## **[2. Glossary](https://docs.spring.io/spring-cloud-gateway/docs/2.2.5.RELEASE/reference/html/" \l "glossary)**

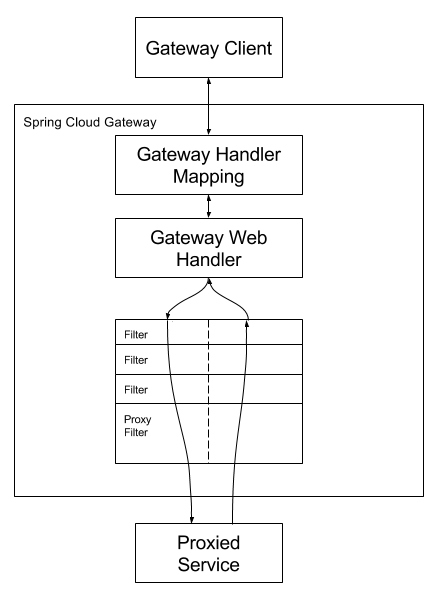
****Route****: The basic building block of the gateway. It is defined by an ID, a destination URI, a collection of predicates, and a collection of filters. A route is matched if the aggregate predicate is true.

****Predicate****: This is a [Java 8 Function Predicate](https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html). The input type is a [Spring Framework ServerWebExchange](https://docs.spring.io/spring/docs/5.0.x/javadoc-api/org/springframework/web/server/ServerWebExchange.html). This lets you match on anything from the HTTP request, such as headers or parameters.

****Filter****: These are instances of [Spring Framework GatewayFilter](https://docs.spring.io/spring/docs/5.0.x/javadoc-api/org/springframework/web/server/GatewayFilter.html) that have been constructed with a specific factory. Here, you can modify requests and responses before or after sending the downstream request.

## **[3. How It Works](https://docs.spring.io/spring-cloud-gateway/docs/2.2.5.RELEASE/reference/html/" \l "gateway-how-it-works)**

The following diagram provides a high-level overview of how Spring Cloud Gateway works:



Clients make requests to Spring Cloud Gateway. If the Gateway Handler Mapping determines that a request matches a route, it is sent to the Gateway Web Handler. This handler runs the request through a filter chain that is specific to the request. The reason the filters are divided by the dotted line is that filters can run logic both before and after the proxy request is sent. All “pre” filter logic is executed. Then the proxy request is made. After the proxy request is made, the “post” filter logic is run.

## **附录、应用举例**

自定义路由：

@Bean  
 public RouteLocator customRouteLocator(RouteLocatorBuilder builder, UriConfiguration uriConfiguration) {  
 // 使用负载均衡  
 // String uri = "lb://serviceName";

// 不使用负载均衡  
 // String uri = "http://localhost:9080";  
 String uri = uriConfiguration.getUri();  
  
 return builder.routes()

//单路径多filter  
 .route("test\_two\_filter", r -> r.path("/twofilter")  
 .filters(  
 f -> f.filter(new TestFilter())  
 .filter(new TestFilter())  
 )  
 .uri(uri)  
 )

//添加响应报文头

//修改响应报文体  
 .route("rewrite\_empty\_response", r -> r.path("/")  
 .filters(f -> f  
 .addResponseHeader("X-TestHeader", "rewrite\_empty\_response")  
 .modifyResponseBody(String.class, String.class,  
 (exchange, s) -> {  
 return Mono.*just*("Error!BecauseOfYouCannotAccessThisAddress!");  
 })  
  
 ).uri(uri)  
 )

//单路径单filter  
 .route("default\_path\_to\_httpbin1111",  
 r -> r.path("/")  
 .filters(  
 f -> f.filter(new AuthFilter())  
 )  
 .uri(uri)  
 )  
 // prefixPath 修改请求uri

// stripPrefix修改请求uri

.route("default\_path\_to\_httpbin2222",  
 r -> r.path("/api/\*\*")  
 .filters(f -> f.filter(new AuthFilter())

.prefixPath("/") //prefixPath就是在uri最前面加上指定字符串  
 .stripPrefix(1) //转发时去掉请求地址的服务名前缀  
 .addResponseHeader("X-TestHeader", "foobar2222"))  
 .uri(uri)  
 )  
 // 对domain和uri同时匹配  
 .route(r -> r.host("\*\*.abc.org").and().path("/anything/png")  
 .filters(f ->  
 f.prefixPath("/httpbin")  
 .addResponseHeader("X-TestHeader", "foobar"))  
 .uri(uri)  
 )

// 对domain和请求体同时匹配 *This predicate is BETA and may be subject to change in a future release. // predicate that checks the contents of the request body*

.route("read\_body\_pred", r -> r.host("\*.readbody.org")  
 .and().readBody(String.class,  
 s -> s.trim().equalsIgnoreCase("hi"))  
 .filters(f -> f.prefixPath("/httpbin")  
 .addResponseHeader("X-TestHeader", "read\_body\_pred")  
 ).uri(uri)  
 )  
 // 重写request  
 .route("rewrite\_request\_upper", r -> r.host("\*.rewriterequestupper.org")  
 .filters(f -> f.prefixPath("/httpbin")  
 .addResponseHeader("X-TestHeader", "rewrite\_request\_upper")  
 .modifyRequestBody(String.class, String.class,  
 (exchange, s) -> {  
 return Mono.*just*(s.toUpperCase() + s.toUpperCase());  
 })  
 ).uri(uri)  
 )

// 重写response  
 .route("rewrite\_response\_upper", r -> r.host("\*.rewriteresponseupper.org")  
 .filters(f -> f.prefixPath("/httpbin")  
 .addResponseHeader("X-TestHeader", "rewrite\_response\_upper")  
 .modifyResponseBody(String.class, String.class,  
 (exchange, s) -> {  
 return Mono.*just*(s.toUpperCase());  
 })  
 ).uri(uri)  
 )

// 使用自定义限流器  
 .route(r -> r.order(-1)  
 .host("\*\*.throttle.org").and().path("/get")  
 .filters(f -> f.prefixPath("/httpbin")  
 .filter(new ThrottleGatewayFilter()  
 .setCapacity(1)  
 .setRefillTokens(1)  
 .setRefillPeriod(10)  
 .setRefillUnit(TimeUnit.*SECONDS*)))  
 .uri(uri)  
 )  
 .build();  
 }

sentinel的用法：

官方文档： <https://github.com/alibaba/Sentinel/wiki/%E4%BB%8B%E7%BB%8D>

https://github.com/alibaba/Sentinel/wiki/%E5%A6%82%E4%BD%95%E4%BD%BF%E7%94%A8#Dashboard

sentinel-dashboard官方文档： <https://github.com/alibaba/Sentinel/wiki/%E6%8E%A7%E5%88%B6%E5%8F%B0>

系统全局过滤器执行顺序（名称、order）

RemoveCachedBodyFilter                        HIGHEST\_PRECEDENCE = Integer.MIN\_VALUE

AdaptCachedBodyGlobalFilter                  HIGHEST\_PRECEDENCE = Integer.MIN\_VALUE

NettyWriteResponseFilter                        -1

ForwardPathFilter                                      0

GatewayMetricsFilter                                0

RouteToRequestUrlFilter                         10000

WeightCalculatorWebFilter                     10001

LoadBalancerClientFilter                        10100

WebsocketRoutingFilter                        LOWEST\_PRECEDENCE -1

NettyRoutingFilter                                  LOWEST\_PRECEDENCE =Integer.MAX\_VALUE

ForwardRoutingFilter                             LOWEST\_PRECEDENCE =Integer.MAX\_VALUE