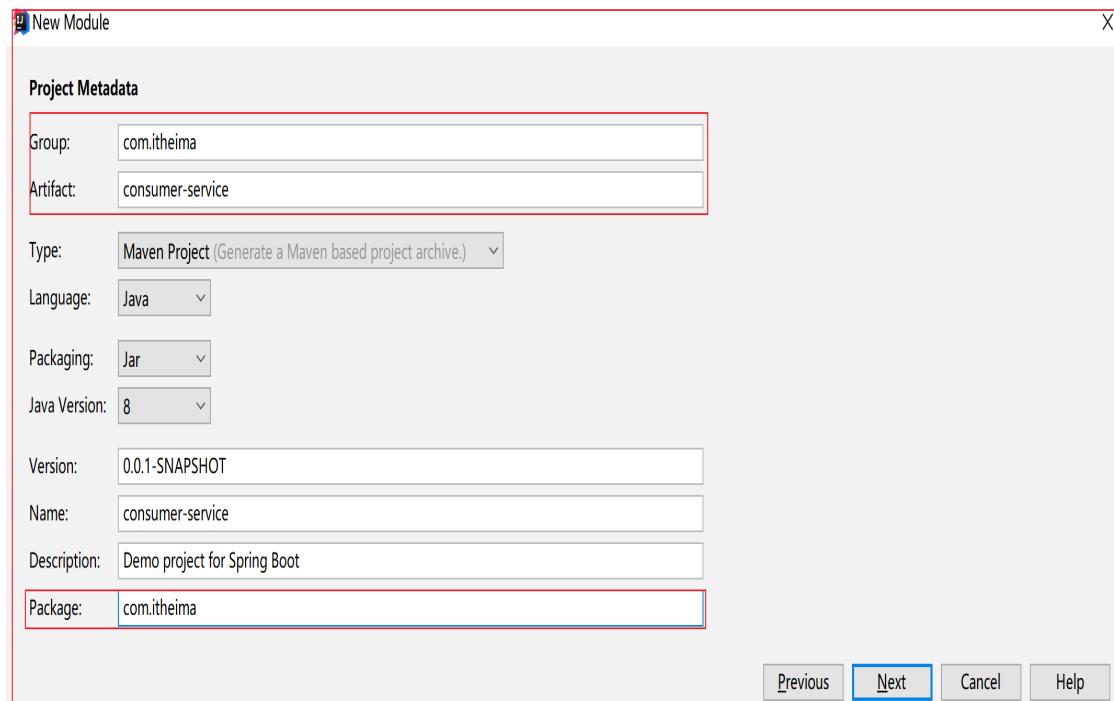
4. 配置user-service服务

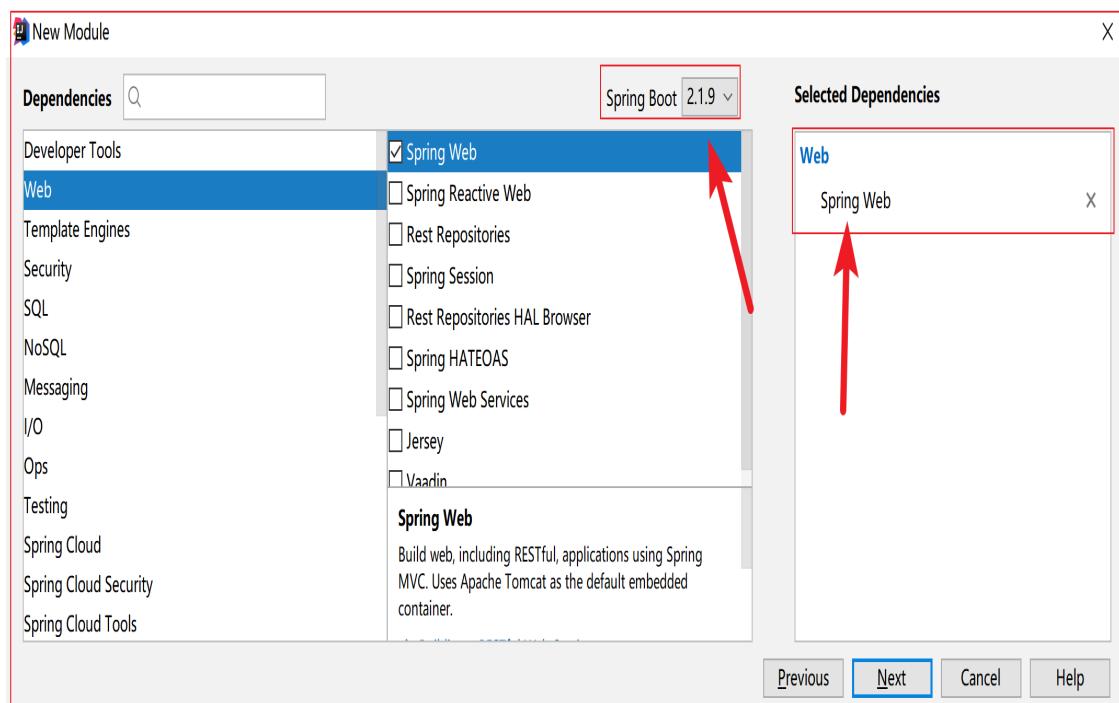
```
#自定义端口号
server.port=9091
#应用名称
spring.application.name=user-service
#db配置
spring.datasource.url=jdbc:mysql://127.0.0.1/itheima?
useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=root
```

5. 启动2个端口9091、9092服务，并测试服务user-service



6. 创建项目consumer-service。勾选web的starter





7. 配置RestTemplate对象，注入Spring容器

```
@SpringBootApplication
public class ConsumerServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConsumerServiceApplication.class,
args);
    }
    //配置RestTemplate对象，注入Spring容器
    @Bean
    public RestTemplate restTemplate(){return new RestTemplate();}
}
```

8. consumer-service配置文件

```
# 自定义端口号
server.port=8081
# 应用名
spring.application.name=consumer-service
```

9. 搭建service层和controller层，通过RestTemplate对象发送请求访问提供者接口

consumer-service C:\ideaworkspace\spring-cloud-alibaba-第2遍\consumer-service

```

    \src
      \main
        \java
          \com.itheima
            controller
            pojo
            service

```

ConsumerServiceApplication

```

// 表现层代码
@RestController
public class ConsuerController {
    @Autowired
    private UserService userService;
    @RequestMapping("/consumer/{id}")
    public User consumerUser(@PathVariable("id") Integer id){
        return userService.findUserByIdWithUserService(id);
    }
}

// 业务层代码
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private RestTemplate restTemplate;
    @Override
    public User findUserByIdWithUserService(Integer id) {
        // 使用ribbon负载均衡器，同时配置了注册中心
        // 才可以使用服务名称访问对应服务，暂时使用host加端口访问
        String url = "http://localhost:9091/user/findById/" + id;
        User user = restTemplate.getForObject(url, User.class);
        return user;
    }
}

```

10. 测试消费者服务consumer-service



第二步：服务注册到Nacos注册中心

1. 在pom.xml中，添加SpringCloud和Nacos相关依赖

```
<!--版本号配置定义-->
<properties>
    <java.version>1.8</java.version>
    <!--SpringCloud阿里巴巴版本-->

    <spring.cloud.alibaba.version>2.1.0.RELEASE</spring.cloud.alibaba.version>
    <!--SpringCloud版本-->
    <spring.cloud.version>Greenwich.SR4</spring.cloud.version>
</properties>
<dependencies>
    <!-- 省略部分坐标... -->
    <!--nacos注册中心-->
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
    </dependency>
</dependencies>
<!--BOM 依赖管理清单列表-->
<dependencyManagement>
    <dependencies>
        <!--SpringCloud的所有依赖坐标清单-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring.cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <!--Alibaba的相关所有依赖坐标清单-->
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-dependencies</artifactId>
            <version>${spring.cloud.alibaba.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

2. 配置Nacos注册中心地址，提供者、消费者都配

```
#注册到nacos注册中心
spring.cloud.nacos.discovery.server-addr=127.0.0.1:8848
```

3. 启动引导类中添加 @EnableDiscoveryClient 注解，提供者、消费者都加

```
@SpringBootApplication
@EnableDiscoveryClient//开启注册中心客户端
public class UserServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(UserServiceApplication.class, args);
    }
}
```

4. 消费者服务中，配置开启负载均衡，加@LoadBanlancer注解。

```
@SpringBootApplication
@EnableDiscoveryClient//开启注册中心客户端
public class ConsumerServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConsumerServiceApplication.class,
args);
    }
    //配置RestTemplate对象，注入Spring容器
    @Bean
    @LoadBalanced//开启负载均衡器
    public RestTemplate restTemplate(){
        return new RestTemplate();
    }
}
```

5. 将访问地址改为服务名称

```

@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private RestTemplate restTemplate;
    @Override
    public User findUserByIdWithUserService(Integer id) {
        // 使用ribbon负载均衡器，同时配置了注册中心
        // 才可以使用服务名称访问对应服务
        String url = "http://user-service/user/findById/" + id;
        User user = restTemplate.getForObject(url, User.class);
        return user;
    }
}

```

6. 查看nacos管控台服务列表，测试服务负载均衡访问效果

服务名	分组名称	集群数目	实例数	健康实例数	触发保护阀值	操作
user-service	DEFAULT_GROUP	1	2	2	false	详情 示例代码 删除
consumer-service	DEFAULT_GROUP	1	1	1	false	详情 示例代码 删除

```

{
  "id": 1,
  "username": "wuson",
  "password": "123456",
  "name": "武松",
  "note": "端口变化：9091,9092，说明负载均衡生效"
}

```

```

2020-1-9 06:56:06.906 [nio-8081-exec-3] c.netflix.config.ChainedDynamicProperty : Flipping property: user-service.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLimit = 2147483647
2020-1-9 06:56:07.907 [nio-8081-exec-31_c_n_l_DynamicServerListLoadBalancer] c.netflix.config.ChainedDynamicProperty : DynamicServerListLoadBalancer for client user-service initialized:DynamicServerListLoadBalancer:{NFLoadBalancer{name=user-service,current list of Servers=[192.168.200.1:9092,192.168.200.1:9091]},Load balancer stats=Zone stats: {unknown=[Zone:unknown;Instance count:2;Active connections count: 0;Circuit breaker tripped count: 0;Active connections per server: 0.0];},Server stats: {[Server:192.168.200.1:9091;Zone:UNKNOWN;Total Requests:0;Successive connection failure:0;Total blackout seconds:0;Last connection made:Thu Jan 01 08:00:00 CST 1970;First connection made: Thu Jan 01 08:00:00 CST 1970;Active Connections:0;total failure count in last (1000) msecs:0;average resp time:0.0;98 percentile resp time:0.0;95 percentile resp time:0.0;min resp time:0.0;max resp time:0.0;stdev resp time:0.0]}
, [Server:192.168.200.1:9092;Zone:UNKNOWN;Total Requests:0;Successive connection failure:0;Total blackout seconds:0;Last connection made:Thu Jan 01 08:00:00 CST 1970;First connection made: Thu Jan 01 08:00:00 CST 1970;Active Connections:0;total failure count in last (1000) msecs:0;average resp time:0.0;90 percentile resp time:0.0;95 percentile resp time:0.0;min resp time:0.0;max resp time:0.0;stdev resp time:0.0]
}ServerList:com.alibaba.cloud.nacos.ribbon.NacosServerList@58d85f3c
2020-1-9 06:56:07.907 [erListUpdater-0] c.netflix.config.ChainedDynamicProperty : Flipping property: user-service.ribbon.ActiveConnectionsLimit to use NEXT property: niws.loadbalancer.availabilityFilteringRule.activeConnectionsLimit = 2147483647

```

查看服务详情

The screenshot shows the Nacos service details interface. On the left is a sidebar with navigation options: 配置管理 (Configuration Management), 配置列表 (Configuration List), 历史版本 (History Version), 监听查询 (Listen Query), 服务管理 (Service Management), 服务列表 (Service List), 订阅者列表 (Subscriber List), 命名空间 (Namespace), 集群管理 (Cluster Management), and 节点列表 (Node List). The main area is titled '服务详情' (Service Details) for a service named 'user-service' in group 'DEFAULT_GROUP'. It includes fields for '保护阈值' (Protection Threshold) set to 0, '元数据' (Metadata) which is empty, and '服务路由类型' (Service Routing Type) set to 'none'. Below this is a table titled '集群: DEFAULT' (Cluster: DEFAULT) with two rows of data. The first row has IP '192.168.200.1', port '9092', temporary instance 'true', weight '1', healthy status 'true', and metadata 'preserved.register.source=SPRING_CLOUD'. The second row has the same values. To the right of the table are buttons for '修改服务权重' (Modify Service Weight), '下线服务' (Deactivate Service), and '集群配置' (Cluster Configuration). A red arrow points to the '操作' (Operation) column, and another red arrow points to the '下线' (Deactivate) button in the second row's operation column.

可以在nacos管控台服务详情列表中，修改服务的权重和服务上下线。

2.4 配置中心案例

1、目标：

将提供者服务的配置文件，提取到nacos配置中心中管理

2、实现步骤：

1. 在Nacos管控台的配置管理中，添加提供者服务配置文件
2. 在pom.xml中，添加Nacos配置中心依赖坐标
3. 删除application.properties配置文件，使用bootstrap.properties配置文件
4. bootstrap.properties配置文件中，添加Nacos配置中心地址
5. 启动服务测试访问

3、实现过程：

1. 在Nacos管控台的配置管理中，添加提供者服务配置文件

Nacos 1.1.4

配置管理 | public 搜索结果：共查找到 0 条满足要求的配置。

Data ID: 模糊查询请输入Data ID Group: 模糊查询请输入Group 等级: 高级查询 导出查询结果 导入配置

没有数据

新建配置

Data ID: user-service-dev.properties 数据id: {应用名称}-{环境名称}.properties, 多环境(dev、pro、test)

Group: DEFAULT_GROUP 配置分组

描述: 数据库配置

配置文件格式

配置格式: TEXT JSON XML YAML HTML Properties

配置内容:

```
1 #db配置
2 spring.datasource.url=jdbc:mysql://127.0.0.1/heima?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
3 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
4 spring.datasource.username=root
5 spring.datasource.password=root
```

2. 在pom.xml中，添加Nacos配置中心依赖坐标

```
<!--nacos配置中心依赖-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-config</artifactId>
</dependency>
```

3. 删除application.properties配置文件，使用bootstrap.properties配置文件

user-service D:\362\项目二前置课资料\10-springcloud\02代码\spring-cloud-alibaba\user-service

src

main

java

com.itheima

controller

mapper

pojo

service

UserServiceApplication

resources

static

templates

bootstrap.properties

4. bootstrap.properties配置文件中，添加Nacos配置中心地址

```
#配置中心地址  
spring.cloud.nacos.config.server-addr=127.0.0.1:8848  
#指定配置中心的文件格式，默认properties，如果使用yml需显示声明  
#spring.cloud.nacos.config.file-extension=yml
```

5. 启动服务测试访问

```
... (V2.1.9.RELEASE)

15:11.270 INFO 23860 --- [ restartedMain] c.a.c.n.c.NacosPropertySourceBuilder : Loading nacos data, dataId: 'user-service.properties', group: null
15:11.272 INFO 23860 --- [ restartedMain] b.c.PropertySourceBootstrapConfiguration : Located property source: CompositePropertySource [name=NACOS]
15:11.276 INFO 23860 --- [ restartedMain] com.theima.UserServiceApplication : No active profile set, falling back to default profiles: dev
15:12.331 INFO 23860 --- [ restartedMain] o.s.cloud.context.scope.GenericScope : BeanFactory id=ea0644fe-5c4b-3446-8679-2dc88b155b05
15:13.597 INFO 23860 --- [ restartedMain] trationDelegateSbeanPostProcessorChecker : Bean 'org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebinderAutoConfiguration' of type [null] registered in internal bean 'tationDelegateSbeanPostProcessorChecker'
15:12.407 INFO 23860 --- [ restartedMain] trationDelegateSbeanPostProcessorChecker : Bean 'org.springframework.transaction.annotation.ProxyTransactionMBeanName' of type [null] registered in internal bean 'tationDelegateSbeanPostProcessorChecker'
15:13.289 INFO 23860 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9091 (http)
15:13.193 INFO 23860 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
15:13.191 INFO 23860 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.26]
15:13.549 INFO 23860 --- [ restartedMain] o.a.c.c.c.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
15:13.549 INFO 23860 --- [ restartedMain] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 226
```

注意：

- Data-ID名称一旦填写发布不可修改
 - Data-ID中，应用名称要与对应服务应用名称一致： {应用名称}-{环境名称}.properties
 - Data-ID中，环境名称可以不写
 - 默认加载properties格式配置文件，如果使用yml格式，则需要在配置文件中声明文件格式。

扩展1：配置自动刷新

提供者服务，配置信息无法自动刷新配置

可添加personName=肖战，在controller中获取进行测试。

解决办法：

- ### 1. 需要刷新的controller上添加 @RefreshScope

```

@RestController
@RequestMapping("user")
@RefreshScope//自动刷新配置
public class UserController {
    @Autowired
    private UserService userService;

    @Value("${server.port}")
    private String port;
    //获取配置文件中personName
    @Value("${personName}")
    private String name;

    @RequestMapping("/findById/{id}")
    public User findById(@PathVariable("id") Integer id){
        User user = userService.findById(id);
        //设置端口信息，设置PersonName
        user.setNote("端口：" + port + "； personName：" + name);
        return user;
    }
}

```

2. 修改配置，再次测试

```

{
  "id": 1,
  "username": "wuson",
  "password": "123456",
  "name": "武林客",
  "note": "端口：9091； personName：肖战"
}

```

注意：无需第三方消息队列中间件

扩展2：多环境配置使用

1. 方式一：文件名称命名

我们在Nacos界面新建配置时可以通过文件名来区分环境，当多环境时可以将名称填写为：`user-service-dev.properties`，但是需要在生产者的配置文件中声明：`spring.profiles.active=dev`

nacos配置文件：

配置管理 | public | 查询结果: 共查询到 4 条满足要求的配置。

Data ID:		模糊查询请输入Data ID	Group:	模糊查询请输入Group	查询	高级查询	导出查询结果	导入配置	+
<input type="checkbox"/> Data Id		Group		归属应用:		操作			
<input type="checkbox"/> user-service.properties		DEFAULT_GROUP				详情 示例代码 编辑 删 除 更多			
<input type="checkbox"/> user-service-dev.properties		开发环境配置		DEFAULT_GROUP				详情 示例代码 编辑 删 除 更多	
<input type="checkbox"/> user-service-test.properties		测试环境配置		DEFAULT_GROUP				详情 示例代码 编辑 删 除 更多	
<input type="checkbox"/> user-service-pro.properties		生产环境配置		DEFAULT_GROUP				详情 示例代码 编辑 删 除 更多	

删除 导出选中的配置 克隆 每页显示: 10 < 上一页 1 下一页 >

2. 方式二：Group分组

除了使用配置文件命名方式，我们还可以使用group分组的方式来实现相同效果。

新建配置文件时，默认是DEFAULT_GROUP。

我们可以通过设置不同的group和在bootstrap.properties中指定分组来区别不同的环境。

nacos配置文件：

新建配置

* Data ID:	user-service-prod.properties
* Group:	PROD_GROUP
使用分组来区分配置文件	
更多高级选项	
描述: 分组模式中的, 配置文件	
配置格式: <input type="radio"/> TEXT <input type="radio"/> JSON <input type="radio"/> XML <input type="radio"/> YAML <input type="radio"/> HTML <input checked="" type="radio"/> Properties	
* 配置内容:	<pre> 1 #db配置 2 spring.datasource.url=jdbc:mysql://127.0.0.1/heima?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC 3 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver 4 spring.datasource.username=root 5 spring.datasource.password=root 6 personName=分组模式下的开发环境的肖战 </pre>

bootstrap.properties:

```
#指定配置中心分组
spring.cloud.nacos.config.group=PROD_GROUP
```

```

        \\\
        ) ) )
=/_/_/_/
9.RELEASE)

184 --- [ restartedMain] c.a.c.n.c.NacosPropertySourceBuilder : Loading nacos data, dataId: 'user-service.properties', group: 'PROD_GROUP'
184 --- [ restartedMain] b.c.PropertySourceBootstrapConfiguration : Located property source: CompositePropertySource {name='NACOS', propertySourc
184 --- [ restartedMain] com.itheima.UserServiceApplication : No active profile set, falling back to default profiles: default
184 --- [ restartedMain] o.s.cloud.context.scope.GenericScope : BeanFactory id=df809705-d300-31ba-af87-2ebf9f754d70
184 --- [ restartedMain] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.cloud.autoconfigure.ConfigurationPropertiesRebindC
184 --- [ restartedMain] trationDelegate$BeanPostProcessorChecker : Bean 'org.springframework.transaction.annotation.ProxyTransactionManagementC
184 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 9092 (http)
184 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
184 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.26]
184 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext

```



三、流量防卫兵Sentinel

3.1 Sentinel简介

Sentinel: 分布式系统的流量防卫兵



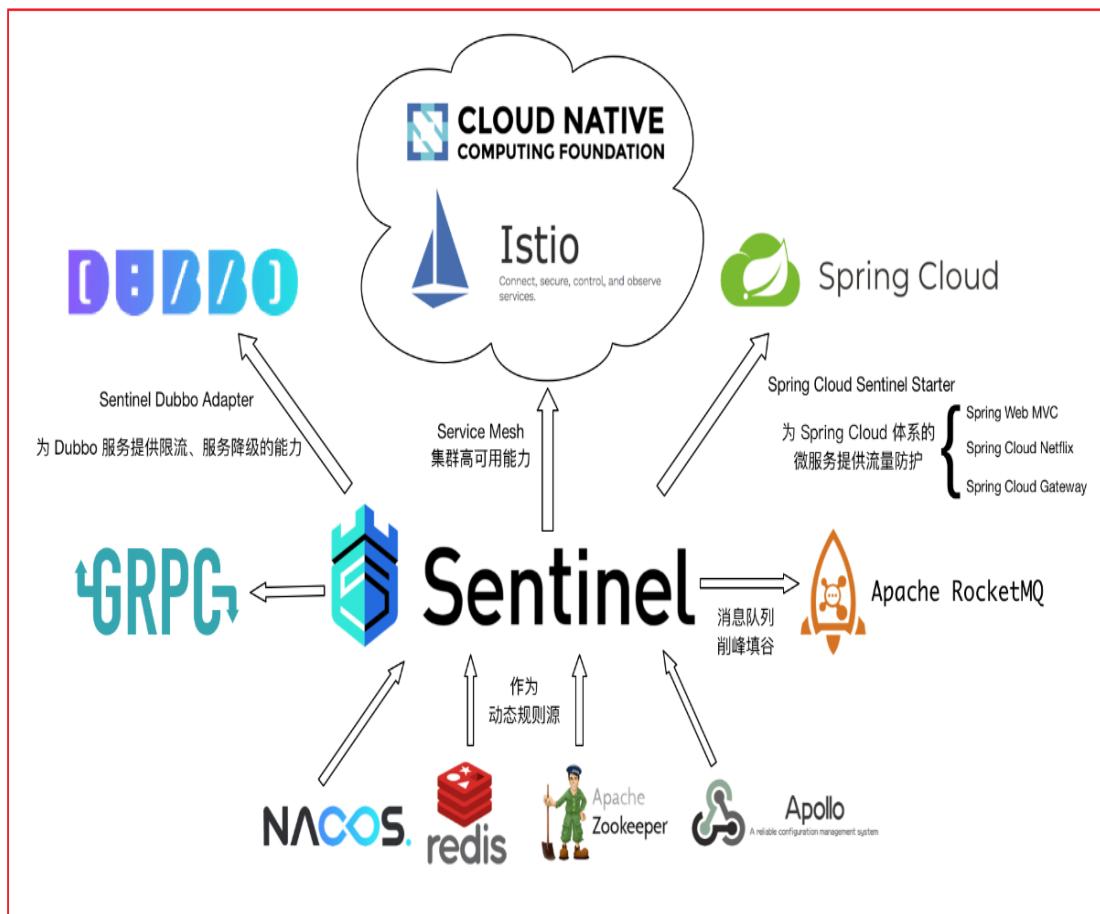
随着微服务的流行，服务和服务之间的稳定性变得越来越重要。Sentinel 是面向分布式服务架构的轻量级流量控制组件，主要以流量为切入点，从限流、流量整形、熔断降级、系统负载保护等多个维度来帮助您保障微服务的稳定性。

- 2012 年，Sentinel 诞生，主要功能为入口流量控制。
- 2013-2017 年，Sentinel 在阿里巴巴集团内部迅速发展，成为基础技术模块，覆盖了所有的核心场景。Sentinel 也因此积累了大量的流量归整场景以及生产实践。
- 2018 年，Sentinel 开源，并持续演进。

1、主要特征

- 丰富的应用场景：Sentinel 承接了阿里巴巴近 10 年的双十一大促流量的核心场景，例如秒杀（即突发流量控制在系统容量可以承受的范围）、消息削峰填谷、集群流量控制、实时熔断下游不可用应用等。
- 完备的实时监控：Sentinel 同时提供实时的监控功能。您可以在控制台中看到接入应用的单台机器秒级数据，甚至 500 台以下规模的集群的汇总运行情况。
- 广泛的开源生态：Sentinel 提供开箱即用的与其它开源框架/库的整合模块，例如与 Spring Cloud、Dubbo、gRPC 的整合。您只需要引入相应的依赖并进行简单的配置即可快速地接入 Sentinel。
- 完善的 SPI 扩展点：Sentinel 提供简单易用、完善的 SPI 扩展接口。您可以通过实现扩展接口来快速地定制逻辑。例如定制规则管理、适配动态数据源等。

Sentinel 的主要特性：



3、Sentinel 分为两个部分：

- 核心库（Java 客户端）不依赖任何框架/库，能够运行于所有 Java 运行时环境，同时对 Dubbo / Spring Cloud 等框架也有较好的支持。
- 控制台（Dashboard）基于 Spring Boot 开发，打包后可以直接运行，不需要额外的 Tomcat 等应用容器。

4、那些公司在用？



3.2 Sentinel控制台安装

1、下载jar

jar包在资料中已经下载完毕

2、启动：

java -jar 启动 默认端口8080，可以通过 -Dserver.port=8086 来修改

```
# 第一种  
java -jar -Dserver.port=8086 sentinel-dashboard-1.6.3.jar  
# 第二种  
java -jar sentinel-dashboard-1.6.3.jar --server.port=8086
```

3、访问测试：

<http://127.0.0.1:8086> 默认账密： sentinel/sentinel



3.3 服务集成Sentinel

目标： 提供者服务和消费者服务， 集成至Sentinel

实现步骤：

1. 在pom.xml添加sentinel依赖坐标
2. 配置文件添加sentinel地址
3. 必须访问服务接口， <http://localhost:9091/user/findById/1>,
<http://localhost:8081/consumer/1>
4. 然后在<http://127.0.0.1:8086>, sentinel中才能看到对应服务

实现过程： 生产者服务和消费者服务

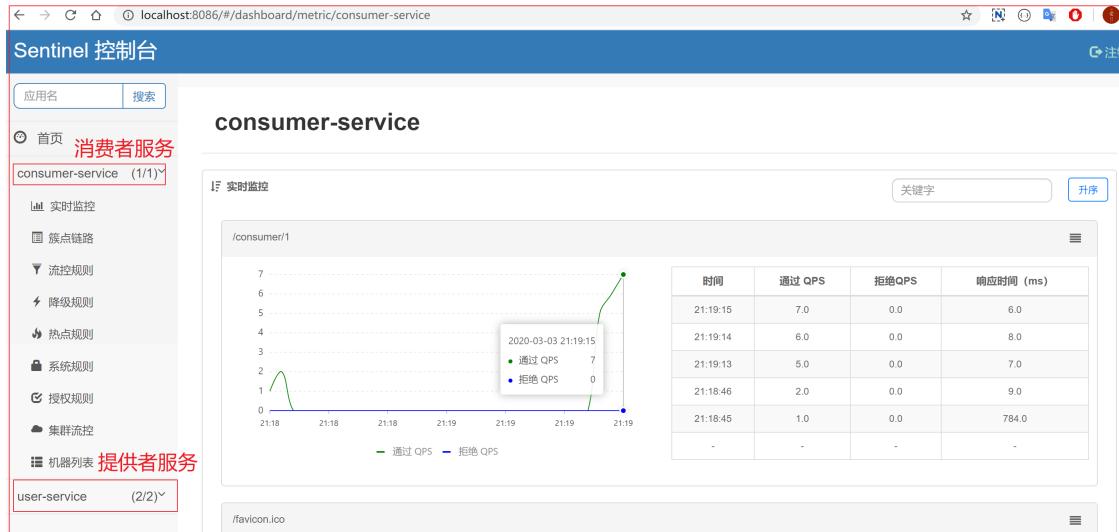
1. 在pom添加sentinel依赖

```
<!--sentinel客户端依赖-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-sentinel</artifactId>
</dependency>
```

2. 配置文件添加sentinel地址

```
#sentinel注册地址
spring.cloud.sentinel.transport.dashboard=127.0.0.1:8086
#服务接收sentinel的配置端口
spring.cloud.sentinel.transport.port=8719
```

3. 访问服务接口 <http://localhost:19091/user/1>,
<http://localhost:8081/consumer/1>
4. 然后在<http://127.0.0.1:8086>, sentinel中才能看到对应服务



3.4 演示：服务降级和限流

目标：消费者服务降级和限流效果演示

实现步骤：

1. 在消费者服务中，添加服务降级和限流处理方法
2. 在sentinel控制台中，添加服务降级配置
3. 测试服务降级
4. 在sentinel控制台中，添加服务限流配置
5. 测试服务限流

实现过程：

1. 在消费者服务中，添加服务降级和限流处理方法

```
@Service
public class UserServiceImpl implements UserService {
    @Autowired
    private RestTemplate restTemplate;
    /**
     * @SentinelResource 注解作用：设置服务降级处理方法和服务限流处理方法
     * value属性 设置被降级或限流方法
     * fallback属性 设置服务降级处理方法
     * blockHandler属性 设置服务限流处理方法
     */
    @Override
    @SentinelResource(value = "findUserByIdWithUserService", fallback
= "fallbackMethod", blockHandler = "blockMethod")
    public User findUserByIdWithUserService(Integer id) {
        // 使用ribbon负载均衡器，同时配置了注册中心
        // 才可以使用服务名称访问对应服务
        String url = "http://user-service:9091/user/findById/" + id;
        User user = restTemplate.getForObject(url, User.class);
        return user;
    }
    // 创建服务降级方法，与被降级方法返回值保持一致，参数保持一致
    public User fallbackMethod(Integer id) {
        User user = new User();
        user.setNote("服务降级处理了~");
        return user;
    }
    // 创建服务限流方法，与被降级方法返回值保持一致，参数保持一致，除了参数保持一致，还需要加一个参数blockException
    public User blockMethod(Integer id) {
        User user = new User();
        user.setNote("服务限流处理了~");
        return user;
    }
}
```

2. 在sentinel控制台中，添加服务降级配置

Sentinel 控制台

consumer-service

资源名	通过QPS	拒绝QPS	线程数	平均RT	分钟通过	分钟拒绝	操作
sentinel_default_context	0	0	0	0	0	0	<button>+ 流控</button> <button>+ 降级</button> <button>+ 热点</button> <button>+ 授权</button>
sentinel_web_context	0	0	0	7	0	0	<button>+ 流控</button> <button>+ 降级</button> <button>+ 热点</button> <button>+ 授权</button>
/consumer/2	0	0	0	0	4	0	<button>+ 流控</button> <button>+ 降级</button> <button>+ 热点</button> <button>+ 授权</button>
findUserByIdWithUserService	0	0	0	0	1	3	<button>+ 流控</button> <button>+ 降级</button> <button>+ 热点</button> <button>+ 授权</button>
/consumer/1	0	0	0	0	0	0	<button>+ 流控</button> <button>+ 降级</button> <button>+ 热点</button> <button>+ 授权</button>
/favicon.ico	0	0	0	0	3	0	<button>+ 流控</button> <button>+ 降级</button> <button>+ 热点</button> <button>+ 授权</button>

Sentinel 控制台

consumer-service

新增降级规则

出现异常

资源名:

降级策略: RT 异常比例 异常数

异常数: 异常数

时间窗口: 降级时间间隔, 单位秒

方便测试降级效果: 异常数1个, 时间窗口: 1秒

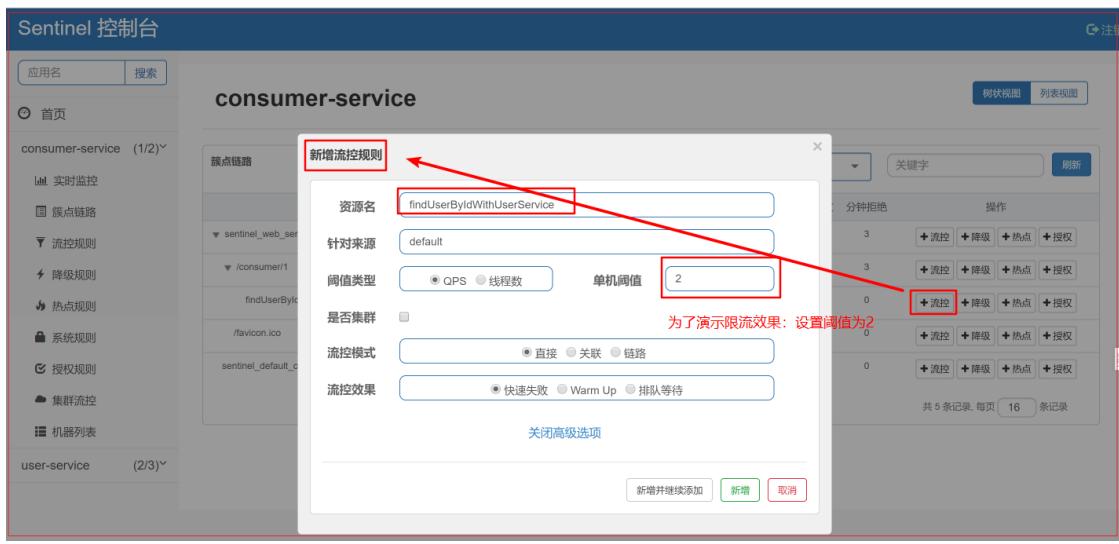
新增并继续添加 新增 取消

3. 测试服务降级

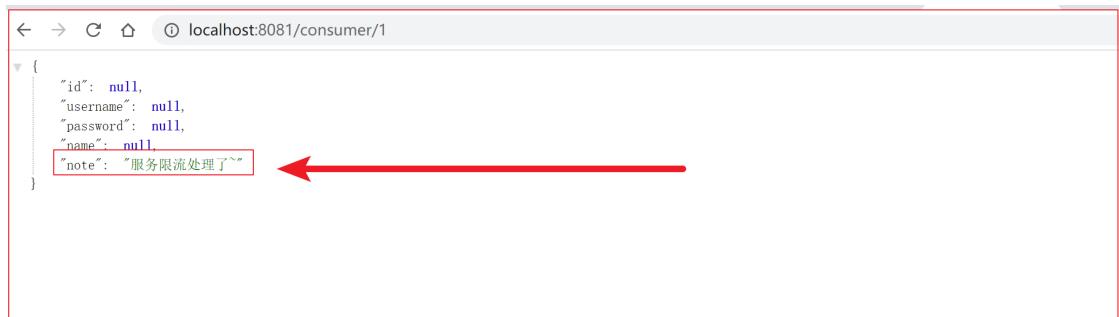
localhost:8081/consumer/1

```
{ "id": null, "username": null, "password": null, "note": "服务降级处理了~" }
```

4. 在sentinel控制台中, 添加服务限流配置



5. 测试服务限流



注意：

- 服务降级处理方法，返回值与参数必须与被降级方法保持一致
- 服务限流处理方法，返回值与参数必须与被降级方法保持一致，参数除保留原来之外，还需添加BlockException参数。
- 服务降级设置资源：findUserByIdWithUserService
- 服务降级设置资源：findUserByIdWithUserService, /consumer/1