21 Gateway 中 Predicate 和 Filter 的用法

更新时间：2019-07-05 09:56:55

> 学习知识要善于思考，思考，再思考。
>
> —— 爱因斯

经过前、两节的学习，大家对服务网关和 Spring Cloud Gateway 的使用有了一定的了解，这节课我们继续学习 Spring Cloud Gateway，了解其最关键的两个功能点 Predicate 和 Filter。

Predicate 和 Filter 是 Spring Cloud Gateway 的核心，通过这两个功能点的灵活配置使用，Spring Cloud Gateway 的使用变得高效、简单。Predicate 的核心作用是路由选择，通过一些列的规则配置，让我们知道哪些请求可以被某个规则转发；Filter 是过滤器，在 Predicate 删选出某些请求需要转发时，Filter 负责在这些请求的执行前或者执行后做一些处理，比如安全校验、参数处理等。

用一句话来总结就是：**Predicate 帮助选择哪些请求需要处理，Filter 给选择出来的请求做一些改动**。接下来我们学习这两个功能点的使用。

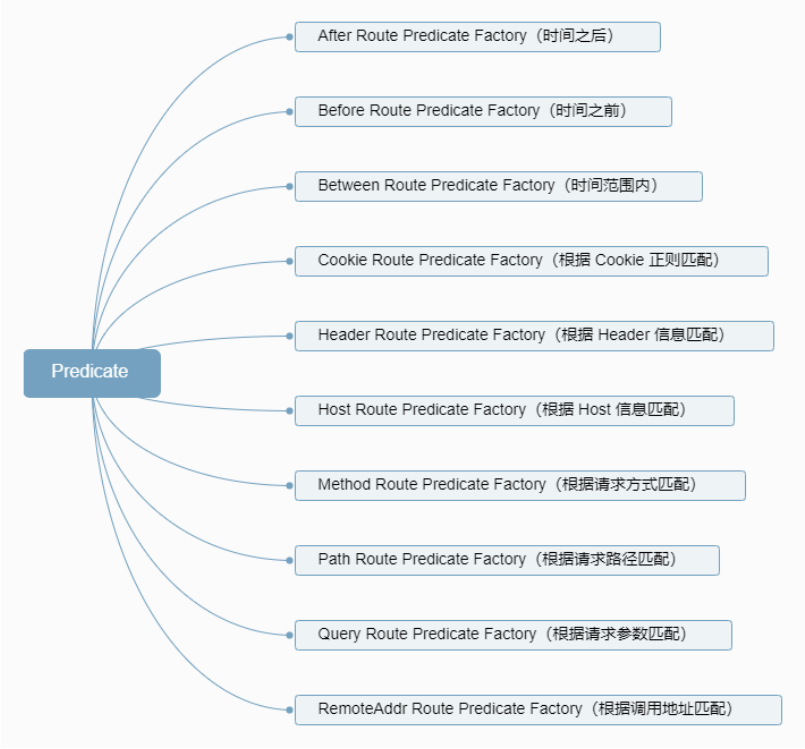
Predicate

Spring Cloud Gateway 是通过 Spring WebFlux 的 HandlerMapping 做为底层支持来匹配到转发路由，Spring Cloud Gateway 内置了很多 Predicates 工厂，这些 Predicates 工厂通过不同的 HTTP 请求参数来匹配，多个 Predicates 工厂可以组合使用。

Predicate 介绍

Predicate 来源于 Java 8，是 Java 8 中引入的一个函数，Predicate 接受一个输入参数，返回一个布尔值结果。该接口包含多种默认方法来将 Predicate 组合成其他复杂的逻辑（比如：与，或，非）。可以用于接口请求参数校验、判断新老数据是否有变化需要进行更新操作。

在 Spring Cloud Gateway 中 Spring 利用 Predicate 的特性实现了各种路由匹配规则，有通过 Header、请求参数等不同的条件来进行作为条件匹配到对应的路由。Spring Cloud Gateway 中内置了几种 Predicate 的实现，如下图：



可以看出 Spring Cloud Gateway 内置的 Predicate 已经非常丰富，足够满足我们日常的绝大部分工作，接下来选择几个有代表性的 Predicate 进行演示：

通过时间匹配

Predicate 支持设置一个时间，在请求进行转发的时候，可以通过判断在这个时间之前或者之后进行转发。比如我们现在设置只有在2019年1月1日才会转发到我的网站，在这之前不进行转发，我就可以这样配置：

```
spring:
  cloud:
    gateway:
      routes:
      - id: time_route
        uri: http://www.ityouknow.com/
        predicates:
          - After=2018-04-22T06:06:06+08:00[Asia/Shanghai]
```

Spring 是通过 ZonedDateTime 来对时间进行的对比，ZonedDateTime 是 Java 8 中日期时间功能里，用于表示带时区的日期与时间信息的类，ZonedDateTime 支持通过时区来设置时间，中国的时区是：Asia/Shanghai。

After Route Predicate 是指在这个时间之后的请求都转发到目标地址。上面的示例是指，请求时间在 2018年4月22日6点6分6秒之后的所有请求都转发到地址`https://www.ityouknow.com`。+08:00 是指时间和 UTC 时间相差八个小时，时间地区为`Asia/Shanghai`。

添加完路由规则之后，访问地址`http://localhost:8080`会自动转发到`https://www.ityouknow.com`。

其它的两个时间判断和上面类似，只需要替换对应的 predicates 值即可。

```
predicates:
 - After=2018-04-22T06:06:06+08:00[Asia/Shanghai]
 - Before=2018-04-22T06:06:06+08:00[Asia/Shanghai]
 - Between=2018-04-22T06:06:06+08:00[Asia/Shanghai], 2019-04-22T06:06:06+08:00[Asia/Shanghai]
```

通过请求方式匹配

请求方式即使页面表单的请求类型，比如：POST、GET、PUT、DELETE ，Spring Cloud Gateway 内置了 Predicate 可根据不同的请求方式来选择路由。

我们来配置一个 Get 请求方式的转发，注释掉上面的配置，在配置文件添加以下内容：

```
spring:
  cloud:
    gateway:
      routes:
      - id: method_route
        uri: http://www.ityouknow.com/
        predicates:
           - Method=GET
```

修改完成之后重新启动项目，我们在 windows 系统下打开 cmd 命令行，使用 curl 命令来测试。

```
# curl 默认是以 GET 的方式去请求
curl http://localhost:8080
```

测试返回页面代码，证明匹配到路由，我们再以 POST 的方式请求测试。

```
curl -X POST http://localhost:8080
```

返回 404 没有找到，证明没有匹配上路由。

通过请求路径匹配

Path Route Predicate 接收一个匹配路径的参数来判断是否走路由。

```
spring:
  cloud:
    gateway:
      routes:
      - id: path_route
        uri: http://ityouknow.com
        predicates:
        - Path=/foo/{segment}
```

如果请求路径符合要求，则此路由将匹配，例如：/foo/1 或者 /foo/bar。

使用 curl 测试，命令行输入:

```
curl http://localhost:8080/foo/1
curl http://localhost:8080/foo/xx
curl http://localhost:8080/boo/xx
```

经过测试第一和第二条命令可以正常获取到页面返回值，最后一个命令报404，证明路由是通过指定路由来匹配。

通过请求参数匹配

Query Route Predicate 支持传入两个参数，一个是属性名一个为属性值，属性值可以是正则表达式。

```
spring:
  cloud:
    gateway:
      routes:
      - id: query_route
        uri: http://ityouknow.com
        predicates:
        - Query=smile
```

这样配置，只要请求中包含 smile 属性的参数即可匹配路由。

使用 curl 测试，命令行输入:

```
curl localhost:8080?id=1
curl localhost:8080?smile=x&id=2
```

经过测试发现只要请求汇总带有 smile 参数即会匹配路由，不带 smile 参数则不会匹配。

还可以将 Query 的值以键值对的方式进行配置，这样在请求过来时会对属性值和正则进行匹配，匹配上才会走路由。

```
spring:
  cloud:
    gateway:
      routes:
      - id: query_route
        uri: http://ityouknow.com
        predicates:
        - Query=keep, pu.
```

这样只要当请求中包含 keep 属性并且参数值是以 pu 开头的长度为三位的字符串才会进行匹配和路由。

使用 curl 测试，命令行输入:

```
curl localhost:8080?keep=pub
```

测试可以返回页面代码，将 keep 的属性值改为 pubx 再次访问就会报 404，证明路由需要匹配正则表达式才会进行路由。

组合使用

在我们日常工作中，往往会使用多个 Predicate 来进行判断，Spring Cloud Gateway 支持同时配置多个 Predicate 条件，各种 Predicates 同时存在于同一个路由时，请求必须同时满足所有的条件才被这个路由匹配。

我们来测试一个组合使用的案例：

```
spring:
  cloud:
    gateway:
      routes:
       - id: method_path_time
         uri: http://ityouknow.com
         predicates:
         - Method=GET
         - Query=foo, ba.
         - After=2018-01-20T06:06:06+08:00[Asia/Shanghai]
```

我们使用以下命令测试：

```
curl localhost:8080?foo=baa
```

使用 Post 方式提交、修改 foo 的时间或者更新 After 的时间为 2020 年均会返回 404，这说明了三个条件都必须同时满足后才可执行。

其它几个 Predicate 的使用方式和上述基本类似，只需要按照其语法修改对于的 Predicate 条件即可。同时需要注意的是：**一个请求满足多个路由条件时，请求只会被首个成功匹配的路由转发**。

Filter 介绍

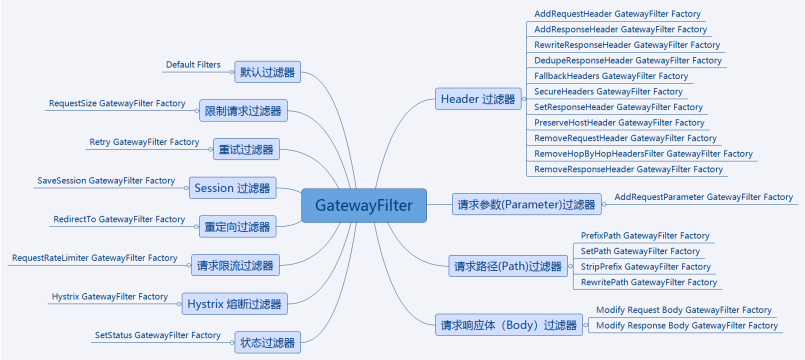
Spring Cloud Gateway 的 Filter 的生命周期不像 Zuul 的那么丰富，它只有两个：“pre” 和 “post”。

- PRE： 这种过滤器在请求被路由之前调用。我们可利用这种过滤器实现权限管理、安全校验、记录调试信息等。
- POST：这种过滤器在路由到微服务以后执行。这种过滤器可用来为响应添加标准的 HTTP Header、收集统计信息和指标、将响应从微服务发送给客户端等。

Spring Cloud Gateway 的 Filter 分为两种：GatewayFilter 与 GlobalFilter。GlobalFilter 会应用到所有的路由上，而 GatewayFilter 将应用到单个路由或者一个分组的路由上。

Gateway Filter

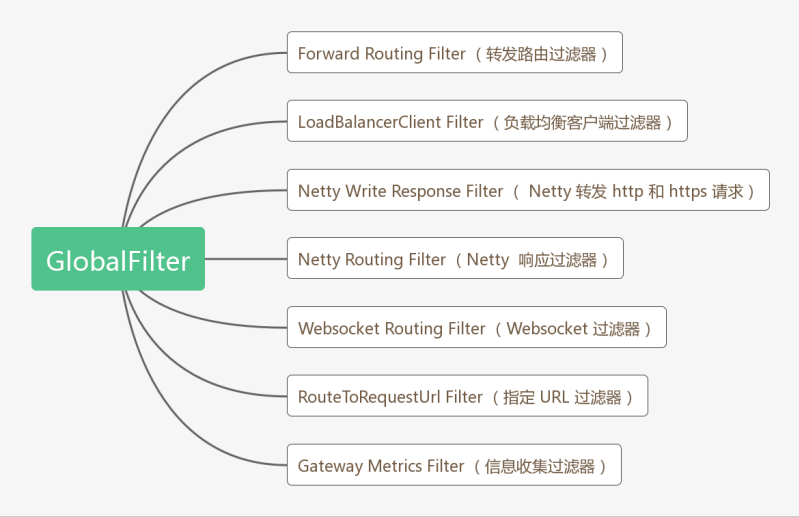
Spring Cloud Gateway 内置了 25 种 GatewayFilter 和一个 Default Filters。按照不同作用方式我们将它划分为 12 大类。



Global Filter

Spring Cloud Gateway 内置了7种 GlobalFilter，比如 Netty Routing Filter、LoadBalancerClient Filter、Websocket Routing Filter 等，根据名字即可看出这些 Filter 的作用。

多个 Global Filter 可以通过 @Order 或者 getOrder() 方法指定每个 Global Filter 的执行顺序，order 值越小，Global Filter 执行的优先级越高。



利用 GatewayFilter 可以修改 Http 的请求或者响应，或者根据请求或者响应做一些特殊的限制，更多时候我们会利用 GatewayFilter 做一些具体的路由配置，接下来我们通过示例来学习。

AddRequestParameter GatewayFilter

AddRequestParameter GatewayFilter 是匹配的请求中添加相关参数，可以用在需要在特定请求中添加参数的场景中。

复用上节课的示例项目，在 provider-1 项目中添加 foo() 方法，代码如下：

```
@RequestMapping("/foo")
public String foo(String foo) {
    return "hello "+foo+"!";
}
```

provider-1 项目中添加 foo() 方法，多添加一个感叹号用于区别。

```
@RequestMapping("/foo")
public String foo(String foo) {
    return "hello "+foo+"!! ";
}
```

在 consumer 项目中，添加对 foo 的调用。

HelloService 中添加以下代码：

```
@FeignClient("provider")
public interface HelloService {
    @GetMapping("/foo")
    String foo(@RequestParam("foo") String foo);
}
```

HelloController 中添加以下代码：

```
@GetMapping("/foo")
public String foo(String foo) {
    return helloService.foo(foo);
}
```

gateway 项目中配置文件 application.yml 的内容修改如下：

```
server:
  port: 8888
spring:
  application:
    name: gateway
  cloud:
    gateway:
     discovery:
        locator:
         enabled: true
     routes:
     - id: add_request_parameter_route
       uri: lb://provider
       filters:
        - AddRequestParameter=foo, bar
       predicates:
         - Method=GET
eureka:
  client:
    service-url:
      defaultZone: http://localhost:1111/eureka/
logging:
  level:
    org.springframework.cloud.gateway: debug
```

关键配置解释：

- `spring.cloud.gateway.discovery.locator.enabled`，是否与服务发现组件进行结合，通过 serviceId 转发到具体的服务实例。默认为 false，设为 true 便开启通过服务中心的自动根据 serviceId 创建路由的功能。
- `spring.cloud.gateway.routes.uri=lb://provider`，配置路由转发到名为 provider 的服务提供者。
- `spring.cloud.gateway.routes.filters`，配置需要执行 Filter 的具体实现。
- `spring.cloud.gateway.routes.filters.AddRequestParameter`，配置 AddRequestParameter Filter 给匹配请求添加参数。
- `spring.cloud.gateway.routes.predicates`，请求的删选条件， Filter 需要和 Predicate 配合使用。

以上改造配置完成之后，依次启动 eureka、consumer、 provider-1 、 provider-2 和 gateay 项目。

首先我们直接调用 consumer 中的 foo 方法，访问地址：`http://localhost:5002/foo` 查看返回结果：

```
hello null!
```

说明 provider 项目端并没有接受到 foo 参数的值，也就是请求中并没有添加 foo 参数值，接下来我们通过网关来调用 foo 方法。浏览器访问地址：`http://localhost:8888/foo`，页面交替返回信息如下：

```
hello bar!
hello bar!!
```

通过上面实验可以得知，通过路由进行转发时，在请求中添加了 foo 参数和值，也就意味着在请求过程中的 Filter 已经添加生效。

这里默认使用了全局过滤器 LoadBalancerClient ， 当路由配置中 uri 所用的协议为 lb 时（以 `uri: lb://provider` 为例），gateway 将使用 LoadBalancerClient 把 provider 通过 eureka 解析为实际的主机和端口，并进行负载均衡。

其它的 GatewayFilter 使用方式和上述方式比较类似，这里不再一一列举，使用时按照要求的语法配置即可。

小结

Spring Cloud Gateway 有非常强大的 Predicate 选择机制，内置的 Predicates 实现已经满足了我们绝大部分工作场景。同时 Spring Cloud Gateway 也提供了请求过程中的各种 Filters ，其中 Filter 又区分为 GatewayFilter 和 GlobalFilter，GatewayFilter 作用于特定请求，GlobalFilter 作用于全局，实际工作中我们根据需求来选择使用。

参考链接：https://cloud.spring.io/spring-cloud-gateway/spring-cloud-gateway.html

本文作者：纯洁的微笑、江南一点雨

20 Spring Cloud Gateway 快速实践

22 初识 Spring Cloud Config