# 第 5 章

# 网 关 组 件

**本章主要内容：**

- 网关概述
- 网关快速入门
- 路由转发中的负载均衡
- 过滤器
- 网关中配置 Cors 跨域
- 灰度发布

网关可以作为所有 API 服务请求的接入点，同时可以作为所有后端业务服务的聚合点，通过网关可以实现安全、验证、路由、过滤、流控等策略，并对所有 API 服务和策略进行统一管理。本章重点讲述网关的路由转发功能、过滤器、跨域配置和灰度发布。

## 5.1 网关组件概述

在 Spring Cloud 体系架构中，需要部署一个单独的网关服务对外提供访问入口，然后网关服务根据配置好的规则将请求转发至具体的后端服务。这个任务由 Spring Cloud 的网关组件实现。Spring Cloud Gateway 是 Spring Cloud 的全新子项目，该项目意在提供简单方便、可扩展的统一 API 路由管理方式。

网关效果图如图 5-1 所示。

Gateway 作为网关组件，主要有以下作用。

（1）统一入口：为所有微服务提供一个唯一的入口，网关起到外部和内部隔离的作用，保障了后台服务的安全性。

（2）鉴权校验：识别每个请求的权限，拒绝不符合要求的请求。

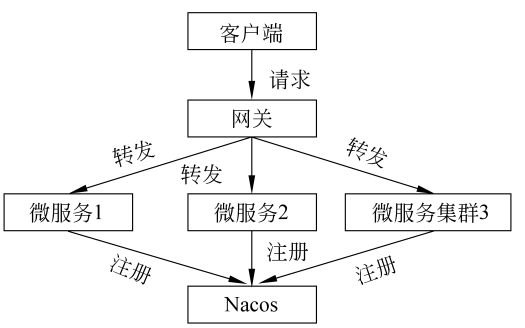


图 5-1 网关效果图

（3）动态路由：动态地将请求路由到不同的后端集群中。

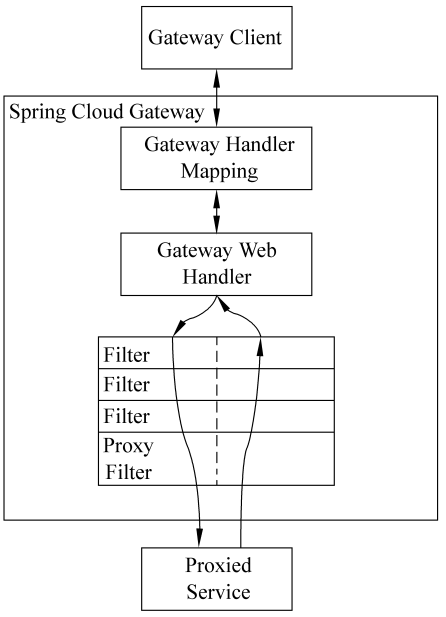


图 5-2　网关的工作流程

（4）减少客户端与服务器端的耦合：服务可以独立发展，通过网关层进行映射。

网关的工作流程如图 5-2 所示。客户端向 Spring Cloud Gateway 发出请求。如果在 Gateway Handler Mapping 中找到与请求相匹配的路由，则将其发送到 Gateway Web Handler。Handler 再通过指定的过滤器链来将请求发送到实际的服务，以便执行业务逻辑。

Gateway 中的一些基本概念。

**1. Route（路由）**

路由是网关的基本单元，由 ID、URI、一组 Predicate、一组 Filter 组成，最基本的功能是只要满足 Predicate 中的条件就路由转发到 URI 中。

**2. Predicate（断言）**

用来匹配来自 HTTP 请求的任何内容，作为路由转发的判断条件，包括 Path、Query、Method、Header 等。

**3. Filter（过滤器）**

过滤器是路由转发请求时所经过的过滤逻辑，可用于修改请求和响应。

## 5.2　网关组件快速入门

本节通过实例来讲解网关的基本路由转发功能。

### 5.2.1　准备微服务项目

准备一个 userservice 项目，复制第 4 章的 userservice 项目，再准备一个 orderservice 项目，复制第 4 章的 orderservice 项目，将端口修改为 8082。下面一步为这两个微服务项目创建统一网关及实现路由转发等功能。

### 5.2.2　创建网关项目实现简单路由功能

创建 Spring Boot 3.0.2 项目，命名为 gateway，添加网关应用程序入口类，代码如下：

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
```

```
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
<dependency>
    <groupId>com.alibaba.cloud</groupId>
    <artifactId>spring-cloud-starter-alibaba-nacos-discovery</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-loadbalancer</artifactId>
</dependency>
```

application.yml 配置文件中的代码如下：

```
server:
  port: 8080

spring:
  application:
    name: gateway
  cloud:
    nacos:
      discovery:
        server-addr: localhost:8848
    gateway:
      routes:
        -id: user1_route
          uri: http://localhost:8081/user/{id}
          predicates:
            -Path=/user/{id}

        -id: order_route
          uri: http://localhost:8082/order/{id}
          predicates:
            -Path=/order/{id}
```

这里配置的重点是 routes，表示路由，可以配置多条路由，每条路由用 id、uri 和 predicates 进行描述，其中 id 是路由标识，只要使用不重复的字符串即可，uri 表示转发的目标地址，predicates 称为路由断言，即路由转发需要匹配的条件，predicates 下面又有多种，其中 Path 表示路径匹配，即如果客户端发出的请求路径跟这里描述的路径匹配，则转发到 uri 指定的目标地址。例如，如果客户端发出的请求路径是 http://localhost:8080/user/1，跟 predicates 下的 Path＝/user/{id} 匹配，就会将此请求转发到 uri：http://localhost:8081/user/1。

启动 Nacos，启动 userservice 和 orderservice 项目，使用浏览器访问 http://localhost:8080/user/1，结果如图 5-3 所示，显然请求被转发到了 http://localhost:8081/user/1。接

着使用浏览器访问 http://localhost:8080/order/1，结果如图 5-4 所示，显示请求被转发到了 http://localhost:8082/order/1。

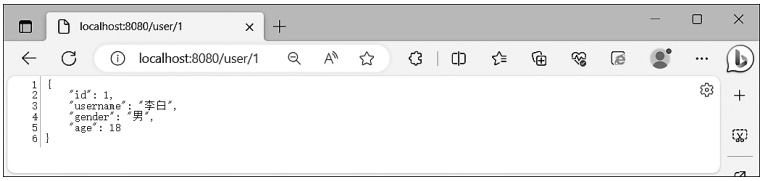


图 5-3　访问 http://localhost:8080/user/1


图 5-4　路由转发

## 5.3　实现路由转发中的负载均衡

如果转发的目标地址是由多个实例组成的微服务集群，则网关还可以通过配置实现负载均衡，网关会先将请求转发到负载均衡器，再由负载均衡器进行负载。业务架构如图 5-5 所示。

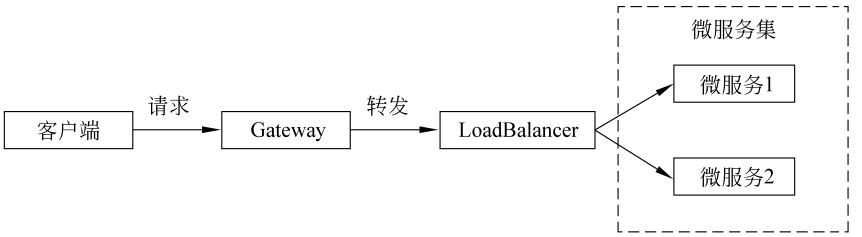


图 5-5　业务架构

下面演示一个负载均衡的实例。

(1) 将第 4 章的 userservice2 复制过来，并把端口修改为 8083。为了区分这两个相同服务名称的微服务，分别在各自的控制器的 findUserById 方法添加代码 System.out.println("userservice:8081")和 System.out.println("userservice:8083")。为了观察控制台的输出日志，两个项目都在 application.yml 文件中添加日志配置，代码如下：

```
logging:
  level:
```

```
    com.sike.controller.UserController: DEBUG
```

（2）打开项目网关，修改 application.yml 配置文件，将 uri 由 http 开头的表示形式改为以 lb 开头的表示形式，并且用服务名代替 localhost:8081，其中 lb 表示负载均衡的意思。具体的代码如下：

```
uri: http://localhost:8081/user/{id}
```

修改的代码如下：

```
uri: lb://userservice/user/{id}
```

（3）重启两个 userservice 项目及网关项目，使用浏览器访问 localhost:8080/user/1，观察发现只有其中一个 userservice 项目控制台有输出，再次访问，会发现另外一个也有输出了。继续多次访问，可以发现会交替访问两个微服务项目，这表示以轮询的方式进行了负载均衡。

## 5.4　过滤器

Spring Cloud Gateway 中的过滤器从接口实现上分为两种，一种是 GatewayFilter，GatewayFilter 又可分为路由过滤器和默认过滤器，前者只针对一个路由进行过滤，后者针对多个路由进行过滤；另一种是 GlobalFilter，称为全局过滤器，针对所有路由进行过滤。如果要配置过滤器，则首先要学习路由断言。

### 5.4.1　路由断言

路由断言表示路由转发的条件，当条件匹配后才会被转发，可以有多个条件，如果有多个条件，则在多个条件都匹配的情况下才会被转发。常见的条件如表 5-1 所示。

**表 5-1　路由断言的条件**

| 匹配方式 | 说　明 | 样　例 |
| --- | --- | --- |
| Before | 某个时间点之前 | Before=2023-06-28T00:00:00+08:00[Asia/Shanghai] |
| After | 某个时间点之后 | After=2023-06-28T00:00:00+08:00[Asia/Shanghai] |
| Between | Before +After | Between=2023-06-28T00:00:00+08:00[Asia/Shanghai], 2023-07-01T00:00:00+08:00[Asia/Shanghai] |
| Cookie | Cookie 值 | Cookie=sike, admin |
| Header | Header 值 | Header=X-Request-Id, \d+ |
| Host | 主机名 | Host=**.sike.com |

续表

| 匹配方式 | 说　明 | 样　例 |
| --- | --- | --- |
| Method | 请求方式 | Method=GET |
| Query | 请求参数 | Query=aaa，bbb |
| Path | 请求路径 | Path=/user/{userId} |
| RemoteAddr | 请求 IP | RemoteAddr=192.168.0.8/24 |
| Weight | 权重 | Weight=groupName1，9 |

示例代码如下：

```
predicates:
  - Path=/user/{id}
  - Method=GET
```

这表示以 GET 方式访问 http://localhost:8080/user/{id}才会被转发，如果以 POST 方式访问，则不转发，示例代码如下：

```
predicates:
  - Path=/user/{id}
  - Before=2024-06-28T00:00:00+08:00[Asia/Shanghai]
```

表示 2024 年 6 月 28 日前访问上述地址就会被转发，如果在设定的时间之后访问，则不会被转发。如果将 Before 修改为 After，则相反，示例代码如下：

```
predicates:
  - Path=/user/{id}
  - RemoteAddr=192.168.1.34
```

表示只接受来自 RemoteAddr 中指定的 IP 地址的请求。

### 5.4.2　路由过滤器

Spring Cloud Gateway 内置了多种路由过滤器，它们都由 GatewayFilter 接口的工厂类来产生。路由过滤器可用于修改进入的 HTTP 请求和返回的 HTTP 响应，例如可以给请求添加一个参数，或添加一个请求头。下面通过案例学习常用路由过滤器的用法。

（1）修改网关项目，对 application.yml 配置文件中的第 1 个路由的配置进行修改，代码如下：

```
routes:
  - id: user1_route
    uri: http://localhost:8081/user/{id}
    predicates:
```

```
        - Path=/user/{id}
      filters:
        - AddRequestParameter=username,admin
```

可见这条路由添加了一个 filters 的配置，表示给这个路由添加一个路由过滤器，其中的关键字 AddRequestParameter 表示添加一个请求参数，键是 username，值是 admin，这相当于原来的请求路径 http://localhost:8080/user/1 变成了 http://localhost:8080/user/1?username=admin。

(2) 修改 userservice 项目的控制器的 findUserById 方法，以便接收和处理这个参数，修改后的控制器中的代码如下：

```
@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/{id}")
    public User findUserById(@PathVariable Integer id, String username){
        System.out.println("userservice 项目中获取的 username:"+username);
        return userService.getById(id);
    }
}
```

(3) 重启有关项目，使用浏览器访问 http://localhost:8080/user/1，如果控制台输出的内容为“userservice 项目中获取的 username:admin”，就表示过滤器执行成功。

### 5.4.3　路由过滤器工厂

Spring Cloud Gateway 包含许多内置的 GatewayFilter 工厂。上述配置中的 AddRequestParameter 就是其中的一种，更多的内置过滤器工厂及其作用如表 5-2 所示。

**表 5-2　内置过滤器工厂及其作用**

| 过滤器工厂 | 作　　用 | 参　　数 |
| --- | --- | --- |
| AddRequestHeader | 为原始请求添加 Header | Header 的名称及值 |
| AddRequestParameter | 为原始请求添加请求参数 | 参数名称及值 |
| AddResponseHeader | 为原始响应添加 Header | Header 的名称及值 |
| DedupeResponseHeader | 剔除响应头中重复的值 | 需要去重的 Header 名称及去重策略 |
| Hystrix | 为路由引入 Hystrix 的断路器保护 | HystrixCommand 的名称 |

续表

| 过滤器工厂 | 作　　用 | 参　　数 |
| --- | --- | --- |
| FallbackHeaders | 为 fallbackUri 的请求头中添加具体的异常信息 | Header 的名称 |
| PrefixPath | 为原始请求路径添加前缀 | 前缀路径 |
| PreserveHostHeader | 为请求添加一个 preserveHostHeader＝true 的属性，路由过滤器会检查该属性以决定是否要发送原始的 Host | 无 |
| RequestRateLimiter | 用于对请求限流，限流算法为令牌桶 | keyResolver 、rateLimiter、statusCode、denyEmptyKey、emptyKeyStatus |
| RedirectTo | 将原始请求重定向到指定的 URL | HTTP 状态码及重定向的 URL |
| RemoveHopByHop-HeadersFilter | 为原始请求删除 IETF 组织规定的一系列 Header | 默认就会启用，可以通过配置指定仅删除哪些 Header |
| RemoveReguestHeader | 为原始请求删除某个 Header | Header 名称 |
| RemoveResponseHeader | 为原始响应删除某个 Header | Header 名称 |
| RewritePath | 重写原始的请求路径 | 原始路径正则表达式及重写后路径的正则表达式 |
| RewriteResponseHeader | 重写原始响应中的某个 Header | Header 名称，值的正则表达式，重写后的值 |
| SaveSession | 在转发请求之前，强制执行 WebSession::save 操作 | 无 |
| secureHeaders | 为原始响应添加一系列起安全作用的响应头 | 无，支持修改这些安全响应头的值 |
| SetPath | 修改原始的请求路径 | 修改后的路径 |
| SetResponseHeader | 修改原始响应中某个 Header 的值 | Header 名称，修改后的值 |
| SetStatus | 修改原始响应的状态码 | HTTP 状态码，可以是数字，也可以是字符串 |
| StripPrefix | 用于截断原始请求的路径 | 使用数字表示要截断的路径的数量 |
| Retry | 针对不同的响应进行重试 | retries、statuses、methods、series |
| RequestSize | 设置允许接收最大请求包的大小。如果请求包大小超过设置的值，则返回 413 Payload Too Large | 请求包大小，单位为字节，默认值为 5MB |
| ModifyRequestBody | 在转发请求之前修改原始请求体内容 | 修改后的请求体内容 |
| ModifyResponseBody | 修改原始响应体的内容 | 修改后的响应体内容 |

### 5.4.4　默认过滤器

上面只针对其中一个路由进行过滤器配置，也可以针对多个（一组）路由统一设置一个过滤器，方法是在配置文件的 routes 的同一级别添加 default-filters 配置项，参考代码如下：

```
gateway:
  routes:
    - id: user1_route
      uri: http://localhost:8081/user/{id}
      predicates:
        - Path=/user/{id}#
    - id: order_route
      uri: http://localhost:8082/order/{id}
      predicates:
        - Path=/order/{id}
  default-filters:
    - AddRequestParameter=username,admin
```

这样上述两个路由都同时适用同一个过滤器，两个路由都会添加请求参数。参考 userservice 项目的 findUserById 方法修改 orderservice 项目的控制器中的 findOrderById 方法，以便接收和处理 username 参数的输出。重启有关项目，访问 localhost:8080/order/1，结果 orderservice 项目的控制台输出：orderservice 项目中获取的 username:admin，这表明 orderservice 项目也被添加了过滤器。

### 5.4.5　全局过滤器

GatewayFilter 路由过滤器并不能实现业务逻辑的判断与处理，要实现业务逻辑的判断与处理需要用到全局过滤器 GlobalFilter。GlobalFilter 是一个全局的过滤器，作用于所有的路由。GlobalFilter 过滤器不需要配置，在系统初始化时加载，并作用在每个路由上，最终通过 GatewayFilterAdapter 包装成 GatewayFilterChain 可识别的过滤器。下面为 Gateway 项目添加全局过滤器以实现特定的功能。

（1）在网关项目中创建 GlobalFilter1 类，实现 GlobalFilter 接口，代码如下：

```
@Order(0)
@Component
public class GlobalFilter1 implements GlobalFilter {
    @Override
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain
chain) {
        MultiValueMap<String, String> queryParams =
exchange.getRequest().getQueryParams();
        String username = queryParams.getFirst("username");
```

```
            System.out.println("全局过滤器中的 username:"+username);
            if("admin".equals(username)){
                System.out.println("用户名正确,可以转发");
                return chain.filter(exchange);
            }
            System.out.println("用户名错误,拒绝转发");
            exchange.getResponse().setStatusCode(HttpStatus.UNAUTHORIZED);
            return exchange.getResponse().setComplete();
        }
    }
```

这段代码的意思是获取请求参数 username,判断它的值是不是等于 admin,如果是,则放行,否则拦截,并返回 401 未授权状态码。注解@Order 的作用是如果有多个过滤器,则通过@Order 注解中的数字来决定优先级,数字越小优先级越高。注解@Component 的作用是项目启动时会把 GlobalFilter1 对象放入 Spring 容器中,这样它所定义的全局过滤器就会生效并监听请求。

(2) 在 application.yml 文件中注释掉之前的有关路由过滤器的配置。

(3) 运行测试。使用浏览器访问 http://localhost:8080/user/1,结果如图 5-6 所示,提示 401 未授权错误,并且控制台输出:

```
全局过滤器中的 username:null。
用户名错误,拒绝转发。
```

这是因为被全局过滤器拦截,检查请求参数中并没有 username=admin,所以被拦截。

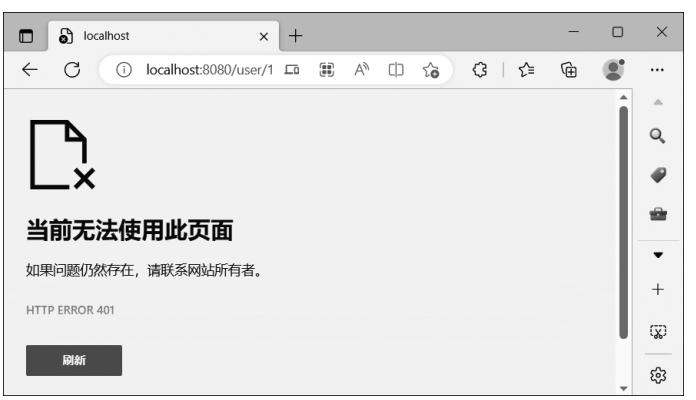


图 5-6 浏览器访问 http://localhost:8080/user/1

在浏览器网址栏添加请求参数?username=admin,即完整的请求路径变为 http://localhost:8080/user/1?username=admin,再次访问,结果如图 5-7 所示,可以正常访问了,并且网址栏输出:

```
全局过滤器中的 username:admin。
用户名正确,可以转发。
```