博学谷
www.boxuegu.com

第六章 Spring Cloud Gateway 服务网关

一样的在线教育，不一样的教学品质

博学谷
www.boxuegu.com

目录 Contents

核心概念与工作流程

博学谷
www.boxuegu.com

小节导学

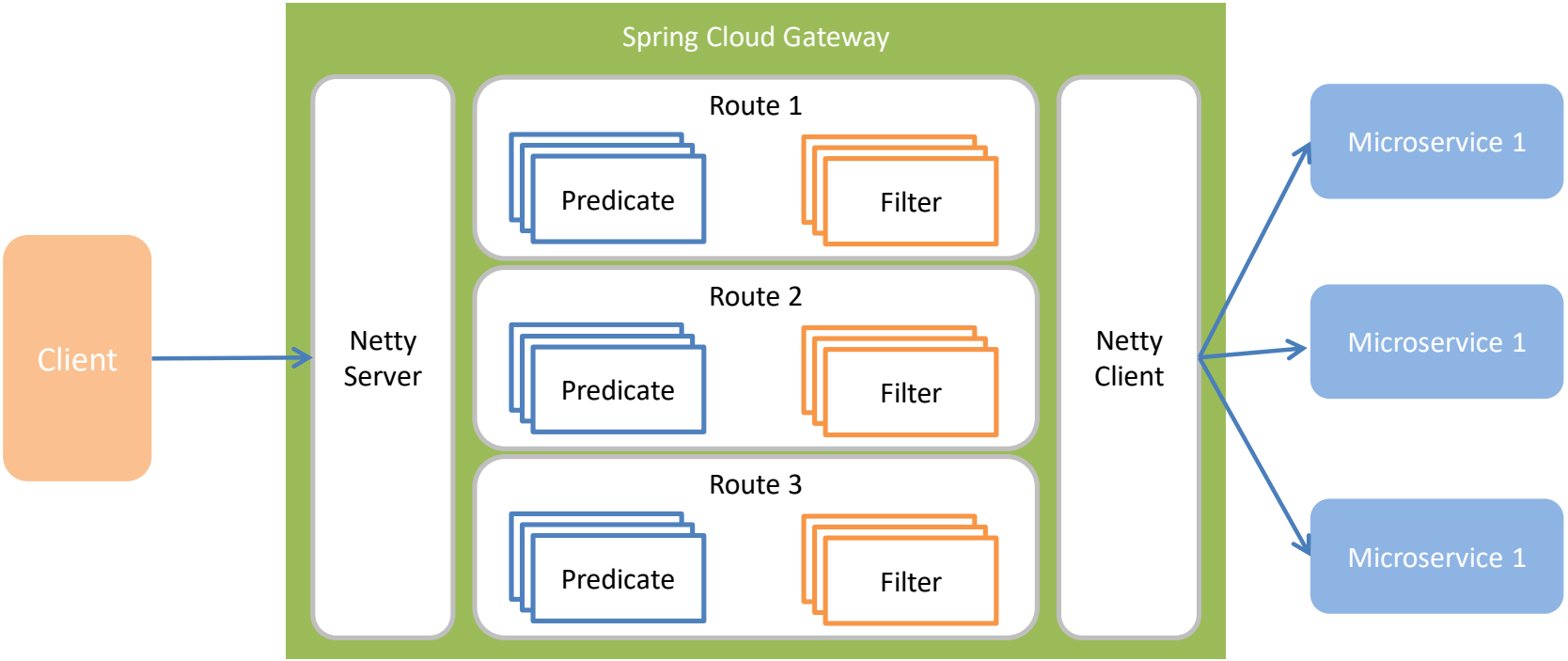
“网关”是整个系统的入口，是**守门人**，所处位置非常关键，我们需要好好认识一下网关，例如：

1. 网关是由哪些部分构成的?
2. “路由”、“过滤器”都是做什么的?
3. 请求从进入网关，到得到响应结果，请求在内部都经历了什么?

本节我们就要弄清楚这些问题，对网关有个清晰的认识。

- 核心概念
- 工作流程

1. 核心概念


Spring Cloud Gateway
Client
Netty Server
Route 1
Predicate
Filter
Route 2
Predicate
Filter
Route 3
Predicate
Filter
Netty Client
Microservice 1
Microservice 1
Microservice 1

1. 核心概念

核心构成

赵四

接 赵四

接 刘能

接 大拿

进入流程

体检

存包

更衣

离开流程

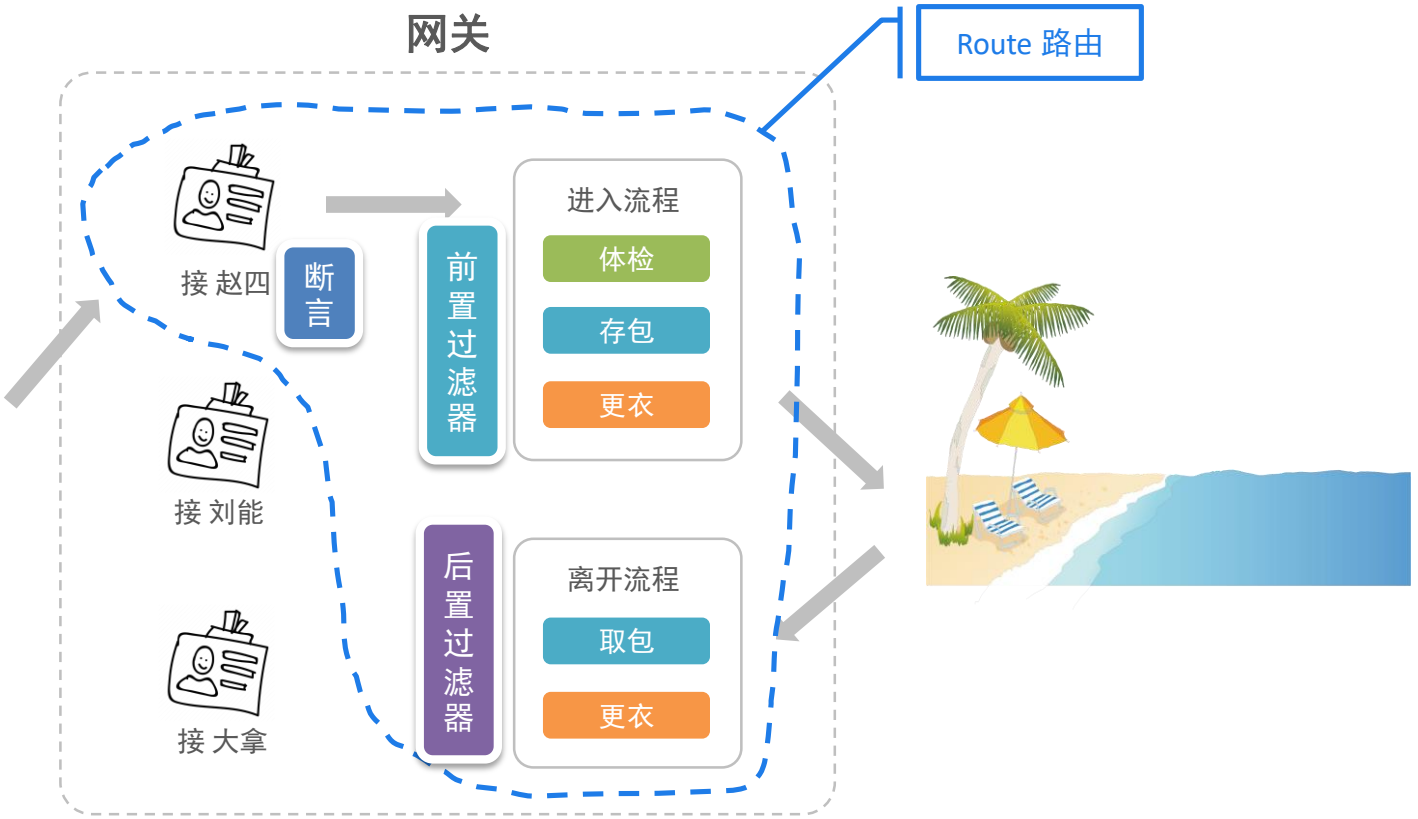
取包

更衣

1. 核心概念

核心构成

赵四



1. 核心概念

博学谷
www.boxuegu.com

核心构成

- Route（路由）

  就像是一个**向导**。Gateway 包含多个 Route，每个 Route 包含：

  1. ID - 路由编号，唯一
  2. URI - 目的地 URI，即请求最终被转发的目的地
  3. Order - 当请求匹配多个路由时，使用顺序小的
  4. Predicate - 一组断言
  5. Filter - 一组过滤器

- Predicate（断言）

  就是**匹配条件**，满足条件才会被 Route 路由到目的地 URI。可以匹配请求中的任何内容，如头或参数。

- Filter（过滤器）

  作用类似**拦截加工**，经过过滤器的请求和响应，都可以进行修改处理。

1. 核心概念

示例

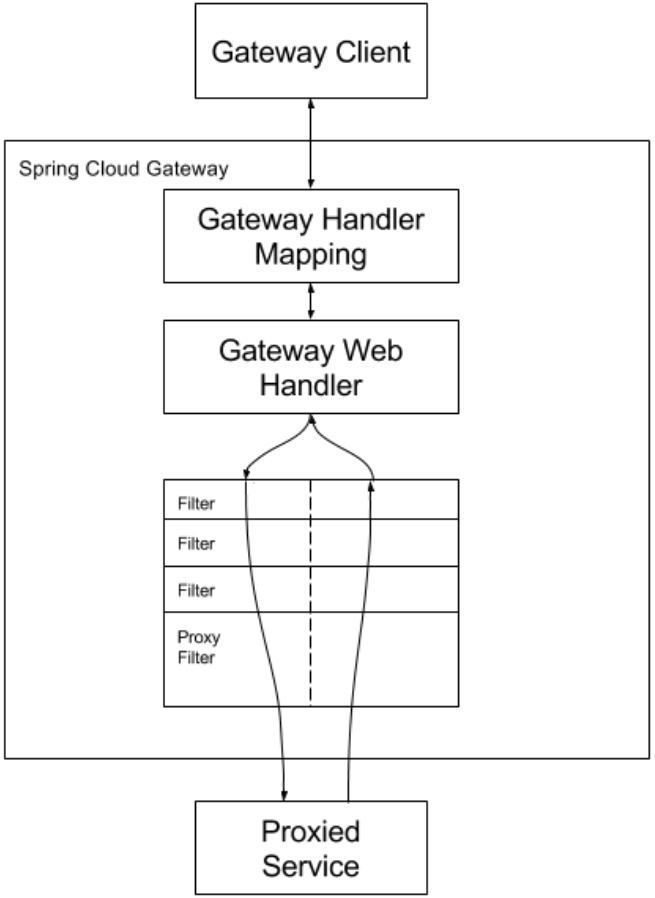
```
spring:
  cloud:
    gateway:
      routes:
        - id: test_route
          uri: http://xxx.com
          predicates:
            - Path=/user/query
          filters:
            - AddRequestHeader=xxx
```

含义

访问路径“/user/query”，就会匹配 “test_route”这个路由。会用“AddRequestHeader”这个过滤器进行处理，然后转发到“http://xxx.com”。

2. 工作流程

博学谷
www.boxuegu.com



核心概念与工作流程-总结

博学谷
www.boxuegu.com

重难点

1. Gateway 中的重要组成部分，各部分的工作职责
2. Gateway 的核心工作流程

博学谷
www.boxuegu.com

目录 Contents

服务路由

博学谷
www.boxuegu.com

小节导学

现在“网关”已经成为整个系统的入口，客户端都是通过 Gateway 来访问服务，那么**具体怎么访问服务呢?**

SpringCloud Gateway 提供了一个非常方便的用法，不需要写任何代码，**根据服务名**就可以自动转发。

本节我们就实践一下这种最基础的 Gateway 用法，以便对整体结构有个大体的认识。

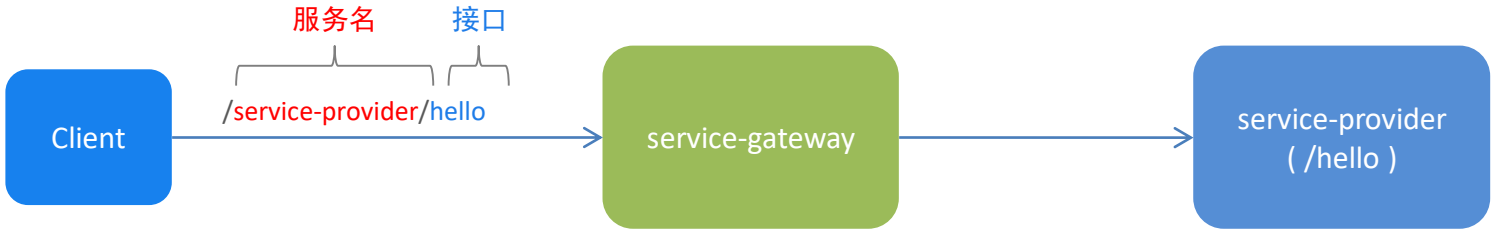
- 实践服务路由

1. 实践服务路由

目标

一个 service-provider，包含一个接口“/hello”

一个 service-gateway，客户端访问 Gateway 的“/hello”，转发到 service-provider 的“/hello”



实践步骤

1. 创建服务 service-provider，整合注册中心 Nacos，并提供一个接口“/hello”
2. 创建服务 service-gateway，整合注册中心 Nacos，与 Gateway
3. 测试 Gateway 路由转发

1. 实践服务路由

博学谷
www.boxuegu.com

实践步骤

1. 创建 service-provider，整合注册中心 Nacos，并提供一个接口“/hello”

Nacos 依赖：

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

Nacos 配置：

```
nacos:
   discovery:
      server-addr: localhost:8848
```

注解：`@EnableDiscoveryClient`

测试接口：

```
@RestController
public class TestController {
    @GetMapping("/hello")
    public String hello() {  return "hello"; }
}
```

1. 实践服务路由

博学谷
www.boxuegu.com

实践步骤

2. 创建 service-gateway，整合注册中心 Nacos，与 Gateway

依赖：

```
<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>
```

1. 实践服务路由

博学谷
www.boxuegu.com

实践步骤

2. 创建 service-gateway，整合注册中心 Nacos，与 Gateway

```
server:
  port: 8081
spring:
  application:
    name: service-gateway
  cloud:
    nacos:
      discovery:
        server-addr: localhost:8848
    gateway:
      discovery:
        locator:
          enabled: true
```

1. 实践服务路由

博学谷
www.boxuegu.com

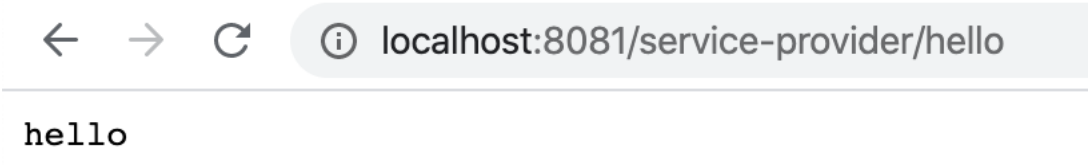
实践步骤

3. 测试 Gateway 路由转发

通过 service-gateway 访问“/hello”

```
http://localhost:8081/service-provider/hello
```

效果：



如果启动多个 service-provider 实例，会自动实现负载均衡

服务路由-总结

博学谷
www.boxuegu.com

重难点

1. 理解接入 Gateway 之后的整体结构

2. 掌握 Gateway 通过服务发现机制自动路由的实现方式

博学谷
www.boxuegu.com

目录 Contents

内置路由断言

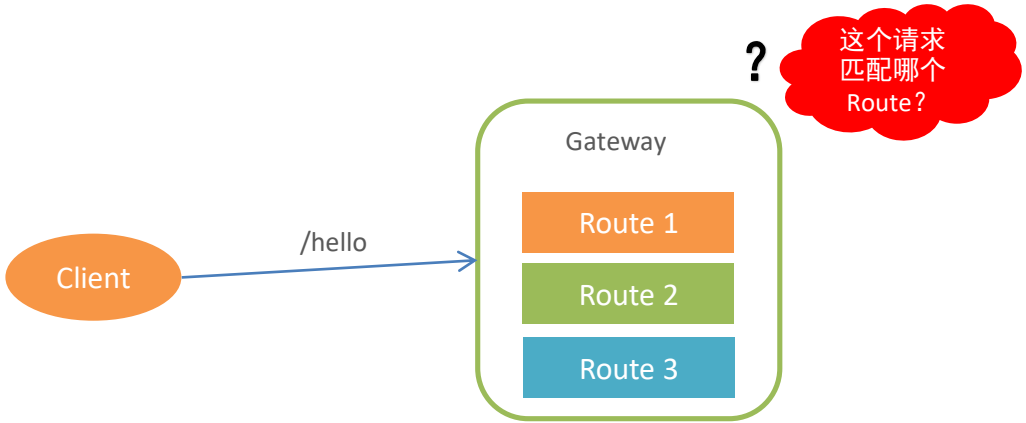
小节导学

前面介绍 Gateway 核心概念的时候，我们知道了 Predicate（路由断言）是 Gateway 的重要组成部分。

Spring Cloud Gateway 包括很多路由断言，当 HTTP 请求进入 Gateway 之后，路由断言会根据配置的路由规则对请求进行断言匹配，成功则路由转发。

Gateway 内置了非常丰富的路由断言，本节我们就实践几种最常用的断言方式。

- After
- Between
- Cookie
- Header
- Method
- Path
- Query
- Weight



1. After

博学谷
www.boxuegu.com

作用

当请求进来的时间 > 配置的时间，匹配成功，否则失败。（与之对应的断言是 Before，用法一致）

配置

```
routes:
  - id: route-hello
    uri: http://localhost:8001/hello
    predicates:
      - After=2030-01-28T12:41:19.524+08:00[Asia/Shanghai]
```

配置的时间要求是 UTC 时间格式，生成方法：

```
ZonedDateTime.now().format(DateTimeFormatter.ISO_ZONED_DATE_TIME); // 现在
ZonedDateTime.now().plusHours(1).format(DateTimeFormatter.ISO_ZONED_DATE_TIME); // 1小时之后
ZonedDateTime.now().minusHours(1).format(DateTimeFormatter.ISO_ZONED_DATE_TIME); // 1小时之前
```

2. Between

博学谷
www.boxuegu.com

作用

当请求进来的时间在配置的时间**之间**时，匹配成功，否则失败。

配置

```
routes:
  - id: route-hello
    uri: http://localhost:8001/hello
    predicates:
      - Between=2020-01-28T12:41:19.524+08:00[Asia/Shanghai],2030-01-28T12:41:19.524+08:00[Asia/Shanghai]
```

3. Cookie

博学谷
www.boxuegu.com

作用

请求中的 Cookie 符合配置时，匹配成功，否则失败。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: cookie_route
        uri: http://localhost:8001/hello
        predicates:
        - Cookie=test, ch.p
```

Cookie 配置的是 name, value, 其中 “value” 可以是**正则表达式**的形式。

4. Header

博学谷
www.boxuegu.com

作用

请求中的 Header 符合配置时，匹配成功，否则失败。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: header_route
        uri: http://localhost:8001/hello
        predicates:
        - Header=X-Request-Id, \d+
```

Header 配置的是 name, value, 其中 “value” 可以是**正则表达式**的形式。

5. Method

博学谷
www.boxuegu.com

作用

HTTP 请求的 Method 在配置之中时，匹配成功，否则失败。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: method_route
        uri: http://localhost:8001/hello
        predicates:
        - Method=GET,POST
```

6. Path

博学谷
www.boxuegu.com

作用

HTTP 请求的路径符合配置的路径模式时，匹配成功，否则失败。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: path_route
        uri: http://localhost:8001/hello
        predicates:
        - Path=/red/{segment},/blue/{segment}
```

{segment} – 占位符

在 Gateway 过滤器中可以取到其对应的值：

```
Map<String, String> uriVar = ServerWebExchangeUtils
            .getPathPredicateVariables(exchange);
String segment = uriVar.get("segment");
```

7. Query

博学谷
www.boxuegu.com

作用

HTTP 请求参数符合配置时，匹配成功，否则失败。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: query_route
        uri: http://localhost:8001/hello
        predicates:
        - Query=green
```

请求中有“green”参数即可匹配。

```
spring:
  cloud:
    gateway:
      routes:
      - id: query_route
        uri: http://localhost:8001/hello
        predicates:
        - Query=red, gree.
```

请求有“red”参数，且值符合正则“gree.”时为匹配。

8. Weight

博学谷
www.boxuegu.com

作用

实现权重路由。例如一个服务有新、旧2个版本，访问地址一样，新服务需要小范围的运行一段时间，80%的流量给旧版本，20%的流量给新版本，这就需要网关按照权重分配转发的流量。

配置

```
gateway:
    routes:
    - id: test_v1
        uri: http://localhost:8001
        predicates:
        - Path=/test
        - Weight=service_weight, 8
    - id: test_v2
        uri: http://localhost:8002
        predicates:
        - Path=/test
        - Weight=service_weight, 2
```

8. Weight

博学谷
www.boxuegu.com

验证

1. 创建2个服务，作为新版本服务、旧版本服务

旧版本 端口：8001，接口：

```
@GetMapping("/test")
public String testv1(){
    return "v1 旧版本";
}
```

新版本 端口：8002，接口：

```
@GetMapping("/test")
public String testv2(){
    return "v2 新版本";
}
```

2. gateway 中配置权重路由

3. 测试，访问 http://localhost:8081/test，就可以体验到输出的**“v1”占绝大多数。**

内置路由断言-总结

博学谷
www.boxuegu.com

重难点

1. 理解路由断言的作用
2. 掌握 After、Between、Cookie、Header、Method、Path、Query、Weight 这些常用断言的用法
3. 理解 Weight 权重路由的作用、场景

博学谷
www.boxuegu.com

目录 Contents

自定义路由断言

博学谷
www.boxuegu.com

小节导学

上节学习一些 Spring Cloud Gateway 内置的路由断言，已经很丰富了，但在实际场景中，难免会有无法满足的场景。例如：

1. “Query”断言中，只能判断一组参数和值，如果需要判断2组怎么办？
2. “Between”断言中，判断的是精准的时间范围，如果需要每天夜里12点至4点不允许下单怎么办？

Spring Cloud Gateway 官方文档中并没有说明如何自定义断言，但我们可以学习模仿内置断言，本节我们就开发一个自己的断言。

- 内置断言开发思路分析
- 自定义断言开发

1. 内置断言开发思路分析

```
public class AfterRoutePredicateFactory extends AbstractRoutePredicateFactory<AfterRoutePredicateFactory.Config> {
    public static final String DATETIME_KEY = "datetime";

    public AfterRoutePredicateFactory() { super(AfterRoutePredicateFactory.Config.class); }

    public List<String> shortcutFieldOrder() { return Collections.singletonList("datetime"); }

    public Predicate<ServerWebExchange> apply(AfterRoutePredicateFactory.Config config) {
        ZonedDateTime datetime = config.getDatetime();
        return (exchange) -> {
            ZonedDateTime now = ZonedDateTime.now();
            return now.isAfter(datetime);
        };
    }

    public static class Config {
        @NotNull
        private ZonedDateTime datetime;

        public Config() {
        }

        public ZonedDateTime getDatetime() { return this.datetime; }

        public void setDatetime(ZonedDateTime datetime) { this.datetime = datetime; }
    }
}
```

核心点

- 集成抽象类 AbstractRoutePredicateFactory，是泛型，需要指定一个类
- 此处使用了一个内部类 Config，对应的是配置文件中 After 的值。
- shortcutFieldOrder 方法用于定义 After 值中各项的顺序
- apply 方法处理断言逻辑
- 此类需要定义为 Bean

2. 自定义断言开发

博学谷
www.boxuegu.com

需求

请求参数中必须包含这2个参数：“red”、“blue”，而且其值必须为“y”。

路由断言形式：

```
spring:
  cloud:
    gateway:
      routes:
      - id: query_route
        uri: http://localhost:8001/hello
        predicates:
        - QueryParams=red,y,blue,y
```

自定义路由断言-总结

博学谷
www.boxuegu.com

重难点

1. 理解内置路由断言的分析思路
2. 掌握自定义断言的开发关键点
3. 掌握自定义断言的开发方法

博学谷
www.boxuegu.com

目录 Contents

内置过滤器

博学谷
www.boxuegu.com

小节导学

前面介绍 Gateway 核心概念的时候，我们知道了 Filter（过滤器）和 Predicate（断言）一样，也是 Gateway 的重要组成部分。

当请求转发给服务之前，和收到服务的响应之后，都可以被 Filter 处理。

Gateway 内置了非常丰富的过滤器，本节我们就实践几种最常用的过滤方式。

- AddRequestHeader
- AddRequestParameter
- AddResponseHeader
- RemoveRequestHeader
- StripPrefix
- RewritePath
- LoadBalancerClientFilter

1. AddRequestHeader

博学谷
www.boxuegu.com

作用

接收2个参数：name、value，作为新的请求头，添加到当前请求中。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_header_route
        uri: http://localhost:8001/test/head
        predicates:
        - Method=GET
        filters:
        - AddRequestHeader=X-Request-red, blue
```

1. AddRequestHeader

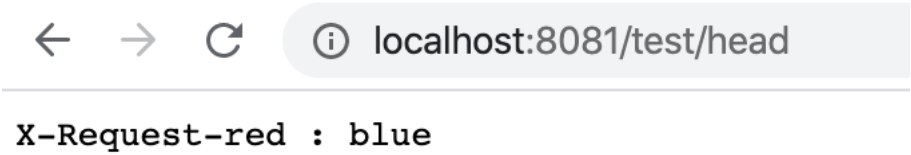
博学谷
www.boxuegu.com

验证

service-provider 中添加接口“/test/head”

```
@GetMapping("/test/head")
public String testGatewayHead(HttpServletRequest request, HttpServletResponse response){
    String head=request.getHeader("X-Request-red");
    return "X-Request-red : "+head;
}
```

service-gateway 中接口“http://localhost:8081/test/head”



2. AddRequestParameter

博学谷
www.boxuegu.com

作用

接收2个参数：name、value，作为新的请求参数，添加到当前请求中。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_parameter_route
        uri: http://localhost:8001/test/param
        predicates:
        - Method=GET
        filters:
        - AddRequestParameter=red, blue
```

2. AddRequestParameter

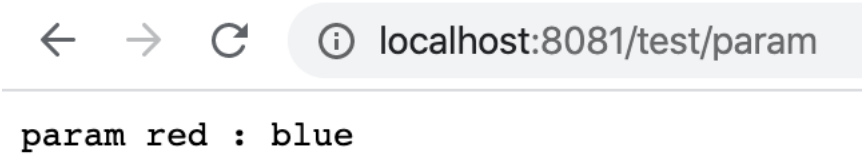
博学谷
www.boxuegu.com

验证

service-provider 中添加接口“/test/param”

```
@GetMapping("/test/param")
public String testGatewayParam(HttpServletRequest request, HttpServletResponse response) {
    String val = request.getParameter("red");
    return "param red : " + val;
}
```

service-gateway 中接口“http://localhost:8081/test/param”



3. AddResponseHeader

博学谷
www.boxuegu.com

作用

接收2个参数：name、value，作为新的响应头信息，添加到当前请求的响应中。

配置

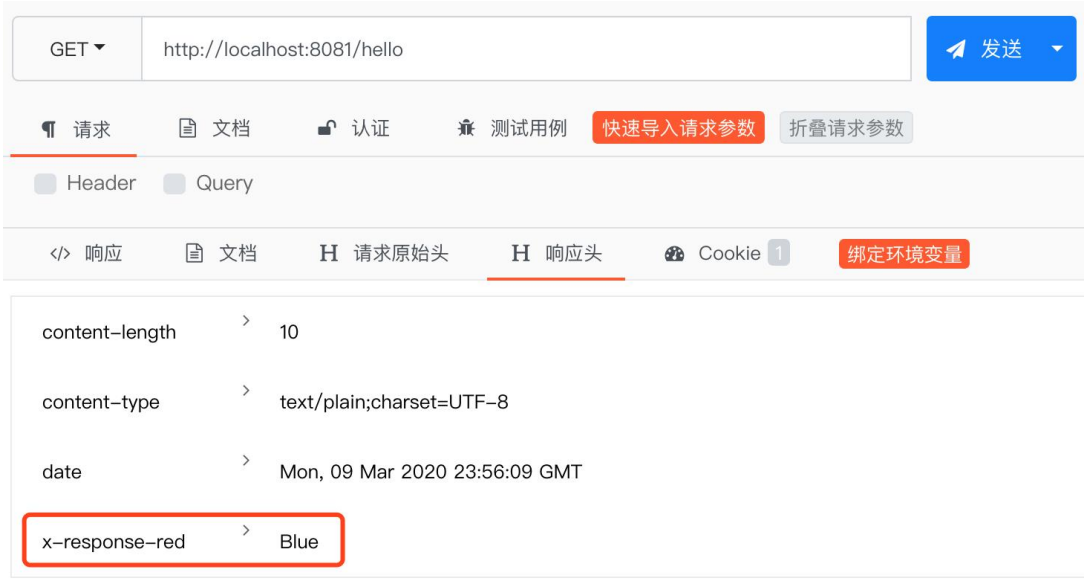
```
spring:
  cloud:
    gateway:
      routes:
      - id: add_response_header_route
        uri: http://localhost:8001/hello
        predicates:
        - Method=GET
        filters:
        - AddResponseHeader=X-Response-Red, Blue
```

3. AddResponseHeader

博学谷
www.boxuegu.com

验证

访问“http://localhost:8081/hello”，在其响应信息中可以看到我们添加的内容。



4. RemoveRequestHeader

博学谷
www.boxuegu.com

作用

接收1个参数：name，如果当前请求中有这个头信息，就删除。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: removerequestheader_route
        uri: http://localhost:8001/hello
        predicates:
        - Method=GET
        filters:
        - RemoveRequestHeader=X-Request-red
```

5. StripPrefix

博学谷
www.boxuegu.com

作用

接收1个参数：数字，含义为从请求路径中截取掉前面的几个部分。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: strip_prefix_route
        uri: http://localhost:8001/hello
        predicates:
        - Path=/red/**
        filters:
        - StripPrefix=2
```

验证

访问“http://localhost:8081/red/blue/hello”，路径中的“/red/blue”会被截取掉，相当于访问“/hello”

6. RewritePath

博学谷
www.boxuegu.com

作用

接收2个参数：正则表达式、替换字符串，实际上就是对请求路径做正则替换。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: rewritepath_route
        uri: http://localhost:8001/hello
        predicates:
        - Path=/red/**
        filters:
        - RewritePath=/red(?<segment>/?.*), $\{segment}
```

验证

访问“http://localhost:8081/red/hello”，路径“/red/hello”会被替换为“/hello”，相当于访问“/hello”

7. LoadBalancerClientFilter

博学谷
www.boxuegu.com

作用

以负载均衡的方式获取实际的 uri 地址。

配置

```
spring:
  cloud:
    gateway:
      routes:
      - id: myRoute
        uri: lb://service-provider
        predicates:
        - Path=/**
```

工作方式

在配置中可以发现，并没有使用 filter，因为这里使用的是 **GlobalFilter（全局过滤器）。**

全局过滤器是**应用于所有请求**的，而不是像普通过滤器只作用于单个路由。

LoadBalancerClientFilter 发现 uri 的前缀为“**lb**”的时候，就会使用 LoadBalancerClient 获取服务实例的 IP、port，替换为 uri，达到负载均衡的效果。

内置过滤器-总结

重难点

1. 理解路由过滤器的工作思路
2. 掌握 AddRequestHeader、AddRequestParameter、AddResponseHeader、RemoveRequestHeader、StripPrefix、RewritePath、LoadBalancerClientFilter 常用过滤器的作用与应用方法

博学谷
www.boxuegu.com

目录 Contents

自定义过滤器

博学谷
www.boxuegu.com

小节导学

上节学习了一些 Spring Cloud Gateway 内置的过滤器，同样很丰富，但像路由断言一样，实际场景中也会有无法满足的需求。所以，也需要我们有开发过滤器的能力。内置的过滤器分为：

1. GatewayFilter 普通过滤器（作用于某个路由 Route）
2. GlobalFilter 全局过滤器（作用于所有路由 Route）

那么我们自己开发过滤器也分为2种：

1. 自定义普通过滤器
2. 自定义全局过滤器

本节我们的目标就是自己开发过滤器，思路和自定义路由断言时一样，可以先学习一下内置过滤器是怎么开发的。

- 自定义普通过滤器
- 自定义全局过滤器

1. 自定义普通过滤器

博学谷
www.boxuegu.com

普通过滤器代码分析

```
public class AddRequestHeaderGatewayFilterFactory
    extends AbstractNameValueGatewayFilterFactory {

    public AddRequestHeaderGatewayFilterFactory() {
    }

    public GatewayFilter apply(NameValueConfig config) {
        return (exchange, chain) -> {
            ServerHttpRequest request = exchange.getRequest().mutate()
                .header(config.getName(), config.getValue()).build();
            return chain.filter(exchange.mutate().request(request).build());
        };
    }
}
```

1. 自定义普通过滤器

博学谷
www.boxuegu.com

普通过滤器代码分析

核心代码：

- 继承了 “AbstractNameValueGatewayFilterFactory”
- 过滤器的核心方法就是 “apply”，其参数为“NameValueConfig”，可以方便的取得配置文件中指定的 name、value
- “exchange.getRequest().mutate()”可以获取 request 进行修改
- “exchange.mutate()”可以修改 exchange
- “chain.filter”方法用于传递过滤器

1. 自定义普通过滤器

博学谷
www.boxuegu.com

需求

输出调用服务的耗时，请求服务前记录时间点，响应完成后计算所用的毫秒数。

配置形式：

```
spring:
  cloud:
    gateway:
      routes:
      - id: rewritepath_route
        uri: http://localhost:8001/hello
        predicates:
          - Method=GET
        filters:
          - AddDebug=a,b
```

1. 自定义普通过滤器

博学谷
www.boxuegu.com

实现

```
@Component
public class AddDebugGatewayFilterFactory extends AbstractNameValueGatewayFilterFactory {
    @Override
    public GatewayFilter apply(NameValueConfig config) {
        return (exchange, chain) -> {
            System.out.println("--- 进入 AddDebugGatewayFilterFactory");
            exchange.getAttributes().put("Time", System.currentTimeMillis());
            return chain.filter(exchange).then(
                    Mono.fromRunnable(() -> {
                        Long start = exchange.getAttribute("Time");
                        Long end = System.currentTimeMillis();
                        System.out.println("共耗时 : " + (end - start) + "ms");
                    })
            );
        };
    }
}
```

2. 自定义全局过滤器

博学谷
www.boxuegu.com

全局过滤器代码分析

```
public class LoadBalancerClientFilter implements GlobalFilter, Ordered {
    public static final int LOAD_BALANCER_CLIENT_FILTER_ORDER = 10100;
    private static final Log log = LogFactory.getLog(LoadBalancerClientFilter.class);
    protected final LoadBalancerClient loadBalancer;
    private LoadBalancerProperties properties;

    public LoadBalancerClientFilter(LoadBalancerClient loadBalancer, LoadBalancerProperties
        this.loadBalancer = loadBalancer;
        this.properties = properties;
    }

    public int getOrder() { return 10100; }

    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain) {...}
```

2. 自定义全局过滤器

博学谷
www.boxuegu.com

全局过滤器代码分析

核心代码：

- 实现了2个接口 “GlobalFilter”、“Ordered”
- “filter”方法处理过滤逻辑
- “getOrder”方法定义了此过滤器的优先级，数字越小，优先级越高
- “exchange”用法与普通过滤器中介绍的一样

2. 自定义全局过滤器

博学谷
www.boxuegu.com

需求

验证请求头中是否有“token”，如果没有，则返回“未认证”状态码。

此全局过滤器不需要配置。

自定义过滤器-总结

博学谷
www.boxuegu.com

重难点

1. 理解自定义过滤器的思路
2. 掌握自定义普通过滤器的开发思路、核心技术点
3. 掌握自定义全局过滤器的开发思路、核心技术点

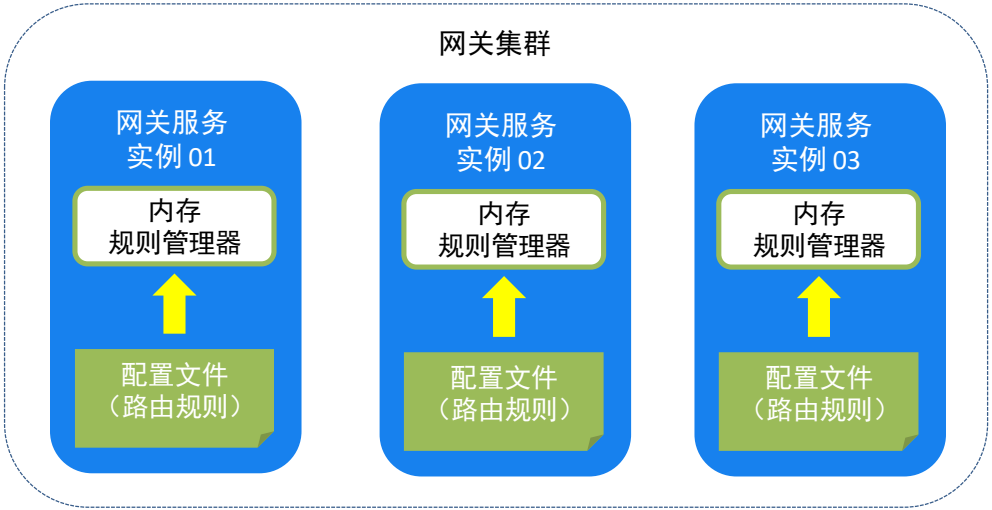
博学谷
www.boxuegu.com

目录 Contents

动态路由

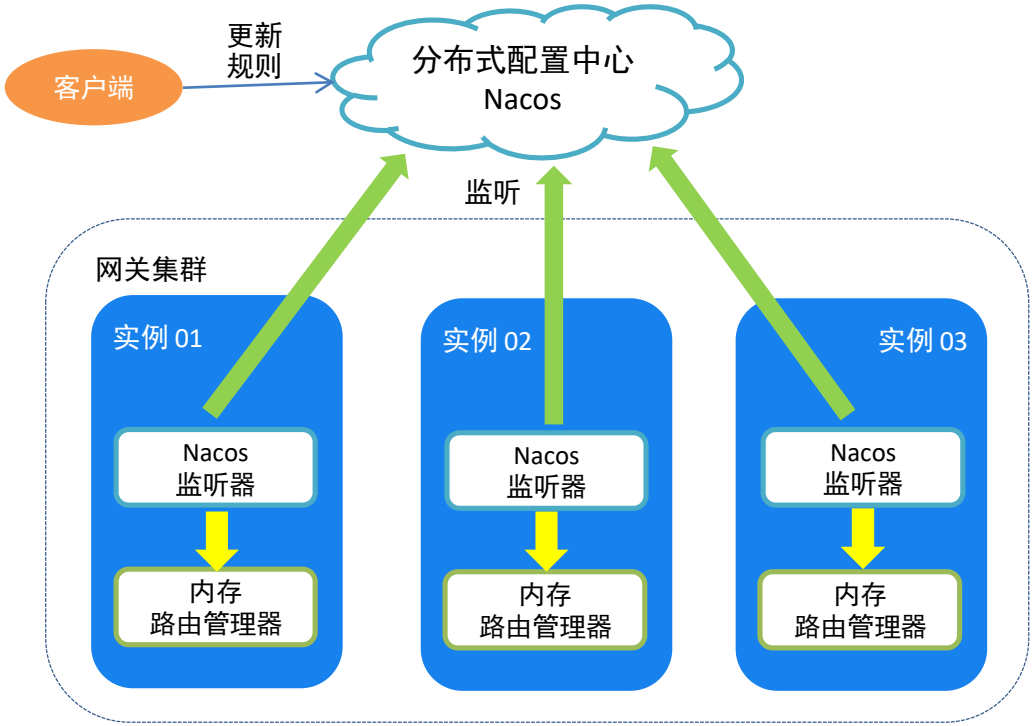
小节导学

路由规则是网关的核心内容，配置在应用的属性配置文件中，启动的时候将路由规则加载到内存，属于静态路由方式。在高可靠架构中，网关服务都会部署多个实例，这时静态路由方式就些不足，例如更新路由规则时，需要重启所有的网关服务实例，造成系统中断。



动态路由

小节导学



本节我们采用 Nacos 来实现动态路由功能。

- 动态路由实现思路分析
- 动态路由开发

1. 动态路由实现思路分析

博学谷
www.boxuegu.com

核心代码

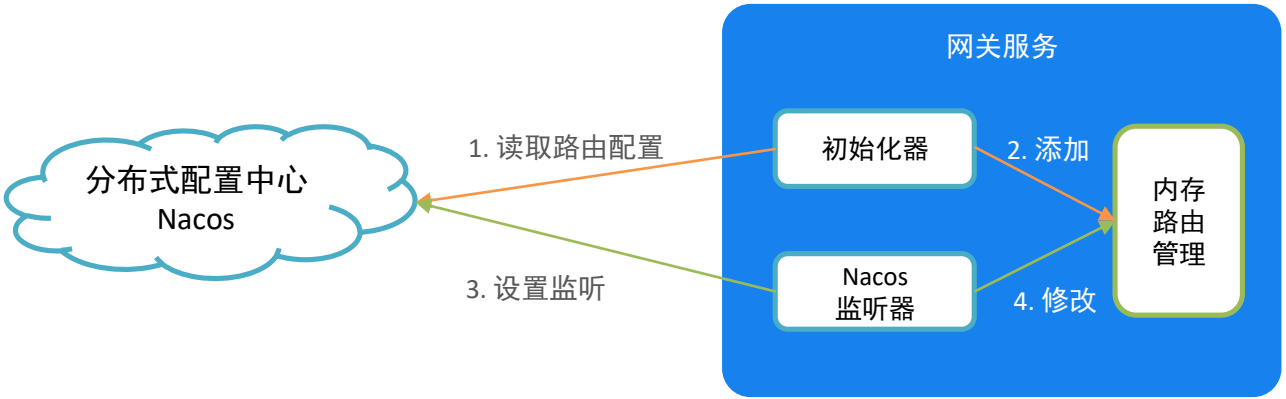
Gateway 提供了修改路由的接口 RouteDefinitionWriter, 有了这个接口才能动态修改路由

```
public interface RouteDefinitionWriter {
    Mono<Void> save(Mono<RouteDefinition> route);
    Mono<Void> delete(Mono<String> routeId);
}

public class InMemoryRouteDefinitionRepository implements RouteDefinitionRepository {
    private final Map<String, RouteDefinition> routes =
            Collections.synchronizedMap(new LinkedHashMap());
    public Mono<Void> save(Mono<RouteDefinition> route) { … }
    ……
}
```

1. 动态路由实现思路分析

实现方案



2. 动态路由开发

开发步骤

验证

1. Nacos 中创建路由配置
2. 启动网关服务，应正确加载 Nacos 中的路由配置
3. Nacos 中修改路由配置，应用应正确更新路由配置

动态路由-总结

博学谷
www.boxuegu.com

重难点

1. 理解动态路由的概念以及场景
2. 掌握路由配置相关源码的分析思路
3. 掌握动态路由的设计结构，以及开发流程

博学谷
www.boxuegu.com

目录 Contents

集成 Sentinel

博学谷
www.boxuegu.com

小节导学

我们学习 Sentinel 时，针对的是 API、指定的资源名做流控、降级等规则。

Gateway 中只是配置一系列的路由规则，没有 API 这类的资源，如何使用 Sentinel 做防护呢?

本节我们就解决这个问题，看 Gateway 与 Sentinel 如何集成在一起。

- Gateway 的资源维度
- 集成 Sentinel 实践

1. Gateway 的资源维度

博学谷
www.boxuegu.com

资源维度

Sentinel 1.6 开始适配了 Gateway，提供2种资源维度：

- **Route 维度** - 即在 Spring 配置文件中配置的路由条目，资源名为对应的 routeId，例如：

```
routes:
        - id: product_route
          uri: http://localhost:8001/
          predicates:
            - Path=/product/**
```

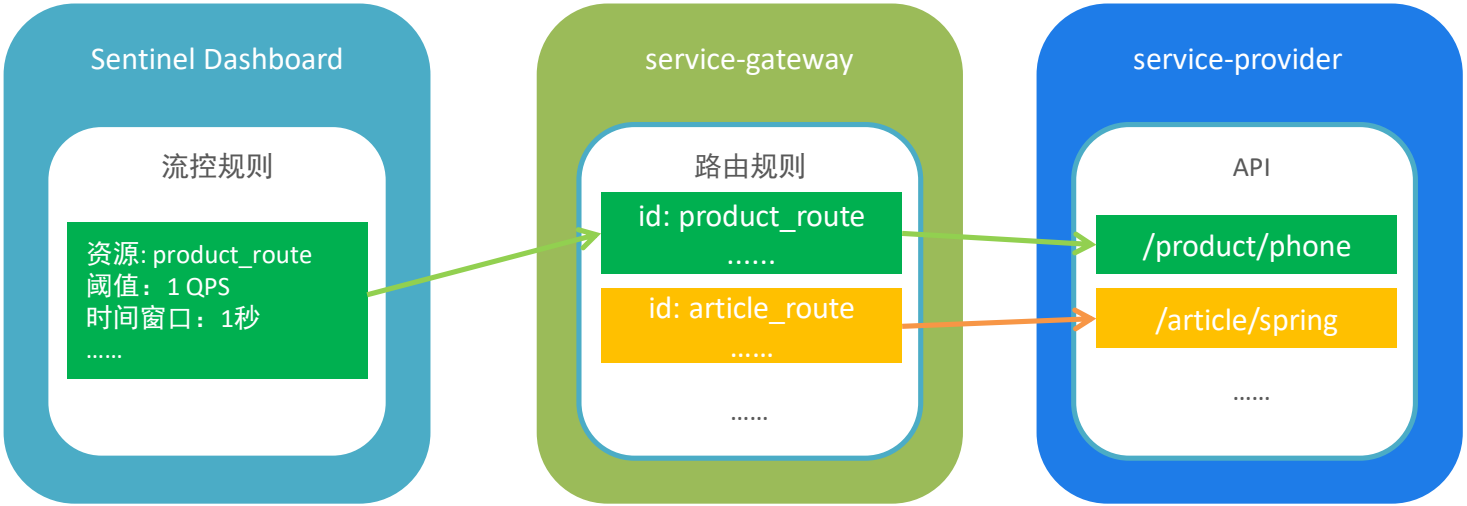
- **自定义 API 维度** - 用户可以利用 Sentinel 提供的 API 来自定义一些 API 分组

  例如，请求 path 模式为 **/foo/**** 和 **/baz/**** 的都归到 **my_api** 这个 API 分组下面，限流的时候可以针对 **my_api** 进行限流。

2. 集成 Sentinel 实践

博学谷
www.boxuegu.com

整体结构



2. 集成 Sentinel 实践

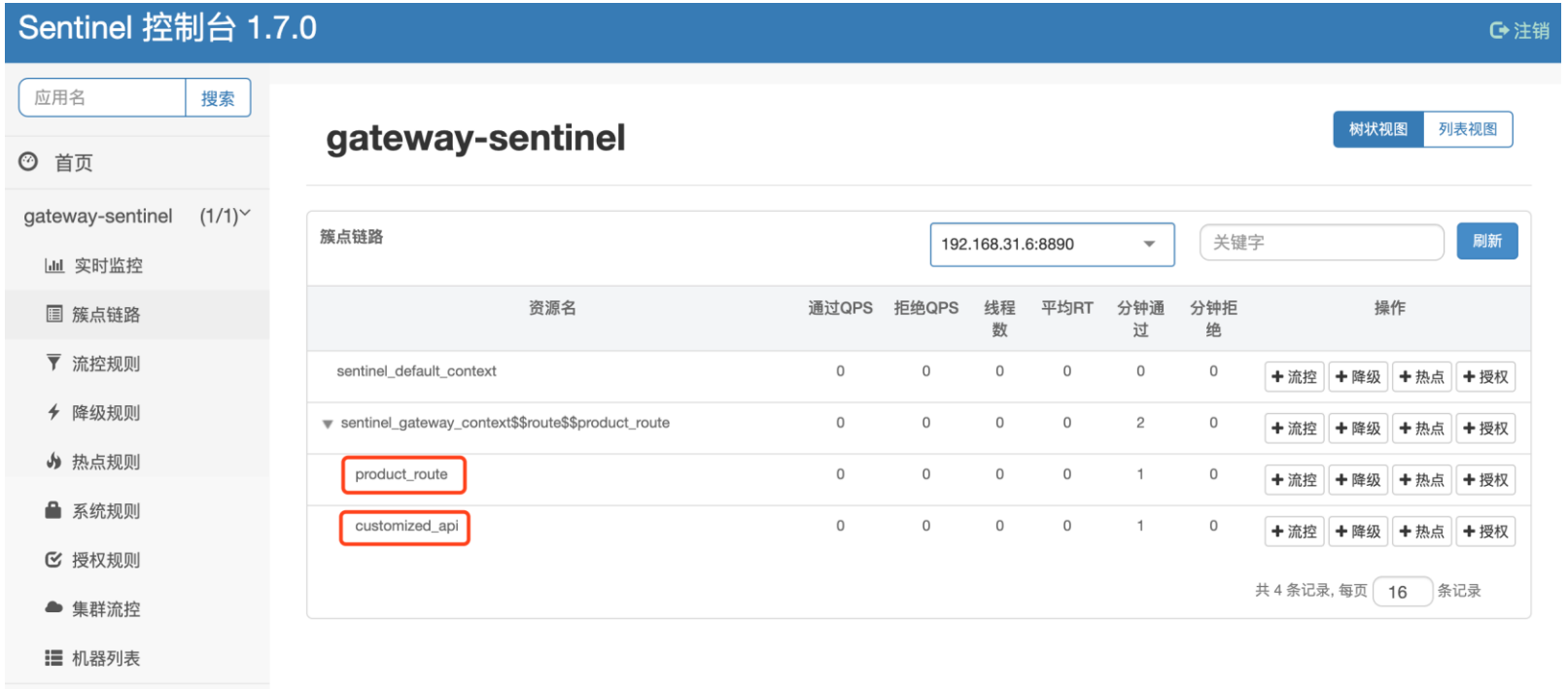
博学谷
www.boxuegu.com

实践步骤

1. 创建 service-provider，提供测试 API
2. 创建 service-gateway 网关服务，配置 Route 规则，并整合 Sentinel
3. Sentinel 控制台中针对 Route 类型资源进行限流测试
4. service-gateway 中定义 API 组合资源
5. Dashboard 中针对 API 组合资源进行限流测试

2. 集成 Sentinel 实践

实践效果



集成 Sentinel-总结

博学谷
www.boxuegu.com

重难点

1. 理解 Sentinel 中 Gateway 的2个资源维度
2. 掌握 Sentinel + Gateway 的结构形式
3. 掌握 Gateway 集成 Sentinel 的开发流程

一样的在线教育，不一样的教学品质