服务网关

## 一、网关介绍

API网关出现的原因是微服务架构的出现，不同的微服务一般会有不同的网络地址，而外部客户端可能需要调用多个服务的接口才能完成一个业务需求，如果让客户端直接与各个微服务通信，会有以下的问题：

（1）客户端会多次请求不同的微服务，增加了客户端的复杂性。

（2）存在跨域请求，在一定场景下处理相对复杂。

（3）认证复杂，每个服务都需要独立认证。

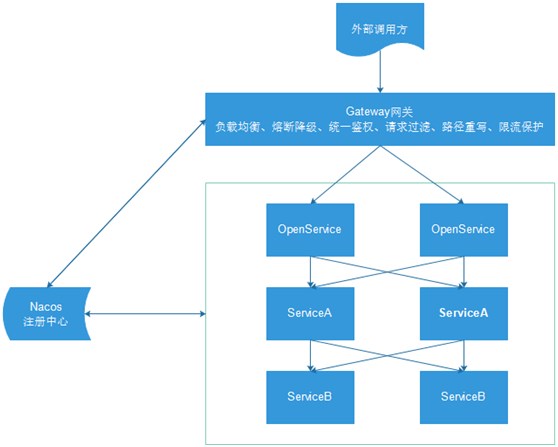
（4）难以重构，随着项目的迭代，可能需要重新划分微服务。例如，可能将多个服务合并成一个或者将一个服务拆分成多个。如果客户端直接与微服务通信，那么重构将会很难实施。

（5）某些微服务可能使用了防火墙 / 浏览器不友好的协议，直接访问会有一定的困难。

以上这些问题可以借助 API 网关解决。API 网关是介于客户端和服务器端之间的中间层，所有的外部请求都会先经过API 网关这一层。也就是说，API 的实现方面更多的考虑业务逻辑，而安全、性能、监控可以交由 API 网关来做，这样既提高业务灵活性又不缺安全性。

## 二、Spring Cloud Gateway介绍

Spring cloud gateway是spring官方基于Spring 5.0、Spring Boot2.0和Project Reactor等技术开发的网关，Spring Cloud Gateway旨在为微服务架构提供简单、有效和统一的API路由管理方式，Spring Cloud Gateway作为Spring Cloud生态系统中的网关，目标是替代Netflix Zuul，其不仅提供统一的路由方式，并且还基于Filer链的方式提供了网关基本的功能，例如：安全、监控/埋点、限流等。



## 三、使用

### 1、创建service\_gateway模块

在总的父模块中，搭建service\_gateway模块，并修改pom.xml文件：

1. **<dependencies>**
2. **<dependency>**
3. **<groupId>**org.springframework.cloud**</groupId>**
4. **<artifactId>**spring-cloud-starter-gateway**</artifactId>**
5. **</dependency>**
7. <!-- 服务注册 -->
8. **<dependency>**
9. **<groupId>**com.alibaba.cloud**</groupId>**
10. **<artifactId>**spring-cloud-starter-alibaba-nacos-discovery**</artifactId>**
11. **</dependency>**
13. **</dependencies>**

### 2、在resources下添加配置文件

application.properties：

1. # 服务端口
2. server.port=80
3. # 服务名
4. spring.application.name=service-gateway
6. # nacos服务地址
7. spring.cloud.nacos.discovery.server-addr=localhost:8848
9. #使用服务发现路由
10. spring.cloud.gateway.discovery.locator.enabled=true
12. #设置路由id
13. spring.cloud.gateway.routes[0].id=service-hosp
14. #设置路由的uri
15. spring.cloud.gateway.routes[0].uri=lb://service-hosp
16. #设置路由断言,代理servicerId为auth-service的/auth/路径
17. spring.cloud.gateway.routes[0].predicates= Path=/\*/hosp/\*\*
19. #设置路由id
20. spring.cloud.gateway.routes[1].id=service-cmn
21. #设置路由的uri
22. spring.cloud.gateway.routes[1].uri=lb://service-cmn
23. #设置路由断言,代理servicerId为auth-service的/auth/路径
24. spring.cloud.gateway.routes[1].predicates= Path=/\*/cmn/\*\*

### 3、添加启动类

1. **package** com.xlh.yygh.gateway;
3. **import** org.springframework.boot.SpringApplication;
4. **import** org.springframework.boot.autoconfigure.SpringBootApplication;
6. @SpringBootApplication
7. **public** **class** ServerGatewayApplication {
8. **public** **static** **void** main(String[] args) {
9. SpringApplication.run(ServerGatewayApplication.**class**, args);
10. }
11. }

### 4、跨域处理

#### 4.1 为什么有跨域问题？

跨域不一定都会有跨域问题。因为跨域问题是浏览器对于ajax请求的一种安全限制：一个页面发起的ajax请求，只能是与当前页域名相同的路径，这能有效的阻止跨站攻击。

因此：跨域问题是针对ajax的一种限制。但是这却给我们的开发带来了不便，而且在实际生产环境中，肯定会有很多台服务器之间交互，地址和端口都可能不同。

#### 4.2 解决跨域问题

**全局配置类实现**

1. **package** com.xlh.yygh.gateway.config;
3. **import** org.springframework.context.annotation.Bean;
4. **import** org.springframework.context.annotation.Configuration;
5. **import** org.springframework.web.cors.CorsConfiguration;
6. **import** org.springframework.web.cors.reactive.CorsWebFilter;
7. **import** org.springframework.web.cors.reactive.UrlBasedCorsConfigurationSource;
8. **import** org.springframework.web.util.pattern.PathPatternParser;
10. //处理跨域
11. @Configuration
12. **public** **class** CrossConfig {
14. @Bean
15. **public** CorsWebFilter corsFilter() {
16. CorsConfiguration config = **new** CorsConfiguration();
17. config.addAllowedMethod("\*");
18. config.addAllowedOrigin("\*");
19. config.addAllowedHeader("\*");
21. UrlBasedCorsConfigurationSource source = **new** UrlBasedCorsConfigurationSource(**new** PathPatternParser());
22. source.registerCorsConfiguration("/\*\*", config);
24. **return** **new** CorsWebFilter(source);
25. }
27. }

**目前我们已经在网关做了跨域处理，那么service服务就不需要再做跨域处理了，将之前在controller类上添加过@CrossOrigin标签的去掉，防止程序异常。**