

# 微服务-分布式-Java栈-学习笔记

2022Q1学习笔记

## 联系我

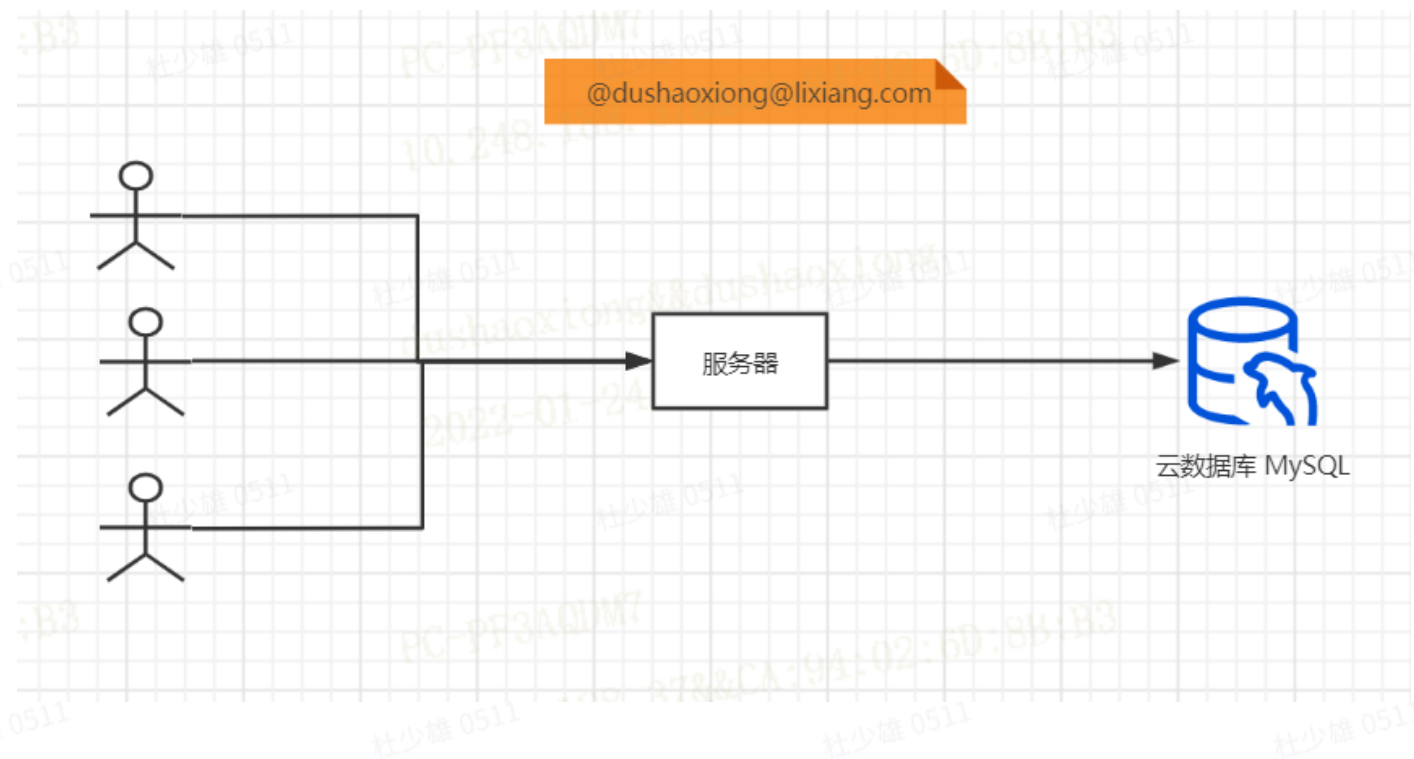
- 工作邮箱: [dushaoxiong@lixiang.com](mailto:dushaoxiong@lixiang.com)
- 个人邮箱: [email@shaoxiongdu.cn](mailto:email@shaoxiongdu.cn)
- 个人博客: [shaoxiongdu.cn](http://shaoxiongdu.cn)
- GitHub地址: [www.github.com/shaoxiongdu](https://github.com/shaoxiongdu)

配套demo项目: <https://github.com/shaoxiongdu/spring-cloud-demo>

## 一、微服务

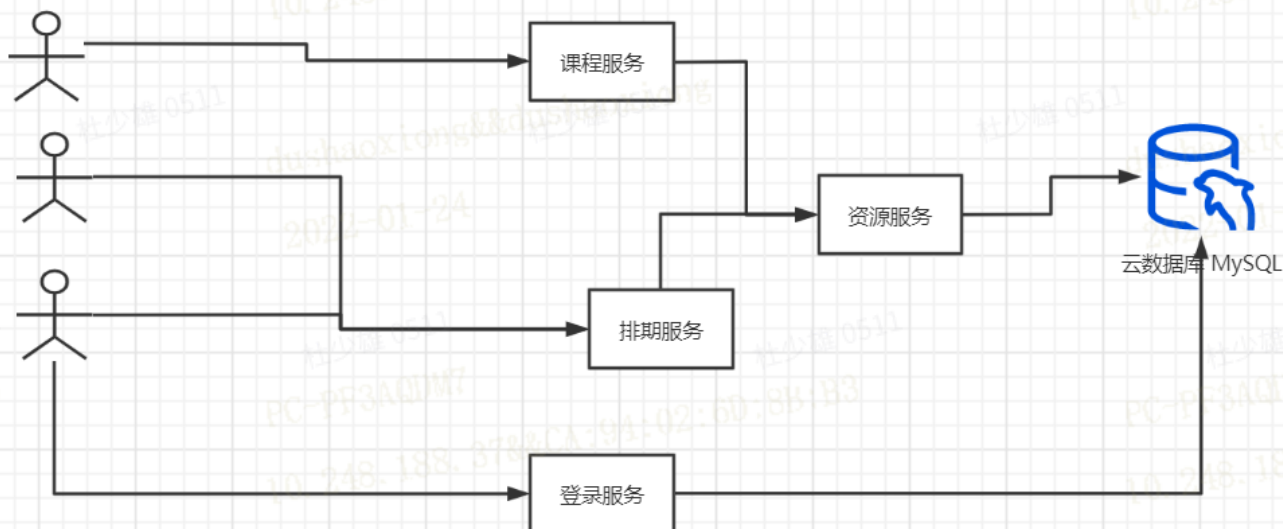
### 1. 单体架构

将业务的所有功能集中在一个项目中开发，打成一个包部署。

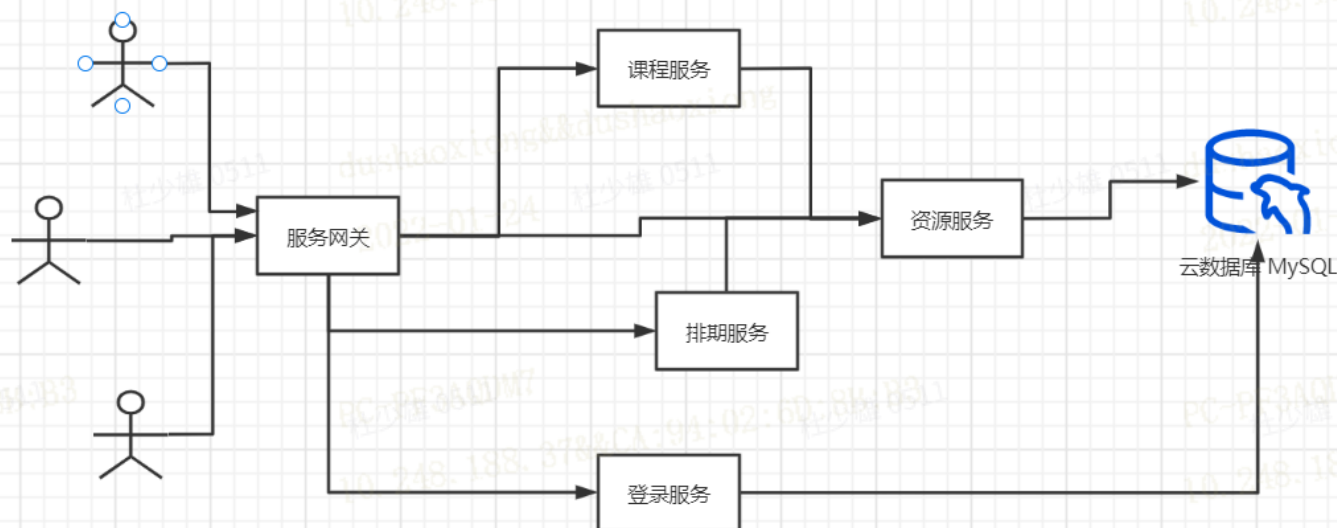


### 2. 分布式架构

根据业务功能对系统做拆分，每个业务功能模块作为独立项目开发，称为一个服务。



### 3. 微服务架构



- 单一职责：微服务拆分粒度更小，每一个服务都对应唯一的业务能力，做到单一职责
- 自治：团队独立、技术独立、数据独立，独立部署和交付
- 面向服务：服务提供统一标准的接口，与语言和技术无关
- 隔离性强：服务调用做好隔离、容错、降级，避免出现级联问题

微服务的上述特性其实是在给分布式架构制定一个标准，进一步降低服务之间的耦合度，提供服务的独立性和灵活性。做到高内聚，低耦合。

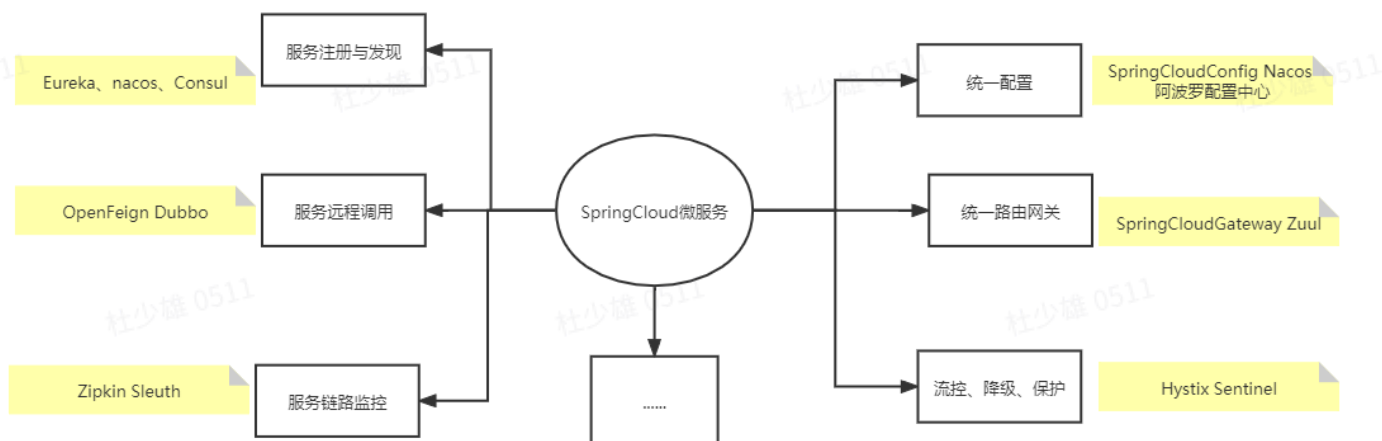
因此，可以认为**微服务是一种经过良好架构设计的分布式架构方案。**

但方案该怎么落地？选用什么样的技术栈？

全球的互联网公司都在积极尝试自己的微服务落地方案。  
其中在Java领域最引人注目的就是SpringCloud提供的方案了。

## 4. SpringCloud

@dushaoxiong@lixiang.com



## 5. 总结

单体架构：简单方便，高度耦合，扩展性差，适合小型项目。例如：学生管理系统

分布式架构：松耦合，扩展性好，但架构复杂，难度大。适合大型互联网项目，例如：京东、淘宝

微服务：一种良好的分布式架构方案

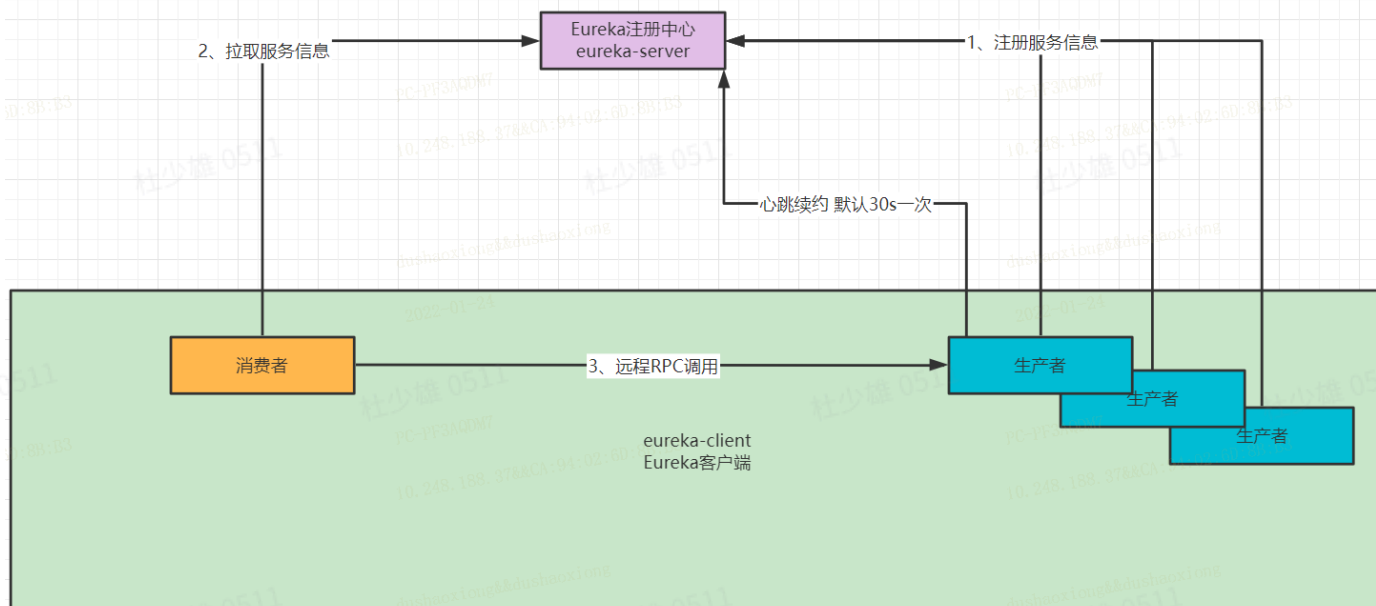
①优点：拆分粒度更小、服务更独立、耦合度更低

②缺点：架构非常复杂，运维、监控、部署难度提高

SpringCloud是微服务架构的一站式解决方案，集成了各种优秀微服务功能组件

## 二、服务注册中心

### 1. Eureka



#### a. 消费者如何得知生产者的实际地址？

- 生产者启动之后，将自己的ip端口等信息注册到Eureka服务端。（服务注册）
- Eureka保存生产者服务名称和IP端口等信息
- 消费者根据需要的生产者名称从Eureka服务端拉取列表，然后远程调用（服务拉取）

#### b. 消费者如何从多个生产者实例中选择具体的实例？

- 从生产者实例列表中利用负载均衡算法选中一个实例地址

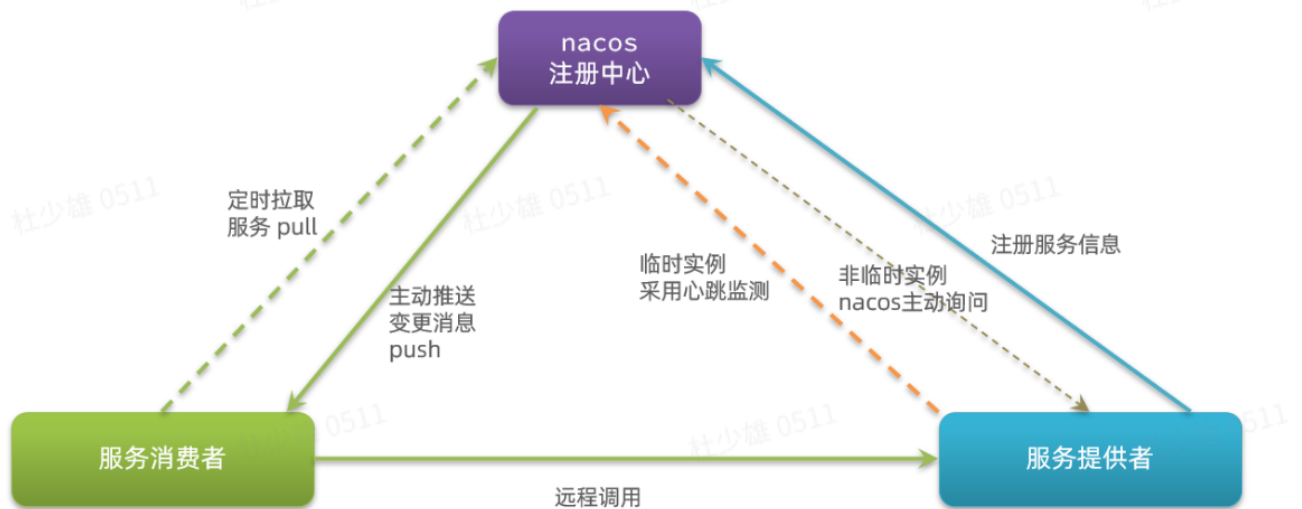
#### c. 消费者如何得知某个生产者实例是否依然健康，是不是已经宕机？

- 生产者会每隔一段时间（默认30秒）向eureka-server发起请求，报告自己状态，称为心跳
- 当超过一定时间没有发送心跳时，eureka-server会认为微服务实例故障，将该实例从服务列表中剔除
- 消费者拉取服务时，就能将故障实例排除了

## 2. Nacos

国内公司一般都推崇阿里巴巴的技术，比如注册中心，SpringCloudAlibaba也推出了一个名为Nacos的注册中心。

Nacos是阿里巴巴的产品，现在是SpringCloud中的一个组件。相比Eureka功能更加丰富，在国内受欢迎程度较高。



## 1. 使用方式

- 下载nacos 启动
- 项目导入依赖

### XML

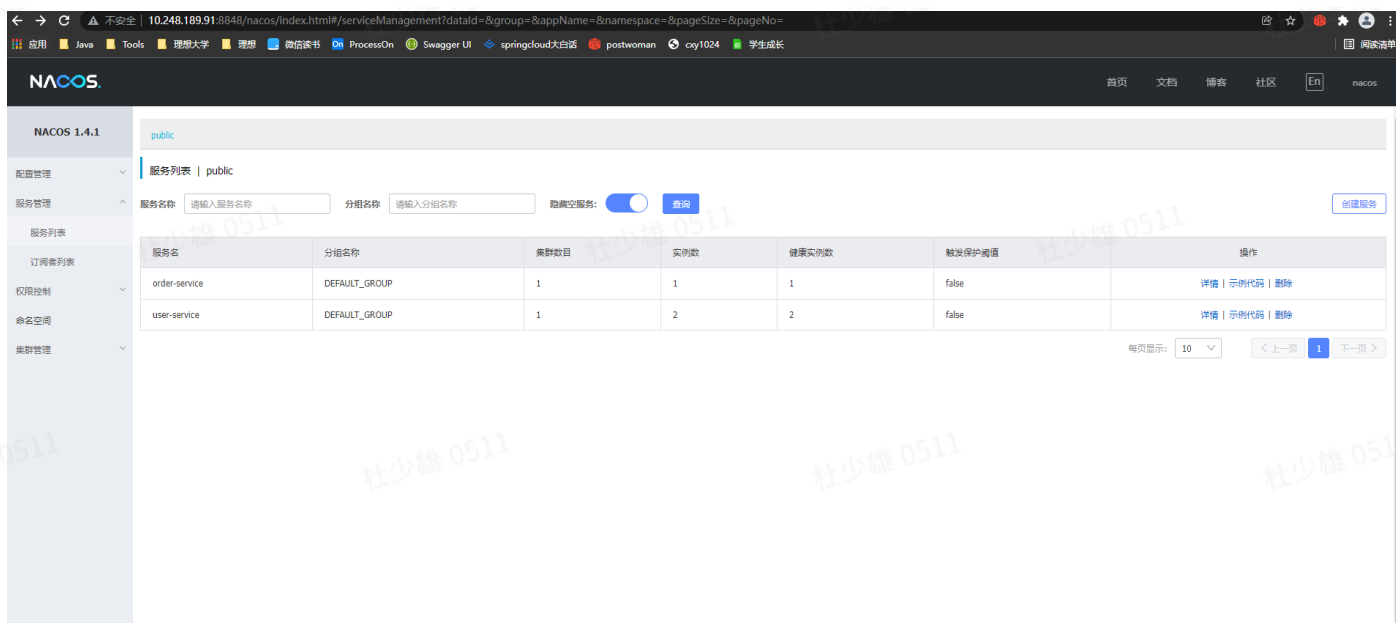
```
1 <dependency>
2   <groupId>com.alibaba.cloud</groupId>
3   <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
4 </dependency>
```

### c. 配置nacos地址

### YAML

```
1 spring:
2   cloud:
3     nacos:
4       server-addr: localhost:8848
```

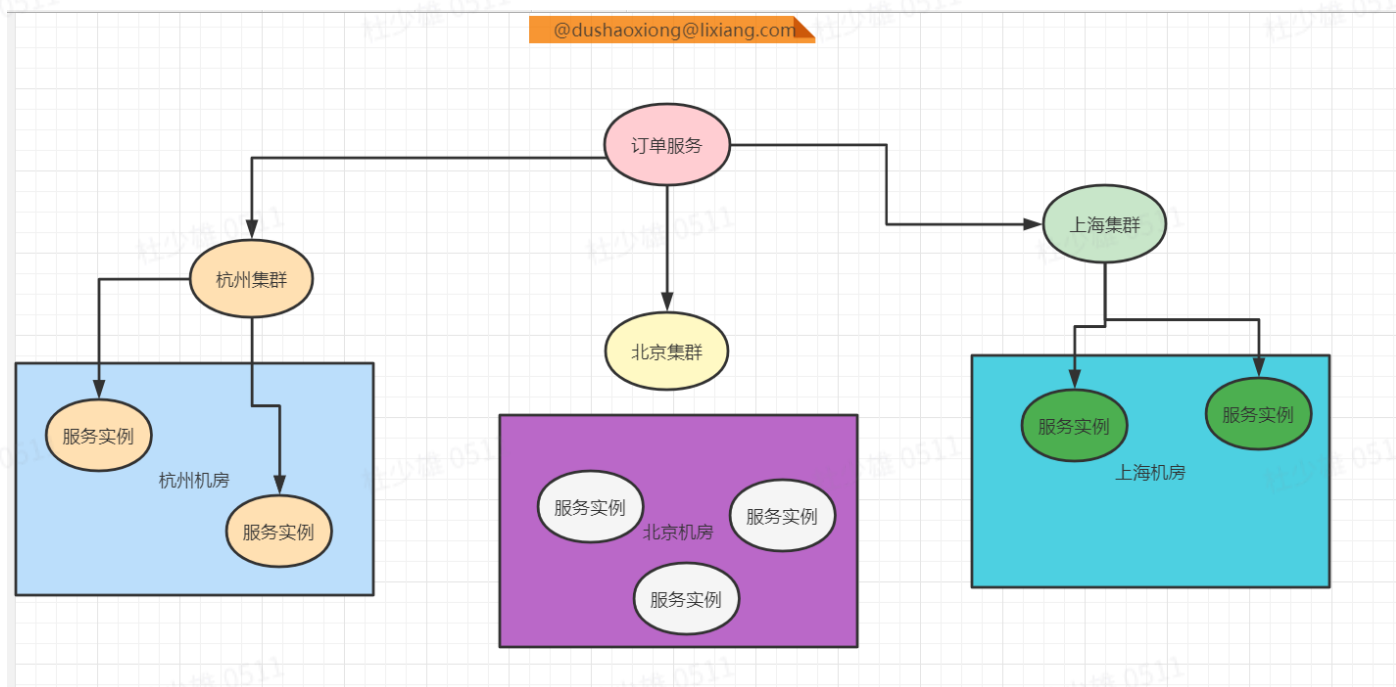
- 启动服务，即可在nacos的控制页面找到对应的微服务注册信息。



e. 通过远程RPC消费者即可调用生产者的服务。

## 2. 集群概念

Nacos就将同一机房内的实例 划分为一个**集群**。

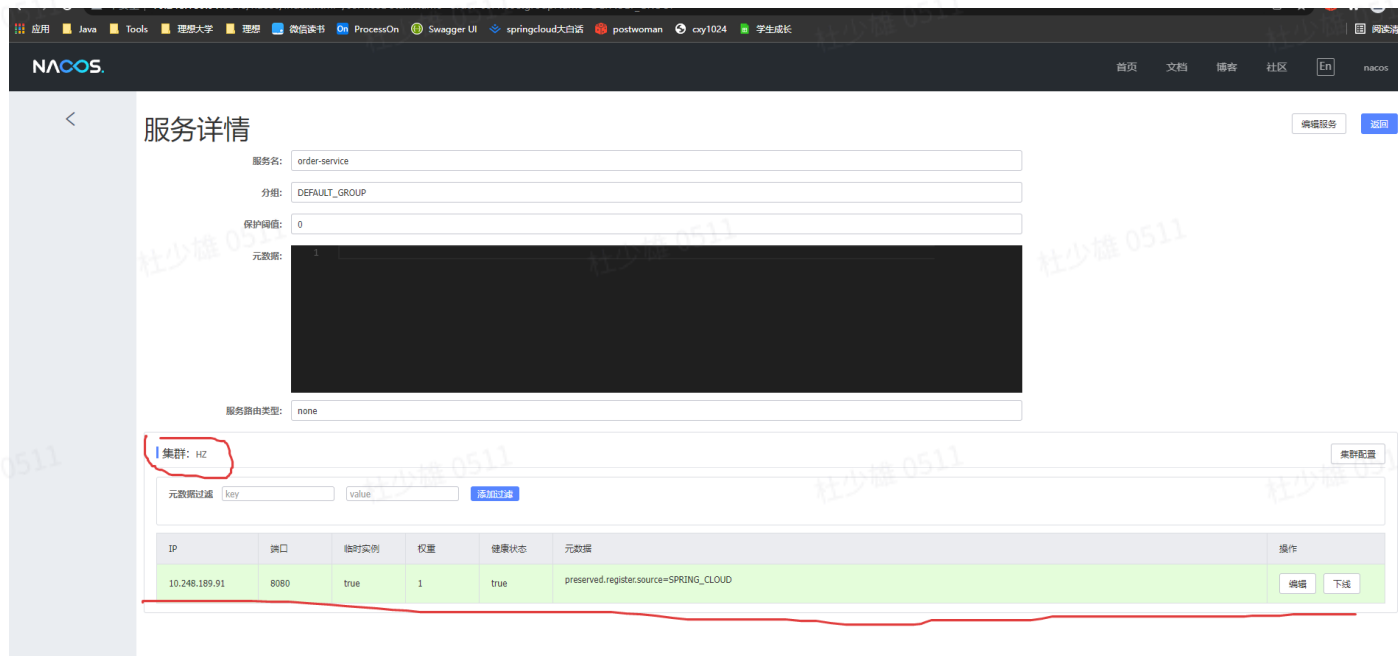


微服务互相访问时，应该尽可能访问同集群实例，因为本地访问速度更快。当本集群内不可用时，才访问其它集群。

通过配置文件即可设置当前服务所属的集群。

## YAML

```
1 spring:
2   cloud:
3     nacos:
4       server-addr: localhost:8848
5       discovery:
6         cluster-name: HZ # 集群名称
```



## 3. Nacos与Eureka对比

### a. 共同点

- 都支持服务注册和服务拉取
- 都支持服务提供者心跳方式做健康检测

### b. 不同点

Nacos的服务实例分为两种类型：

- Nacos支持服务端**主动检测提供者**状态：临时实例采用心跳模式，非临时实例采用主动检测模式
- 临时实例心跳不正常会被剔除，非临时实例则不会被剔除
- Nacos支持**服务列表变更的消息推送模式**，服务列表更新更及时
- Nacos集群默认采用AP方式，当集群中存在非临时实例时，采用CP模式；Eureka采用AP方式

## 三、服务远程调用

## 1. Feign远程调用

Feign是一个声明式的http客户端，官方地址：<https://github.com/OpenFeign/feign>

其作用就是帮助我们优雅的实现http请求的发送。

### Feign makes writing java http clients easier

gitter join chat PASSED maven central 11.1

Feign is a Java to HTTP client binder inspired by [Retrofit](#), [JAXRS-2.0](#), and [WebSocket](#). Feign's first goal was reducing the complexity of binding [Denominator](#) uniformly to HTTP APIs regardless of [ReSTfulness](#).

### 1.1 使用方式

#### 1. 引入依赖

XML

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

#### 2.SpringBoot启动类加入注解

Java

```
1 @EnableFeignClients
```

#### 3.编写Feign的客户端



## Kotlin

```
1
2 /**
3  * @author dushaoxiong@lixiang.com
4  * @version 1.0
5  * @date 2022/2/14 18:47
6  */
7 @FeignClient("user-service")
8 public interface UserClient {
9     /**
10      * 查询用户信息
11      * @param id 用户Id
12      * @return 用户信息
13      */
14     @GetMapping("/user/{id}")
15     User findById(@PathVariable("id") Long id);
16 }
```

4.直接使用: 其他地方自动注入UserClient接口 即可拿到对应的代理实现 进行远程调用

### 1.2 自定义配置

可通过yml配置文件，也可通过Java代码。以Java代码为例。

Feign可以支持很多的自定义配置，如下表所示：

	A	B	C
1	类型	作用	说明
2	<b>feign.Logger.Level</b>	修改日志级别	包含四种不同的级别： NONE、 BASIC、 HEADERS、 FULL
3	feign.codec. Decoder	响应结果的解析器	http远程调用的结果做解析，例如解析json字符串为java对象
4	feign.codec. Encoder	请求参数编码	将请求参数编码，便于通过http请求发送
5	feign. Contract	支持的注解格式	默认是SpringMVC的注解
6	feign. Retryer	失败重试机制	请求失败的重试机制，默认是没有，不过会使用Ribbon的重试

日志级别：

- NONE：不记录任何日志信息，这是默认值。
- BASIC：仅记录请求的方法，URL以及响应状态码和执行时间
- HEADERS：在BASIC的基础上，额外记录了请求和响应的头信息
- FULL：记录所有请求和响应的明细，包括头信息、请求体、元数据。

一般情况下，默认值就能满足我们使用，如果要自定义时，只需要创建自定义的@Bean覆盖默认Bean即可。

## TypeScript

```
1
2 /**
3  * @author dushaoxiong@lixiang.com
4  * @version 1.0
5  * @date 2022/2/14 18:51
6  */
7 public class FeignClientConfiguration {
8
9     @Bean
10     public Logger.Level feignLogLevel(){
11         return Logger.Level.BASIC;
12     }
13
14 }
```

然后将上方的配置类设置到启动类的注解EnableFeignClients属性中即可。

### 1.3 Feign使用优化

Feign底层发起http请求，依赖于其它的框架。其底层客户端实现包括：

- URLConnection：默认实现，不支持连接池
- Apache HttpClient：支持连接池
- OKHttp：支持连接池

因此提高Feign的性能主要手段就是使用**连接池**代替默认的URLConnection。

这里我们用Apache的HttpClient来演示。

#### 1) 引入依赖

在order-service的pom文件中引入Apache的HttpClient依赖：

#### XML

```
1 <!--httpClient的依赖 -->
2 <dependency>
3     <groupId>io.github.openfeign</groupId>
4     <artifactId>feign-httpclient</artifactId>
5 </dependency>
```

#### 2) 配置连接池

在application.yml中添加配置：

## YAML

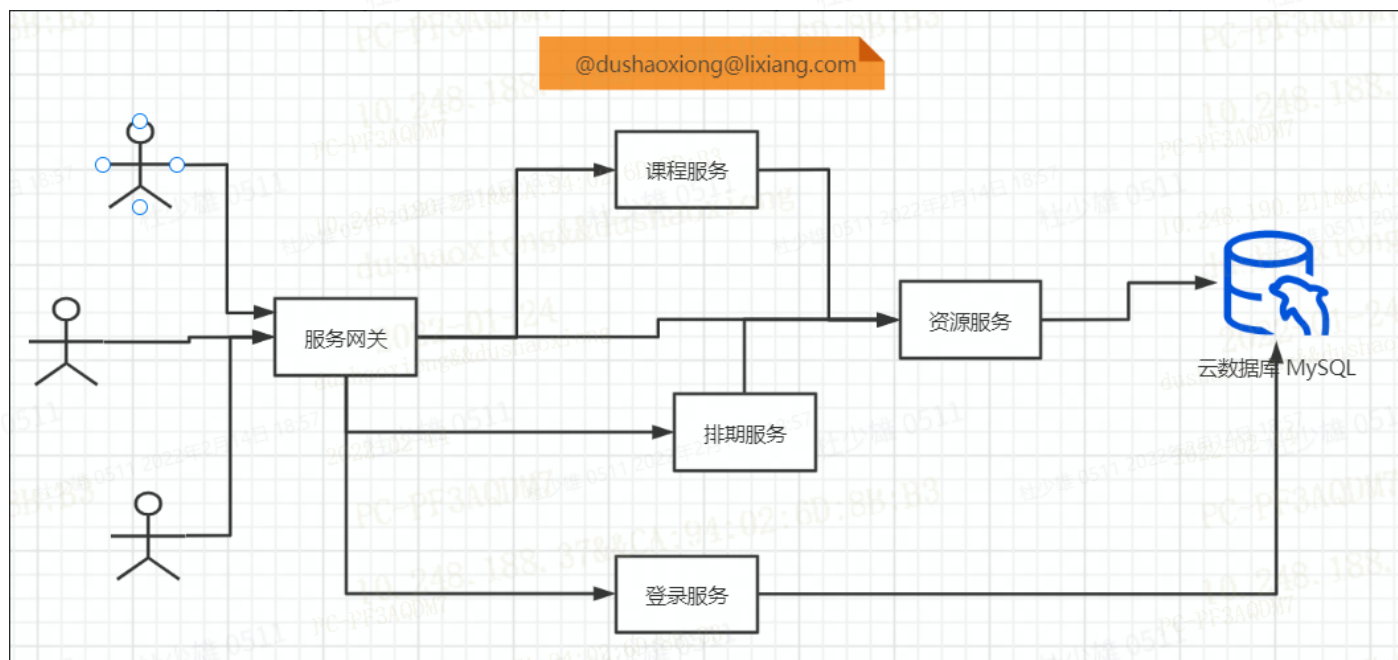
```
1 feign:
2   client:
3     config:
4       default: # default全局的配置
5       loggerLevel: BASIC # 日志级别, BASIC就是基本的请求和响应信息
6   httpclient:
7     enabled: true # 开启feign对HttpClient的支持
8     max-connections: 200 # 最大的连接数
9     max-connections-per-route: 50 # 每个路径的最大连接数
```

3) 这样通过SpringBoot的自动配置, 即可设置地产连接池为HttpClient。

## 四、服务网关

### 1. 服务网关简介

服务网关是我们服务的守门神, 所有微服务的统一入口。



核心功能包括

- **权限控制**: 网关作为微服务入口, 需要校验用户是否有请求资格, 如果没有则进行拦截。
- **路由和负载均衡**: 一切请求都必须先经过这里, 但网关不处理业务, 而是根据某种规则, 把请求转发到某个微服务, 这个过程叫做路由。当然路由的目标服务有多个时, 还需要做负载均衡。
- **限流**: 当请求流量过高时, 在网关中按照下流的微服务能够接受的速度来放行请求, 避免服务压力过大。

在SpringCloud中网关的实现包括两种:

- gateway
- zuul

Zuul是基于Servlet的实现，属于阻塞式编程。而SpringCloudGateway则是基于Spring5中提供的WebFlux，属于响应式编程的实现，具备更好的性能。

## 1.2 SpringCloud Gateway

最后更新时间： 2022年2月14日 19点13分