# 基本概念

Spring Cloud Gateway是基于Spring生态系统之上构建的API网关，包括：Spring 5.x，Spring Boot 2.x和Project Reactor。Spring Cloud Gateway旨在提供一种简单而有效的方法来路由到API，并为它们提供跨领域的关注点，例如：安全性，监视/指标，限流等。

# 服务网关

API Gateway（APIGW / API 网关），顾名思义是系统对外的唯一入口。API网关封装了系统内部架构，为每个客户端提供定制的API。 近几年来移动应用与企业间互联需求的兴起。从以前单一的Web应用，扩展到多种使用场景，且每种使用场景对后台服务的要求都不尽相同。这不仅增加了后台服务的响应量，还增加了后台服务的复杂性。随着微服务架构概念的提出，API网关成为了微服务架构的一个标配组件。

# 为什么要使用网关

微服务的应用可能部署在不同机房，不同地区，不同域名下。此时客户端（浏览器/手机/软件工具）想要请求对应的服务，都需要知道机器的具体 IP 或者域名 URL，当微服务实例众多时，这是非常难以记忆的，对于客户端来说也太复杂难以维护。此时就有了网关，客户端相关的请求直接发送到网关，由网关根据请求标识解析判断出具体的微服务地址，再把请求转发到微服务实例，这其中的记忆功能就全部交由网关来操作了。

# 网关的作用

# 核心概念

## 路由（Route）

路由是网关最基础的部分，路由信息由 ID、目标 URI、一组断言和一组过滤器组成。如果断言路由为真，则说明请求的 URI 和配置匹配。

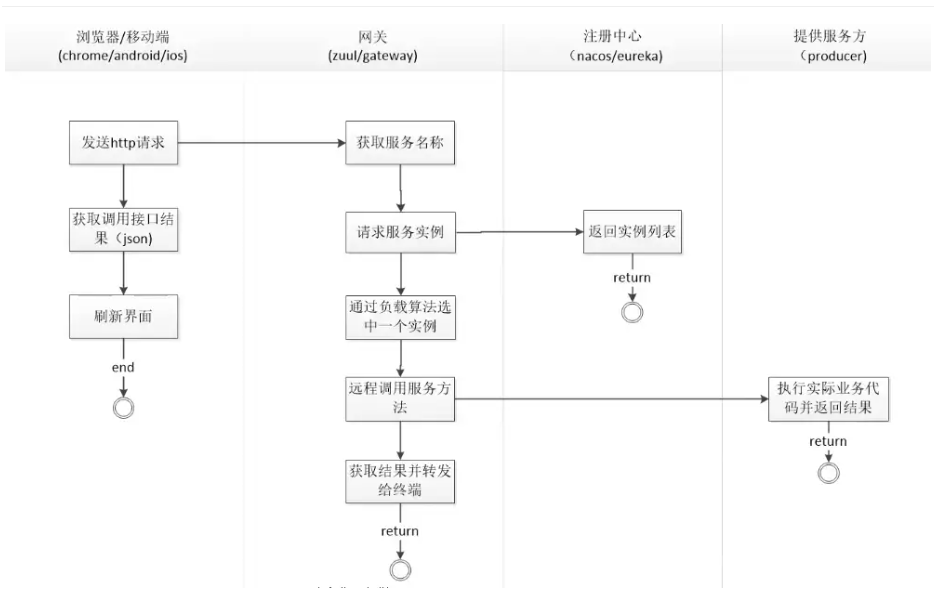
## 断言（Predicate）

Java8 中的断言函数。Spring Cloud Gateway 中的断言函数输入类型是 Spring 5.0 框架中的 ServerWebExchange。Spring Cloud Gateway 中的断言函数允许开发者去定义匹配来自于 Http Request 中的任何信息，比如请求头和参数等。

## 过滤器（Filter）

一个标准的 Spring Web Filter。Spring Cloud Gateway 中的 Filter 分为两种类型，分别是 Gateway Filter 和 Global Filter。过滤器将会对请求和响应进行处理。

# 网关活动图



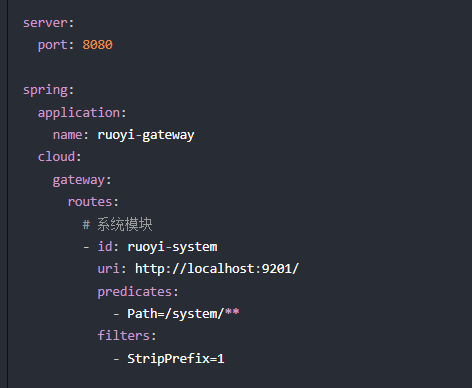
# 使用网关

## 1、添加依赖

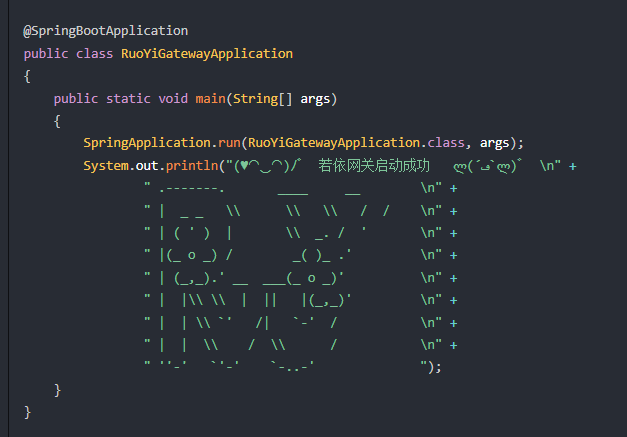


spring-cloud-starter-gateway

## 2、resources/application.yml配置文件



## 3、网关启动类



# 路由规则

## 基本概念

Spring Cloud Gateway创建Route对象时， 使用RoutePredicateFactory创建Predicate对象，Predicate对象可以赋值给Route。

1、Spring Cloud Gateway包含许多内置的Route Predicate Factories。

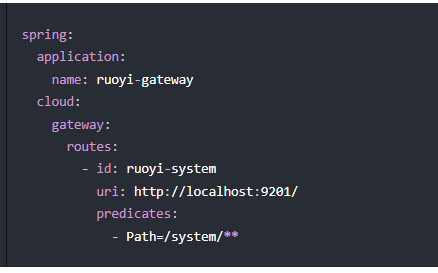
2、所有这些断言都匹配 HTTP 请求的不同属性。

3、多个Route Predicate Factories可以通过逻辑与（and）结合起来一起使用。

路由断言工厂RoutePredicateFactory包含的主要实现类包括Datetime、请求的远端地址、路由权重、请求头、Host 地址、请求方法、请求路径和请求参数等类型的路由断言。

## Path

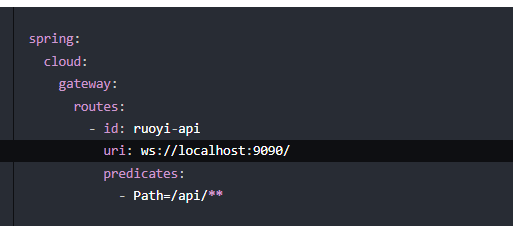
匹配请求路径。



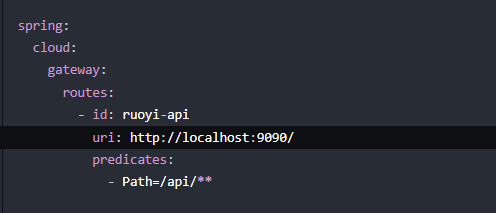
# 路由配置

在spring cloud gateway中配置uri有三种方式，包括：

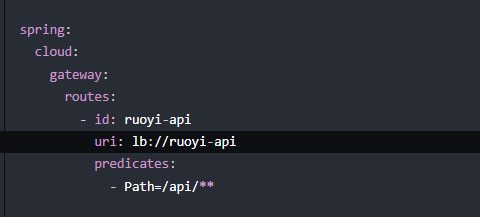
## websocker配置



## http地址配置



## 注册中心配置（重点）



### lowerCaseServiceId: true

配置项用于指定是否将服务 ID 转换为小写形式，当启用了 lowerCaseServiceId 配置后，Spring Cloud Gateway 将会在进行服务路由匹配时将服务 ID 转换为小写形式，以便进行大小写不敏感的匹配。

### StripPrefix=0

在Spring Cloud中，StripPrefix=0是一个用于路由配置的参数选项，用来指定是否从请求URL中去掉前缀路径。下面是对该参数的详细解释：

当我们构建微服务架构时，通常会使用API网关（如Spring Cloud Gateway、Zuul等）来转发客户端的请求到后端的微服务实例。在这个过程中，有时候我们需要对请求路径进行处理，以满足特定的业务需求。

StripPrefix就是其中之一的参数选项。它用于决定在将请求发送到后端服务时，是否从请求URL中删除前缀路径。下面是对StripPrefix=0的解释：

当StripPrefix=0时，表示不删除任何前缀路径。即原始请求中的前缀路径将被保留，不做任何修改。

当StripPrefix=1时，表示删除一个前缀路径。即原始请求中的第一个路径分段将被删除。例如，/api/users/123 将变成 /users/123。

当StripPrefix=2时，表示删除两个前缀路径。即原始请求中的前两个路径分段将被删除。例如，/api/v1/users/123 将变成 /users/123。

通过设置StripPrefix参数，我们可以根据具体情况来调整请求转发的路径，以适配后端微服务的路由规则和处理机制。需要注意的是，StripPrefix只在API网关中有效，用于定义请求转发的规则。不同的API网关实现和版本可能会稍有不同，具体的参数配置和使用方式可能会有所差异。

### RequestTrace=true

在Spring Cloud中，RequestTrace=true是一个用于请求跟踪的参数选项，用于启用请求追踪功能。下面是对该参数的详细解释：

当构建分布式系统并使用微服务架构时，了解请求的完整调用链非常重要。请求跟踪（Request Tracing）是一种技术，用于追踪和记录一个请求在分布式系统中的传递路径和执行过程。通过请求跟踪，我们可以获得请求在各个微服务之间的流转情况，并在必要时进行故障排查和性能优化。RequestTrace就是其中之一的参数选项。通过设置RequestTrace=true，我们可以启用请求跟踪功能，从而对请求进行完整的追踪和记录。启用请求跟踪后，每个接收到的请求都会被分配一个唯一的请求ID，它将随着请求在系统中传递，并在不同的微服务之间携带。每个微服务都会将自身的处理信息和相关的上下文（如开始时间、结束时间、耗时等）记录下来，并将其与请求ID关联起来。最终我们可以通过请求ID将整个请求的调用链串联起来，并进行分析和监控。需要注意的是，具体的请求跟踪实现可能会依赖于不同的组件或库，如Spring Cloud Sleuth、Zipkin等。因此在使用RequestTrace=true之前，需要确保请求跟踪的相关组件已正确配置和集成到系统中。

### PreserveHostHeader

用于指定是否保留原始请求的 Host 头部信息。默认情况下，Spring Cloud Gateway 会将传入请求的 Host 头部信息替换为网关的主机和端口。这样做是为了确保请求能够正确路由到后端服务。但在某些场景下，需要保留原始请求的 Host 头部信息，以便后端服务能够正确处理请求。

# 限流配置

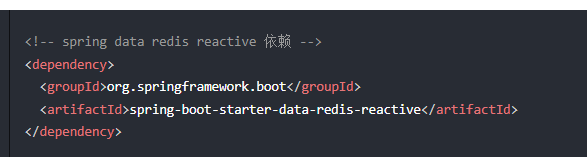
## 基本概念

顾名思义限流就是限制流量。通过限流我们可以很好地控制系统的 QPS，从而达到保护系统的目的。常见的限流算法有：计数器算法，漏桶（Leaky Bucket）算法，令牌桶（Token Bucket）算法。

Spring Cloud Gateway官方提供了RequestRateLimiterGatewayFilterFactory过滤器工厂，使用Redis 和Lua脚本实现了令牌桶的方式。

## 使用方法

### 1、添加依赖



spring-boot-starter-data-redis-reactive

### 2、限流规则，根据URI限流

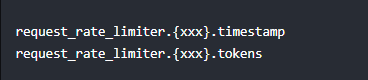


### 3、编写URI限流规则配置类



### 4、测试服务验证限流

启动网关服务RuoYiGatewayApplication.java和系统服务RuoYiSystemApplication.java。因为网关服务有认证鉴权，可以设置一下白名单/system/\*\*在进行测试，多次请求会发现返回HTTP ERROR 429，同时在redis中会操作两个key，表示限流成功。



### 5、其他限流规则

#### 参数限流

key-resolver: "#{@parameterKeyResolver}"

@Bean

public KeyResolver parameterKeyResolver()

{

return exchange -> Mono.just(exchange.getRequest().getQueryParams().getFirst("userId"));

}

#### IP限流

key-resolver: "#{@ipKeyResolver}"

@Bean

public KeyResolver ipKeyResolver()

{

return exchange -> Mono.just(exchange.getRequest().getRemoteAddress().getHostName());

}

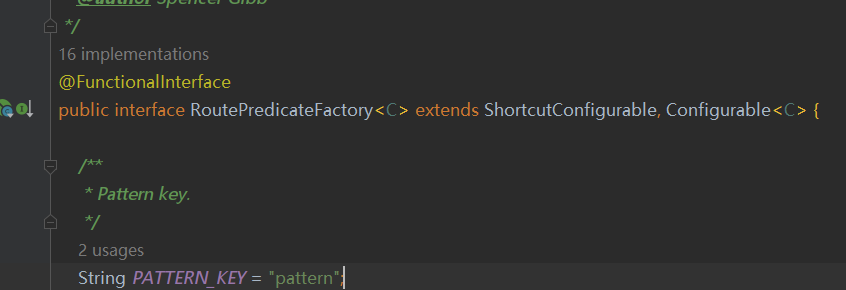
# 接口

## RoutePredicateFactory

### 基本概念

RoutePredicateFactory类是用于定义路由断言的工厂类。在Spring Cloud Gateway中，路由断言用于判断给定的请求是否满足某个条件，并根据条件的结果进行路由操作。

### 源码



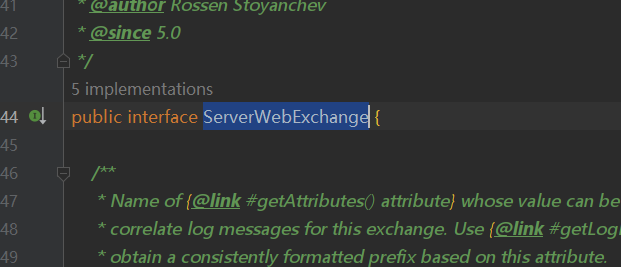
### 方法

#### default Predicate<ServerWebExchange> apply(Consumer<C> consumer)

## ServerWebExchange

### 基本概念

### 源码



### 方法

## ServerHttpRequest

## GatewayRuleManager

## GatewayFilter

## GlobalFilter

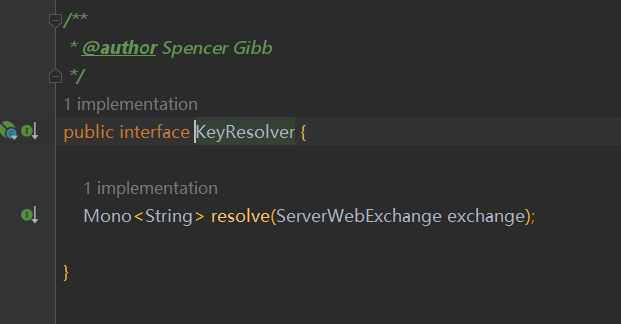
## GatewayFilterChain

## KeyResolver

### 基本概念

KeyResolver 接口用于定义从请求中提取限流或路由的键（Key）的策略。它是限流和路由功能中的一部分，用于根据请求的特征生成一个唯一的键，以便进行限流或路由的匹配。

### 源码



### 方法

#### Mono<String> resolve(ServerWebExchange exchange)

方法接收一个 ServerWebExchange 对象作为参数，表示当前的 HTTP 请求和响应。通过解析 ServerWebExchange 对象，KeyResolver 实现类可以从请求中提取出一个唯一的键。

### 示例代码



在上述示例中，我们实现了一个自定义的 KeyResolver 类，并重写了 resolve 方法。在该方法中，我们使用请求的 IP 地址作为键。

要在 Spring Cloud Gateway 中使用该 KeyResolver，你需要将它注册到限流或路由规则中。以下是一个配置示例：



在上述示例中，我们将自定义的 KeyResolver 注册为一个 Bean，并通过 key-resolver 参数指定了该解析器。

### 总结

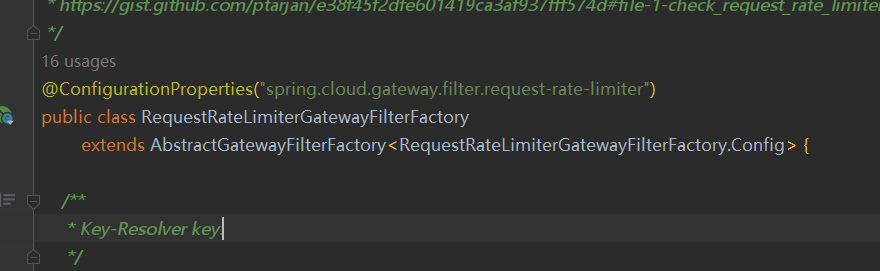
KeyResolver 接口用于定义从请求中提取限流或路由的键的策略。它允许根据请求的特征生成一个唯一的键，以便进行限流或路由的匹配。

# 类

## RequestRateLimiterGatewayFilterFactory

### 基本概念

### 源码

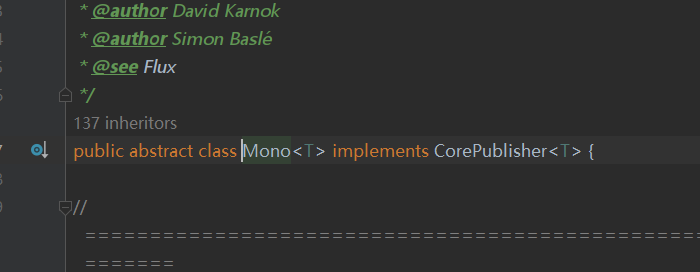


## Mono

### 基本概念

Mono是一个基于Reactor Mono的响应式编程库，用于在Spring应用程序中实现响应式编程。Reactor Mono是一个响应式流，它表示异步的单个结果值。Mono是Reactor核心库的一部分，也是响应式编程的重要组成部分。Mono可以通过将非阻塞代码包装到Mono对象中，来实现敏捷、可靠和高性能的应用程序。它提供了一种响应式的方式来处理异步任务，例如网络I/O、消息传递和数据库访问等，让开发者可以更容易地构建高效的事件驱动型应用程序。总之Spring Cloud Mono是一种用于实现响应式编程的库，它利用Reactor Mono来实现敏捷、可靠和高性能的应用程序，并提供了响应式的方式来处理异步任务。

### 源码



### 方法

## HttpClientProperties

### 基本概念

# 问题