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

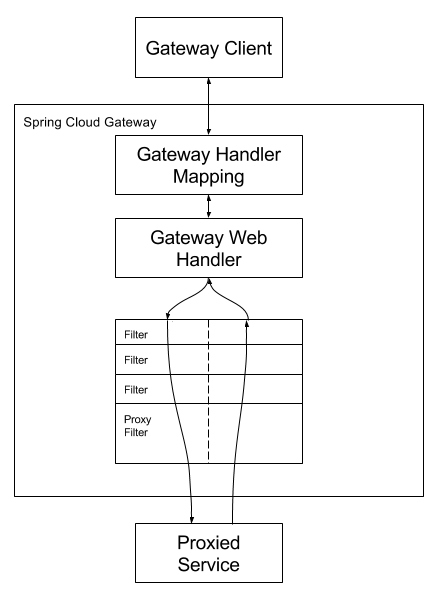
****Route****: The basic building block of the gateway. It is defined by an ID, a destination URI, a collection of predicates, and a collection of filters. A route is matched if the aggregate predicate is true.

****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**

1）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();  
 }

2）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>

3）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

4）查看规则列表：

<http://localhost:8719/getRules?type=flow>

[{"clusterConfig":{"fallbackToLocalWhenFail":true,"sampleCount":10,"strategy":0,"thresholdType":0,"windowIntervalMs":1000},"clusterMode":false,"controlBehavior":0,"count":0.0,"grade":1,"limitApp":"default","maxQueueingTimeMs":500,"resource":"default\_path\_to\_httpbin2222","strategy":0,"warmUpPeriodSec":10}]

<http://localhost:9990/v1/flow/rules?app=concurrent-gateway&ip=192.168.11.1&port=8719>

{"success":true,"code":0,"msg":"success","data":[{"id":7,"app":"concurrent-gateway","ip":"192.168.11.1","port":8719,"limitApp":"default","resource":"default\_path\_to\_httpbin2222","grade":1,"count":0.0,"strategy":0,"refResource":null,"controlBehavior":0,"warmUpPeriodSec":10,"maxQueueingTimeMs":500,"clusterMode":false,"clusterConfig":{"flowId":null,"thresholdType":0,"fallbackToLocalWhenFail":true,"strategy":0,"sampleCount":10,"windowIntervalMs":1000},"gmtCreate":null,"gmtModified":null}]}

5）规则持久化：

<https://github.com/alibaba/Sentinel/wiki/%E5%8A%A8%E6%80%81%E8%A7%84%E5%88%99%E6%89%A9%E5%B1%95>

6）Sentinel限流规则参数说明:

**Sentinel 1.6.0 引入了 Sentinel API Gateway Adapter Common 模块**，此模块中包含网关限流的规则和自定义 API 的实体和管理逻辑：

**1、GatewayFlowRule**：网关限流规则，**这个根据网关的自身的路由场景设计的**，可以针对不同 route 或自定义的 API 分组进行限流，**支持针对请求中的参数、Header、来源 IP 等进行定制化的限流。**

**2、ApiDefinition**：用户自定义的 API 定义分组，**可以看做是一些 URL 匹配的组合**。比如我们可以定义一个 API 叫 myapi，请求 path 模式为 /foo/\*\* 和 /baz/\*\* 的都归到 myapi 这个 API 分组下面。限流的时候可以针对**这个自定义的 API 分组维度进行限流**。

**网关限流规则 GatewayFlowRule**

# 

**字段解释**如下：

**1、resource：资源名称**

网关中的 route 名称或者用户自定义的API 分组名称。

**2、resourceMode**

规则是针对 **API Gateway 的route**（RESOURCEMODEROUTEID）还是用户在 Sentinel 中**自定义的API 分组**（RESOURCEMODECUSTOMAPI\_NAME），默认是route。

**3、grade**

**限流指标维度**，同限流规则的grade字段。

**4、count：限流阈值5、intervalSec**

统计时间窗口，单位是秒，默认是1 秒（目前仅对参数限流生效）。

**6、controlBehavior**

流量整形的控制效果，同限流规则的 controlBehavior 字段，目前支持**快速失败和匀速排队**两种模式，**默认是快速失败**。

**7、burst**

应对突发请求时额外允许的请求数目（目前仅对参数限流生效）。

**8、maxQueueingTimeoutMs**

匀速排队模式下的最长排队时间，单位是毫秒，仅在匀速排队模式下生效。

**9、paramItem参数限流配置**。若不提供，则代表不针对参数进行限流，该网关规则将会被转换成普通流控规则；否则会转换成热点规则。其中的字段：**parseStrategy**：**从请求中提取参数的策略**，目前支持四种模式提取来源 \* IP（PARAMPARSESTRATEGYCLIENTIP） \* Host（PARAMPARSESTRATEGYHOST） \* 任意 Header（PARAMPARSESTRATEGYHEADER） \* 任意 URL 参数（PARAMPARSESTRATEGYURLPARAM）

**fieldName**：若提取策略选择 Header 模式或 URL 参数模式，则需要指定对应的 header 名称或 URL 参数名称。**pattern 和 matchStrategy**：为参数匹配特性预留

可以通过 GatewayRuleManager.loadRules(rules)手动加载网关规则，或通过 GatewayRuleManager.register2Property(property)注册动态规则源动态推送（推荐方式）

**网关流控实现原理**

# **IMG_256**

上图的**整体流程**如下：

1、**外部请求进入API Gateway时会经过Sentinel实现的filter**，其中会依次进行 路由/API 分组匹配、请求属性解析和参数组装。

2、**Sentinel 会根据配置的网关流控规则来解析请求属性**，并依照参数索引顺序组装参数数组，最终传入SphU.entry(res, args) 中。

3、Sentinel API Gateway **Adapter Common 模块向 Slot Chain 中添加了一个 GatewayFlowSlot**，专门用来做网关规则的检查。

4、**GatewayFlowSlot会从GatewayRuleManager中提取生成的热点参数规则**，根据传入的参数依次进行规则检查。若某条规则不针对请求属性，则会在**参数最后一个位置置入预设的常量**，**达到普通流控的效果**。**注意：**

当通过 GatewayRuleManager 加载**网关流控规则GatewayFlowRule时**，无论是否针对请求属性进行限流，Sentinel底层都会将**网关流控规则转化为热点参数规则ParamFlowRule**，存储在GatewayRuleManager 中，与正常的热点参数规则相隔离。转换时Sentinel会根据请求属性配置，**为网关流控规则设置参数索引idx**，并同步到**生成的热点参数规则**中

[回到顶部](https://www.cnblogs.com/Soy-technology/p/11589016.html" \l "_labelTop)

## ****总结****

从Sentinel的**1.6.0版本**开始，**提供了SpringCloud Gateway的适配模块**，可以提供两种资源维度的限流：

1、**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

2、**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