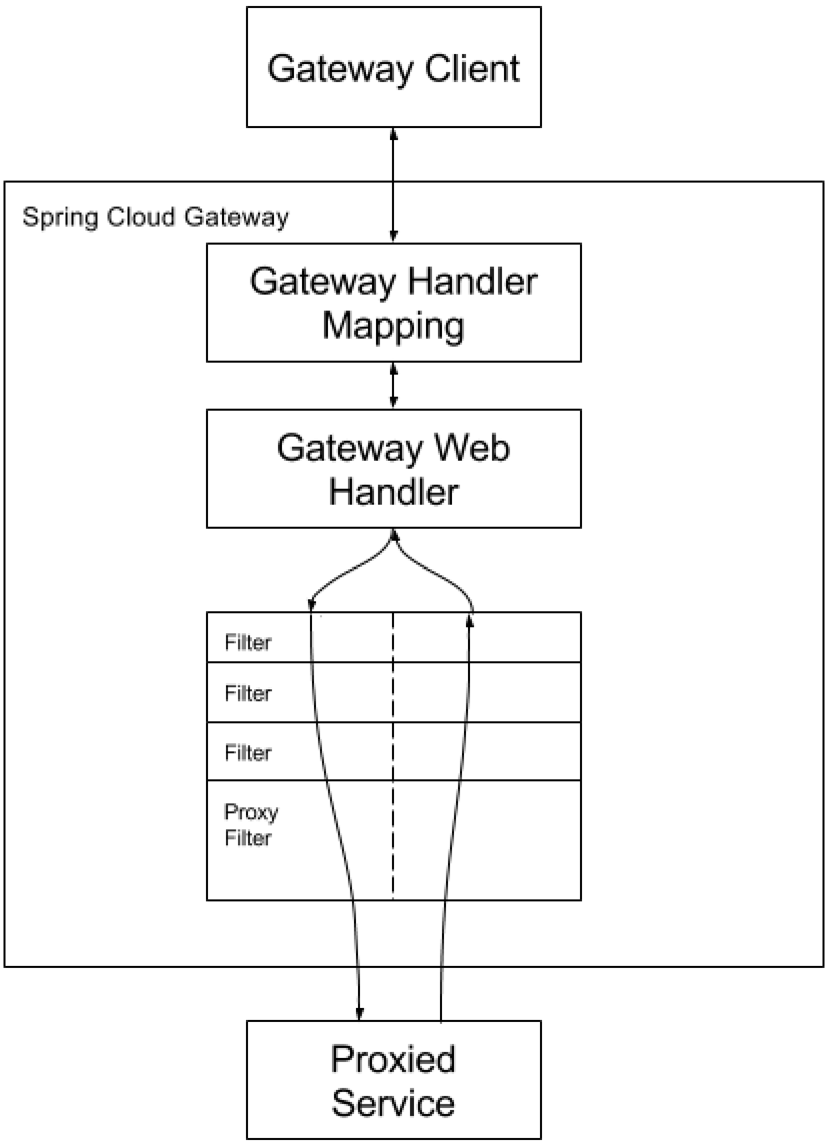
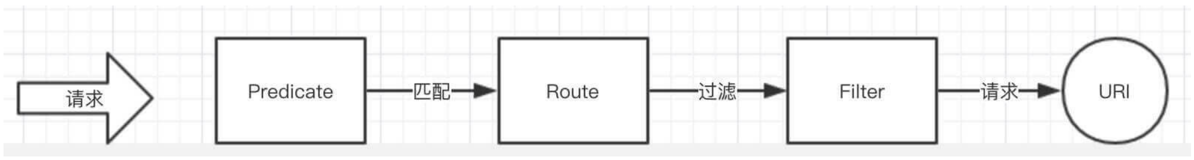
Spring-Cloud-Gateway源码系列学习

版本 v2.2.6.RELEASE

Spring-Cloud-Gateay工作流程





基础组件学习

Route

> Spring-Cloud-Gateway最基础组件

```
public class Route implements Ordered {

    private final String id;

    //路由目标uri，最终被转发的目的地
    private final URI uri;

    //序号，越小优先级越高，实现Ordered的getOrder方法返回order
    private final int order;

    //谓语，匹配Route的前置条件
    private final AsyncPredicate<ServerWebExchange> predicate;

    //过滤器列表，比如可以修改请求头等
    private final List<GatewayFilter> gatewayFilters;

    //
    private final Map<String, Object> metadata;
}
```

AsyncPredicate

> 异步谓词，java8函数式，可以理解Spring-Cloud-Gateway写的Java8 Predicate加强版，ServerWebExchangeUtils#toAsyncPredicate可以把一个Predicate转成AsyncPredicate

```
public interface AsyncPredicate<T> extends Function<T, Publisher<Boolean>> {

    /*
    * 与操作，具体实现逻辑在内部类AndAsyncPredicate.apply
    * Mono.from(left.apply(t)).flatMap(
    *               result -> !result ? Mono.just(false) :
Mono.from(right.apply(t)))
    */
    default AsyncPredicate<T> and(AsyncPredicate<? super T> other) {
        return new AndAsyncPredicate<>(this, other);
    }

    /*
    * 取反操作，具体逻辑在内部类NegateAsyncPredicate的apply
    * Mono.from(predicate.apply(t)).map(b -> !b)
    */
    default AsyncPredicate<T> negate() {
        return new NegateAsyncPredicate<>(this);
    }

    /*
    * 或操作，具体逻辑在内部类OrAsyncPredicate的apply
    * Mono.from(left.apply(t)).flatMap(
    *               result -> result ? Mono.just(true) :
Mono.from(right.apply(t)));
    */
    default AsyncPredicate<T> or(AsyncPredicate<? super T> other) {
        return new OrAsyncPredicate<>(this, other);
    }

    /*
    * Default操作，传入DefaultAsyncPredicate的是一个java8 Predicate，即
GatewayPredicate
    * Mono.just(delegate.test(t));
    */
    static AsyncPredicate<ServerWebExchange> from(
            Predicate<? super ServerWebExchange> predicate) {
        return new DefaultAsyncPredicate<>
(GatewayPredicate.wrapIfNeeded(predicate));
    }
}
```

GatewayFilter 与 GatewayFilterChain

> 拦截器，责任链设计模式

```
public interface GatewayFilter extends ShortcutConfigurable {

    String NAME_KEY = "name";
    String VALUE_KEY = "value";

    //当前拦截器处理
    Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain chain);
}
public interface GatewayFilterChain {

    //调用GatewayFilterChain#filter传递到下个Filter处理
    Mono<Void> filter(ServerWebExchange exchange);

}
```

Spring-Cloud-Gateway配置元信息

GatewayProperties

> “spring.cloud.gateway” 前缀的 properties 会绑定 到GatewayProperties

```
@ConfigurationProperties("spring.cloud.gateway")
@Validated
public class GatewayProperties {

    private final Log logger = LogFactory.getLog(getClass());

    //路由列表
    @NotNull
    @Valid
    private List<RouteDefinition> routes = new ArrayList<>();

    //默认拦截器，会作用于每个Route
    private List<FilterDefinition> defaultFilters = new ArrayList<>();

    //Http的text/event-stream与application/stream+json
    private List<MediaType> streamingMediaTypes = Arrays
            .asList(MediaType.TEXT_EVENT_STREAM,
MediaType.APPLICATION_STREAM_JSON);
}
```

RouteDefinition

> 对Route的信息定义，最终会被RouteLocator解析成Route

```
@Validated
public class RouteDefinition {

    private String id;
    //PredicateDefinition类型的谓语列表
    @NotEmpty
    @Valid
    private List<PredicateDefinition> predicates = new ArrayList<>();

    //FilterDefinition类型的拦截器列表
    @Valid
    private List<FilterDefinition> filters = new ArrayList<>();

    //路由目标uri，最终被转发的目的地
    @NotNull
    private URI uri;

    //元数据
    private Map<String, Object> metadata = new HashMap<>();

    //序号
    private int order = 0;
}
```

PredicateDefinition

> 对配置predicate的信息定义，最终会被解析成AsyncPredicate或Predicate

```
/*
* 配置示例：
* spring:
*  cloud:
*   gateway:
*     routes:
*     - id: add_request_header_route
*       uri: http://example.org
*       order: 999
*       predicates:
*       - Path=/echo,/test
*/
@Validated
public class PredicateDefinition {

    //对应Path
    @NotNull
    private String name;

    //key是固定字符串 _genkey_ + 数组元素下标，value是/echo和/test
    //key是根据构造方法的NameUtils.generateName(i)，而
NameUtils.GENERATED_NAME_PREFIX="_genkey_"
    private Map<String, String> args = new LinkedHashMap<>();
}
```

FilterDefinition

> 对配置filters的信息定义，最终会被解析成GatewayFilter

```
/*
* 配置示例：
* spring:
```

```
*   cloud:
*    gateway:
*      routes:
*      - id: add_request_header_route
*        uri: http://example.org
*        filters:
*        - AddRequestHeader=X-Request-Foo, Bar
*/
@Validated
public class FilterDefinition {

    //对应AddRequestHeader
    @NotNull
    private String name;

    //key是固定字符串 _genkey_ + 数组元素下标，value是X-Request-Foo和Bar
    private Map<String, String> args = new LinkedHashMap<>();
}
```

Predicate进一步理解

RoutePredicateFactory

> 所有 predicate factory 的顶级接口，职责就是生产 Predicate

```
@FunctionalInterface
public interface RoutePredicateFactory<C> extends ShortcutConfigurable,
Configurable<C> {

    /**
     * Pattern key.
     */
    String PATTERN_KEY = "pattern";

    //生成一个java8的Predicate
    default Predicate<ServerWebExchange> apply(Consumer<C> consumer) {
        C config = newConfig();
        consumer.accept(config);
        beforeApply(config);
        return apply(config);
    }

    //生产异步谓词AsyncPredicate，支持与、或、非操作
    default AsyncPredicate<ServerWebExchange> applyAsync(Consumer<C> consumer) {
        C config = newConfig();
        consumer.accept(config);
        beforeApply(config);
        return applyAsync(config);
    }

    default void beforeApply(C config) {
    }

    //由子类实现，RoutePredicateFactory有很多实现类，这也是它为什么可以支持Before/After某
个时间进行判断的原因
```

```
    Predicate<ServerWebExchange> apply(C config);

    //生成并把java8的Predicate转成AsyncPredicate
    default AsyncPredicate<ServerWebExchange> applyAsync(C config) {
        return toAsyncPredicate(apply(config));
    }

}
```

RoutePredicateFactory的常用实现类

- AfterRoutePredicateFactory：某个时间后的请求会匹配成功
- BeforeRoutePredicateFactory：某个时间前的请求会匹配成功
- BetweenRoutePredicateFactory：两个时间间的请求会匹配成功，开区间
- CookieRoutePredicateFactory：根据Cookie进行匹配
- HeaderRoutePredicateFactory：根据Http Header进行匹配
- HostRoutePredicateFactory：根据Host进行匹配
- MethodRoutePredicateFactory：根据Http请求方法进行匹配
- PathRoutePredicateFactory：根据请求路径进行匹配
- QueryRoutePredicateFactory：根据请求携带的参数进行匹配
- RemoteAddrRoutePredicateFactory：根据ip进行匹配

//TODO配置示例

Filter进一步理解

GatewayFilterFactory

> 所有 filter factory 的顶级接口，职责就是生产 GatewayFilter

```
@FunctionalInterface
public interface GatewayFilterFactory<C> extends ShortcutConfigurable,
Configurable<C> {

    /**
     * Name key.
     */
    String NAME_KEY = "name";

    /**
     * Value key.
     */
    String VALUE_KEY = "value";

    default GatewayFilter apply(String routeId, Consumer<C> consumer) {
        C config = newConfig();
        consumer.accept(config);
        return apply(routeId, config);
    }

    default GatewayFilter apply(Consumer<C> consumer) {
        C config = newConfig();
        consumer.accept(config);
        //调用子类实现
        return apply(config);
    }

    //子类实现
    GatewayFilter apply(C config);

    default GatewayFilter apply(String routeId, C config) {
        if (config instanceof HasRouteId) {
            HasRouteId hasRouteId = (HasRouteId) config;
            hasRouteId.setRouteId(routeId);
        }
        //调用子类实现
        return apply(config);
    }

}
```

GatewayFilterFactory常用实现类

- AddRequestHeaderGatewayFilterFactory：为请求Header添加指定值

  ```
  spring:
    cloud:
      gateway:
        routes:
        - id: add_request_header_route
          uri: http://example.org
          filters:
          - AddRequestHeader=X-Request-Foo, Bar
  ```

- RemoveRequestHeaderGatewayFilterFactory：为请求Header去掉指定值

  ```
  spring:
    cloud:
      gateway:
        routes:
        - id: removeresponseheader_route
          uri: http://example.org
          filters:
          - RemoveRequestHeader=X-Response-Foo
  ```

- AddResponseHeaderGatewayFilterFactory：为响应Header添加指定值

  ```
  spring:
    cloud:
      gateway:
        routes:
        - id: removeresponseheader_route
          uri: http://example.org
          filters:
          - AddResponseHeader=X-Response-Foo
  ```

- RemoveResponseHeaderGatewayFilterFactory：为响应Header去掉指定值

```
spring:
  cloud:
    gateway:
      routes:
      - id: removeresponseheader_route
        uri: http://example.org
        filters:
        - RemoveResponseHeader=X-Response-Foo
```

- SetResponseHeaderGatewayFilterFactory：对响应Header进行替换，没有则添加

```
spring:
  cloud:
    gateway:
      routes:
      - id: setresponseheader_route
        uri: http://example.org
        filters:
        - SetResponseHeader=X-Response-Foo, Bar
```

- RemoveNonProxyHeadersGatewayFilterFactory：移除请求 Proxy 相关的 Header的值，包括 Connection、Keep-Alive、Proxy、Authenticate、Proxy-Authorization、TE、Trailer、Transfer-Encoding、Upgrade

```
spring.cloud.gateway.filter.remove-non-proxy-headers.headers = Connection
```

- SecureHeadersGatewayFilterFactory：添加响应 Secure 相关的 Header

```
//Secure 相关的 Header 定义在 SecureHeadersGatewayFilterFactory 代码里
public class SecureHeadersGatewayFilterFactory extends
AbstractGatewayFilterFactory {

    /**
     * Xss-Protection header name.
     */
    public static final String X_XSS_PROTECTION_HEADER = "X-Xss-Protection";

    /**
     * Strict transport security header name.
     */
    public static final String STRICT_TRANSPORT_SECURITY_HEADER = "Strict-
Transport-Security";

    /**
     * Frame options header name.
     */
    public static final String X_FRAME_OPTIONS_HEADER = "X-Frame-Options";

    /**
     * Content-Type Options header name.
     */
    public static final String X_CONTENT_TYPE_OPTIONS_HEADER = "X-Content-
Type-Options";

    /**
```

```
     * Referrer Policy header name.
     */
    public static final String REFERRER_POLICY_HEADER = "Referrer-Policy";

    /**
     * Content-Security Policy header name.
     */
    public static final String CONTENT_SECURITY_POLICY_HEADER = "Content-
Security-Policy";

    /**
     * Download Options header name.
     */
    public static final String X_DOWNLOAD_OPTIONS_HEADER = "X-Download-
Options";

    /**
     * Permitted Cross-Domain Policies header name.
     */
    public static final String X_PERMITTED_CROSS_DOMAIN_POLICIES_HEADER =
"X-Permitted-Cross-Domain-Policies";
}
//使用示例
spring.cloud.gateway.filter.secure-headers=X-Xss-Protection
```

- AddRequestParameterGatewayFilterFactory：为请求添加参数

```
spring:
  cloud:
    gateway:
      routes:
      - id: add_request_parameter_route
        uri: http://example.org
        filters:
        - AddRequestParameter=foo, bar
```

- RewritePathGatewayFilterFactory：根据配置的正则表达式 regexp ，使用配置的 replacement 重写请求 Path

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

- PrefixPathGatewayFilterFactory：在请求路径前加上PrefixPath，重写请求 Path

```yaml
spring:
  cloud:
    gateway:
      routes:
      - id: prefixpath_route
        uri: http://example.org
        filters:
        - PrefixPath=/mypath
```

- SetPathGatewayFilterFactory：依然是重写请求 Path，/foo/bar -> /bar

```yaml
spring:
  cloud:
    gateway:
      routes:
      - id: setpath_route
        uri: http://example.org
        predicates:
        - Path=/foo/{segment}
        filters:
        - SetPath=/{segment}
```

- SetStatusGatewayFilterFactory：设置响应Http Status，如下配置无论什么情况，响应的http状态码为401

```yaml
spring:
  cloud:
    gateway:
      routes:
      - id: setstatusstring_route
        uri: http://example.org
        filters:
        - SetStatus=BAD_REQUEST
      - id: setstatusint_route
        uri: http://example.org
        filters:
        - SetStatus=401
```

- RedirectToGatewayFilterFactory：将响应重定向到指定 URL ，并设置响应状态码为指定 Status 。注意，Status 必须为 3XX 重定向状态码。

```yaml
spring:
  cloud:
    gateway:
      routes:
      - id: prefixpath_route
        uri: http://example.org
        filters:
        - RedirectTo=302, http://www.iocoder.cn
```

- HystrixGatewayFilterFactory：熔断网关过滤器工厂，详细源码研究//TODO

```
spring:
  cloud:
    gateway:
      routes:
      - id: default_path_to_httpbin
        uri: http://127.0.0.1:8081
        order: 10000
        predicates:
        - Path=/**
        filters:
        - Hystrix=myCommandName
```

- RequestRateLimiterGatewayFilterFactory：请求限流网关过滤器工厂

```
spring:
  cloud:
    gateway:
      routes:
      - id: default_path_to_httpbin
        uri: http://127.0.0.1:8081
        order: 10000
        predicates:
        - Path=/**
        filters:
        - RequestRateLimiter=10, 20, #{@principalNameKeyResolver}
//- RequestRateLimiter=10, 20, #{@principalNameKeyResolver}
// 第一个参数：令牌桶上限
// 第二个参数：令牌桶填充平均速率，单位：秒
// 第三个参数：限流键解析器 Bean 对象名字
```