

百亿流量 API 网关设计与实践

百亿流量 API 网关设计与实践

第一部分：API 网关概述

分布式服务架构、微服务架构与 API 网关

什么是 API 网关 (API Gateway)

分布式服务架构、微服务架构与 API 网关

API 网关的技术趋势

API 网关的定义、职能与关注点

API 网关的定义

API 网关的职能

API 网关的关注点

API 网关的分类与技术分析

API 网关的分类

流量网关与 WAF

业务网关

第二部分：开源网关的分析与调研

常见的开源网关介绍

Nginx+Lua

Java

Go

Dotnet

NodeJS

四大开源网关的对比分析

(OpenResty/Kong/Zuul2/SpringCloudGateway 等)

OpenResty/Kong/Zuul2/SpringCloudGateway 重要特性对比

OpenResty/Kong/Zuul2/SpringCloudGateway 性能测试对比

开源网关的技术总结

开源网关的测试分析

各类网关的 demo 与测试

第三部分：百亿流量交易系统 API 网关设计

百亿流量交易系统 API 网关的现状和面临问题

百亿流量系统面对的业务现状

网关系统与其他系统的关系

网关系统典型的应用场景

交易系统 API 的特点

交易系统 API 网关面临的问题

业务网关的设计与最佳实践

API 网关 1.0

API 网关 2.0

API 网关的日常监控

推荐外部客户使用 Websocket

API 网关的性能优化

对 API 网关的发展展望

本次分享我们从百亿流量交易系统 API 网关 (API Gateway) 的现状和面临问题出发，阐述微服务架构与 API 网关的关系，理顺流量网关与业务网关的脉络，带来最全面的 API 网关知识与经验。内容涉及：

第一部分：API 网关概述

- 分布式服务架构、微服务架构与 API 网关
- API 网关的定义与职能、关注点
- API 网关的分类与技术分析

第二部分：开源网关的分析与调研

- 常见的开源网关介绍
- 四大开源网关的对比分析 (OpenResty/Kong/Zuul2/SpringCloudGateway 等)
- 开源网关的技术总结

第三部分：百亿流量交易系统 API 网关设计

- 百亿流量 API 网关的现状和面临问题
- 业务网关的设计与最佳实践
- 对 API 网关的发展展望

第一部分：API 网关概述

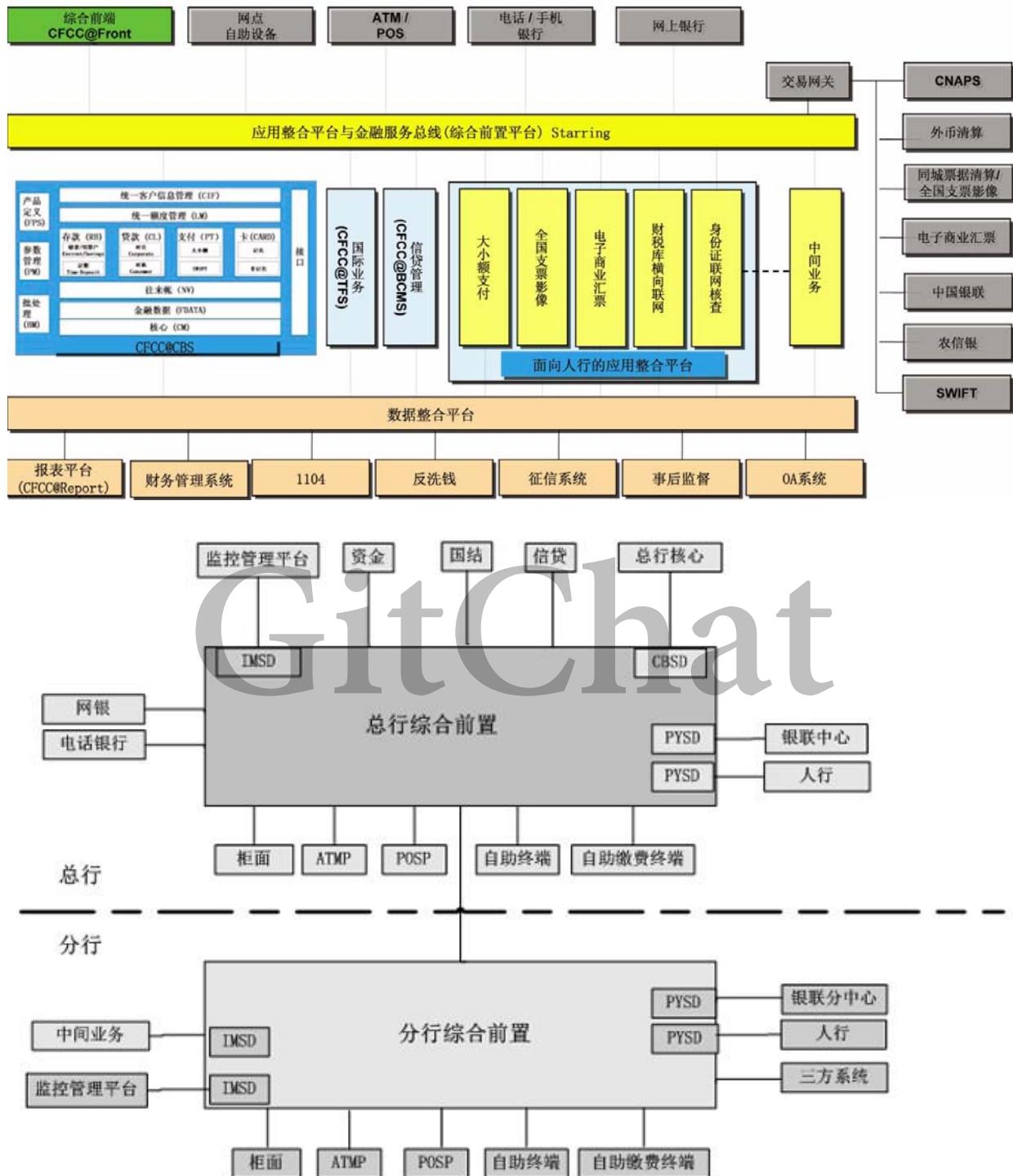
计算机科学领域的任何问题都可以通过增加一个间接的中间层来解决。 —— David Wheeler

分布式服务架构、微服务架构与 API 网关

什么是 API 网关 (API Gateway)

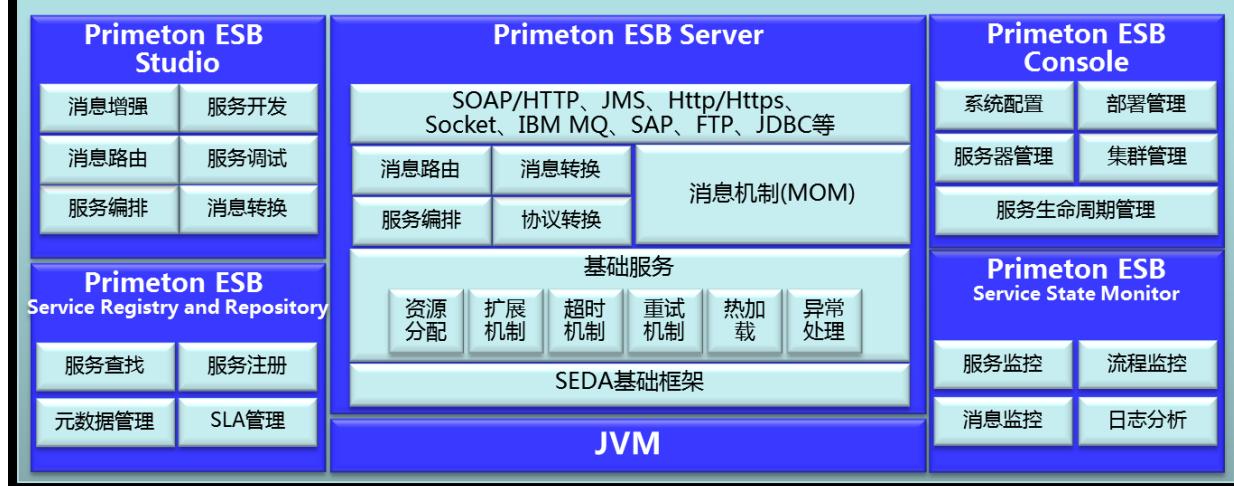
其实网关跟面向服务架构 (Service Oriented Architecture, SOA) 和微服务架构 (MicroServices Architecture, MSA) 有很深的渊源。

十多年以前，银行等金融机构完成全国业务系统大集中以后，分散的系统都变得集中，同时也带来各种问题：业务发展过快如何应对，对接系统过多如何集成和管理。为了解决这些问题，业界实现了作用于渠道与业务系统之间的中间层网关，即综合前置系统，由其适配各类渠道和业务，处理各种协议接入、路由与报文转换、同步异步调用等。

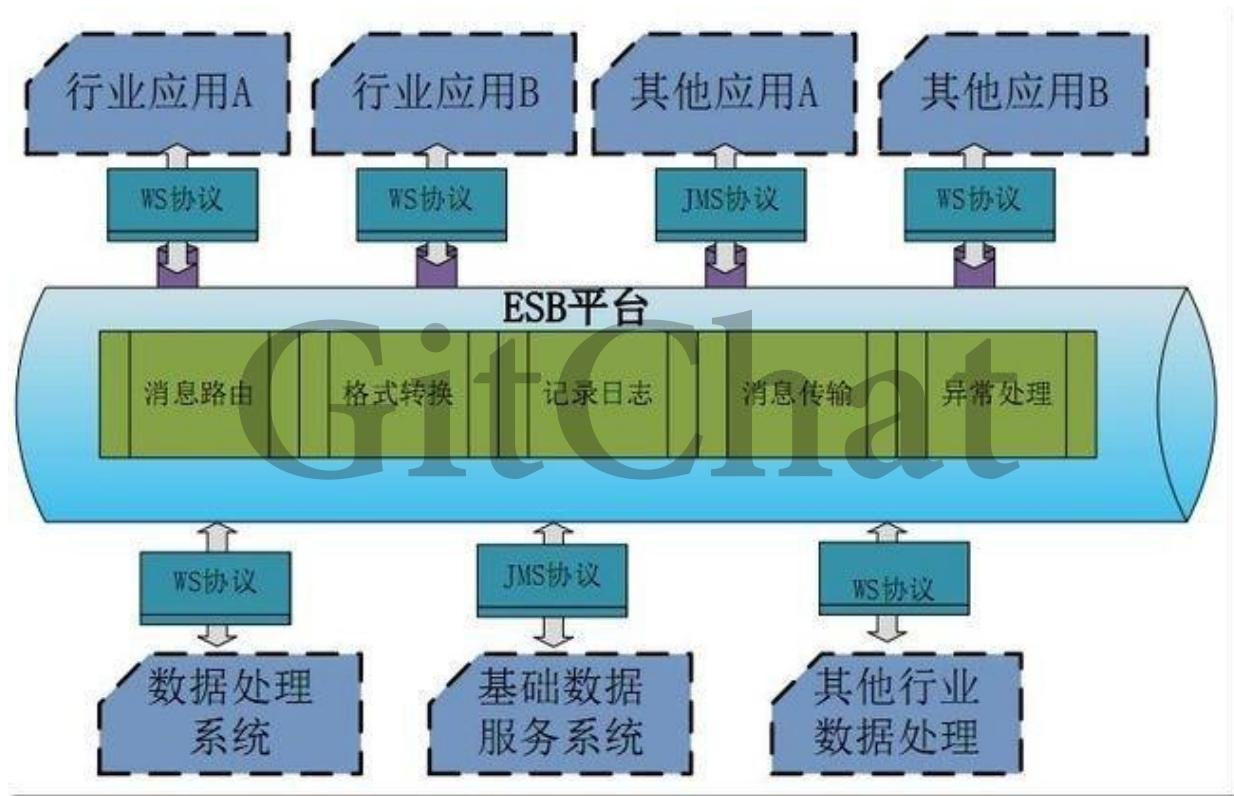


人们基于 SOA 的理念，在综合前置的基础上，进一步增加了服务的元数据管理、注册、中介、编排、治理等功能，逐渐形成了企业服务总线（ESB, Enterprise Service Bus）。

Primeton ESB™



(作者参与设计开发的 Primeton ESB 系统)



面向服务架构 (SOA) 是一种建设企业 IT 生态系统的架构指导思想。SOA 的关注点是服务，服务最基本的业务功能单元，由平台中立性的接口契约来定义。通过将业务系统服务化，可以将不同模块解耦，各种异构系统间可以轻松实现服务调用、消息交换和资源共享。

不同于以往的孤立业务系统，SOA 强调整个企业 IT 生态环境是一个大的整体。整个 IT 生态中的所有业务服务构成了企业的核心 IT 资源。各系统的业务拆解为不同粒度和层次的模块和服务，服务可以组装到更大的粒度，不同来源的服务可以编排到同一个处理流程，实现非常复杂的集成场景和更加丰富的业务功能。

SOA 从更高的层次对整个企业 IT 生态进行统一的设计与管理，应用软件被划分为具有不同功能的服务单元，并通过标准的软件接口把这些服务联系起来，以 SOA 架构实现的企

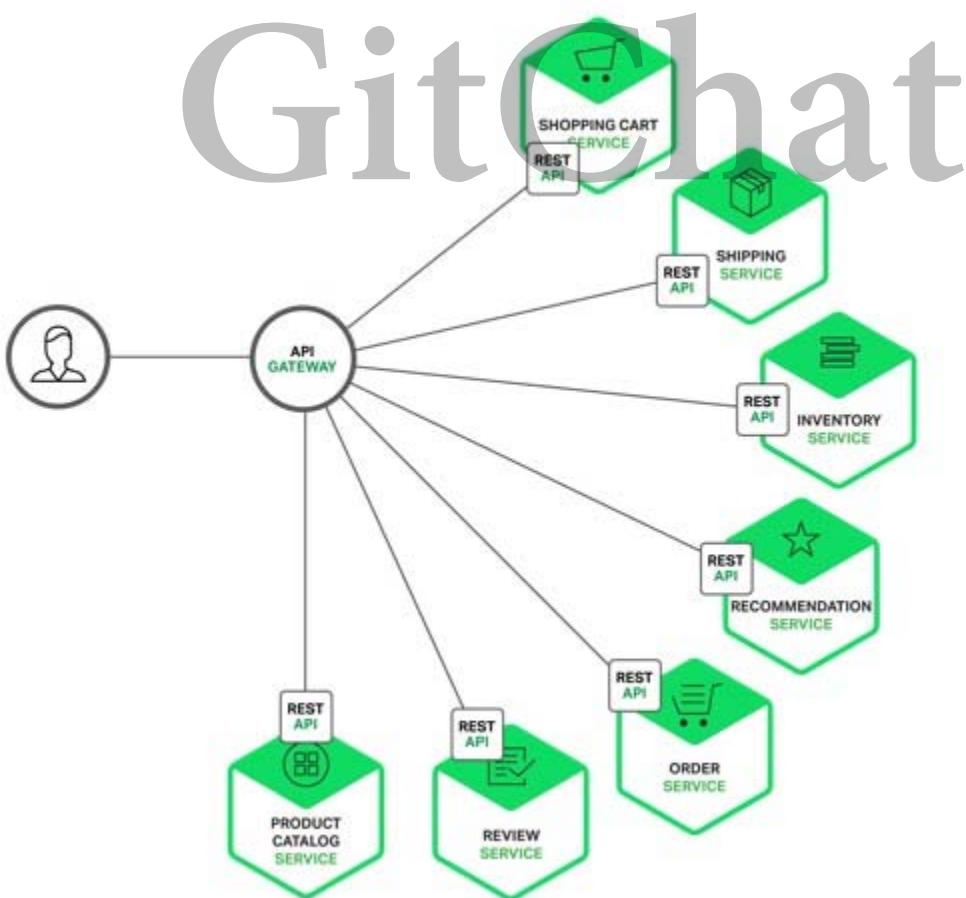
业应用可以更灵活快速地响应企业业务变化，实现新旧软件资产的整合和复用，降低软件整体拥有成本。

当然基于 ESB 这种集中式管理的 SOA 方案也存在着种种问题，特别是面向互联网技术领域的爆发式发展的情况下。

分布式服务架构、微服务架构与 API 网关

而近年来，随着互联网技术的飞速发展，为了解决以 ESB 这种集中式管理的 SOA 方案的种种问题，以 Apache Dubbo（2011 年开源后）与新近出现的 Spring Cloud 为代表的分布式技术的出现，给了 SOA 实现的另外一个选择：去中心化的分布式服务架构（DSA）。分布式服务架构技术不再依赖于具体的服务中心容器技术（比如 ESB），而是将服务寻址和调用完全分开，这样就不需要通过容器作为服务代理，在运行期实现最搞笑的直连调用。

进而又在此基础上随着 REST、Docker 容器化、领域建模、自动化测试运维等领域的的发展，逐渐形成了微服务架构（MSA）。在微服务架构里，服务的粒度被进一步细分，各个业务服务可以被独立的设计、开发、测试、部署和管理。这时，各个独立部署单元可以用不同的开发测试团队维护，可以使用不同的编程语言和技术平台进行设计，这就要求必须使用一种语言和平台无关的服务协议作为各个单元间的通讯方式。

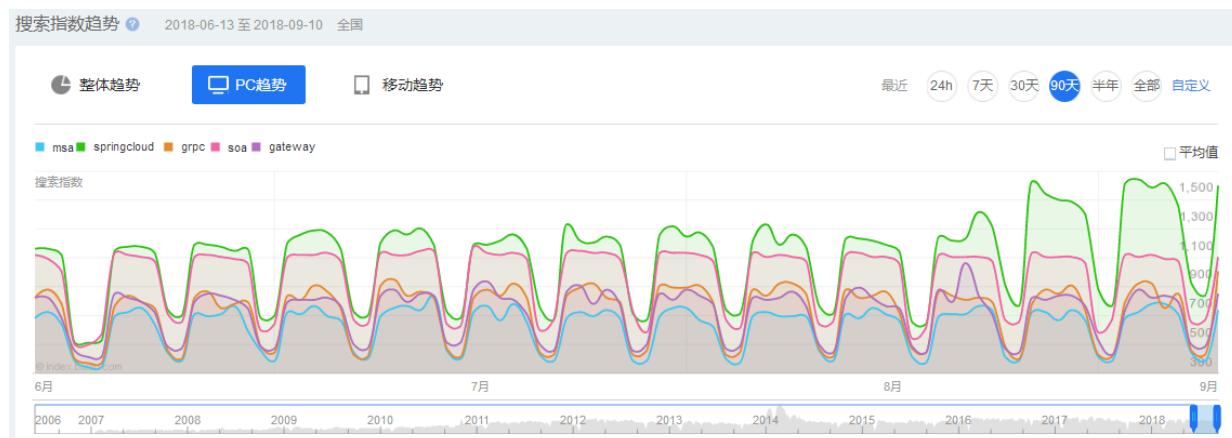


我们可以看到微服务架构中，由于系统和服务的细分，导致系统结构变得非常复杂，REST API 由于其简单、高效、跨平台、易开发、易测试、易集成，成为了不二选择。此时一个类似综合前置的系统就产生了，这就是 API 网关（API Gateway）。API 网关作为

分散在各个业务系统微服务的 API 聚合点和统一接入点，外部请求通过访问这个接入点，即可访问内部所有的 REST API 服务。

跟 SOA/ESB 类似，企业内部向外暴露的所有业务服务能力，都可以通过 API 网关上管理的 API 服务来得以体现，所以 API 网关上也就聚合了企业所有直接对外提供的 IT 业务能力。

API 网关的技术趋势



我们从百度指数趋势看到，SpringCloud 和 SOA 非常火，MSA、gRPC、Gateway 也都有着非常高的关注度，而且这些技术的搜索趋势都正相关。

另一方面，我们可以通过 Github 的搜索来看，Gateway 类型的项目也非常多。

<https://github.com/search?o=desc&p=1&q=gateway&s=stars&type=Repositories>

可以看到，前 10 页的 100 个项目，使用 Go 语言实现的 Gateway 差不多占一半，语言分类上来看：

Go > NodeJS/JavaScript > Java > Lua > C/C++ > PHP > Python/Ruby/Perl

API 网关的定义、职能与关注点

API 网关的定义

网关的角色是作为一个 API 架构，用来保护、增强和控制对于 API 服务的访问。

The role of a Gateway in an API architecture is to protect, enrich and control access to API services.

— <https://github.com/strongloop/microgateway>

API 网关是一个处于应用程序或服务（提供 REST API 接口服务）之前的系统，用来管理授权、访问控制和流量限制等，这样 REST API 接口服务就被 API 网关保护起来，对所有的调用者透明。因此，隐藏在 API 网关后面的业务系统就可以专注于创建和管理服务，而不用去处理这些策略性的基础设施。

这样，网关系统就可以代理业务系统的业务服务 API。此时网关接受外部其他系统的服务调用请求，也需要访问后端的实际业务服务。在接受请求的同时，可以实现安全相关的系统保护措施。在访问后端业务服务的时候，可以根据相关的请求信息做出判断，路由到特定的业务服务上，或者调用多个服务后聚合成新的数据返回给调用方。网关系统也可以把请求的数据做一些过程和预处理，同理也可以把返回给调用者的数据做一些过滤和预处理，即根据需要对请求头/响应头、请求报文/响应报文做一些修改处理。如果不做这些额外的处理，最简单直接的代理服务 API 功能，我们一般叫做透传。

同时，由于 REST API 的语言无关性，我们可以看出基于 API 网关，我们的后端服务可以是任何异构系统，不论是 Java、Dotnet、Python，还是 PHP、ROR、NodeJS 等，只要是支持 REST API，就可以被 API 网关管理起来。

API 网关的职能

API网关的四大职能



一般来说，API 网关有四大职能：

- 请求接入：作为所有 API 接口服务请求的接入点，管理所有的接入请求；
- 业务聚合：作为所有后端业务服务的聚合点，所有的业务服务都可以在这里被调用；
- 中介策略：实现安全、验证、路由、过滤、流控，缓存等策略，进行一些必要的中介处理；

- 统一管理：提供配置管理工具，对所有 API 服务的调用生命周期和相应的中介策略进行统一管理。

API 网关的关注点

通过以上的分析可以看出，API 网关不是一个典型的业务系统，而是一个为了让业务系统更专注与业务服务本身，给 API 服务提供更多附加能力的一个中间层。

这样在设计和实现 API 网关时，两个目标需要考虑：

1. 开发维护简单，节约人力成本和维护成本。这要求我们使用非常成熟的简单可维护的技术体系。
2. 高性能，节约设备成本，提高系统吞吐能力。这要求我们需要针对 API 网关的特点，进行一些特定的设计和权衡。

当并发量小的时候，这些都不是问题。然后一旦系统的 API 访问量非常大的时候，这些都会成为关键的问题。

海量并发的 Gateway 最重要的三个关注点：

1. 保持大规模的 inbound 请求接入能力（长短连接），比如基于 netty 实现。
2. 最大程度的复用 outbound 的 HTTP 连接能力，比如基于 HttpClient4 的 asynchronousHttpclient 实现。
3. 方便灵活地实现安全、验证、过滤、聚合、限流、监控等各种策略。

API 网关的分类与技术分析

API 网关的分类

如果我们对于上述的目标和关注点进行更深入的思考，就会发现一个很重要的问题：所有需要考虑的问题和功能可以分为两类。

一类是全局性的，跟具体的后端业务系统和服务完全无关的部分，比如安全策略、全局性流控策略、流量分发策略等。

一类是针对具体的后端业务系统，或者是服务和业务有一定关联性的部分，并且一般被直接部署在业务服务的前面。

这样，随着互联网的复杂业务系统的发展，这两类功能集合逐渐形成了现在常见的两种网关系统：流量网关和业务网关。

网关分类与功能



流量网关与 WAF

我们定义全局性的、跟具体的后端业务系统和服务完全无关的策略网关，即为流量网关。这样流量网关关注于全局流量的稳定与安全，具体比如防止各类 SQL 注入，黑白名单控制，接入请求到业务系统的 Loadbalance 等，通常有如下的一些通用性功能：

- 全局性流控
- 日志统计
- 防止 SQL 注入
- 防止 Web 攻击
- 屏蔽工具扫描
- 黑白名单控制

等等。

通过这个功能清单，我们可以发现，流量网关的功能跟 Web 应用防火墙（WAF）非常类似。WAF一般是基于 Nginx/OpenResty 的 ngx_lua 模块开发的 Web 应用防火墙。

WAF一般代码很简单，关注于使用简单，高性能和轻量级。简单的说就是在 Nginx 本身的代理能力以外，添加了安全相关功能。一句话来描述其原理，就是解析 HTTP 请求（协议解析模块），规则检测（规则模块），做不同的防御动作（动作模块），并将防御过程（日志模块）记录下来。

一般的 WAF 具有如下功能：

- 防止 SQL 注入，本地包含，部分溢出，fuzzing 测试，XSS/SSRF 等 Web 攻击
- 防止 Apache Bench 之类压力测试工具的攻击
- 屏蔽常见的扫描黑客工具，扫描器
- 屏蔽图片附件类目录执行权限、防止 webshell 上传
- 支持 IP 白名单和黑名单功能，直接将黑名单的 IP 访问拒绝

- 支持 URL 白名单，将不需要过滤的 URL 进行定义
- 支持 User-Agent 的过滤、支持 CC 攻击防护、限制单个 URL 指定时间的访问次数
- 支持支持 Cookie 过滤，URL 与 URL 参数过滤
- 支持日志记录，将所有拒绝的操作，记录到日志中去

几个 WAF 开源实现

以上 WAF 的内容主要参考如下两个项目：

- <https://github.com/unixhot/waf>
- https://github.com/loveshell/ngx_lua_waf

流量网关的开源实例，还可以参考著名的开源项目 Kong（基于 OpenResty）。

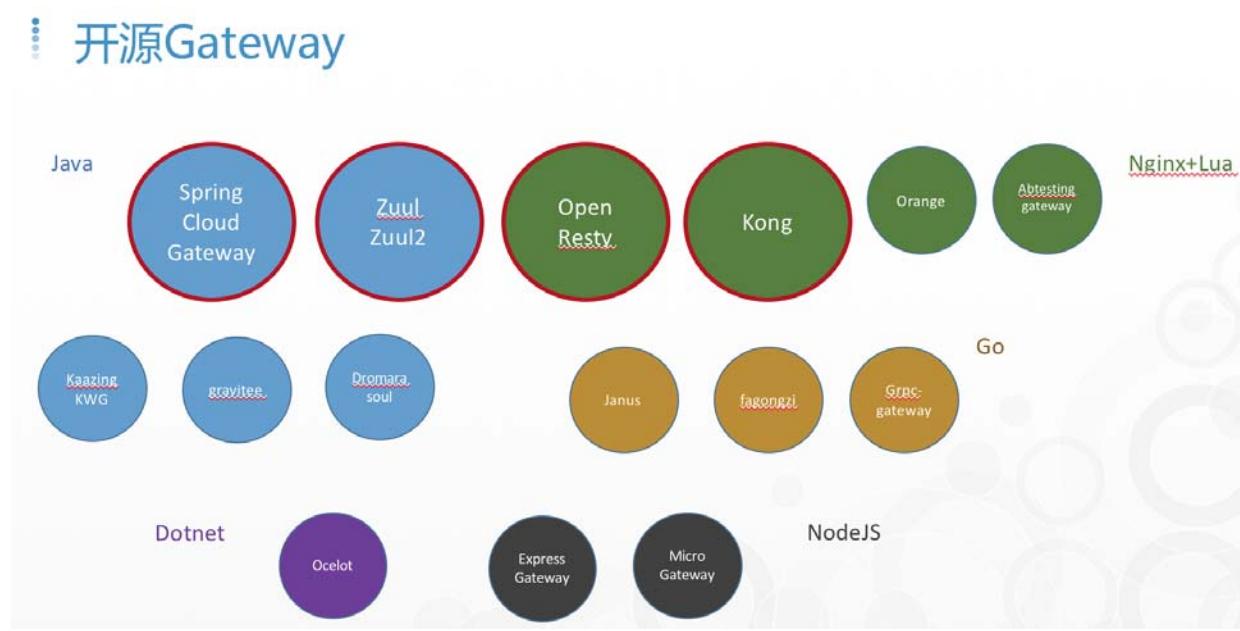
业务网关

我们定义针对具体的后端业务系统，或者是服务和业务有一定关联性的策略网关，即为业务网关。比如针对某个系统、某个服务或者某个用户分类的流控策略，针对某一类服务的缓存策略，针对某个具体系统的权限验证方式，针对某些用户条件判断的请求过滤，针对具体几个相关API的数据聚合封装等等。

业务网关一般部署在流量网关之后，业务系统之前，比流量网关更靠近系统。我们大部分情况下说的 API 网关，狭义上指的是业务网关。并且如果系统的规模不大，我们也会将两者合二为一，使用一个网关来处理所有的工作。具体的业务网关设计实现，将在下面的篇章详细介绍。

第二部分：开源网关的分析与调研

常见的开源网关介绍



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：Open Resty、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 4 个：

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

Nginx+Lua

Open Resty

项目地址：<http://openresty.org/>

OpenResty® 是一个基于 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。

OpenResty® 通过汇聚各种设计精良的 Nginx 模块（主要由 OpenResty 团队自主开发），从而将 Nginx 有效地变成一个强大的通用 Web 应用平台。这样，Web 开发人员和系统工程师可以使用 Lua 脚本语言调动 Nginx 支持的各种 C 以及 Lua 模块，快速构造出足以胜任 10K 乃至 1000K 以上单机并发连接的高性能 Web 应用系统。

OpenResty® 的目标是让你的 Web 服务直接跑在 Nginx 服务内部，充分利用 Nginx 的非阻塞 I/O 模型，不仅仅对 HTTP 客户端请求，甚至于对远程后端诸如 MySQL、PostgreSQL、Memcached 以及 Redis 等都进行一致的高性能响应。

以上介绍来自于 [OpenResty 网站中文版](#)。简单的说，OpenResty 基于 Nginx，集成了 Lua 语言和 Lua 的各种工具库，可用的第三方模块，这样我们就在 Nginx 既有的高效 HTTP 处理的基础上，同时获得了 Lua 提供的动态扩展能力。因此，我们可以做出各种符合我们需要的网关策略的 Lua 脚本，以其为基础实现我们的网关系统。

Kong

项目地址：

- <https://konghq.com/>
- <https://github.com/kong/kong>

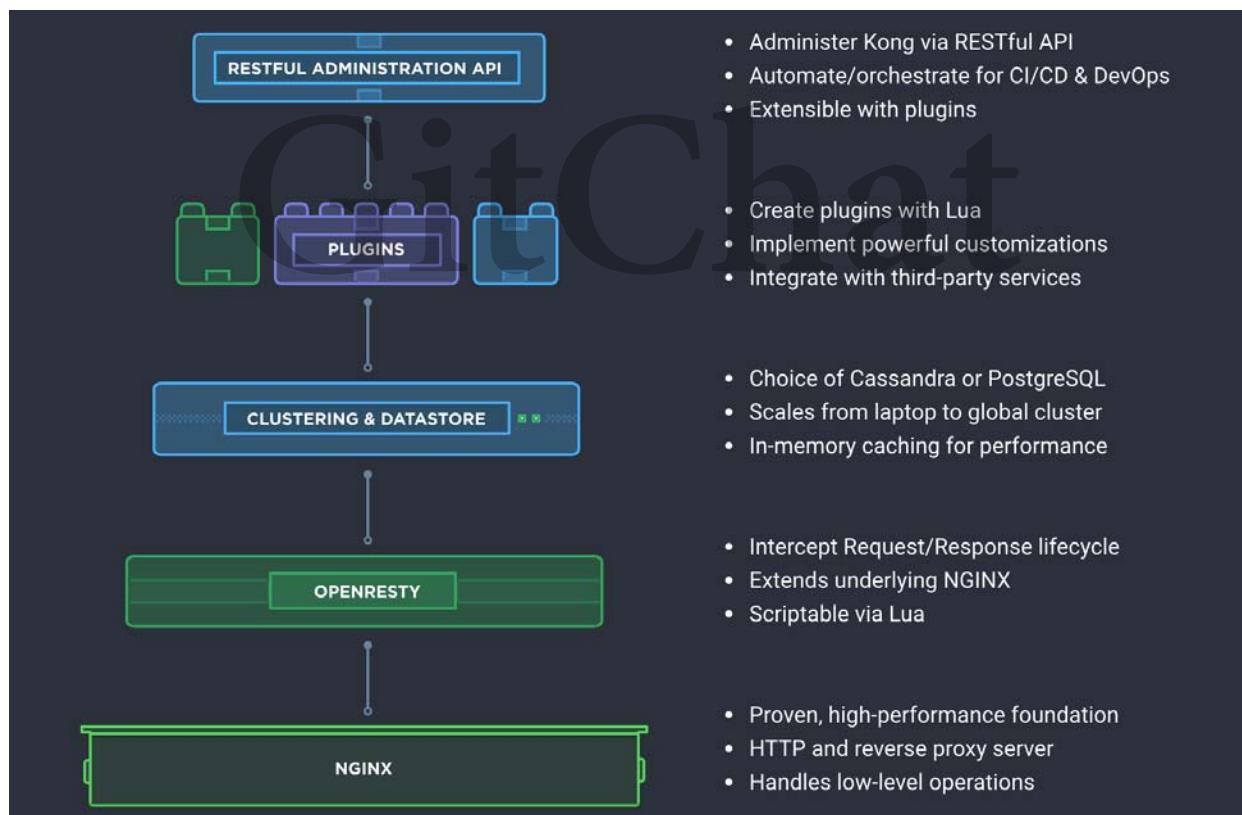
Kong 基于 OpenResty，是一个云原生、快速、可扩展、分布式的微服务抽象层（Microservice Abstraction Layer），也叫 API 网关（API Gateway），在 Service Mesh 里也叫 API 中间件（API Middleware）。

Kong 开源于 2015 年，核心价值在于高性能和扩展性。从全球 5000 强的组织统计数据来看，Kong 是现在依然在维护的，在生产环境使用最广泛的 API 网关。

Kong 宣称自己是世界上最流行的开源微服务 API 网关（The World's Most Popular Open Source Microservice API Gateway）。

核心优势：

- 可扩展：可以方便的通过添加节点水平扩展，这意味着可以在很低的延迟下支持很大的系统负载。
- 模块化：可以通过添加新的插件来扩展 Kong 的能力，这些插件可以通过 RESTful Admin API 来安装和配置。
- 在任何基础架构上运行：Kong 可以在任何地方都能运行，比如在云或混合环境中部署 Kong，单个或全球的数据中心。



ABTestingGateway

项目地址：

<https://github.com/CNSRE/ABTestingGateway>

ABTestingGateway 是一个可以动态设置分流策略的网关，关注与灰度发布相关领域，基于 Nginx 和 ngx-lua 开发，使用 Redis 作为分流策略数据库，可以实现动态

调度功能。

ABTestingGateway 是新浪微博内部的动态路由系统 dygateway 的一部分，目前已经开源。在以往的基于 Nginx 实现的灰度系统中，分流逻辑往往通过 rewrite 阶段的 if 和 rewrite 指令等实现，优点是性能较高，缺点是功能受限、容易出错，以及转发规则固定，只能静态分流。ABTestingGateway 则采用 ngx-lua，通过启用 lua-shared-dict 和 lua-resty-lock 作为系统缓存和缓存锁，系统获得了较为接近原生 Nginx 转发的性能。

功能特性：

- 支持多种分流方式，目前包括 iprange、uidrange、uid 尾数和指定 uid 分流
- 支持多级分流，动态设置分流策略，即时生效，无需重启
- 可扩展性，提供了开发框架，开发者可以灵活添加新的分流方式，实现二次开发
- 高性能，压测数据接近原生 Nginx 转发
- 灰度系统配置写在 Nginx 配置文件中，方便管理员配置
- 适用于多种场景：灰度发布、AB 测试和负载均衡等

据了解，美团内部的 Oceanus 也是基于 Nginx 和 ngx_lua 扩展实现，主要提供服务注册与发现、动态负载均衡、可视化管理、定制化路由、安全反扒、session ID 复用、熔断降级、一键截流和性能统计等功能。

Java

Zuul/Zuul2

GitChat

项目地址：<https://github.com/Netflix/zuul>

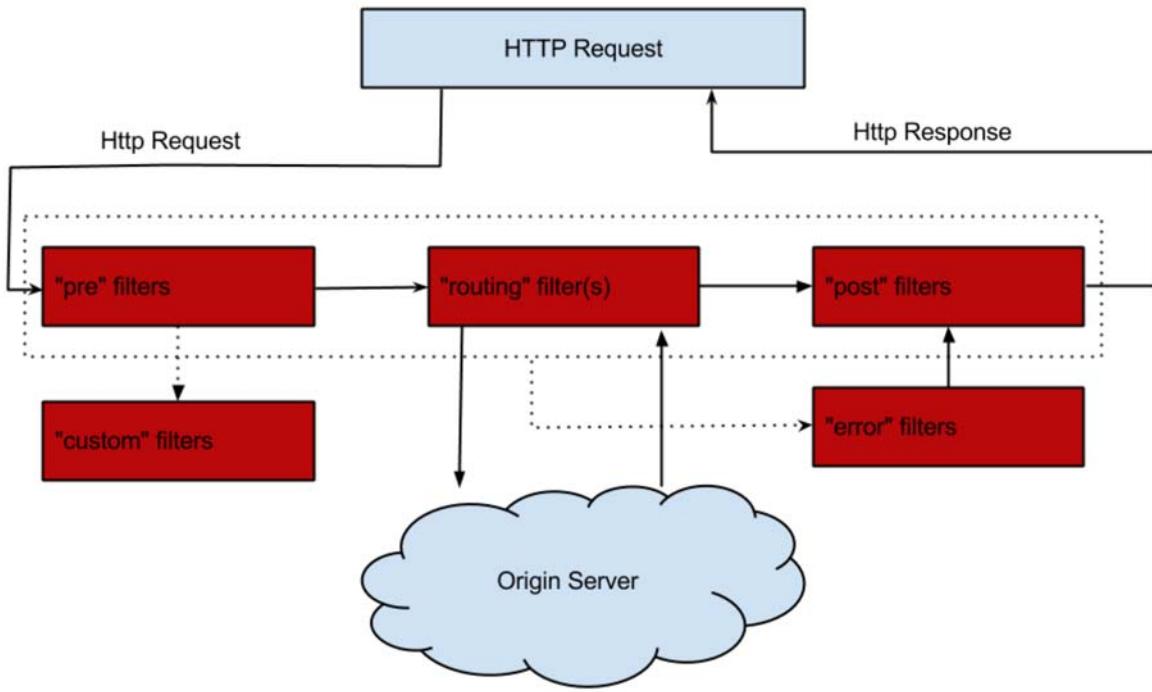
Zuul 是 Netflix 开源的 API 网关系统，它的主要设计目标是动态路由、监控、弹性和平安全。

Zuul 的内部原理可以简单看做是很多不同功能 filter 的集合（PS：作为对比，ESB 也可以简单被看做是管道（channel）和过滤器（filter）的集合），这些 filter 可以使用 Groovy 或其他基于 JVM 的脚本编写（当然 Java 也可以编写），放置在指定的位置，然后可以被 Zuul Server 轮询发现变动后动态加载并实时生效。

Zuul 目前有两个大的版本，1.x 和 2.x，这两个版本差别很大。

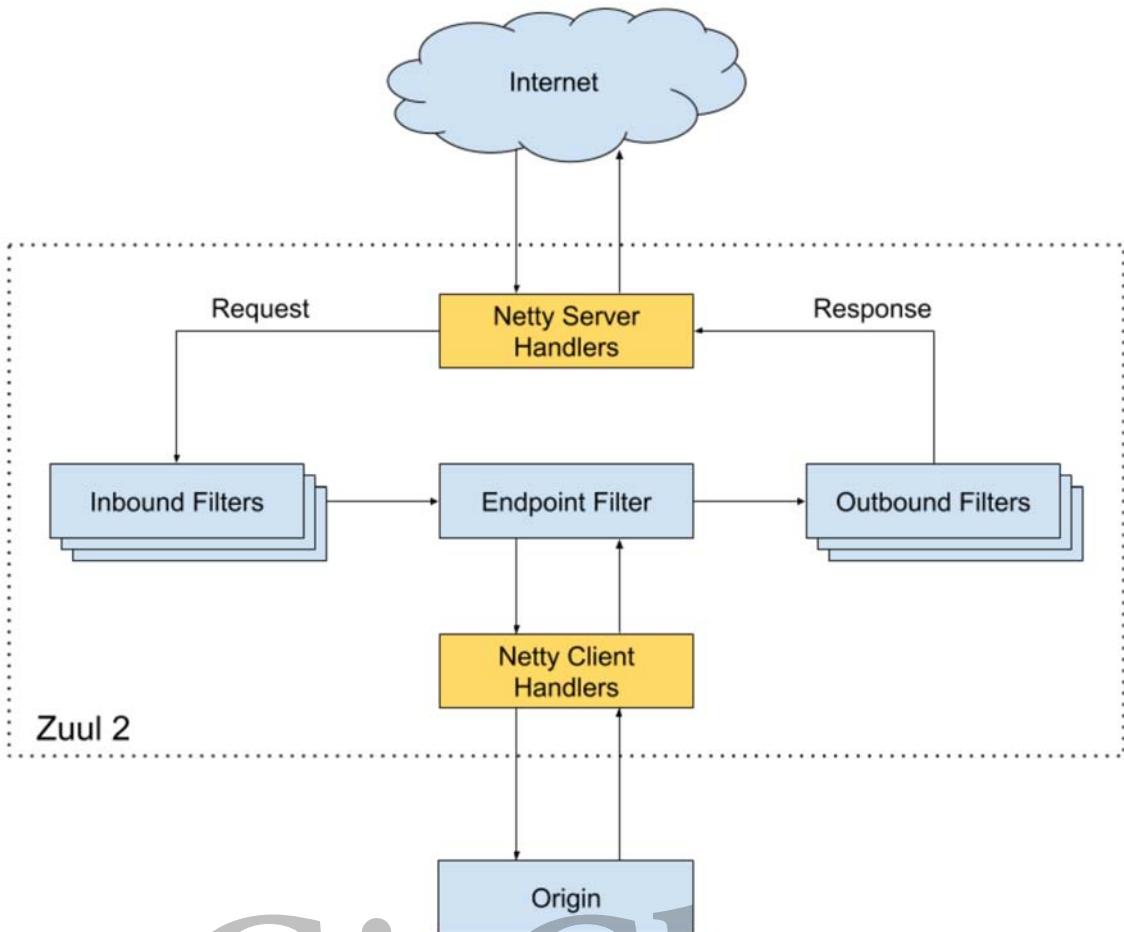
Zuul 1.x 基于同步 IO，也是 Spring Cloud 全家桶的一部分，可以方便的配合 Spring Boot/Spring Cloud 配置和使用。

在 Zuul 1.x 里，filter 的种类和处理流程可以参见下图，最主要的就是 pre、routing、post 这三种过滤器，分别作用于调用业务服务 API 之前的请求处理、直接响应、调用业务服务 API 之后的响应处理。



(Zuul 1.x 示意图)

Zuul 2.x 最大的改进就是基于 Netty Server 实现了异步 IO 来接入请求，同时基于 Netty Client 实现了到后端业务服务 API 的请求。这样就可以实现更高的性能、更低的延迟。此外也调整了 filter 类型，将原来的三个核心 filter 显式命名为：Inbound Filter、Endpoint Filter 和 Outbound Filter。



(Zuul 2.x 示意图)

Zuul 2.x 核心功能：

- Service Discovery
- Load Balancing
- Connection Pooling
- Status Categories
- Retries
- Request Passport
- Request Attempts
- Origin Concurrency Protection
- HTTP/2
- Mutual TLS
- Proxy Protocol
- GZip
- WebSockets

Spring Cloud Gateway

项目地址：

<https://github.com/spring-cloud/spring-cloud-gateway/>

Spring Cloud Gateway 基于 Java 8、Spring 5.0、Spring Boot 2.0、Project Reactor，发展的比 Zuul 2 要早，目前也是 Spring Cloud 全家桶的一部分。

Spring Cloud Gateway 可以看做是一个 Zuul 1.x 的升级版和代替品，比 Zuul 2 更早的使用 Netty 实现异步 IO，从而实现了一个简单、比 Zuul 1.x 更高效的、与 Spring Cloud 紧密配合的 API 网关。

Spring Cloud Gateway 里明确的区分了 Router 和 Filter，并且一个很大的特点是内置了非常多的开箱即用功能，并且都可以通过 SpringBoot 配置或者手工编码链式调用来使用。

比如内置了 10 种 Router，使得我们可以直接配置一下就可以随心所欲的根据 Header、或者 Path、或者 Host、或者 Query 来做路由。

比如区分了一般的 Filter 和全局 Filter，内置了 20 种 Filter 和 9 种全局 Filter，也都可以直接用。当然自定义 Filter 也非常方便。

核心特性：

- Able to match routes on any request attribute.
- Predicates and filters are specific to routes.
- Hystrix Circuit Breaker integration.
- Spring Cloud DiscoveryClient integration
- Easy to write Predicates and Filters
- Request Rate Limiting
- Path Rewriting

gravitee gateway

项目地址：

- <https://gravitee.io/>
- <https://github.com/gravitee-io/gravitee-gateway>

Kaazing WebSocket Gateway

项目地址：

- <https://github.com/kaazing/gateway>
- <https://kaazing.com/products/websocket-gateway/>

Kaazing WebSocket Gateway 是一个专门针对和处理 Websocket 的网关，其宣称提供世界一流的企业级 WebSocket 服务能力。

具体如下特性：

- 标准 WebSocket 支持，支持全双工的双向数据投递
- 线性扩展，无状态架构意味着可以部署更多机器来扩展服务能力
- 验证，鉴权，单点登录支持，跨域访问控制
- SSL/TLS 加密支持
- Websocket keepalive 和 TCP 半开半关探测
- 通过负载均衡和集群实现高可用
- Docker 支持
- JMS/AMQP 等支持
- IP 白名单
- 自动重连和消息可靠接受保证
- Fanout 处理策略
- 实时缓存等

Dromara soul

项目地址：<https://github.com/Dromara/soul>

Go

fagongzi

项目地址：

<https://github.com/fagongzi/gateway>



fagongzi gateway 是一个 Go 实现的功能全面的 API Gateway，自带了一个 Rails 实现的 Web UI 管理界面。

功能特性：

- 流量控制
- 熔断
- 负载均衡
- 服务发现
- 插件机制
- 路由(分流，复制流量)
- API 聚合
- API 参数校验
- API 访问控制 (黑白名单)
- API 默认返回值
- API 定制返回值
- API 结果 Cache
- JWT Authorization
- API Metric 导入 Prometheus

- API 失败重试
- 后端 server 的健康检查
- 开放管理 API(GRPC、Restful)
- 支持 Websocket 协议

Janus

项目地址:

<https://github.com/hellofresh/janus>

Janus 是一个轻量级的 API Gateway 和管理平台，它能帮你实现控制谁，什么时候，如何访问这些 REST API，同时它也记录了所有的访问交互细节和错误。

使用 Go 实现 API 网关的一个好处在于，一般只需要一个单独的二进制文件即可运行，没有复杂的依赖关系（No dependency hell）。

功能特性:

- 热加载配置，不需要重启网关进程
- HTTP 连接的优雅关闭
- 支持 OpenTracing，从而可以进行分布式跟踪
- 支持 HTTP/2
- 可以针对每一个 API 实现断路器
- 重试机制
- 流控，可以针对每一个用户或者 key
- CORS 过滤，可以针对具体的 API
- 多种开箱即用的验证协议支持，比如 JWT、OAuth2.0 和 Basic Auth
- docker image 支持

Dotnet

Ocelot

项目地址:

<https://github.com/ThreeMammals/Ocelot>

核心特性:

- 路由
- 请求聚合
- 服务发现（基于 Consul 或 Eureka）
- 服务 Fabric
- WebSockets

- 验证与鉴权
- 流控
- 缓存
- 重试策略与 QoS
- 负载均衡
- 日志与跟踪
- 请求头、Query 字符串转换
- 自定义的中间处理
- 配置和管理 REST API

NodeJS

Express Gateway

项目地址：

- <https://github.com/ExpressGateway/express-gateway>
- <https://www.express-gateway.io/>

Express Gateway 是一个基于 NodeJS 开发， Express 和 Express 中间件实现的 REST API 网关。

核心特性：

GitChat

- 动态中心化配置
- API 消费者和凭证管理
- 插件机制
- 分布式数据存储
- 命令行工具 CLI

microgateway

项目地址：

- <https://github.com/strongloop/microgateway>
- <https://developer.ibm.com/apiconnect>

StrongLoop 是 IBM 的一个子公司， Microgateway 网关基于 Node.js/Express 和 Nginx 构建，作为 IBM API Connect，同时也是 IBM 云生态的一部分。

Microgateway 是一个聚焦于开发者，可扩展的网关框架，它可以增强我们对微服务和 API 的访问能力。

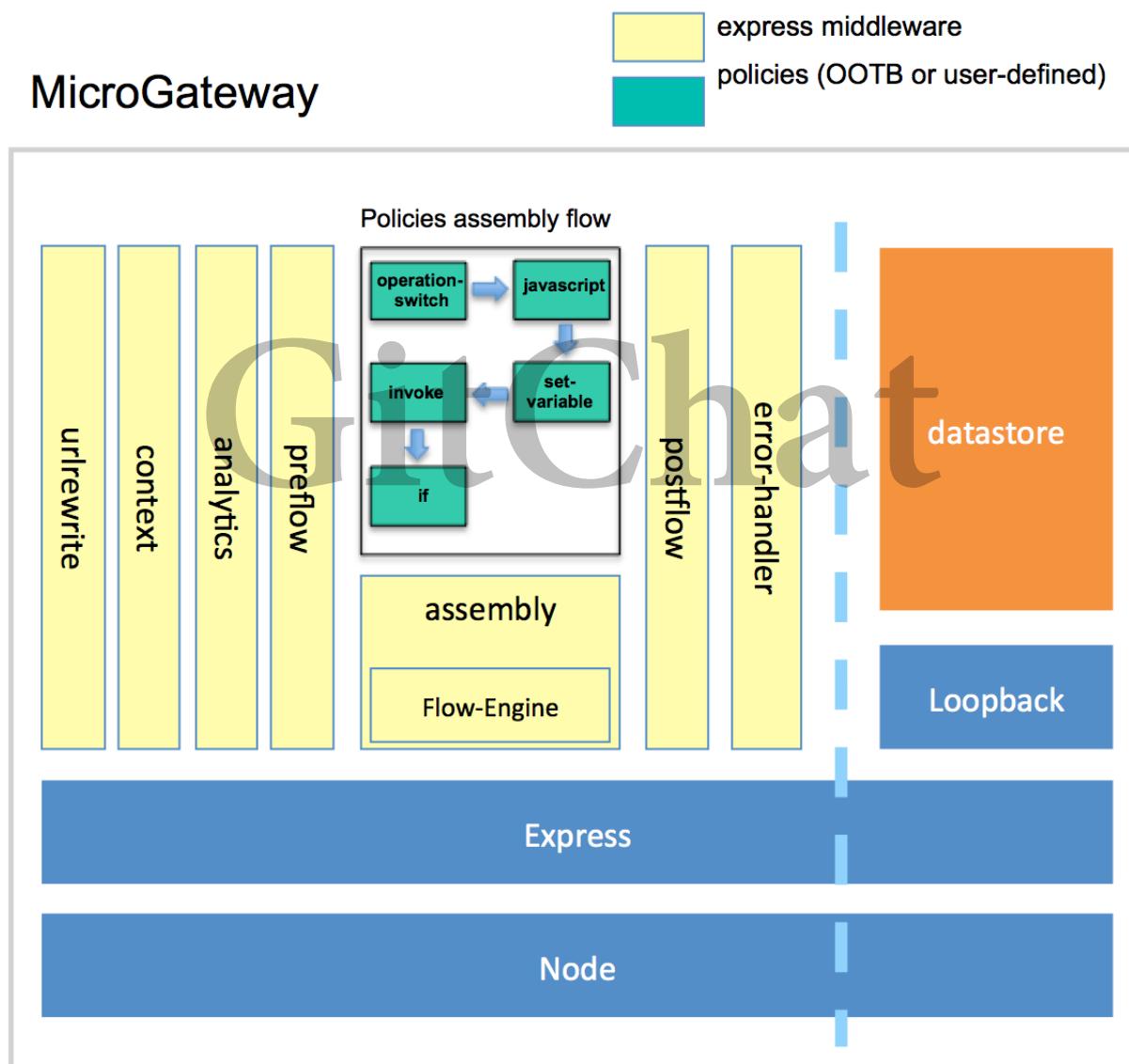
核心特性：

- 安全和控制，基于 Swagger (OpenAPI) 规范
- 内置了多种网关策略，API Key 验证、流控、OAuth2.0、JavaScript 脚本支持
- 使用 Swagger 扩展 (API Assembly) 实现网关策略 (安全、路由、集成等)
- 方便地自定义网关策略

此外，Microgateway 还有几个特性：

- 通过集成 Swagger，实现基于 Swagger API 定义的验证能力
- 使用 datastore 来保持需要处理的 API 数据模型
- 使用一个流式引擎来处理多种策略，使得 API 设计者可以更好的控制 API 的生命周期

核心架构如下图所示：



四大开源网关的对比分析 (OpenResty/Kong/Zuul2/SpringCloudGateway 等)

OpenResty/Kong/Zuul2/SpringCloudGateway 重要特性对比

网关	限流	鉴权	监控	易用性	可维护性	成熟度
Spring Cloud Gateway	可以通过 IP, 用户, 集群限流, 提供了相应的接口进行扩展	普通鉴权、auth2.0	Gateway Filter	Metrics	简单易用	spring 系列 可扩展, 强大, 易配置, 维护性好
Zuul2	可以通过配置文件配置集群限流和单服务器限流亦可通过 filter 实现限流扩展	filter 中实现	filter 中实现	参考资料较少	可维护性较差	开源不久, 资料少
OpenResty	需要 lua 开发	需要 lua 开发	需要开发	简单易用, 但是需要进行的 lua 开发很多	可维护性较差, 将来需要维护大量的 lua 脚本	很成熟资料很多
Kong	根据秒, 分, 时, 天, 月, 年, 根据用户进行限流。可在原码的基础上进行开发	普通鉴权, Key Auth 鉴权, HMAC, auth2.0	可上报 datadog, 记录请求数量, 请求数据量, 应答数据量, 接收于发送的时间间隔, 状态码数量, kong 内运行时间	简单易用, api 转发通过管理员接口配置, 开发需要 lua 脚本	“可维护性较差, 将来需要维护大量的 lua 库”	相对成熟, 用户问题汇总, 社区, 插件开源

GitChat

以限流功能为例：

- Spring Cloud Gateway 目前提供了基于 Redis 的 RateLimiter 实现，使用的算法是令牌桶算法，通过 yml 文件进行配置；
- Zuul2 可以通过配置文件配置集群限流和单服务器限流亦可通过 filter 实现限流扩展；
- OpenResty 可以使用 resty.limit.count、resty.limit.conn、resty.limit.req 来实现限流功能可实现漏桶或令牌通算法；

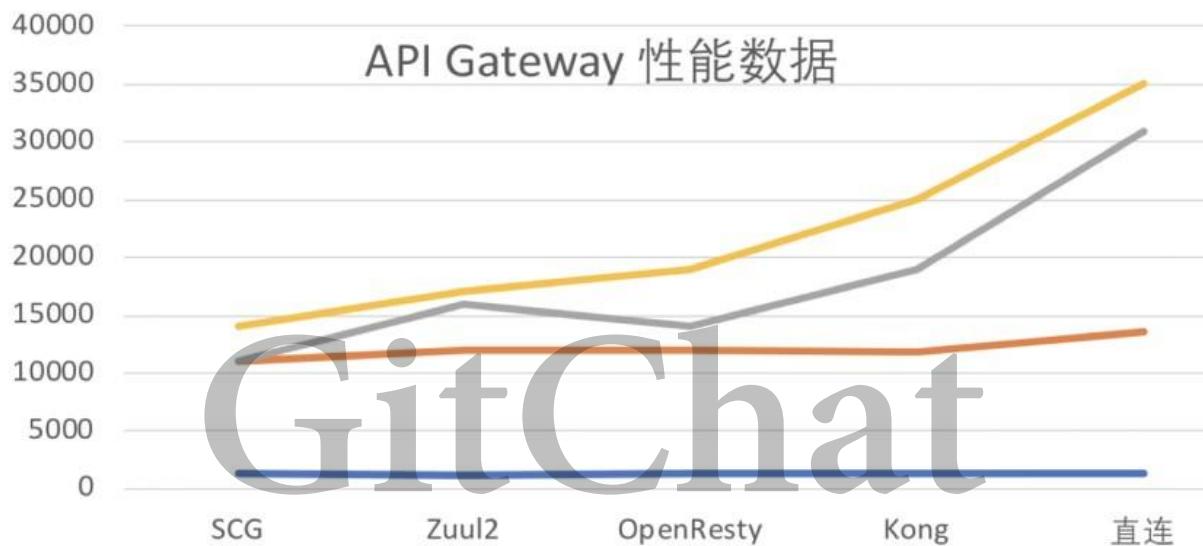
- Kong 拥有基础限流组件，可在基础组件源代码基础上进行 lua 开发。

对 Zuul/Zuul2/Spring Cloud Gateway 的一些功能点分析可以参考 Spring Cloud Gateway 作者 Spencer Gibb 的文章：

<https://spencergibb.netlify.com/preso/detroit-cf-api-gateway-2017-03/>

OpenResty/Kong/Zuul2/SpringCloudGateway 性能测试对比

分别使用 3 台 4Core16G 内存的机器，作为 API 服务提供者、Gateway、压力机，使用 wrk 作为性能测试工具，对 OpenResty/Kong/Zuul2/SpringCloudGateway 进行简单小报文的情况进行性能测试。



(Spring Cloud Gateway、Zuul2、OpenResty、Kong 的性能对比)

上图中 y 轴坐标是 QPS，x 轴是一个 Gateway 的数据，每根线是一个场景下的不同网关数据，测试结论如下：

- 实测情况是性能 SCG~Zuul2 << OpenResty ~< Kong << Direct (直连)；
- Spring Cloud Gateway、Zuul2 的性能差不多，大概是直连的 40%；
- OpenResty、Kong 差不多，大概是直连的 60-70%；
- 大并发下，例如模拟 200 并发用户、1000 并发用户时，Zuul2 会有很大概率返回出错。

开源网关的技术总结

开源网关的测试分析

脱离场景谈性能，都是耍流氓。性能就像温度，不同的场合下标准是不一样的。同样是 18 摄氏度，老人觉得冷，小孩觉得很合适，企鹅觉得热，冰箱里的蔬菜可能要坏了。

同样基准条件下，不同的参数和软件，相对而言的横向比较，才有价值。比如同样的机器（比如 16G 内存/4Core），同样的 server（用 Spring Boot，配置路径 api/hello 返回一个 helloworld），同样的压测方式和工具（比如用 WRK，10 线程，20 并发连接），我们测试直接访问 server 得到的极限 QPS（QPS-Direct，29K）；和配置了一个 Spring Cloud Gateway 做网关访问的极限 QPS（QPS-SCG，11K）、同样方式配置一个 Zuul2 做网关压测得到的极限 QPS（QPS-Zuul2，13K），Kong 得到的极限 QPS（QPS-Kong，21K），OpenResty 得到的极限 QPS（QPS-OR，19K），这个对比就有意义了。

Kong 的性能非常不错，非常适合做流量网关，并且对于 service、route、upstream、consumer、plugins 的抽象，也是自研网关值得借鉴的。

对于复杂系统，不建议业务网关用 Kong，或者更明确的说是不建议在 Java 技术栈的系统深度定制 Kong 或 OpenResty，主要是工程性方面的考虑。举个例子：假如我们有很多个不同业务线，鉴权方式五花八门，都是与业务多少有点相关的。这时如果把鉴权在网关实现，就需要维护大量的 Lua 脚本，引入一个新的复杂技术栈是一个成本不低的事情。

Spring Cloud Gateway/Zuul2 对于 Java 技术栈来说比较方便，可以依赖业务系统的一些 common jar。Lua 不方便，不光是语言的问题，更是复用基础设施的问题。另外，对于网关系统来说，性能不是差一个数量级，问题不大，多加 2 台机器就可以搞定。

目前测试的总结来看，如果服务都是 2ms 级别，直连的性能假如是 100，Kong 可以到 60，OpenResty 是 50，Zuul2 和 Spring Cloud Gateway 是 35，如果服务本身的 latency 大一点，这些个差距会逐步缩小。

目前来看 Zuul2 的坑还是比较多的：

1. 不成熟，没文档，刚出不久，还没有太多的实际应用案例
2. 高并发时出错率较高，1000 并发时我们的测试场景近 50% 的出错

所以简单使用或者轻度定制业务网关系统，目前比较建议使用 Spring Cloud Gateway 作为基础骨架。

各类网关的 demo 与测试

以上测试用到的模拟服务和网关 demo 代码，大部分可以在这里找到：

<https://github.com/kimmking/spring-cloud-gateway-demo>

这里也简单模拟了一个 NodeJS 做的 Gateway，加了 keep-alive 和 pool，demo 的性能测试结果大概是直连的 1/9，也就是 Spring Cloud Gateway 或 Zuul2 的 1/4 左右。

第三部分：百亿流量交易系统 API 网关设计

百亿流量交易系统 API 网关的现状和面临问题

百亿流量系统面对的业务现状

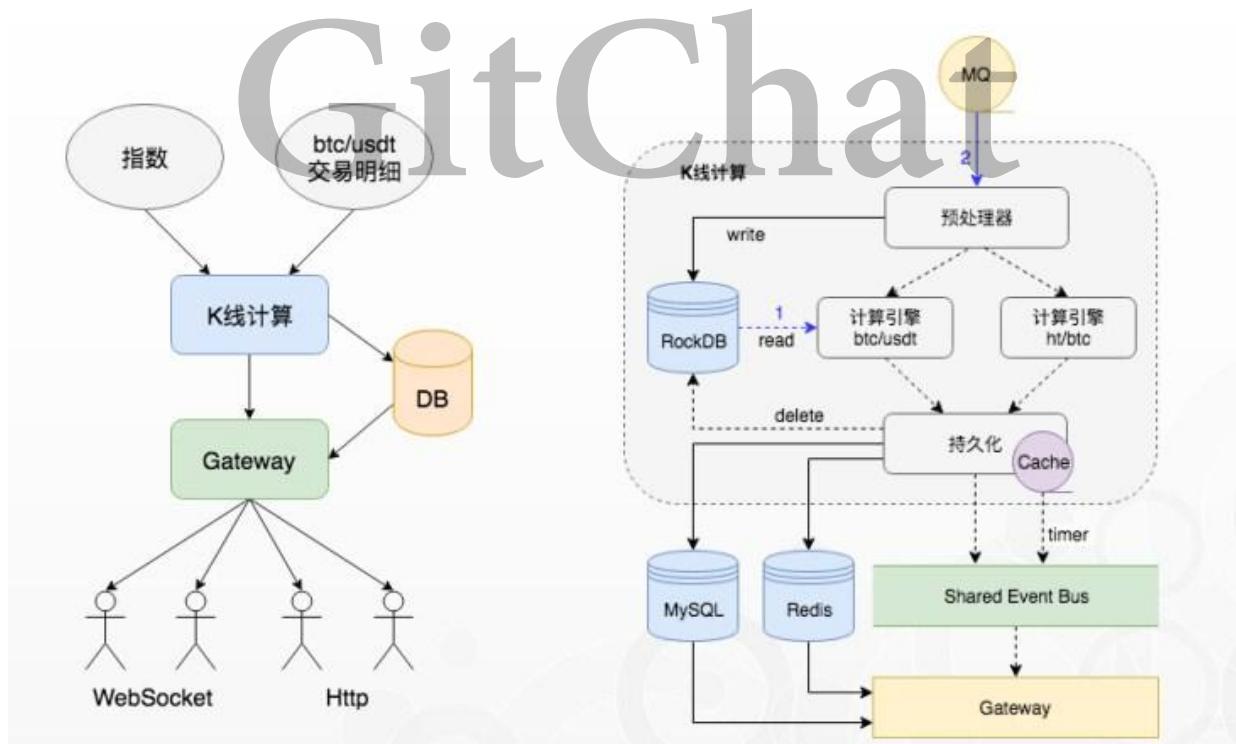


我们目前面临的现状是日常十几万的并发在线长连接数（不算短连接），每天长连接总数 3000 万+，每天 API 的调用次数超过 100 亿，每天交易订单数 1.5 亿。

在这个情况下，API 网关设计的一个重要目标就是：如何借助 API 网关为各类客户提供精准、专业、个性化的服务，保障客户实时的获得业务系统的数据和业务能力。

网关系统与其他系统的关系

我们的业务里，API 网关系统与其他系统的关系大致如下图所示：



网关系统典型的应用场景

我们的 API 网关系统为 Web 端、移动 APP 端客户提供服务，同时也为大量 API 客户提供 API 调用服务，同时支持 REST API 和 WebSocket 协议。

作为实时交易系统的前置系统，必须精准及时为客户提供最新的行情和交易信息。一旦出现数据的延迟或者错误，都会给客户造成无法挽回的损失。

另外针对不同的客户和渠道，网关系统需要提供不同的安全、验证、流控、缓存策略，同时可以随时聚合不同视角的数据进行预处理，保障系统的稳定可靠和数据的实时精确。

...	6530.96 ≈¥44802.38	-0.09% ▼	6537.65 ≈¥44848.27	6526.50 ≈¥44771.79	11317
...	454.00 ≈¥3114.44	-0.32% ▼	455.62 ≈¥3125.55	454.00 ≈¥3114.44	70453
...	222.83 ≈¥1528.61	-0.54% ▼	224.60 ≈¥1540.75	222.62 ≈¥1527.17	635772
...	11.2571 ≈¥77.22	-0.28% ▼	11.3231 ≈¥77.67	11.2571 ≈¥77.22	622264
...	57.92 ≈¥397.33	-0.20% ▼	58.09 ≈¥398.49	57.86 ≈¥396.91	174810
...	5.4061 ≈¥37.08	-0.64% ▼	5.4414 ≈¥37.32	5.4036 ≈¥37.06	10035098
...	0.069391 ≈¥0.47	-0.24% ▼	0.069581 ≈¥0.47	0.069322 ≈¥0.47	35740833



交易系统 API 的特点

作为一个全球性的交易系统，API 的特点总结如下：

- 访问非常集中：最核心的一组 API，占据了访问量的一半以上
- 访问非常频繁：QPS 非常高，日均访问量非常大
- 数据格式固定：交易系统处理的数据格式非常固定
- 报文数据量小：每次请求传输的数据一般不超过 10K
- 用户全世界分布：客户分布在全世界的各个国家
- 分内部调用和外部调用：除了 API 客户直接调用的 API，其他的 API 都是由内部其他系统调用的
- 7x24 小时不间断服务：系统需要提供高可用、不间断的服务能力，以满足不同时区客户的交易和自动化策略交易
- 外部用户有一定技术能力：外部 API 客户，一般是自己集成我们的 API，实现自己的交易系统

交易系统 API 网关面临的问题

问题 1：流量的不断增加

如何合理控制流量，如何应对突发流量，怎么样最大程度的保障系统稳定，都是重要的问题。特别网关作为一个直接面对客户的系统，任何问题都会放大百倍。很多千奇百怪的重来没人遇到的问题都随时可能出现。

问题 2：网关系统越来越复杂

现有的业务网关经过多年发展，里面有大量的业务嵌入，并且存在很多个不同的业务网关，相互之间没有任何关系，也没有沉淀出基础设施。

同时技术债务太多，系统里硬编码实现了全局性网关策略以及很多业务规则，导致维护成本较大。

问题 3：API 网关管理比较困难

海量并发下 API 的监控指标设计和数据的收集也是一个不小的问题。7x24 小时运行的技术支持也导致维护成本较高。

问题 4：推送还是拉取的选择

使用短连接还是长连接，REST API 还是 WebSocket？

业务渠道较多（多个不同产品线的 Web、App、API 等形成十几个不同的渠道），导致用户的使用行为难以控制。

GitChat 业务网关的设计与最佳实践

API 网关 1.0

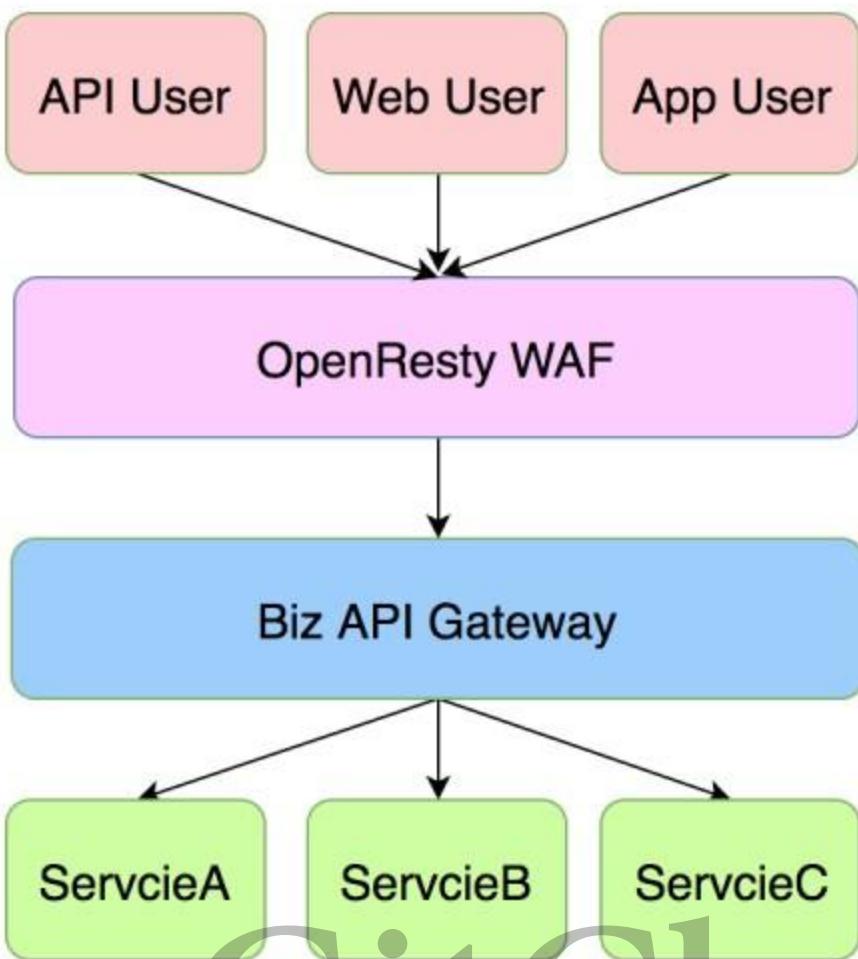
我们的 API 网关 1.0 版本是多年前开发的，是直接使用 OpenResty 定制的，全局的安全测试、流量的路由转发策略、针对不同级别的限流等都是直接用 Lua 脚本实现。

这样就导致在经历了业务飞速发展以后，系统里存在了非常多的相同功能或不同功能的 Lua 脚本，每次上线或维护都需要找到影响的其中几个或几十个 Lua 脚本，进行策略调整，非常不方便，策略控制的粒度也不够细。

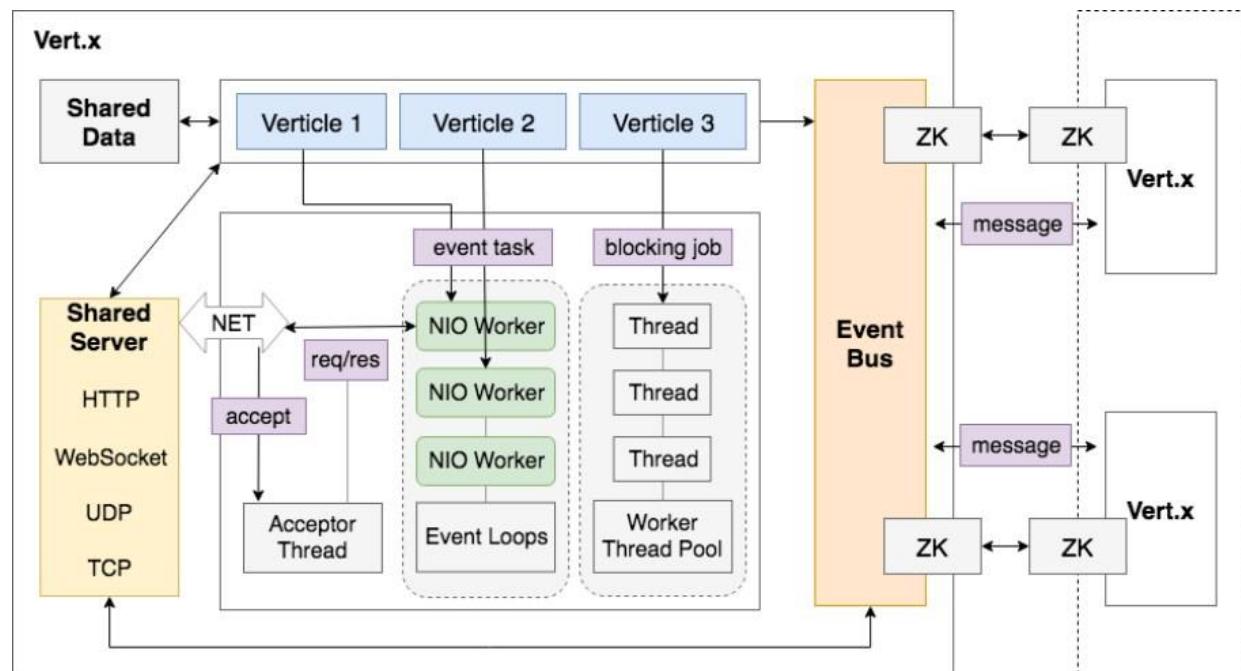
API 网关 2.0

在区分了流量网关和业务网关以后，2017 年开始实现了流量网关和业务网关的分离，流量网关继续使用 OpenResty 定制，只保留少量全局性，不经常改动的配置功能和对应的 Lua 脚本。

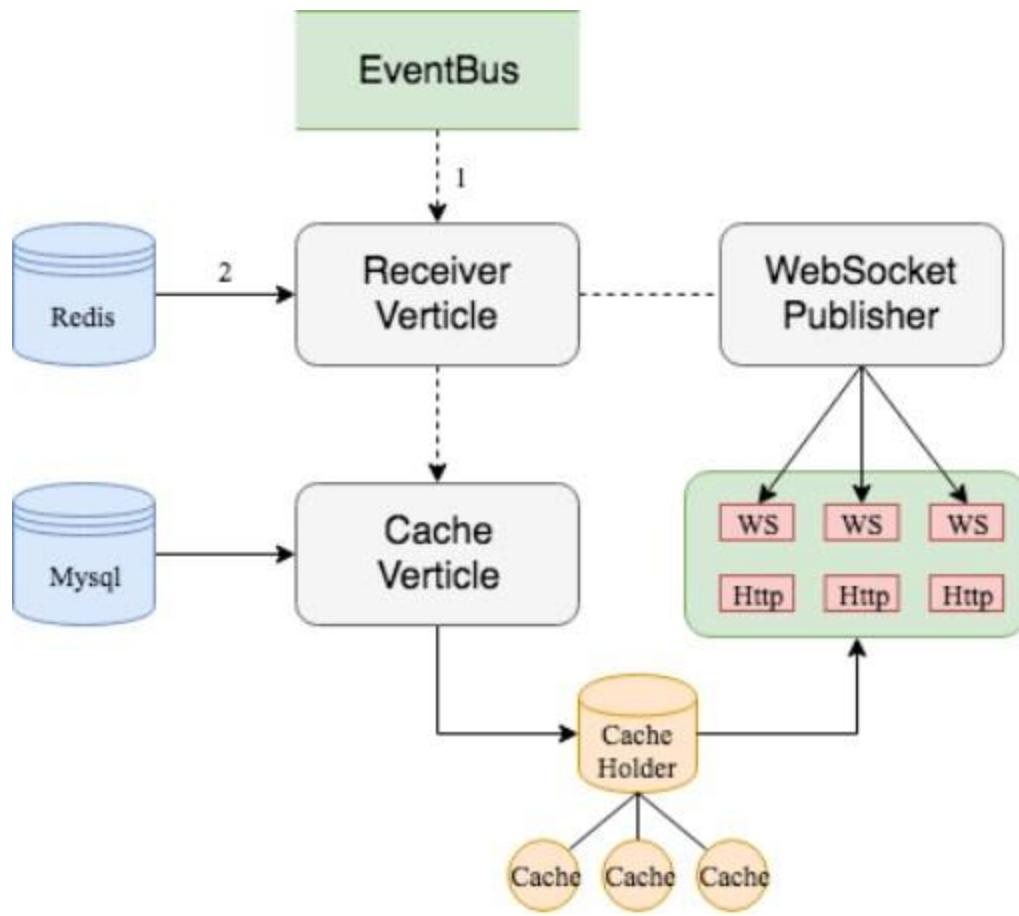
业务网关使用 Vert.x 实现的 Java 系统，部署在流量网关和后端业务服务系统之间，利用 Vert.x 的反应式编程能力和异步非阻塞 IO 能力、分布式部署的扩展能力，这样就初步解决了问题 1 和问题 2。



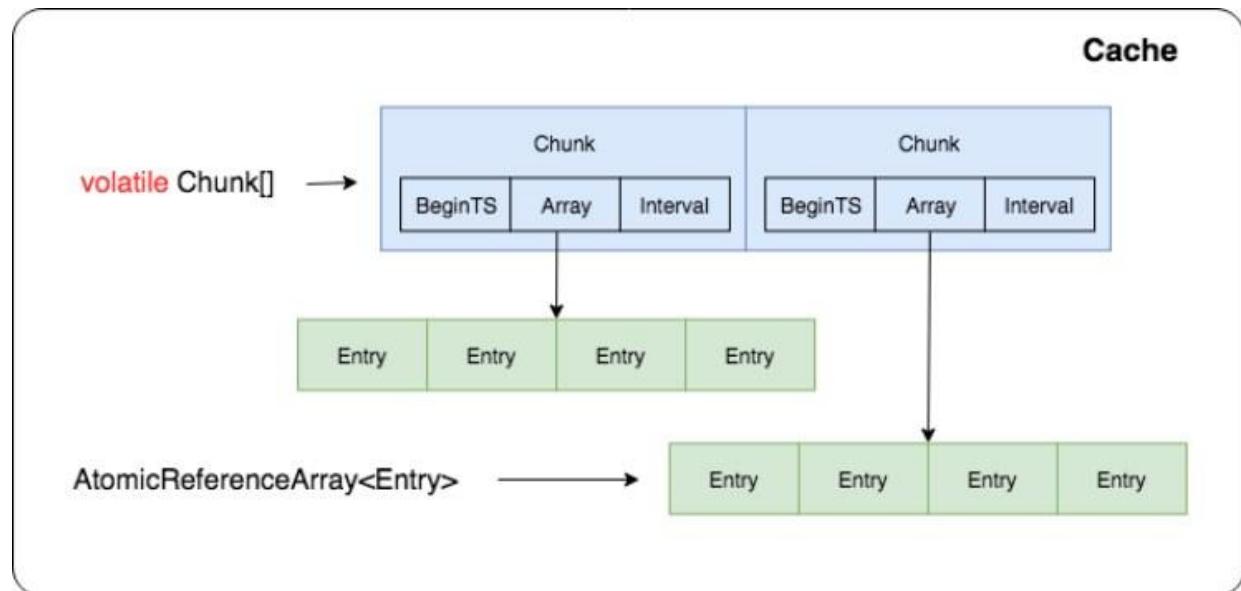
Vert.x 是一个基于事件驱动和异步非阻塞 IO、运行于 JVM 上的框架，如下图所示。在 Vert.x 里，Verticle 是最基础的开发和部署单元，不同的 Vert.x 可以通过 Event Bus 传递数据，进而方便的实现高并发性能的网络程序。关于 Vert.x 原理的分析可以参考阿里宿何的 blog： <https://www.sczyh30.com/tags/Vert-x/>



Vert.x 同时也很好的支持 Websocket 协议，所以可以方便的实现支持 REST API 和 Websocket、完全异步的网关系统。



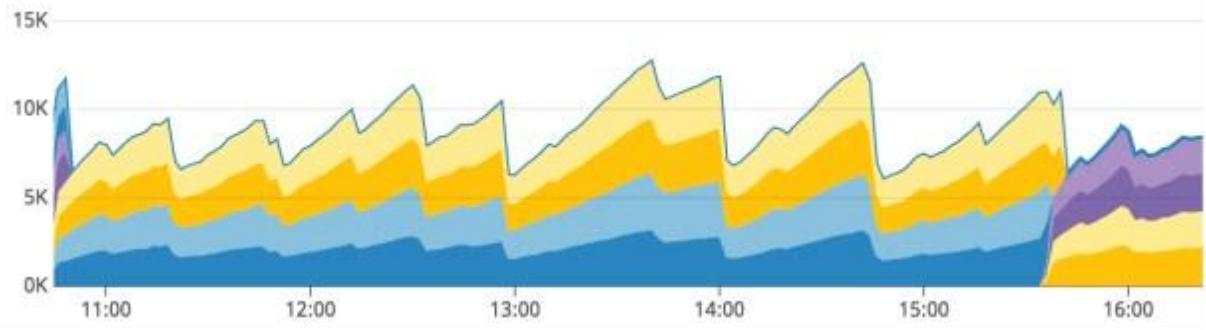
一个高性能的 API 网关系统，缓存是必不可少的部分。无论是分发冷热数据，降低对业务系统的压力，还是作为中间数据源，为服务聚合提供高效可复用的业务数据，都发挥了巨大作用。而一个优秀、高效的缓存系统，也必须是需要针对所承载的业务数据特点，进行特定设计和实现的。



API 网关的日常监控

我们使用多种工具对 API 进行监控和管理，全链路访问跟踪、连接数统计分析、全世界重要国家和城市的波测访问统计。网关技术团队每时每刻都关注着数据的变化趋势。各个业务系统研发团队，每天安排专人关注自己系统的 API 性能，推进性能问题解决和持续优化。这就初步解决了问题 3。

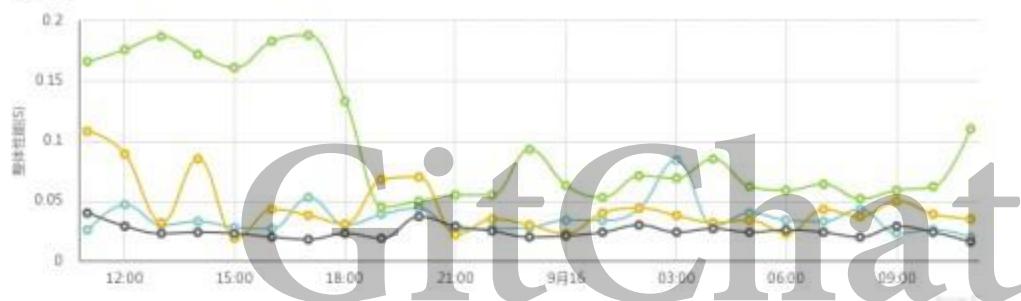
客户端 Websocket 连接数



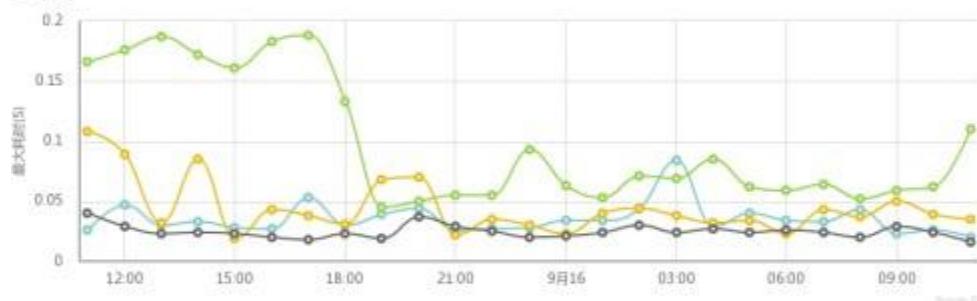
Gateway 服务器 CPU -



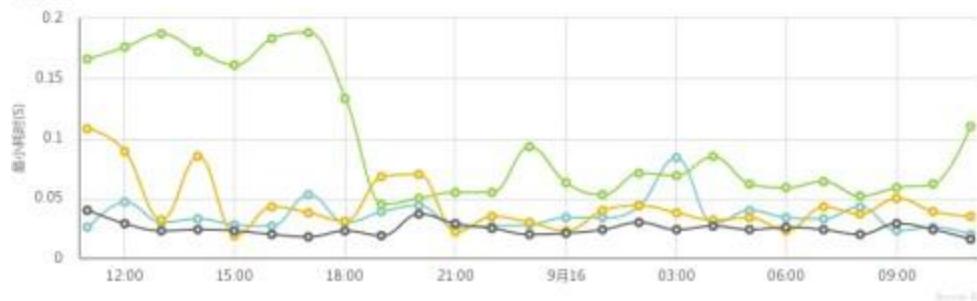
整体性能



最大耗时



最小耗时





推荐外部客户使用 Websocket

由于外部客户需要自己通过 API 网关调用 API 服务来集成业务服务能力到自己的系统，各个客户的技术能力和系统处理能力有较大差异，使用行为也各有不同。对于不断发展变动的交易业务数据，客户调用 API 频率太低则会影响数据实时性，调用频率太高则可能会浪费双方的系统资源。同时利用 Websocket 的消息推送特点，我们可以在网关系统控制客户接受消息的频率、单个用户的连接数量等，随时根据业务系统的情况动态进行策略调整。综合考虑，Websocket 是一个比 REST API 更加实时可靠，更加易于管理的方式。通过逐步协助和鼓励客户使用 Websocket 协议上，基本解决了问题 4。

API 网关的性能优化

API 网关系统作为 API 的统一接入点，为了给用户提供最优质的用户体验，必须长期做性能优化工作。

不仅 API 网关自己做优化，同时可以根据监控情况，时刻发现各业务系统的 API 服务能力，以此为出发点，推动各个业务系统不断优化 API 性能。

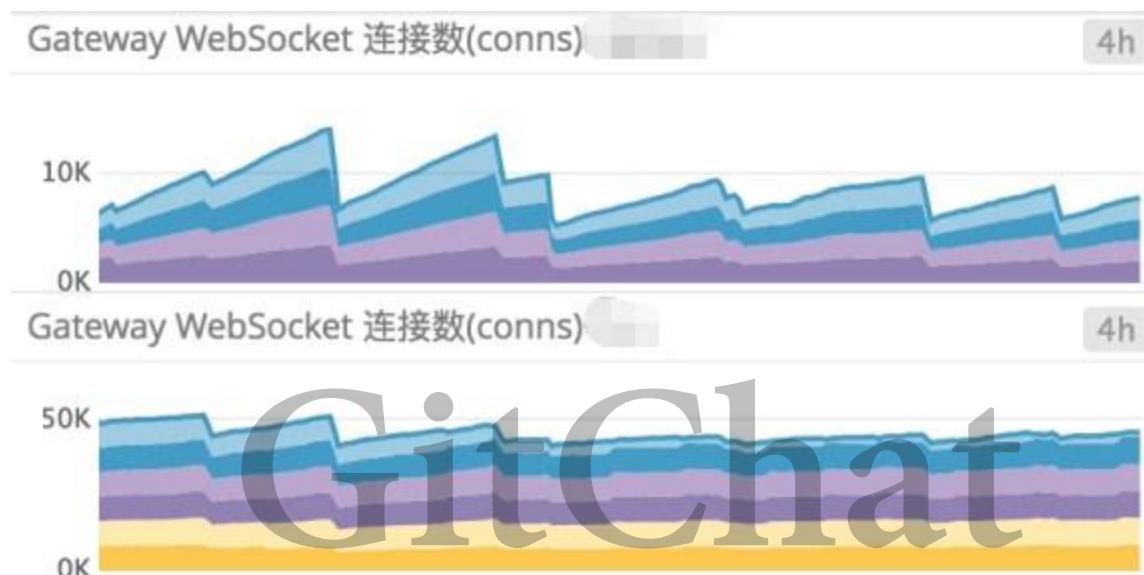
在此举一个具体的例子，某个网关系统发现连接经常剧烈抖动（如下图所示），严重影响系统的稳定性、浪费系统资源，经过排除发现：

1. 有爬虫 IP 不断爬取我们的交易数据，且这些 IP 所在网段都没有在平台产生任何实际交易，最高单爬虫 IP 的每日新建连接近 100 万次，平均每秒 10 几次；
2. 有部分 API 客户的程序存在 bug，且处理速度有限，不断的断开并重新连接，尝试重新对 API 数据进行处理，严重影响了客户的用户体验。

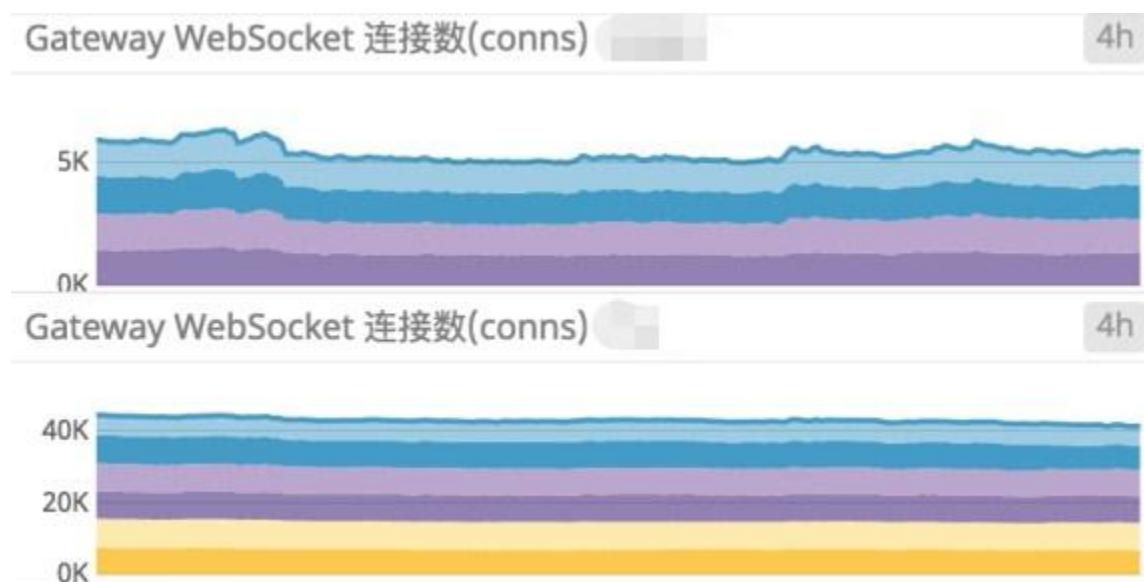
针对如上分析，我们采取了几个处理方式：

1. 对于每天认定的爬虫 IP，加入黑名单，直接在流量网关限制其访问我们的 API 网关；
2. 对于存在 bug 的 API 客户，协助对方进行问题定位和 bug 修复，增强客户使用信心；
3. 对于处理速度和技术能力有限的客户，基于定制的 Websocket 服务，使用滑动时间窗口算法，在业务数据变化非常大时，对分发的消息进行批量优化；
4. 对于未登录和识别身份的 API 调用，流量网关实现全局的流控策略，增加缓存时间和限制调用次数，保障系统稳定；
5. 业务网关则根据 API 服务的重要等级和客户的分类，进一步细化和实时控制网关策略，最大程度保障核心业务和客户的使用。

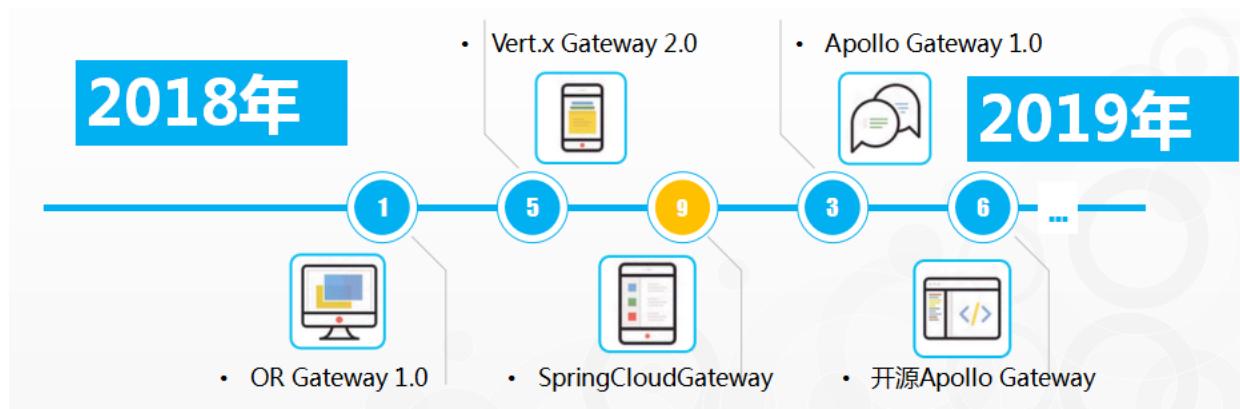
优化前：



优化后：



对 API 网关的发展展望



1. 现有的 API Gateway 是以 Vert.x 为基础、结合业务自研的网关系统 Gateway 2.0。
2. 目前计划年底前基于 Spring Cloud 和 Spring Cloud Gateway 实现新一代微服务架构的网关系统 Gateway 3.0。
3. 计划明年上半年将整合了流量网关和业务网关、并增加了很多开箱即用功能组件的微服务架构网关，作为 Apollo Gateway 1.0 开源。

期待大家的共同参与 (kimmking@163.com) 。

GitChat