5 种 API 网关技术选型，yyds!

戳一戳→ 程序员的成长之路 2022-10-08 00:00 发表于广东

阅读本文大概需要 20 分钟。

来自：**https://developer.aliyun.com/article/889271**

本文准备围绕七个点来讲网关，分别是网关的基本概念、网关设计思路、网关设计重点、流量网关、业务网关、常见网关对比，对基础概念熟悉的朋友可以根据目录查看自己感兴趣的部分。

什么是网关

网关，很多地方将网关比如成门，没什么问题，但是需要区分网关与网桥的区别，

网桥 工作在数据链路层，在不同或相同类型的LAN之间存储并转发数据帧，必要时进行链路层上的协议转换。可连接两个或多个网络，在其中传送信息包。

网关 是一个大概念，不具体特指一类产品，只要连接两个不同的网络都可以叫网关，网桥一般只转发信息，而网关可能进行包装。

网关通俗理解

根据网关的特性，举个例子：

假如你要去找集团老板(这儿只是举个例子)，大家都知道老板肯定不是谁想见就能见的，也怕坏人嘛，那么你去老板所在的办公楼，假如是集团总部，大楼这个门就充当了网关的角色，大门一般都有看门员，看门员会做哪些事情呢？

首先所有想见老板的人肯定都得从这个门进(统一入口)，这个门相当于将办公室和外界隔离了，主要为了保护里面的安全以及正常工作，来到这个门之后，门卫肯定会让你出示相关证件(鉴权检验)，意思就是判断你要见老板这个请求是否合理，如果不合理直接就拒绝了，让你回家等消息，如果鉴权之后，发现你找老板其实只是为了和他谈谈两元店的生意，门卫会跟你说这个用不着找老板，你去集团投资部就行了(动态路由，将请求路由到不同的后端集群中)，此时会对你进行一些包装，例如给你出具一个访问证类似的，然后告诉你路该怎么走，等等。

你看看，网关的作用是不是就是这三个，最终目的就是减少你与集团的耦合，具体到计算机上就是减少客户端与服务端的耦合，如果没有网关意味着所有请求都会直接调用服务器上的资源，这样耦合太强了，服务器出了问题，客户端会直接报错，例如老板换工作的地方了，如果没有网关你直接去原来的地方找，肯定会被告知老板不在这儿。

为什么需要网关

当使用单体应用程序架构时，客户端（Web 或移动端）通过向后端应用程序发起一次 REST 调用来获取数据。负载均衡器将请求路由给 N 个相同的应用程序实例中的一个。然后应用程序会查询各种数据库表，并将响应返回给客户端。微服务架构下，单体应用被切割成多个微服务，如果将所有的微服务直接对外暴露，势必会出现安全方面的各种问题，另外内外耦合严重。

客户端可以直接向每个微服务发送请求，其问题主要如下：

- 客户端需求和每个微服务暴露的细粒度 API 不匹配。
- 部分服务使用的协议不是Web友好协议。可能使用 Thrift 二进制 RPC，也可能使用 AMQP 消息传递协议。
- 微服务难以重构。如果合并两个服务，或者将一个服务拆分成两个或更多服务，这类重构就非常困难了。

服务端的各个服务直接暴露给客户端调用势必会引起各种问题。同时，服务端的各个服务可扩展和伸缩性很差。API 网关是微服务架构中的基础组件，位于接入层之下和业务服务层之上，如前所述的这些功能适合在 API 网关实现。

网关与服务器集群

回到我们服务器上，下面图介绍了网关(Gateway)作用，可知 Gateway 方式下的架构，可以细到为每一个服务的实例配置一个自己的 Gateway，也可以粗到为

一组服务配置一个，甚至可以粗到为整个架构配置一个接入的 Gateway。于是，整个系统架构的复杂度就会变得简单可控起来。

这张图展示了一个多层 Gateway 架构，其中有一个总的 Gateway 接入所有的流量(流量网关)，并分发给不同的子系统，还有第二级 Gateway 用于做各个子系统的接入 Gateway(业务网关)。可以看到，网关所管理的服务粒度可粗可细。通过网关，我们可以把分布式架构组织成一个星型架构，由网络对服务的请求进行路由和分发。下面来聊聊好的网关应该具备哪些功能，也就是网关设计模式。

网关设计思路

一个网关需要有以下的功能:

1. 请求路由

网关一定要有请求路由的功能。这样一来，对于调用端来说，也是一件非常方便的事情。因为调用端不需要知道自己需要用到的其它服务的地址，全部统一地交给 Gateway 来处理。

2. 服务注册

为了能够代理后面的服务，并把请求路由到正确的位置上，网关应该有服务注册功能，也就是后端的服务实例可以把其提供服务的地址注册、取消注册。一般来说，注册也就是注册一些 API 接口。比如，HTTP 的 Restful 请求，可以注册

相应 API 的 URI、方法、HTTP 头。这样，Gateway 就可以根据接收到的请求中的信息来决定路由到哪一个后端的服务上。

3. 负载均衡

因为一个网关可以接收多个服务实例，所以网关还需要在各个对等的服务实例上做负载均衡策略。简单点就是直接 Round-Robin 轮询，复杂点的可以设置上权重进行分发，再复杂一点还可以做到 session 粘连。

4. 弹力设计

网关还可以把弹力设计中的那些异步、重试、幂等、流控、熔断、监视等都可以实现进去。这样，同样可以像 Service Mesh 那样，让应用服务只关心自己的业务逻辑（或是说数据面上的事）而不是控制逻辑（控制面）。

5. 安全方面

SSL 加密及证书管理、Session 验证、授权、数据校验，以及对请求源进行恶意攻击的防范。错误处理越靠前的位置就是越好，所以，网关可以做到一个全站的接入组件来对后端的服务进行保护。当然，网关还可以做更多更有趣的事情，比如：灰度发布、API聚合、API编排。

灰度发布

网关完全可以做到对相同服务不同版本的实例进行导流，还可以收集相关的数据。这样对于软件质量的提升，甚至产品试错都有非常积极的意义。

API 聚合

使用网关可以将多个单独请求聚合成一个请求。在微服务体系的架构中，因为服务变小了，所以一个明显的问题是，客户端可能需要多次请求才能得到所有的数据。这样一来，客户端与后端之间的频繁通信会对应用程序的性能和规模产生非常不利的影响。于是，我们可以让网关来帮客户端请求多个后端的服务（有些场景下完全可以并发请求），然后把后端服务的响应结果拼装起来，回传给客户端（当然，这个过程也可以做成异步的，但这需要客户端的配合）。

API 编排

同样在微服务的架构下，要走完一个完整的业务流程，我们需要调用一系列API，就像一种工作流一样，这个事完全可以通过网页来编排这个业务流程。我们可能通过一个 DSL 来定义和编排不同的 API，也可以通过像 AWS Lambda 服务那样的方式来串联不同的 API。

网关设计重点

网关设计重点主要是三个， 高性能、高可用、高扩展:

1. 高性能

在技术设计上，网关不应该也不能成为性能的瓶颈。对于高性能，最好使用高性能的编程语言来实现，如 C、C++、Go 和 Java。网关对后端的请求，以及对前端的请求的服务一定要使用异步非阻塞的 I/O 来确保后端延迟不会导致应用程序中出现性能问题。C 和 C++ 可以参看 Linux 下的 epoll 和 Windows 的 I/O Completion Port 的异步 IO 模型，Java 下如 Netty、Spring Reactor 的 NIO 框架。

2. 高可用

因为所有的流量或调用经过网关，所以网关必须成为一个高可用的技术组件，它的稳定直接关系到了所有服务的稳定。网关如果没有设计，就会成变一个单点故障。因此，一个好的网关至少要做到以下几点。

- 集群化 。网关要成为一个集群，其最好可以自己组成一个集群，并可以自己同步集群数据，而不需要依赖于一个第三方系统来同步数据。
- 服务化 。网关还需要做到在不间断的情况下修改配置，一种是像 Nginx reload 配置那样，可以做到不停服务，另一种是最好做到服务化。也就是说，得要有自己的 Admin API 来在运行时修改自己的配置。
- 持续化 。比如重启，就是像 Nginx 那样优雅地重启。有一个主管请求分发的主进程。当我们需要重启时，新的请求被分配到新的进程中，而老的进程处理完正在处理的请求后就退出。

3. 高扩展

因为网关需要承接所有的业务流量和请求，所以一定会有或多或少的业务逻辑。而我们都知道，业务逻辑是多变和不确定的。比如，需要在网关上加入一些和业务相关的东西。因此，一个好的 Gateway 还需要是可以扩展的，并能进行二次开发的。当然，像 Nginx 那样通过 Module 进行二次开发的固然可以。

另外，在运维方面，网关应该有以下几个设计原则。

- 业务松耦合，协议紧耦合。在业务设计上，网关不应与后面的服务之间形成服务耦合，也不应该有业务逻辑。网关应该是在网络应用层上的组件，不应该处理通讯协议体，只应该解析和处理通讯协议头。另外，除了服务发现外，网关不应该有第三方服务的依赖。
- 应用监视，提供分析数据。网关上需要考虑应用性能的监控，除了有相应后端服务的高可用的统计之外，还需要使用 Tracing ID 实施分布式链路跟踪，并统计好一定时间内每个 API 的吞吐量、响应时间和返回码，以便启动弹力设计中的相应策略。
- 用弹力设计保护后端服务。网关上一定要实现熔断、限流、重试和超时等弹力设计。如果一个或多个服务调用花费的时间过长，那么可接受超时并返回一部分数据，或是返回一个网关里的缓存的上一次成功请求的数据。你可以考虑一下这样的设计。
- **DevOps**。因为网关这个组件太关键了，所以需要 DevOps 这样的东西，将其发生故障的概率降到最低。这个软件需要经过精良的测试，包括功能和性能的测试，还有浸泡测试。还需要有一系列自动化运维的管控工具。

网关设计注意事项

1. 不要在网关中的代码里内置聚合后端服务的功能，而应考虑将聚合服务放在网关核心代码之外。可以使用 Plugin 的方式，也可以放在网关后面形成一个 Serverless 服务。
2. 网关应该靠近后端服务，并和后端服务使用同一个内网，这样可以保证网关和后端服务调用的低延迟，并可以减少很多网络上的问题。这里多说一句，网关处理的静态内容应该靠近用户（应该放到 CDN 上），而网关和此时的动态服务应该靠近后端服务。
3. 网关也需要做容量扩展，所以需要成为一个集群来分担前端带来的流量。这一点，要么通过 DNS 轮询的方式实现，要么通过 CDN 来做流量调度，或者通过更为底层的性能更高的负载均衡设备。
4. 对于服务发现，可以做一个时间不长的缓存，这样不需要每次请求都去查一下相关的服务所在的地方。当然，如果你的系统不复杂，可以考虑把服务发现的功能直接集成进网关中。
5. 为网关考虑 bulkhead 设计方式。用不同的网关服务不同的后端服务，或是用不同的网关服务前端不同的客户。

另外，因为网关是为用户请求和后端服务的桥接装置，所以需要考虑一些安全方面的事宜。具体如下：

1. 加密数据。可以把 SSL 相关的证书放到网关上，由网关做统一的 SSL 传输管理。
2. 校验用户的请求。一些基本的用户验证可以放在网关上来做，比如用户是否已登录，用户请求中的 token 是否合法等。但是，我们需要权衡一下，网关是否需要校验用户的输入。因为这样一来，网关就需要从只关心协议头，到需要关心协议体。而协议体中的东西一方面不像协议头是标准的，另一方面解析协议体还要耗费大量的运行时间，从而降低网关的性能。对此，我想说的是，看具体需求，一方面如果协议体是标准的，那么可以干；另一方面，对于解析协议所带来的性能问题，需要做相应的隔离。
3. 检测异常访问。网关需要检测一些异常访问，比如，在一段比较短的时间内请求次数超过一定数值；还比如，同一客户端的 4xx 请求出错率太高……对于这样的一些请求访问，网关一方面要把这样的请求屏蔽掉，另一方面需要发出警告，有可能会是一些比较重大的安全问题，如被黑客攻击。

流量网关

流量网关，顾名思义就是控制流量进入集群的网关，有很多工作需要在这一步做，对于一个服务集群，势必有很多非法的请求或者无效的请求，这时候要将请求拒之门外，降低集群的流量压力。

定义全局性的、跟具体的后端业务应用和服务完全无关的策略网关就是上图所示的架构模型——流量网关。流量网关通常只专注于全局的Api管理策略，比如全局

流量监控、日志记录、全局限流、黑白名单控制、接入请求到业务系统的负载均衡等，有点类似防火墙。**Kong** 就是典型的流量网关。

下面是kong的架构图，来自官网：https://konghq.com

这里需要补充一点的是，业务网关一般部署在流量网关之后、业务系统之前，比流量网关更靠近业务系统。通常API网指的是业务网关。有时候我们也会模糊流量网关和业务网关，让一个网关承担所有的工作，所以这两者之间并没有严格的界线。

业务网关

当一个单体应用被拆分成许许多多的微服务应用后，也带来了一些问题。一些与业务非强相关的功能，比如权限控制、日志输出、数据加密、熔断限流等，每个微服务应用都需要，因此存在着大量重复的代码实现。而且由于系统的迭代、人员的更替，各个微服务中这些功能的实现细节出现了较大的差异，导致维护成本变高。另一方面，原先单体应用下非常容易做的接口管理，在服务拆分后没有了一个集中管理的地方，无法统计已存在哪些接口、接口定义是什么、运行状态如何。

网关就是为了解决上述问题。作为微服务体系中的核心基础设施，一般需要具备接口管理、协议适配、熔断限流、安全防护等功能，各种开源的网关产品（比如zuul）都提供了优秀高可扩展性的架构、可以很方便的实现我们需要的一些功能、比如鉴权、日志监控、熔断限流等。

与流量网关相对应的就是业务网关，业务网关更靠近我们的业务，也就是与服务器应用层打交道，那么有很多应用层需要考虑的事情就可以依托业务网关，例如在线程模型、协议适配、熔断限流，服务编排等。下面看看业务网关体系结构:

从这个途中可以看出业务网关主要职责以及所做的事情， 目前业务网关比较成熟的 API 网关框架产品有三个 分别是:Zuul1、Zuul2 和 SpringCloud Gateway， 后面再进行对比。

常见网关对比

既然对比，就先宏观上对各种网关有一个了解，后面再挑一些常用的或者说应用广泛的详细了解。

目前常见的开源网关大致上按照语言分类有如下几类：

- **Nginx+lua**：OpenResty、Kong、Orange、Abtesting gateway 等
- **Java**：Zuul/Zuul2、Spring Cloud Gateway、Kaazing KWG、gravitee、Dromara soul 等
- **Go**：Janus、fagongzi、Grpc-gateway
- **Dotnet**：Ocelot
- **NodeJS**：Express Gateway、Micro Gateway

按照使用数量、成熟度等来划分，主流的有 5个：

- OpenResty
- Kong
- Zuul、Zuul2
- Spring Cloud Gateway

1. OpenResty

OpenResty是一个流量网关，根据前面对流量网关的介绍就可以知道流量网关的指责。

OpenResty基于 Nginx与 Lua 的高性能 Web 平台，其内部集成了大量精良的 Lua 库、第三方模块以及大多数的依赖项。用于方便地搭建能够处理超高并发、扩展性极高的动态 Web 应用、Web 服务和动态网关。

通过揉和众多设计良好的 Nginx 模块，OpenResty 有效地把 Nginx 服务器转变为一个强大的 Web 应用服务器，基于它开发人员可以使用 Lua 编程语言对 Nginx 核心以及现有的各种 Nginx C 模块进行脚本编程，构建出可以处理一万以上并发请求的极端高性能的 Web 应用

OpenResty 最早是顺应 OpenAPI 的潮流做的，所以 Open 取自“开放”之意，而Resty便是 REST 风格的意思。虽然后来也可以基于 ngx_openresty 实现任何形式的 web service 或者传统的 web 应用。

也就是说 Nginx 不再是一个简单的静态网页服务器，也不再是一个简单的反向代理了。第二代的 openresty 致力于通过一系列 nginx 模块，把nginx扩展为全功能的 web 应用服务器。

ngx_openresty 是用户驱动的项目，后来也有不少国内用户的参与，从 openresty.org 的点击量分布上看，国内和国外的点击量基本持平。

ngx_openresty 目前有两大应用目标：

1. 通用目的的 web 应用服务器。在这个目标下，现有的 web 应用技术都可以算是和 OpenResty 或多或少有些类似，比如 Nodejs， PHP 等等。ngx_openresty 的性能（包括内存使用和 CPU 效率）算是最大的卖点之一。
2. Nginx 的脚本扩展编程，用于构建灵活的 Web 应用网关和 Web 应用防火墙。有些类似的是 NetScaler。其优势在于 Lua 编程带来的巨大灵活性。

2. Kong

Kong基于OpenResty开发，也是流量层网关， 是一个云原生、快速、可扩展、分布式的Api 网关。继承了OpenResty的高性能、易扩展性等特点。Kong通过简单的增加机器节点，可以很容易的水平扩展。同时功能插件化，可通过插件来扩展其能力。而且在任何基础架构上都可以运行。具有以下特性：

- 提供了多样化的认证层来保护Api。
- 可对出入流量进行管制。
- 提供了可视化的流量检查、监视分析Api。
- 能够及时的转换请求和相应。
- 提供log解决方案
- 可通过api调用Serverless 函数。

Kong解决了什么问题

当我们决定对应用进行微服务改造时，应用客户端如何与微服务交互的问题也随之而来，毕竟服务数量的增加会直接导致部署授权、负载均衡、通信管理、分析和改变的难度增加。

面对以上问题，API GATEWAY是一个不错的解决方案，其所提供的访问限制、安全、流量控制、分析监控、日志、请求转发、合成和协议转换功能，可以解放开发者去把精力集中在具体逻辑的代码，而不是把时间花费在考虑如何解决应用和其他微服务链接的问题上。

图片来自Kong官网：

可以看到Kong解决的问题。专注于全局的Api管理策略，全局流量监控、日志记录、全局限流、黑白名单控制、接入请求到业务系统的负载均衡等。

Kong的优点以及性能

在众多 API GATEWAY 框架中，Mashape 开源的高性能高可用API网关和API服务管理层——KONG（基于 NGINX+Lua）特点尤为突出，它可以通过插件扩展已有功能，这些插件（使用 lua 编写）在API请求响应循环的生命周期中被执行。于此同时，KONG本身提供包括 HTTP 基本认证、密钥认证、CORS、TCP、UDP、文件日志、API请求限流、请求转发及 NGINX 监控等基本功能。目前，Kong 在 Mashape 管理了超过 15，000 个 API，为 200，000 开发者提供了每月数十亿的请求支持。

Kong架构

Kong提供一些列的服务，这就不得不谈谈内部的架构:

首先最底层是基于Nginx， Nginx是高性能的基础层， 一个良好的负载均衡、反向代理器，然后在此基础上增加Lua脚本库，形成了OpenResty，拦截请求， 响应生命周期，可以通过Lua编写脚本，所以插件比较丰富。

> “
>
> 关于Kong的一些插件库以及如何配置，可以参考简书:开源API网关系统（Kong教程）入门到精通：https://www.jianshu.com/p/a68e45bcadb6
>
> ”

3. Zuul1.0

Zuul是所有从设备和web站点到Netflix流媒体应用程序后端请求的前门。作为一个边缘服务应用程序，Zuul被构建来支持动态路由、监视、弹性和安全性。它还可以根据需要将请求路由到多个Amazon自动伸缩组。

Zuul使用了一系列不同类型的过滤器，使我们能够快速灵活地将功能应用到服务中。

过滤器

过滤器是Zuul的核心功能。它们负责应用程序的业务逻辑，可以执行各种任务。

- **Type**：通常定义过滤器应用在哪个阶段
- **Async**：定义过滤器是同步还是异步
- **Execution Order**：执行顺序
- **Criteria**：过滤器执行的条件
- **Action**：如果条件满足，过滤器执行的动作

Zuul提供了一个动态读取、编译和运行这些过滤器的框架。过滤器之间不直接通信，而是通过每个请求特有的RequestContext共享状态。

下面是Zuul的一些过滤器:

Incoming

Incoming过滤器在请求被代理到Origin之前执行。这通常是执行大部分业务逻辑的地方。例如:认证、动态路由、速率限制、DDoS保护、指标。

Endpoint

Endpoint过滤器负责基于incoming过滤器的执行来处理请求。Zuul有一个内置的过滤器（ProxyEndpoint），用于将请求代理到后端服务器，因此这些过滤器的典型用途是用于静态端点。例如:健康检查响应，静态错误响应，404响应。

Outgoing

Outgoing过滤器在从后端接收到响应以后执行处理操作。通常情况下，它们更多地用于形成响应和添加指标，而不是用于任何繁重的工作。例如:存储统计信息、添加/剥离标准标题、向实时流发送事件、gziping响应。

过滤器类型

下面是与一个请求典型的生命周期对应的标准的过滤器类型：

- **PRE**：路由到Origin之前执行
- **ROUTING**：路由到Origin期间执行
- **POST**：请求被路由到Origin之后执行
- **ERROR**：发生错误的时候执行

这些过滤器帮助我们执行以下功能：

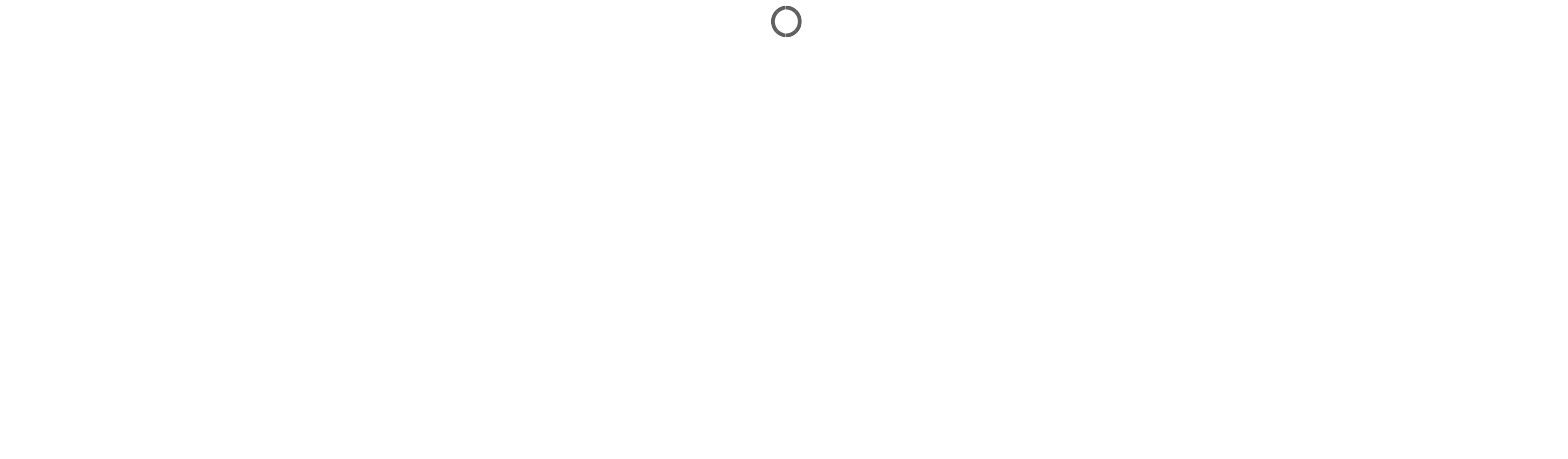
- 身份验证和安全性：识别每个资源的身份验证需求，并拒绝不满足它们的请求
- 监控：在边缘跟踪有意义的数据和统计数据，以便给我们一个准确的生产视图
- 动态路由：动态路由请求到不同的后端集群
- 压力测试：逐渐增加集群的流量，以评估性能
- 限流：为每种请求类型分配容量，并丢弃超过限制的请求
- 静态响应处理：直接在边缘构建一些响应，而不是将它们转发到内部集群

Zuul 1.0 请求生命周期

Netflix宣布了通用API网关Zuul的架构转型。Zuul原本采用同步阻塞架构，转型后叫作Zuul2，采用异步非阻塞架构。Zuul2和Zuul1在架构方面的主要区别在于，Zuul2运行在异步非阻塞的框架上，比如Netty。Zuul1依赖多线程来支持吞吐量的增长，而Zuul 2使用的Netty框架依赖事件循环和回调函数。

4. Zuul2.0

Zuul 2.0 架构图



上图是Zuul2的架构，和Zuul1没有本质区别，两点变化：

1. 前端用Netty Server代替Servlet，目的是支持前端异步。后端用Netty Client代替Http Client，目的是支持后端异步。
2. 过滤器换了一下名字，用Inbound Filters代替Pre-routing Filters，用Endpoint Filter代替Routing Filter，用Outbound Filters代替Post-routing Filters。

Inbound Filters：路由到 Origin 之前执行，可以用于身份验证、路由和装饰请求

Endpoint Filters：可用于返回静态响应，否则内置的ProxyEndpoint过滤器将请求路由到Origin

Outbound Filters：从Origin那里获取响应后执行，可以用于度量、装饰用户的响应或添加自定义header

有两种类型的过滤器：sync 和 async。因为Zuul是运行在一个事件循环之上的，因此从来不要在过滤中阻塞。如果你非要阻塞，可以在一个异步过滤器中这样做，并且在一个单独的线程池上运行，否则可以使用同步过滤器。

上文提到过**Zuul2**开始采用了异步模型

优势 是异步非阻塞模式启动的线程很少，基本上一个CPU core上只需启一个事件环处理线程，它使用的线程资源就很少，上下文切换(Context Switch)开销也少。非阻塞模式可以接受的连接数大大增加，可以简单理解为请求来了只需要进队列，这个队列的容量可以设得很大，只要不超时，队列中的请求都会被依次处理。

不足，异步模式让编程模型变得复杂。一方面Zuul2本身的代码要比Zuul1复杂很多，Zuul1的代码比较容易看懂，Zuul2的代码看起来就比较费劲。另一方面异步模型没有一个明确清晰的请求->处理->响应执行流程(call flow)，它的流程是通过事件触发的，请求处理的流程随时可能被切换断开，内部实现要通过一些关联id机制才能把整个执行流再串联起来，这就给开发调试运维引入了很多复杂性，比如你在IDE里头调试异步请求流就非常困难。另外ThreadLocal机制在这种异步模式下就不能简单工作，因为只有一个事件环线程，不是每个请求一个线程，也就没有线程局部的概念，所以对于CAT这种依赖于ThreadLocal才能工作的监控工具，调用链埋点就不好搞(实际可以工作但需要进行特殊处理)。

总体上，异步非阻塞模式比较适用于IO密集型(IO bound)场景，这种场景下系统大部分时间在处理IO，CPU计算比较轻，少量事件环线程就能处理。

Zuul 与 Zuul 2 性能对比

Netflix给出了一个比较模糊的数据，大致Zuul2的性能比Zuul1好20%左右，这里的性能主要指每节点每秒处理的请求数。为什么说模糊呢？因为这个数据受实际测试环境，流量场景模式等众多因素影响，你很难复现这个测试数据。即便这个20%的性能提升是确实的，其实这个性能提升也并不大，和异步引入的复杂性相比，这20%的提升是否值得是个问题。Netflix本身在其博文22和ppt11中也是有点含糊其词，甚至自身都有一些疑问的。

5. Spring Cloud Gateway

SpringCloud Gateway 是 Spring Cloud 的一个全新项目，该项目是基于 Spring 5.0，Spring Boot 2.0 和 Project Reactor 等技术开发的网关，它旨在为微服务架构提供一种简单有效的统一的 API 路由管理方式。

SpringCloud Gateway 作为 Spring Cloud 生态系统中的网关，目标是替代 Zuul，在Spring Cloud 2.0以上版本中，没有对新版本的Zuul 2.0以上最新高性能版本进行集成，仍然还是使用的Zuul 2.0之前的非Reactor模式的老版本。而为了提升网关的性能，SpringCloud Gateway是基于WebFlux框架实现的，而WebFlux框架底层则使用了高性能的Reactor模式通信框架Netty。

Spring Cloud Gateway 的目标，不仅提供统一的路由方式，并且基于 Filter 链的方式提供了网关基本的功能，例如：安全，监控/指标，和限流。

Spring Cloud Gateway 底层使用了高性能的通信框架**Netty** 。

SpringCloud Gateway 特征

SpringCloud官方，对SpringCloud Gateway 特征介绍如下：

（1）基于 Spring Framework 5，Project Reactor 和 Spring Boot 2.0

（2）集成 Hystrix 断路器

（3）集成 Spring Cloud DiscoveryClient

（4）Predicates 和 Filters 作用于特定路由，易于编写的 Predicates 和 Filters

（5）具备一些网关的高级功能：动态路由、限流、路径重写

从以上的特征来说，和Zuul的特征差别不大。SpringCloud Gateway和Zuul主要的区别，还是在底层的通信框架上。

简单说明一下上文中的三个术语：

Filter（过滤器）

和Zuul的过滤器在概念上类似，可以使用它拦截和修改请求，并且对上游的响应，进行二次处理。过滤器为org.springframework.cloud.gateway.filter.GatewayFilter类的实例。

Route（路由）

网关配置的基本组成模块，和Zuul的路由配置模块类似。一个**Route**模块 由一个ID，一个目标 URI，一组断言和一组过滤器定义。如果断言为真，则路由匹配，目标URI会被访问。

Predicate（断言）：

这是一个 Java 8 的 Predicate，可以使用它来匹配来自 HTTP 请求的任何内容，例如 headers 或参数。断言的 输入类型是一个 ServerWebExchange。

几种网关的对比

<END>

推荐阅读：

为什么某互联网企业开 200w 年薪大家都支持，但是中金开 100w 年薪大家都在骂？

MyBatis的三种分页方式，你用过几种？

互联网初中高级大厂面试题(9个G)

内容包含Java基础、JavaWeb、MySQL性能优化、JVM、锁、百万并发、消息队列、高性能缓存、反射、Spring全家桶原理、微服务、Zookeeper.....等技术栈！

⬇戳阅读原文领取！

朕已阅

阅读原文

喜欢此内容的人还喜欢

React 可组合 API 的设计原则

KooFE前端团队

协程简史，一文讲清楚协程的起源、发展和实现

郭霖

一款Linux、数据库、Redis、MongoDB统一管理平台，有点牛逼了！

开源前线