doi: 10.7690/bgzdh.2025.07.002

一种微服务架构的 API 网关设计

林志达[1]，张华兵[2]，郭献彬[1]，曹小明[2]
（1. 中国南方电网有限责任公司，广州 510670；2. 南方电网数字电网研究院有限公司，广州 510670）

摘要：为解决微服务构建过程中后台服务调用的需求问题，实现服务接口的高效调用，提出利用应用程序接口(application programming interface，API)网关将后台服务封装为 API，利用 API 网关开放给用户调用。对 API 网关的功能性需求和非功能性需求进行分析，并描述每个需求的具体规格和要求；基于 Netty 框架实现请求接入模块，并利用用户活跃度对空闲连接进行过滤，提高对高并发请求的支持能力；结合不同协议调用的特点，基于企业服务总线实现 API 网关对不同协议的适配能力，以功能为单位对 API 网关进行模块化设计。实验结果显示：用户活跃区分策略的使用可以增加服务器的最大连接数；网关的响应时延在 13 ms 以内，极限事务数/秒(transactions per second，TPS)为 3 450，满足性能需求 TPS 3 000 和时延 20 ms 的要求；引入一种网络接口对象(network interface object，NIO)模型与 API 网关进行对比。结果表明：设计的 API 网关满足预期要求，可投入实际应用，可为研究高效调用的 API 网关提供参考。

关键词：微服务架构；API 网关；企业服务总线；服务调用；Netty 框架
中图分类号：TP311.56　　**文献标志码：**A

A Design of API Gateway Based on Microservice Architecture

Lin Zhida[1], Zhang Huabing[2], Guo Xianbin[1], Cao Xiaoming[2]
(1. *China Southern Power Grid Co*., *Ltd*., *Guangzhou* 510670, *China*;
2. *China Southern Power Grid Research Institute Co*., *Ltd*., *Guangzhou* 510670, *China*)

Abstract: In order to solve the demand problem of background service invocation in the process of microservice construction and realize the efficient invocation of service interface, it is proposed to use the application programming interface (API) gateway to encapsulate the background service into API, and use the API gateway to open it to users for invocation. The functional and non-functional requirements of the API gateway are analyzed, and the specific specifications and requirements of each requirement are described. The request access module is implemented based on the Netty framework, and the idle connection is filtered by using the user activity, so as to improve the support capability for high-concurrency requests; Combining with the characteristics of different protocol calls, the adaptation ability of API gateway to different protocols is realized based on enterprise service bus, and the modular design of API gateway is carried out with function as the unit. The experimental results show that the use of user activity differentiation strategy can increase the maximum number of server connections, the response delay of the gateway is less than 13 ms, and the maximum transactions per second (TPS) is 3 450, which meets the requirements of performance requirement TPS 3 000 and delay 20 ms; A network interface object (NIO) model is introduced for comparison with API gateways. The results show that the designed API gateway meets the expected requirements and can be put into practical application, which can provide a reference for the study of efficient API gateway.

Keywords: microservice architecture; API gateway; enterprise service bus; service invocation; Netty framework

0　引言

随着互联网技术的发展，软件系统为人们的生活、学习和工作提供了更加便利的方式，而软件系统架构也由单体式架构转变为面向服务架构，但面向服务架构存在集中化、成本高和维护难的问题。微服务架构具有去中心化和自动化的特点，可以满足企业软件研发的需求，软件系统的开发、部署和维护是相互独立的[1]。但随着业务的扩展和开放合作的需求，软件之间需要相互接入业务，建立软件生态圈，并隔离软件内部系统和外部系统，提升软件的安全性。API 管理平台是通过对系统的后台服务进行统一的管理，适用于企业内部不同平台间服务的对接和调用，对 API 的访问请求进行管理，灵活高效地实现配额管理和服务的调用[2]。随着系统内服务的增多，需要建立有效的 API 管理方式实现对服务的有效管理。

Xu 等[3]提出将边缘计算平台与 API 网关技术相结合，建立微服务安全代理的身份验证机制，该程序可以提供用户身份验证，并允许基于 JSON Web

收稿日期：2024-09-08；**修回日期：**2024-10-09
第一作者：林志达(1983—)，男，广东人。

指令可安全访问边缘计算服务。为将边缘计算平台和 API 网关进行集成，基于 EdgeX Foundry 框架中的 Kong 服务建立微服务安全代理，实现用户的注册服务和配置访问控制。温馨等[4]利用 API 网关作为微服务接口的聚合点，对企业微服务系统提供统一的接口，承担安全控制和路由等业务功能。提出基于 OpenResty 平台建立微服务架构 API 网关，并从负载均衡、路由和访问控制等方面对 API 网关进行设计，为微服务架构提供安全灵活的系统接口方案。吴润[5]研究了利用微服务架构对大型应用程序进行部署和运行，以解决单体交媾的扩展性弱和维护性差的问题。并将对多个微服务的协同运行问题进行分析，提出利用 API 网关建立微服务组合策略。

为解决大型应用程序为服务架构后台调用问题，建立高效的服务接口。对微服务框架的需求进行分析，结合企业服务总线对 API 网关进行模块化设计，增强 API 网关适应不同协议的能力，并基于 Netty 框架对请求接入模块的用户活跃度进行过滤，强化对高并发场景的处理能力。进行相关的测试，验证 API 网关的性能。

1 方法和模型设计

1.1 微服务系统中 API 网关的需求分析

1) 功能性需求。

随着企业规模的增加，传统单体式架构逐渐无法满足企业的需求，利用高性能的微服务开发框架，可以提升软件的开发效率。而微服务所面临的问题是小粒度服务的部署问题，微服务数量的增加，相应的重复工作越多；为减少开发压力，利用微服务架构对系统进行封装，建立规范化开发接口和框架扩展功能[6]。在企业服务化管理过程中，随着服务数量的增多，应用服务治理规范化协调服务间的稳定运行。功能性需求是 API 网关需要实现功能的需求，API 网关的核心能力是对外部的连接请求进行响应，并在请求期内完成多种请求处理策略；因此，可以将 API 网关的功能性微服务分为数据和管理 2 部分：数据功能负责对 API 请求的接入、处理和接触等任务进行处理；管理功能作为治理服务，可以提供标准化的接口实现 API 的调用，提升 API 服务资源的持久化和标准化能力[7]。API 网关的功能性需求包括用户验证、负载均衡、流量监控和日志记录等，对于服务管理功能来说，还包括服务权限管理、流量控制和缓存管理等功能，具体功能性需求如图 1 所示。

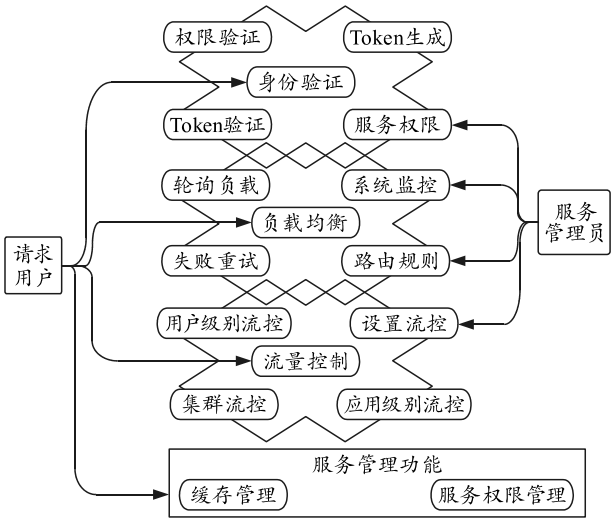


图 1 API 网关功能性需求

2) 非功能性需求。

API 网关除了需要满足上述功能性需求外，还需要满足安全性、高性能、可管理性等非功能性需求。安全性是指使用 HTTPS 协议对服务敏感信息进行传输，对部分高安全性要求的信息进行机密传输，确保请求权限的验证，避免发生越权的请求调用[8]；高性能是指采用异步调用的方式解决超时请求，在原请求调用的基础上，增加的额外时间不超过 20 ms，并且单请求的负载需要在 3 000 TPS 以上；可管理性是指系统的部署需要实现可追踪和可视化，方便管理者随时了解整个系统的服务调用状况[9]；可扩展性是指 API 网关的所有功能模块允许开发者进行扩展开发，实现服务的部署和版本升级，提供更加完善的服务能力。建立的平台化服务管理方案如图 2 所示，实现对服务治理的综合管理。

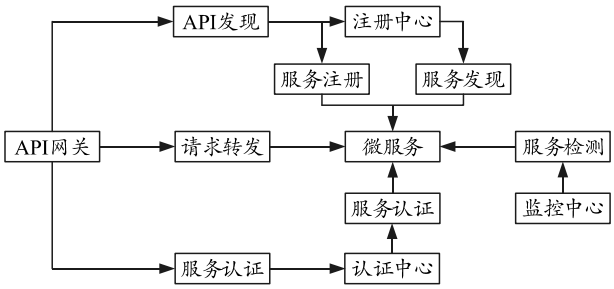


图 2 平台化服务管理方案

API 网关的工作原理是当客户端发起请求时，请求先访问 API 网关服务器，再由网关负责将用户请求发送到实际的服务器地址上，这样就不需要用户直接与网关服务器打交道，而由网关负责。具体的工作流程为：HTTP Client→请求参数解析与转换（验证系统参数→验证业务参数→转换业务参数）→服务调用（API 网关→服务注册）→返回结果转换。

1.2　基于微服务的 API 网关设计与实现

微服务架构根据组网的需求实现服务的扩展，并依据实际的请求状况进行服务模块的动态调整。每个微服务可以提供对外服务，并独立部署，进行自由的组合和调用，更适合利用业务需求去实现服务。并且 API 网关作为整个服务的接口，通过封闭的方式实现具体的服务，用户只需依照标准 API 的形式进行交互，提供高性能和安全性的 API 托管服务，实现企业业务的共享和管理[10]。企业服务总线是将可扩展标记语言(extensible markup language，XML)、传统中间件技术和 Web 服务相结合产生的技术，可以在不同的平台和编程语言条件下进行应用间消息路由、数据转换以及事件处理等操作，企业服务总线可以对不同协议类型信息进行转换适配，并提供负载均衡策略[11]。IBM Integration Bus 是企业服务总线的具体产品，适用于系统和服务的集成，为业务系统间的通信提供安全可靠的支持。平台化服务框架可以推进企业服务治理的规范化，便于对服务管理和成本管理的微服务架构的开发[12]；因此，结合平台化服务框架对 API 网关进行设计，结构如图 3 所示。

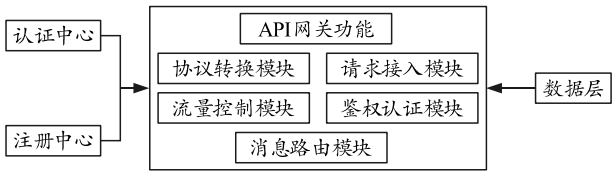


图 3　平台化服务 API 网关架构

1) 请求接入模块。API 网关需要将系统的内部服务通过标准 API 的形式进行提供，实现数据、服务和信息的共享，建立深度合作的生态形式，但随着系统开放性的增大，API 网关需要处理用户高并发请求的情况；因此，基于 Netty 框架筛选请求中的用户活跃度，实现用户请求接入的过滤。

2) 鉴权认证模块。对 API 网关的鉴权认证方式包括通过用户名和密码的扩展鉴权以及基于授权机制的 OAuth 鉴权；扩展鉴权的方式需要使用者提供用户名和密码才可以调用具有扩展鉴权策略的 API，OAuth 鉴权需要用户在特定的开发环境中订阅一个 API，并携带授权标识和生成的 Token 对 API 进行访问，进而实现对 API 权限的管控。

3) 消息路由模块。该模块可以保证调用者准确调用 API 网关中的 API，API 网关中的 API 信息将通过压缩包的方式进行导入和导出。利用四元组的形式对 API 网关中的 API 进行存储，包括命名空间、名称、版本和所述地域。

4) 协议转换模块。API 网关需要对拥有不同协议接口的请求进行响应，支持的协议类型包括 HTTP、REST、DSF 和 SOAP 等。转换过程中将不同编解码类型的消息进行相互转化，然后向服务发出请求。

5) 流量控制模块。API 网关需要具备对高并发请求场景的处理能力，尤其是在拥有大量 API 时，多个用户会在同时发出 API 请求，导致大量请求的涌入，为了解决突发的流量问题，需要对流量进行控制，确保服务的质量。采用配额流量策略可以为每个 API 在固定时间间隔内限定 API 调用次数。配额流量控制策略可以作为 API 网关的一部分接入到平台中，在导入 API 包到网关中时，通过读取其中的配置信息，并限制 API 调用次数。

上述内容中，请求接入模块是 API 网关的重要模块，而 Netty 框架更是该模块的关键技术。Netty 的线程模型由 Boss 和 Worker 线程组成，启动一个服务就会创建一个 Boss 线程，接收到用户请求后 Boss 线程会产生一个 Task，并由 Worker 线程池进行处理，事件请求将在 Selector 进行注册，通过设定查询周期，对询问请求中的空轮询进行限制[13]。Netty 框架在处理连接请求时会采用单独的线程池处理用户请求，线程池中大多数线程是空闲状态，并且 Netty 框架会对用户的活跃度进行分类，通过判断用户的活跃度，将不活跃请求的客户端划分为休眠状态，降低长连接的数量，以及不活跃客户端与微服务架构长连接造成的资源浪费，设计的用户活跃度区分策略如图 4 所示。

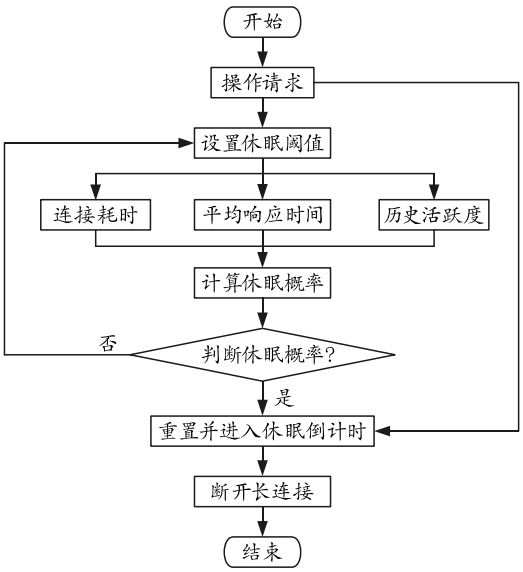


图 4　用户活跃度区分策略流程

Netty 框架使用线程池处理用户请求，利用客户端的平均响应时间，判断用户对消息的敏感程度与信息需求的迫切性，采用休眠的方式避免请求信息阻塞现象的发生，提升系统的响应性能。客户的历史活跃度反应了用户信息请求前后行为的关联程度和连续性，其与服务器的连接时间呈现出正态分布的特点，通信时间大于平均值后，用户对信息的需求性就会下降，客户端进入休眠状态的几率会增加[14]。因此，利用公式对客户端休眠的概率 X 进行计算：

$$X=\frac{|T-t_1|+v_1t_2}{v_2t_3T},\ (T>t_1/2)。\tag{1}$$

式中：T 为用户保持活跃状态的市场；t_1 为客户端历史活跃时长；t_2 为客户端响应时长；t_3 为客户端和服务器建立连接的耗时；v_1 和 v_2 为用户活跃权重，服从正态分布。计算得到用户的活跃时长，并与平均耗时进行比较，结合客户端和服务器连接时间造成的损耗，获得用户操作请求进行休眠状态的概率，当用户的概率值到达限定值时，对客户端进行休眠。若客户端在休眠期内再次产生信息接入请求，则客户端脱离休眠状态，并重新计算休眠概率[15]；因此，利用 Netty 框架建立用户活跃度过滤方案，对用户的连接请求进行过滤，提升 API 网关在高并发请求场景下的运行性能。

同时 API 网关等待发送时间的计算方法如式(2)所示：

$$T(n)=t_{m'}t_m=t_{m'}-t_n+t_n-t_{m'}。\tag{2}$$

式中：T 为时间；n 为个数；m 为分组；m' 为该分组的对应组。根据式(2)可得到 API 网关平均等待时间的计算方法如式(3)所示：

$$\overline{T}(n)=\frac{(n-1)(\gamma+\alpha\beta)+n\alpha(\beta^2-\beta)}{2(1-n\alpha\beta)}。\tag{3}$$

式中：T 为时间；n 为个数；α 为信息到达率；β 为服务信息分组的服务率；γ 为系数。

API 网关 2 个信息组分组所需的时间计算方法如式(4)所示：

$$T_{a,b}=\sum_{i=1}^{n}(\mathrm{L}_{a,b}+\mathrm{T}_{a,b})。\tag{4}$$

式中：T 为时间；n 为个数；终端 a 在接收服务时，b 为终端存储器中平均存储信息分组的个数。

由上述公式可得出 API 网关系统中平均排队队长的计算方法如式(5)所示：

$$\Im=\gamma\frac{1-\alpha\beta}{1-\alpha\beta}。\tag{5}$$

式中：$\Im$ 为队长；α 为信息到达率；β 为服务信息分组的服务率；γ 为系数。

当数据信息与核对一致时，API 网关会将数据反馈给信息系统，反馈时间的计算方法如式(6)所示：

$$T(a)=\alpha\gamma(i-j)/T(a+1)。\tag{6}$$

式中：T 为时间；α 为信息到达率；γ 为系数；i 和 j 分别为发送的时间和接收到的时间；a 为第 a 次发送。

另外，在选择研究数据的时候，要用到特征提取算法，所有备选数据的信息熵计算方法如式(7)所示：

$$H(X)=\sum_{i=1}^{m}P(X_1)(-\log P(X_1))。\tag{7}$$

式中：H 为信息熵；m 为事件；P 为出现该事件的概率。其中单个事件的信息熵计算方法如式(8)所示：

$$I=-\log P(X_n)。\tag{8}$$

式中：I 为单个信息熵；n 为随机数；P 为出现该事件的概率。在选定 API 网关数据之后，关于信息增益的计算方法如式(9)所示：

$$\mathrm{IG}(X)=H(X)-H(X|C)。\tag{9}$$

式中：IG 为信息增益；H 为信息熵；$H(X|C)$为划分熵的信息。由式(9)可得出具体的划分熵计算方法如式(10)所示：

$$H(X\mid C)=\sum_{i=1}^{\infty}P(X_1)(H(X\mid C_1))。\tag{10}$$

式中：$H(X|C)$为划分熵的信息；P 为出现事件的概率。

1.3 实验设计以及参数设置

为了更好地对所设计的 API 网关其功能和性能进行测试，笔者使用比较成熟的测试工具进行测试。主要分为性能测试、安全测试、用户请求接入测试以及用户请求处理测试。具体的压力测试要求如表 1 所示。

表 1 相关参数的介绍

名称	说明
RT	响应时间
TPS	每秒处理事务的能力
QPS	每秒的查询数量
性能测试	测试系统是否满足预期的性能

2　结果与讨论

2.1　API 网关的接入性能测试结果

1) 用户请求接入测试。

笔者在 Windows 环境下利用 LoadRunner 测试工具对设计的 API 网关系统请求接入的性能进行分析，LoadRunner 测试工具的参数均选择系统默认值。测试脚本分别为采用和不采用用户活跃度区分策略对用户请求接入数据进行重复测试，建立 API 网关用户高并发请求场景。测试系统的拒绝访问率、用户活跃区分度和 API 网关拒绝访问率测试结果如图 5 所示。

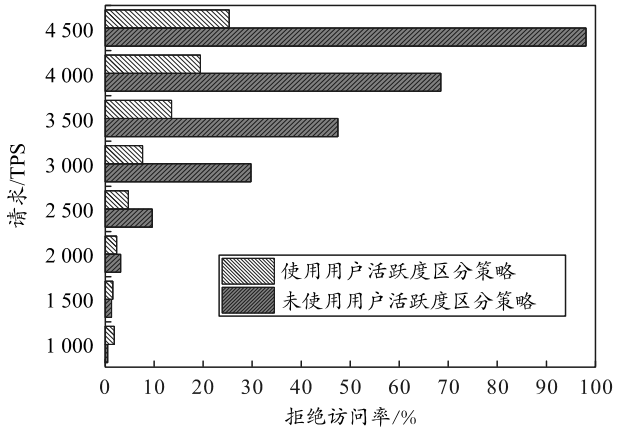


图 5　API 网关接入请求性能测试结果

从图 5 中可以看出：随着用户计入请求数量的增加，未使用用户活跃度区分策略的测试系统在用户请求数达到 2 500 时，拒绝访问率达到 10%；采用用户活跃度区分策略的测试系统在 3 000 TPS 时，拒绝率达到 10%，并发请求数为 4 000 时，拒绝率达到 20%，拒绝率开始明显增加。可见，API 网关的用户高并发请求处理数小于 3 000 TPS，便可以满足高并发场景的要求。

2) 用户请求处理测试。

同理，该实验仍然采用 LoadRunner 测试工具，通过调用协议接入和服务调用的 API，对 API 网关处理用户请求的性能进行分析，使用的用户数据请求类型为北向简单对象访问协议(simple object access protocol，SOAP)接入请求 2 kB 和南向表述性状态转移(representational state transfer，REST)服务应答消息 2 kB，并发用户数为 50 名，进行 30 min 的 API 网关请求，对 API 网关每分钟的每秒事务处理数和响应时间进行记录，测试结果如图 6 所示。可以看出：API 网关每秒处理事务数的最大值为 3 450 TPS，系统响应的最大值为 13 ms。在 3.1 节中测试得到系统的请求性能需求为 3 000 TPS，响应系统的实验要求为 20 ms。可见，设计的 API 网关可以满足接入请求的预期性能，可以投入到微服务架构的应用中。

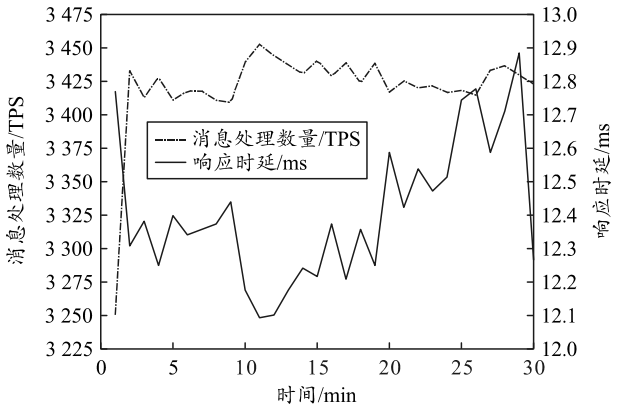
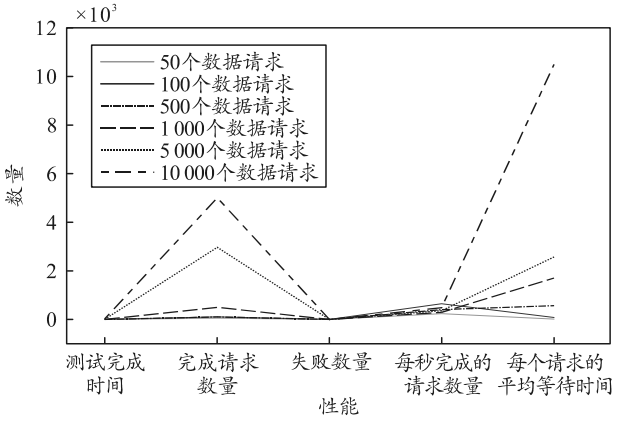


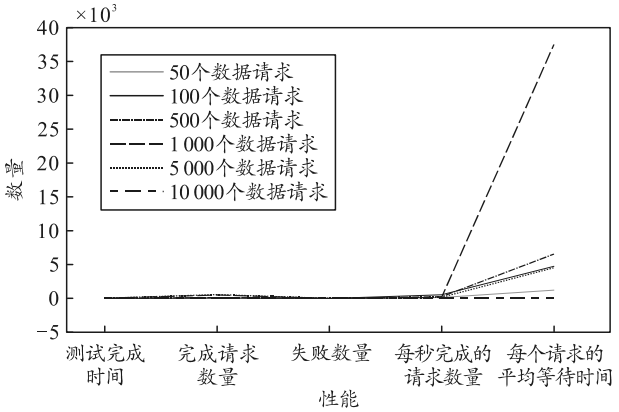
图 6　API 网关每秒处理事务数和响应时延测试

2.2　API 网关与其他模型的性能测试比较

为了更好地分析 API 网关的性能，笔者引入了一种网络接口对象(NIO)模型与之对比。NIO 模型就是以网络接口为基础对象的一种网关模型，并且是近几年网关研究领域的一种热点模型。根据上述设定的参数，API 网关和 NIO 模型具体的性能测试结果如图 7 所示。



(a) API 网关性能测试结果



(b) NIO 模型性能测试结果

图 7　不同网关性能测试结果

从图 7 中 API 网关性能与 NIO 模型性能的测试比较中，可以发现 API 网关在 50、100、500、1 000、5 000 和 10 000 个请求的条件下，数据是处于波动变化中，NIO 模型在同样数据请求的条件下数据波动变化不明显。为了更清楚地比较两者之间的差异，在不同数据的请求下对性能测试结果进行比较，具体如表 2—4 所示。

表 2　100 个数据请求下的性能比较

性能	测试完成时间/ms	完成请求数量	失败数量性能	每秒完成的请求数量	每个请求的平均等待时间/ms
NIO 模型	0.093 8	5	0	533.05	4 750.000
API 网关	0.156 0	5	0	642.74	77.791

表 3　1 000 个数据请求下的性能比较

性能	测试完成时间/ms	完成请求数量	失败数量性能	每秒完成的请求数量	每个请求的平均等待时间/ms
NIO 模型	10.450	500	0	123.40	37 520.123
API 网关	3.403	500	0	293.87	1 701.440

表 4　10 000 个数据请求下的性能比较

性能	测试完成时间/ms	完成请求数量	失败数量性能	每秒完成的请求数量	每个请求的平均等待时间/ms
NIO 模型	23.309	0	0	0	0
API 网关	20.985	5 000	0	476.53	10 492.42

通过将 API 网关和 NIO 模型下的 100、1 000 和 10 000 个数据请求数量对比之后发现，在这三者的请求数量之下，NIO 模型的数据变化波动最大，而 API 网关的数据就比较平稳，可见基于 NIO 模型的系统并发性和稳定性表现一般，并且随着并发量的增长，系统就会出现宕机的风险，无法进行数据的处理，所以相比之下 API 网关的结果更优。

将 API 网关和 NIO 系统不同数据请求下的等待时间进行对比，具体的结果如图 8 所示。

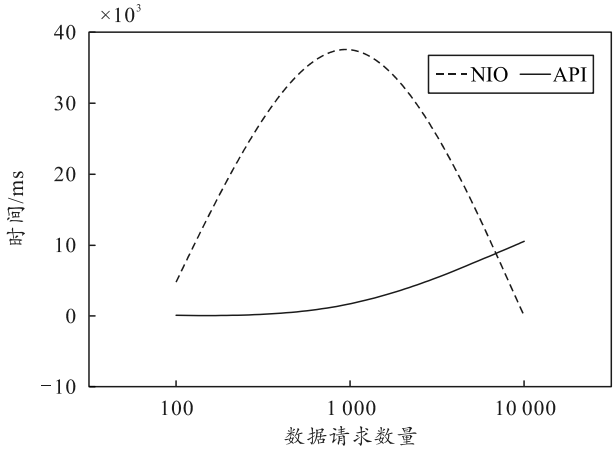


图 9　不同数据请求下的等待时间

从图 8 中可以发现：在 100 个数据请求的情况下，NIO 的等待时间是 4 750 ms，API 的等待时间是 77.791 ms；在 1 000 个数据请求的情况下，NIO 的等待时间是 37 520.132 ms，API 的等待时间是 1 701.447 ms；在 10 000 个数据请求的情况下，NIO 的等待时间是 0，API 的等待时间是 10 492.423 ms。可见，随着数据量的增大，每个请求的等待时间都在不断的增大，但明显可看出基于 NIO 的系统等待时间要明显的多于使用 API 网关的等待时间。

3　结论

为建立高效实用的 API 网关，进一步提升软件的开发效率，笔者采取 API 网关对软件的服务实施封装和控制：首先，笔者对微服务架构平台化服务 API 网关的需求进行了总结，据此构建了 API 网关的整体框架；其次，对设计的 API 网关中的各个模块进行了设计，基于 Netty 框架对用户请求的活跃度进行了筛选；最后，设计实验对设计的 API 网关的性能进行了验证。验证结果表明：1) 利用用户活跃度区分策略能够有效提高系统的请求响应数；2) 将 NIO 模型与 API 网关的性能进行比较时发现，不论是模型的响应时间还是数据的稳定性，API 网关都能呈现最优的效果。本文中的不足之处在于系统的响应性能还需要提高，后续笔者将着重对系统的响应性能进行研究，旨在提出一种高性能的 API 网关。

参考文献：

[1] 黄强文, 曾丹. 基于Spring Cloud和Docker的分布式微服务架构设计[J]. 微型电脑应用, 2019, 35(6): 98−101.

[2] ZUO X, SU Y, WANG Q, et al. An API gateway design strategy optimized for persistence and coupling[J]. Advances in Engineering Software, 2020, 148: 102878.

[3] XU R, JIN W, KIM D. Microservice Security Agent Based on API Gateway in Edge Computing[J]. Sensors, 2019, 19(22): 4905.

[4] 温馨, 樊婧雯, 王富强. 基于OpenResty平台的API网关系统的设计与实现[J]. 信息化研究, 2020, 46(3): 62−68.

[5] 吴润. 基于 API 网关的微服务组合策略研究[J]. 数码世界, 2019(3): 84−86.

[6] 郭栋, 王伟, 曾国荪. 一种基于微服务架构的新型云件PaaS平台[J]. 信息网络安全, 2015, 179(11): 15−20.

[7] VENUGOPAL M. Containerized Microservices architecture[J]. International Journal of Engineering and Computer Science, 2017, 6(11): 23199−23208.

（下转第 36 页）