微服务网关

课程讲义

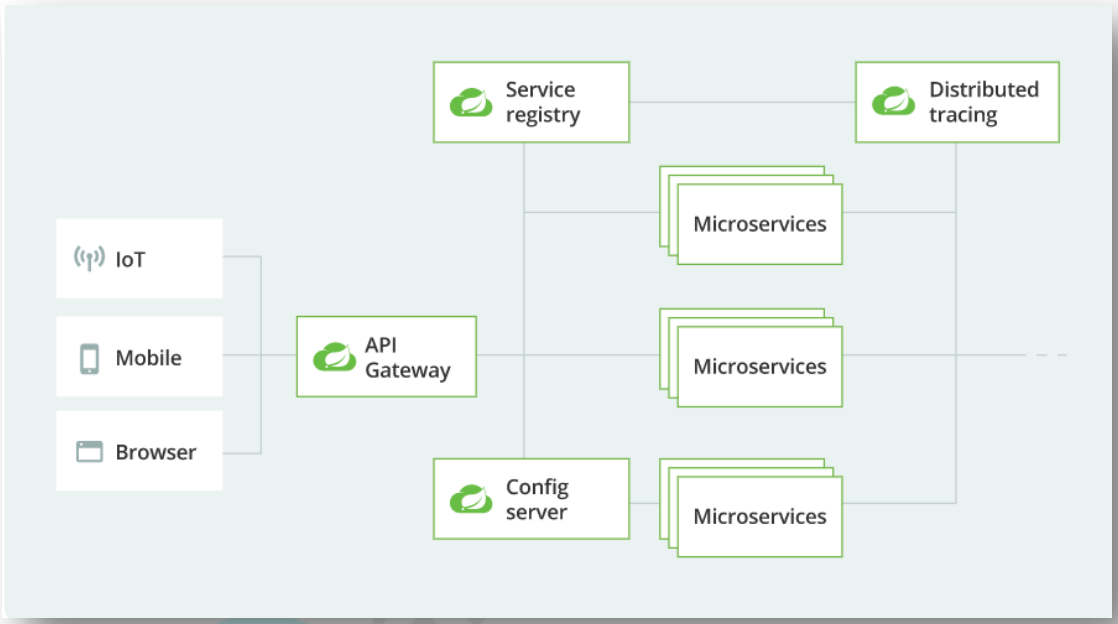
主讲：**Reythor 雷**

2021

微服务网关

1.1概述

1.1.1 网关简介



网关是系统唯一对外的入口，介于客户端与服务器端之间，用于对请求进行鉴权、限流、路由、监控等功能。

1.1.2 最常见的网关

（1） Kong

Kong 是一款基于 OpenResty 的高可用、易扩展由 Mashape 公司开源的 API Gateway 项目。OpenResty 是一款基于 Nginx + Lua 的网关。

（2） Zuul

Zuul1.0 是由 Netflix 开源的 API 网关，基于 servlet 的，使用阻塞 API，它不支持任何长连接。即一个线程处理一次连接请求，这种方式在内部延迟严重，在由于设备故障而引发存

活的连接增多的情况下，占用的线程数量急剧增加的情况。

Zuul2.0 的巨大区别是它运行在异步和无阻塞框架上，客户端与 Zuul 之间为长连接，每个 CPU 内核占用一个线程，通过 Reactor 模型处理所有连接上的请求和响应，请求和响应的生命周期是通过事件和回调来处理的，这种方式减少了线程数量，因此开销较小。

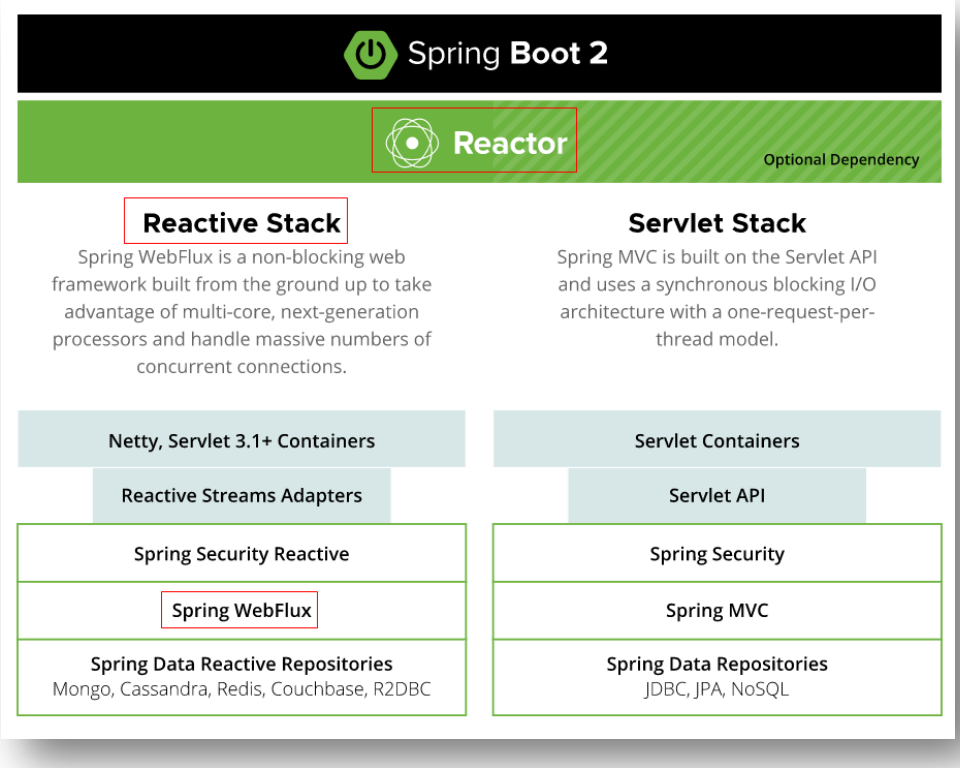
（3） Spring Cloud Gateway

Spring Cloud Gateway 是 Spring Cloud 自己开发的开源 API 网关，建立在 Spring5，Reactor 和 Spring Boot 2 之上，使用非阻塞 API，支持异步开发。是 Zuul 网关的替代者。

1.2 WebFlux 编程基础

Spring Cloud Gateway 无论是我们自己写代码，还是其框架源码，都是基于 WebFlux 进行开发的。所以我们在学习 Spring Cloud Gateway 之前就需要先搞清楚这些概念。

（1） Reactor、Reactive 及 WebFlux 间的关系



（2） Reactor

Reactor 是一种完全基于 Reactive Streams 规范的、全新的库。

（3） Reactive Streams

Reactive Streams 是 Reactive Programming 编程范式的 Java 规范，定义了 Reactive Programming 具体相关接口。

（4） Reactive Programming

Reactive Programming 是一种新的编程范式、编程思想。其不同于我们传统的开发范式。简单来说，其是一种基于流的开发模式。其就是一种编程思想，不是具体实现。

（5） RxJava

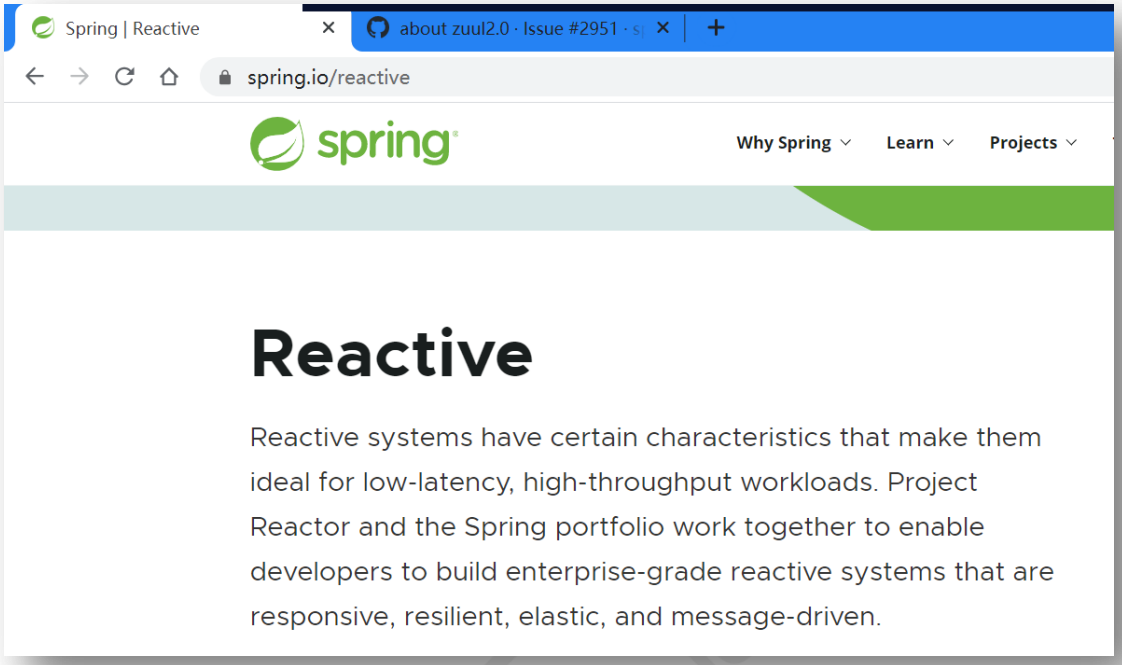
RxJava 也是一套 Reactive Streams 规范库。其是 Reactive Streams 规范的基础，产生于 Reactive Streams 规范之前。RxJava 并不好用。

Reactive Extensions(Rx)，原来是.Net 平台上基于 Reactive Programming 的规范。

（6） RxJava2

RxJava2 是 Reactive Streams 规范生成之后产生的，是基于 Reactive Streams 规范的，但同时为了兼容 RxJava，其又改变了原来 Reactive Streams 的一些内容，导致 RxJava2 仍然很不好用。

（7） Reactive



（8） WebFlux

Spring WebFlux 是一个使用 Reactive 技术栈构建出的框架。

（9） Flux 与 Mono

Flux：一个包含了 0 个或多个元素的异步序列 ——简单理解为一个集合
Mono：一个包含了 0 个或 1 个元素的异步序列 ——简单理解为一个对象

简单来说，Reactive Streams 规范就是对这个异步序列中的元素进行处理的。

1.3 Gateway 工作原理

1.3.1 重要概念

在 Spring Cloud Gateway 中有三个非常重要的概念：

A、 route 路由

路由是网关的最基本组成，由一个路由 id、一个目标地址 url，一组断言工厂及一组 filter 组成。若断言为 true，则请求将经由 filter 被路由到目标 url。

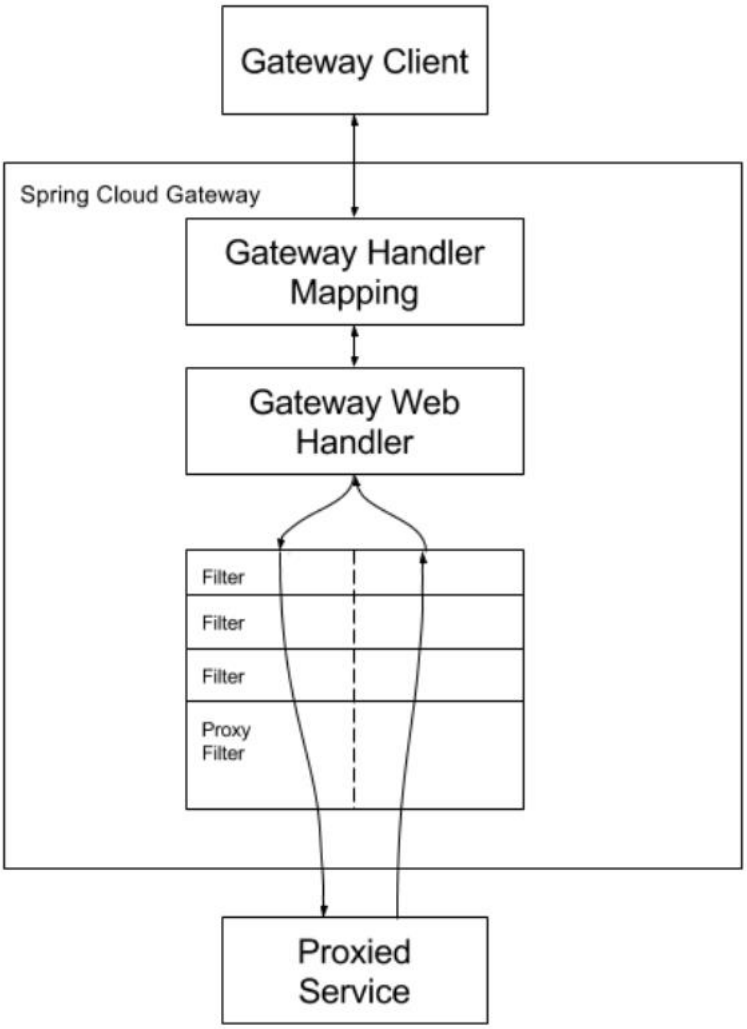
B、 predicate 断言

断言即一个条件判断，根据当前的 http 请求进行指定规则的匹配，比如说 http 请求头，请求时间等。只有当匹配上规则时，断言才为 true，此时请求才会被直接路由到目标地址（目标服务器），或先路由到某过滤器链，经过过程器链的层层处理后，再路由到相应的目标地址（目标服务器）。

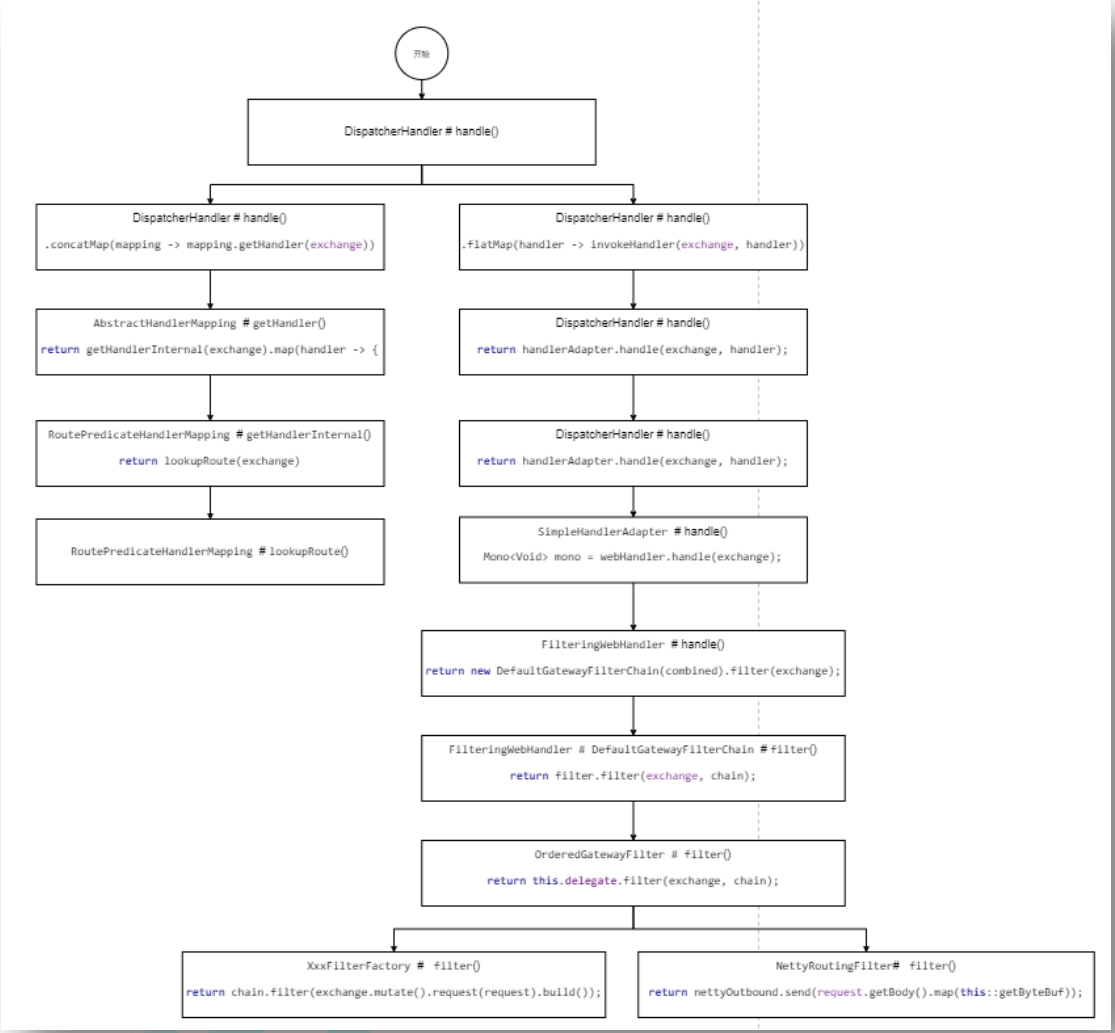
C、 filter 过滤器

对请求进行处理的逻辑部分。当请求的断言为 true 时，会被路由到设置好的过滤器，以对请求或响应进行处理。例如，可以为请求添加一个请求参数，或对请求 URI 进行修改，或为响应添加 header 等。总之，就是对请求或响应进行处理。

1.3.2 工作过程


Gateway Client
Spring Cloud Gateway
Gateway Handler Mapping
Gateway Web Handler
Filter
Filter
Filter
Proxy Filter
Proxied Service

1.4 Gateway 源码解析




```
        // 将一个集合构建为一个 Flux
return Flux.fromIterable(this.handlerMappings)
        // 将 Flux 中的元素由 mapping 变为 Handler
        .concatMap(mapping -> mapping.getHandler(exchange))
        // 从 Flux 中取出一个元素，其返回值为 Mono
        .next()
        // 若当前这个 Mono 中包含的元素为空，则将 createNotFoundError()的返回值作为元素
        .switchIfEmpty(createNotFoundError())
        // 将 Mono 的元素由 handler 变为 invokeHandler()的结果 handlerResult
        .flatMap(handler -> invokeHandler(exchange, handler))
        // 将 Mono 中的元素由 result 变为 handleResult()的结果
        .flatMap(result -> handleResult(exchange, result));
```