微服务——Gateway

作者：strangebob
原文链接：https://ld246.com/article/1682235631653
来源网站：链滴

今天来学习Gateway服务网关

1.Gateway服务网关

Spring Cloud Gateway 是 Spring Cloud 的一个全新项目，该项目是基于 Spring 5.0，Spring Boot 2.0 和 Project Reactor 等响应式编程和事件流技术开发的网关，它旨在为微服务架构提供一种简单有效的统一的 API 路由管理方式。

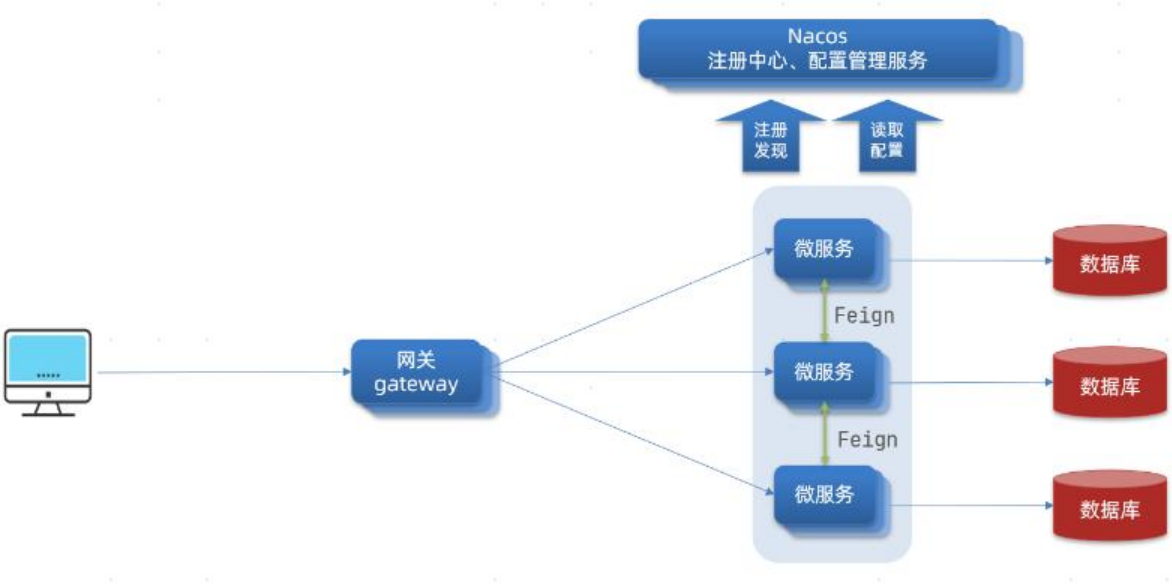
1.1.为什么需要网关

Gateway网关是我们服务的守门神，所有微服务的统一入口。

网关的**核心功能特性**:

- **请求路由**
- **权限控制**
- **限流**

架构图:



权限控制：网关作为微服务入口，需要校验用户是是否有请求资格，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gateway，但网关不处理业务，而是根据某种规则，把请求发到某个微服务，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。

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ebFlux，属于响应式编程的实现，具备更好的性能。

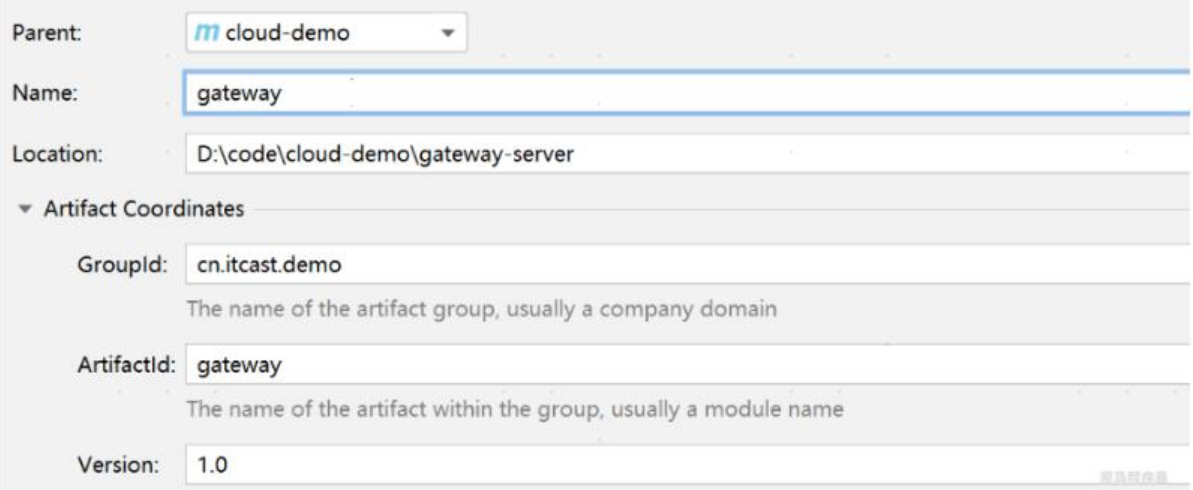
1.2.gateway快速入门

下面，我们就演示下网关的基本路由功能。基本步骤如下：

1. **创建SpringBoot工程gateway，引入网关依赖**
2. **编写启动类**
3. **编写基础配置和路由规则**
4. **启动网关服务进行测试**

1）创建gateway服务，引入依赖

创建服务：



引入依赖：

```
<!--网关-->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
<!--nacos服务发现依赖-->
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
```

2）编写启动类

```
package cn.itcast.gateway;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

```

```
@SpringBootApplication
public class GatewayApplication {

public static void main(String[] args) {
SpringApplication.run(GatewayApplication.class, args);
}
}
```

3）编写基础配置和路由规则

创建application.yml文件，内容如下：

```
server:
  port: 10010 # 网关端口
spring:
  application:
    name: gateway # 服务名称
  cloud:
    nacos:
      server-addr: localhost:8848 # nacos地址
    gateway:
      routes: # 网关路由配置
        - id: user-service # 路由id，自定义，只要唯一即可
          # uri: http://127.0.0.1:8081 # 路由的目标地址 http就是固定地址
          uri: lb://userservice # 路由的目标地址 lb就是负载均衡，后面跟服务名称
          predicates: # 路由断言，也就是判断请求是否符合路由规则的条件
            - Path=/user/** # 这个是按照路径匹配，只要以/user/开头就符合要求
```

我们将符合Path 规则的一切请求，都代理到 uri参数指定的地址。

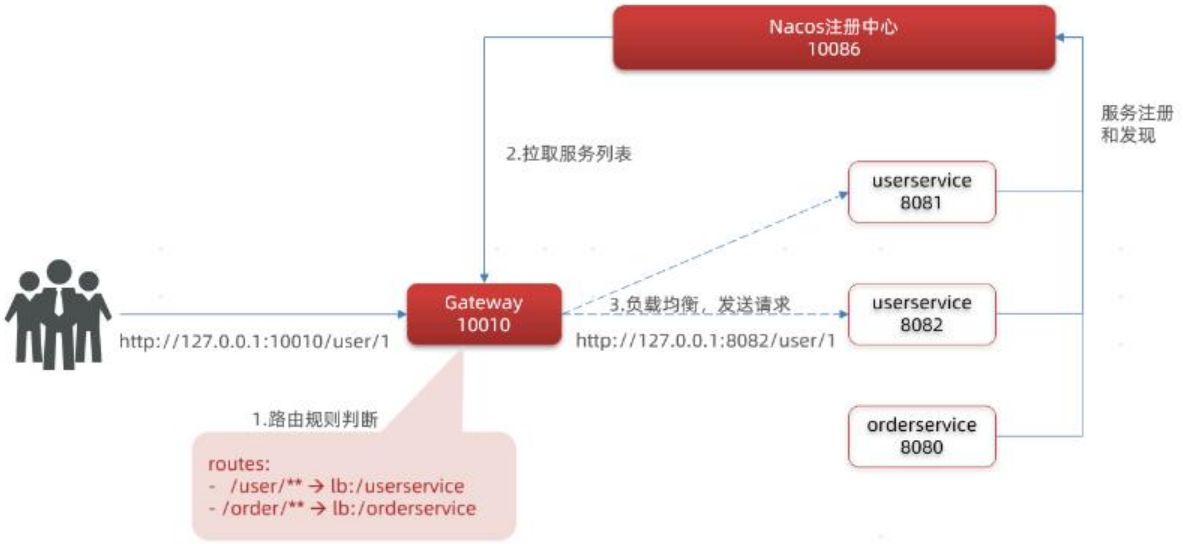
本例中，我们将 **/user/开头的请求，代理到lb://userservice，lb是负载均衡，根据服务名拉取务列表，实现负载均衡。

4）重启测试

重启网关，访问http://localhost:10010/user/1时，符合/user/**规则，请求转发到uri：http://users rvice/user/1，得到了结果：

5）网关路由的流程图

整个访问的流程如下：



总结：

网关搭建步骤：

1. **创建项目，引入nacos服务发现和gateway依赖**
2. **配置application.yml，包括服务基本信息、nacos地址、路由**

路由配置包括：

1. **路由id：路由的唯一标示**
2. **路由目标（uri）：路由的目标地址，http代表固定地址，lb代表根据服务名负载均衡**
3. **路由断言（predicates）：判断路由的规则，**
4. **路由过滤器（filters）：对请求或响应做处理**

接下来，就重点来学习路由断言和路由过滤器的详细知识

1.3.断言工厂

我们在配置文件中写的断言规则只是字符串，这些字符串会被Predicate Factory读取并处理，转变路由判断的条件

例如Path=/user/是按照路径匹配，这个规则是由

org.springframework.cloud.gateway.handler.predicate.PathRoutePredicateFactory类来

处理的，像这样的断言工厂在SpringCloudGateway还有十几个:

名称	说明	示例
After	是某个时间点后的请求	-fter=2037-01-20T17:42:47.789-07:00[America/Denver]
Before	是某个时间点之前的请求	Before=2031-04-13T15:14:47.433+08:00[Asia/Shanghai]

Between 是某两个时间点之前的请求
Between=2037-01-20T17:42:47.789-07:00[America/Denver]，2037-01-21T17:42:47.789-07:00[America/Denver]

Cookie 请求必须包含某些cookie
Cookie=chocolate, ch.p

Header 请求必须包含某些header
Header=X-Request-Id, \d+

Host 请求必须是访问某个host（域名）
Host=**.somehost.org,**.anotherhost.org

Method 请求方式必须是指定方式
Method=GET,POST

Path 请求路径必须符合指定规则
Path=/red/{segment},/blue/**

Query 请求参数必须包含指定参数
*- Query=name, Jack或者- Query=name**

RemoteAddr 请求者的ip必须是指定范围
RemoteAddr=192.168.1.1/24

Weight 权重处理

我们只需要掌握Path这种路由工程就可以了。

1.4.过滤器工厂

GatewayFilter是网关中提供的一种过滤器，可以对进入网关的请求和微服务返回的响应做处理：

1.4.1.路由过滤器的种类

Spring提供了31种不同的路由过滤器工厂。例如：

名称	说明
AddRequestHeader	给当前请求添加一个请求头
RemoveRequestHeader	移除请求中的一个请头
AddResponseHeader	给响应结果中添加一个应头

RemoveResponseHeader	从响应结果中移除有个响应头
RequestRateLimiter	限制请求的流量

1.4.2.请求头过滤器

下面我们以AddRequestHeader 为例来讲解。

需求：给所有进入userservice的请求添加一个请求头：Truth=itcast is freaking awesome!

只需要修改gateway服务的application.yml文件，添加路由过滤即可：

```
spring:
  cloud:
    gateway:
      routes:
      - id: user-service
        uri: lb://userservice
        predicates:
        - Path=/user/**
        filters: # 过滤器
        - AddRequestHeader=Truth, Itcast is freaking awesome! # 添加请求头
```

当前过滤器写在userservice路由下，因此仅仅对访问userservice的请求有效。

1.4.3.默认过滤器

如果要对所有的路由都生效，则可以将过滤器工厂写到default下。格式如下：

```
spring:
  cloud:
    gateway:
      routes:
      - id: user-service
        uri: lb://userservice
        predicates:
        - Path=/user/**
      default-filters: # 默认过滤项
      - AddRequestHeader=Truth, Itcast is freaking awesome!
```

1.4.4.总结

过滤器的作用是什么?

① 对路由的请求或响应做加工处理，比如添加请求头

② 配置在路由下的过滤器只对当前路由的请求生效

defaultFilters的作用是什么?

① 对所有路由都生效的过滤器

1.5.全局过滤器

上一节学习的过滤器，网关提供了31种，但每一种过滤器的作用都是固定的。如果我们希望拦截请求做自己的业务逻辑则没办法实现。

1.5.1.全局过滤器作用

全局过滤器的作用也是处理一切进入网关的请求和微服务响应，与GatewayFilter的作用一样。区别于GatewayFilter通过配置定义，处理逻辑是固定的；而GlobalFilter的逻辑需要自己写代码实现。

定义方式是实现GlobalFilter接口。

```
public interface GlobalFilter {
    /**
     *  处理当前请求，有必要的话通过{@link GatewayFilterChain}将请求交给下一个过滤器处理
     *
     * @param exchange 请求上下文，里面可以获取Request、Response等信息
     * @param chain 用来把请求委托给下一个过滤器
     * @return {@code Mono<Void>} 返回标示当前过滤器业务结束
     */
    Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain);
}
```

在filter中编写自定义逻辑，可以实现下列功能：

- **登录状态判断**
- **权限校验**
- **请求限流等**

1.5.2.自定义全局过滤器

需求：定义全局过滤器，拦截请求，判断请求的参数是否满足下面条件：

- **参数中是否有authorization，**
- **authorization参数值是否为admin**

如果同时满足则放行，否则拦截

实现：

在gateway中定义一个过滤器：

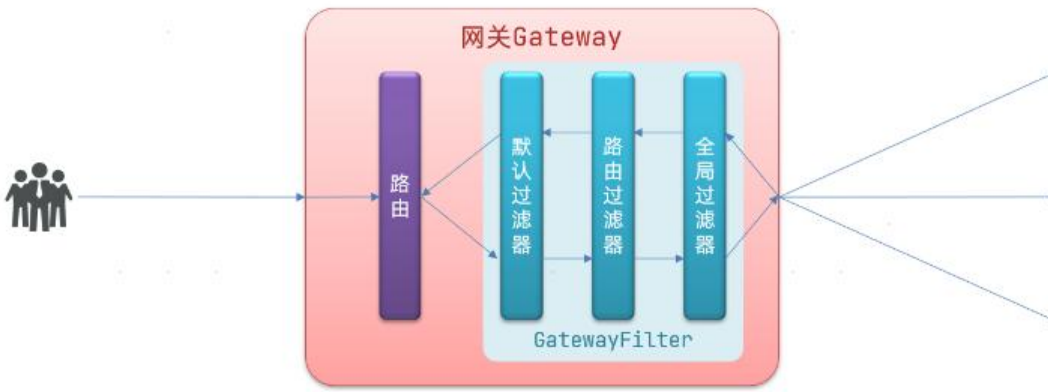
```
package cn.itcast.gateway.filters;

import org.springframework.cloud.gateway.filter.GatewayFilterChain;
import org.springframework.cloud.gateway.filter.GlobalFilter;
import org.springframework.core.annotation.Order;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Component;
import org.springframework.web.server.ServerWebExchange;
import reactor.core.publisher.Mono;
```

```
@Order(-1)
@Component
public class AuthorizeFilter implements GlobalFilter {
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {
        // 1.获取请求参数
        MultiValueMap<String, String> params = exchange.getRequest().getQueryParams();
        // 2.获取authorization参数
        String auth = params.getFirst("authorization");
        // 3.校验
        if ("admin".equals(auth)) {
            // 放行
            return chain.filter(exchange);
        }
        // 4.拦截
        // 4.1.禁止访问，设置状态码
        exchange.getResponse().setStatusCode(HttpStatus.FORBIDDEN);
        // 4.2.结束处理
        return exchange.getResponse().setComplete();
    }
}
```

1.5.3.过滤器执行顺序

请求进入网关会碰到三类过滤器：当前路由的过滤器、DefaultFilter、GlobalFilter

请求路由后，会将当前路由过滤器和DefaultFilter、GlobalFilter，合并到一个过滤器链（集合）中排序后依次执行每个过滤器：



排序的规则是什么呢?

- **每一个过滤器都必须指定一个int类型的order值，** **order值越小，优先级越高，执行顺序越靠前。**
- **GlobalFilter通过实现Ordered接口，或者添加@Order注解来指定order值，由我们自己指定**
- **路由过滤器和defaultFilter的order由Spring指定，默认是按照声明顺序从1递增。**
- **当过滤器的order值一样时，会按照 defaultFilter > 路由过滤器 > GlobalFilter的顺序执行。**

详细内容，可以查看源码：

org.springframework.cloud.gateway.route.RouteDefinitionRouteLocator#getFilters()方法是先载defaultFilters，然后再加载某个route的filters，然后合并。

org.springframework.cloud.gateway.handler.FilteringWebHandler#handle()方法会加载全局过器，与前面的过滤器合并后根据order排序，组织过滤器链

1.6.跨域问题

1.6.1.什么是跨域问题

跨域：域名不一致就是跨域，主要包括：

- **域名不同： ** www.taobao.com 和 www.taobao.org 和 www.jd.com 和 miaosha.jd.com
- **域名相同，端口不同：localhost:8080和localhost8081**

跨域问题：浏览器禁止请求的发起者与服务端发生跨域ajax请求，请求被浏览器拦截的问题

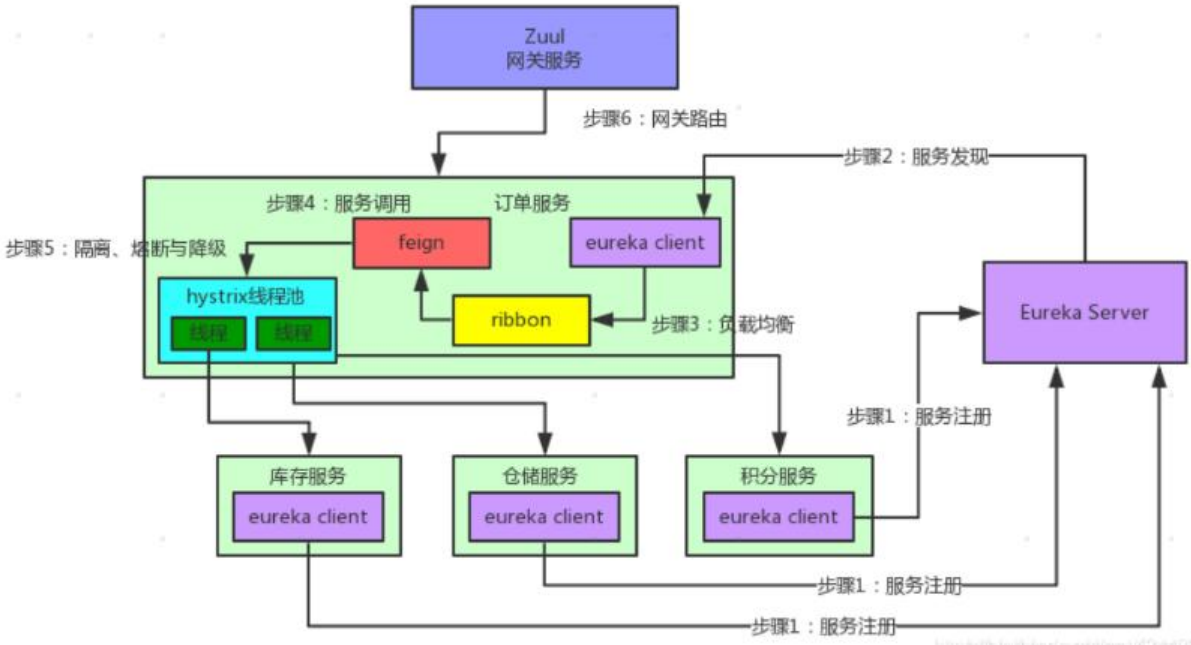
解决方案：CORS，这个以前应该学习过，这里不再赘述了。不知道的小伙伴可以查看https://www.ranyifeng.com/blog/2016/04/cors.html

1.6.2.解决跨域问题

在gateway服务的application.yml文件中，添加下面的配置：

```yaml
spring:
  cloud:
    gateway:
      # 。。。
      globalcors: # 全局的跨域处理
        add-to-simple-url-handler-mapping: true # 解决options请求被拦截问题
        corsConfigurations:
          '[/**]':
            allowedOrigins: # 允许哪些网站的跨域请求
              - "http://localhost:8090"
            allowedMethods: # 允许的跨域ajax的请求方式
              - "GET"
              - "POST"
              - "DELETE"
              - "PUT"
              - "OPTIONS"
            allowedHeaders: "*" # 允许在请求中携带的头信息
            allowCredentials: true # 是否允许携带cookie
            maxAge: 360000 # 这次跨域检测的有效期
```

2.总结



学到这，springCloud五大核心组件基础知识就差不多学习完毕了，还差一个**Hystrix**,留到后面再补吧，下面开始系统的学习一下docker。

原文链接：微服务——Gateway