文章编号: 2095-2163(2021)06-0168-06　　中图分类号: TP393　　文献标志码: A

基于 Netty 的高性能消息中间件设计与实现

王　宁, 张　娜, 于泽川, 苏逸凡, 包晓安
(浙江理工大学 信息学院, 杭州 310018)

摘　要: 随着互联网业务规模的扩大以及访问量的增多,应用服务可能因流量过大而产生崩溃。为解决此问题和保证应用服务器接收消息的准确性,本文设计了一种基于 Netty 的消息中间件。该中间件以 Netty 网络框架为基础实现异步通信,同时为了提升通信双方的编解码速度及缩减整个通信流程的时间,自定义了一种通信协议。该中间件根据本协议特点设计了消费模型及基于文件系统的存储模型,并在消费者端进行了幂等性处理,以提高中间件的可靠性与准确性。实验表明,相比于不使用中间件或使用其它中间件,该中间件在产业化的智慧宿管平台中应用,有效提高了应用服务的响应速率以及接收消息的准确率,保证了智慧宿管平台的平稳运行。

关键词: 消息中间件; Netty; 消费模型; 存储模型; 幂等性处理

Design and implementation of high-performance message middleware based on Netty

WANG Ning, ZHANG Na, YU Zechuan, SU Yifan, BAO Xiaoan
(School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China)

【**Abstract**】With the expansion of Internet and the increasing of the traffic, application service might collapse due to excessive flow. This paper designed a message middleware based on Netty to solve this problem and ensure the accuracy of the message received by the application server. This middleware realized asynchronous communication based on Netty network framework. At the same time, in order to improve the encoding and decoding speed of both sides of communication and reduce the time of the whole communication process, a custom communication protocol was defined. For this middleware designed consumption model and storage model based on file system according to the characteristics of this protocol, and carried out idempotent processing on the consumer side, so as to improve the reliability and accuracy of middleware. The experiment shows that compared with no middleware or other middleware, the application of this middleware in the industrialized smart dormitory administrator platform can effectively improve the response rate of application service and the accuracy rate of receiving messages, and ensure the smooth operation of the smart dormitory administrator platform.

【**Key words**】message middleware; netty; consumption model; storage model; Idempotent treatment

0　引　言

随着互联网技术的迅速发展、物联网应用的快速普及以及大数据时代的来临,传统互联网应用服务已不能满足高并发低延时的需求[1]。当海量物联网设备终端在短时间内连续向业务服务器发送访问请求时,业务服务器无法及时处理这些访问请求,继而造成大量访问请求在服务端的堆积,瞬时数据量激增,最终导致服务器无法及时响应客户端,甚至会出现数据丢失的情况,系统服务的实时性与可靠性出现问题[2]。消息中间件应运而生,通过引入消息中间件,应用服务采用异步通信的方式,可以有效提升服务器的响应速率,同时保证了应用的可靠性。

本文通过对 ActiveMQ、RocketMQ、kafka 等当前流行的消息中间件研究发现:ActiveMQ 默认情况下使用的是消息推送的方式,当数据量骤增的情况下,如果其中一个消费者的消费能力较差,可能导致消息在消费者端产生堆积,无法解决互联网应用服务目前遇到的问题[3-4];而 RocketMQ 与 kafka 使用的消费模式是基于 partition 的,其将每个主题分为一个或多个 partition,每个消费者与其中一个 partition 静态绑定[5],若其中一个 partition 的消费者消费能力较慢时,则无法通过增加消费者数量的方式来加快该 partition 的消费速度,进而导致消息积压的问

基金项目: 浙江省重点研发计划项目(2020C03094);浙江省自然科学基金青年基金(LQ20F050010);国家自然科学基金(6207050141)。

作者简介: 王　宁(1996-),男,硕士研究生,主要研究方向:计算机网络与分布式处理;张　娜(1977-),女,硕士,副教授,主要研究方向:分布式数据处理、软件工程、智能信息处理。

通讯作者: 张　娜　Email:zhangna@ zstu.cn

收稿日期: 2021-02-17

题[6-7]。

综上考虑,本文设计了一种基于 Netty 框架的消息中间件,能够较好的解决上述中间件在应用服务中所产生的问题。本文中间件采用发布/订阅模式,消息生产者 Producer(物联网设备终端)将数据消息发布到中间件服务器 Broker 的消息队列中,再由消息消费者 Consumer 主动从 Broker 的消息队列中拉取并消费。该中间件 Broker 与 Consumer 通信模块基于 Netty 网络框架开发,能够满足短时间内连接数与消息数据急速增多的需求。同时利用 Netty 框架对自定义协议的支持,提出了一种简单的、能够快速解析的通信协议,以进一步增强中间件系统的并发处理能力。本文提出的以 Netty 网络框架为基础实现异步通信的中间件,在本实验室开发的智慧宿管平台中进行了广泛应用并得到了很好的验证。

1　相关技术

1.1　Netty 线程模型

Netty 是一个高性能的异步事件驱动模型的网络框架,其高性能主要得益于其 I/O 模型和线程模型。Netty 的实现主要基于 Reactor 主从多线程模型,将接收客户端请求与处理连接产生的事件这两部分进行了拆分。Netty 服务端使用两个 NioEventLoopGroup,一个叫作 bossGroup 作为主线程池,另一个叫作 workerGroup 作为子线程池。主线程池负责接收客户端请求,子线程池负责处理所有连接产生的事件。当 bossGroup 接收了客户端的连接请求后,会将这个请求封装成 NioSocketChannel,然后注册到 workerGroup 上,此后这个连接产生的所有事件都由 workerGroup 来处理。Netty 中的 NioEventLoopGroup 中包含多个 EventLoop,每个 EventLoop 都聚合了一个多路复用器 Selector,相当于一个 Reactor 单线程模型,从而只需少量的 EventLoop 就可以同时并发处理大量客户端连接。并且为了提升性能,Netty 在 I/O 线程内部采用了串行操作,避免了多线程竞争情况下频繁加锁导致的性能下降问题。因此,每个客户端连接在其生命周期中只会与一个 EventLoop 进行绑定,该连接产生的所有的读写事件都由与之绑定的 EventLoop 进行处理,无需进行线程切换,从而避免了多线程操作导致的锁竞争问题。Netty 线程模型如图 1 所示。

1.2　消息传输模型

Java 消息服务是针对 Java 平台之间消息传输所制定的一套规范,其中提出了点对点(Point To Point,p2p)与发布/订阅(Publish/Subscribe,PUB/SUB)两种消息传输模型[8]。两种传输模型结构如图 2 所示。

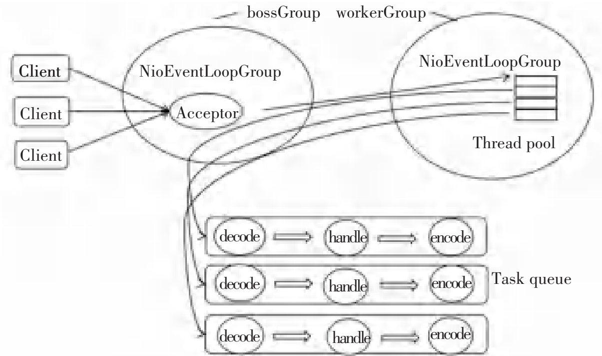
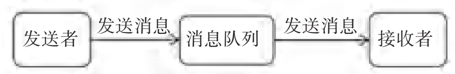


图 1　Netty 线程模型

Fig. 1　Netty thread model

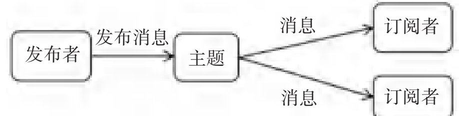
(1)点对点模型:点对点模型主要由发送者、接收者和消息队列 3 部分组成[9-10]。每条消息产生后,都由发送者将其发送到消息队列中,消息队列在收到消息后会将消息保存,并通知接收者,接收者接到队列发送的消费通知后,从消息队列中拉取消息进行消费,并在消费结束后通知消息队列[11]。若消息已经被接收者消费或是超过时间,消息队列便会将这个消息删除。这个模型下的消息,只能被一个接收者消费,并且接收者与发送者之间不存在时间相关性[12]。

(2)发布/订阅模型:发布/订阅模型是由发布者、主题、订阅者 3 个部分组成[13]。消息发布者将一条消息发送到对应主题后,消息订阅者可采用两种方式获取该条消息:一是订阅者主动询问消息服务器,对应主题中是否有未消费的消息;二是消息服务器接收到这条消息后,立即推送给所有订阅了这个主题的订阅者,而无需订阅者去询问。发布/订阅模型允许一条消息被所有的订阅者同时消费,具有一对多的特点。



(a) 点对点模型

(a) The point-to-point model



(b) 发布/订阅模型

(b) Publish/subscribe model

图 2　两种传输模型结构图

Fig. 2　Two kinds of transmission model structure diagram

2 系统设计

2.1 系统结构模型

本文设计的消息中间件由消息生产者 Producer、消息服务器 Broker 以及消息消费者 Consumer 3 部分组成。其中,生产者角色主要为智慧宿管平台系统中智能门锁以及移动终端;消费者角色则为智慧宿管平台系统的后台服务端。中间件与智慧宿管平台的关系模型如图 3 所示。

图 3 中间件与智慧宿管平台关系模型图

Fig. 3 Relationship model diagram between middleware and intelligent residential management platform

本文中间件的消息服务器 Broker 分为 4 个部分:协议制定、消息接收模块、消息消费模型以及消息持久化方式。

2.1.1 协议制定

本文设计的消息中间件是面向智慧宿管平台的,消息生产者主要是智能门锁等终端设备。由于智能门锁这一类的硬件设备具有低功耗、低内存以及芯片级设计等特点,所支持的网络协议也都较为简单,相对于其它网络设备也更接近底层[14]。因此,本文设计的消息中间件系统之间的数据传输都是通过 TCP(Transmission Control Protocol,传输控制协议)协议实现。TCP 协议是一种流式服务,所以各端接收消息数据时,会出现网络传输中最常见的粘包拆包问题。为此本文设计了一种自定义协议,并通过实现 Netty 为自定义协议所提供的两个编解码器抽象类来解决粘包拆包问题。自定义协议组成见表 1。

表 1 协议组成

Tab. 1 Composition of the agreement

字段名	协议版本	消息长度	操作码	门锁设备 SN 号	消息体	校验码
字节数(byte)	1	4	2	1	N	1

2.1.2 消息接收模块

消息服务器 Broker 基于 Netty 开发,核心类为 SmartDepBrokerServer,类结构见表 2。首先初始化启动类 ServerBootstrap,绑定主从线程池 bossGroup 与 workerGroup,并为 workerGroup 预设了 Channel 初始化器。

表 2 SmartDepBrokerServer 类结构

Tab. 2 Class structure of SmartDepBrokerServer

SmartDepBrokerServer		
-brokerServerPort	:int	=6 800
- bootstrap	:ServerBootstrap	
-bossGroup	:NioEventLoopGroup	
-workerGroup	:NioEventLoopGroup	
- handler	:MessageBrokerHandler	
- defaultEventExecutorGroup	: DefaultEventExecutorGroup	
+init( )	: void	
+ start( )	: void	
+ shutdown( )	: void	

Broker 启动后,通过 bossGroup 监听所有客户端的连接请求,进行连接建立。连接建立时,bossGroup 将连接封装成一个 NioSocketChannel,并将其注册到 workerGroup 上;workerGroup 通过 Channel 初始化器为这个连接的 NioSocketChannel 添加预设的编解码器以及消息处理器 MessageBrokerHandler,并监听该 SocketChannel 所有的读写事件。当有数据可读时,先将读取的数据通过解码器按照自定义协议解码,然后通过调用 MessageBrokerHandler 中的 handleMessage 方法对消息数据进行处理。消息处理器 MessageBrokerHandler 类结构见表 3。

表 3 MessageBrokerHandler 类结构

Tab.3 Class structure ofMessageBrokerHandler

MessageBrokerHandler	
-producerHandler	:AtomicReference<ProducerMessageListener>
-consumerHandler	:AtomicReference<ConsumerMessageListener>
-message	:AtomicReference<RequestMessage>
+buildProducerHander( )	:MessageBrokerHandler
+buildConsumerHandler( )	:MessageBrokerHandler
+beforeMessage( )	: void
+handleMessage( )	: void

2.1.3 消息消费模型

目前 Kafka 与 RocketMQ 等一些流行的中间件使用的都是基于 partition 的消费模型,即将一个主题 Topic 分为多个 partition,而每一个 partition 则与一个消费者 Consumer 进行静态绑定,消费者需要消费消息时,只会从与之绑定的 Partition 中获取消息。但是当消费者数量多于 partition 数量时,会导致有些消费者被闲置,浪费了系统资源[15]。并且一旦遇到消息发送方突然大量发送消息的场景,消费者处

理速度较慢,而基于 partition 的消费模型无法通过消费者扩容来提高处理速度,从而导致消息大量堆积。

本文的中间件采用发布/订阅模型,并将所有同属一个主题 Topic 的消息集合起来,形成一个主题队列 Queue。每个主题队列对应一个消费者集群 ConsumerCluster,每个消费者集群中的消费者数量能够根据消息数量动态扩容与缩容。当消息生产者发布消息到消息服务器的对应主题队列后,消息服务器首先调用消息持久化模块,将消息持久化到日志文件中,持久化完成后才会去轮询订阅了该主题的消费者集群 ConsumerCluster,通知集群中的消费者消费,再由消费者主动去消息服务器中拉取消息进行处理消费。由于通知消费者消费消息之前,消息服务器已经将消息持久化到内存中,确保消息不会丢失,实现了消息中间件的高可靠性。消费模型如图 4 所示。

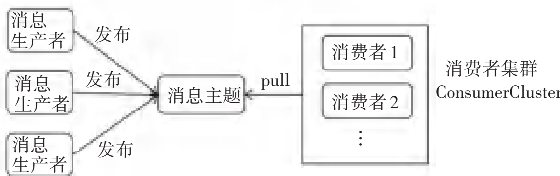


图 4　消费模型图

Fig. 4　Consumption model chart

2.1.4　消息持久化

为了保证中间件的高可靠性,避免消息服务器因意外宕机导致的消息丢失问题,消息服务器需要在重启后能够恢复原先的消息队列,消息中间件一般都需有消息持久化机制[16-17]。本文的消息中间件使用的持久化方式是基于消费模型设计,采用文件存储系统,将消息持久化到日志文件中。每次消息服务器接收到新消息时,会按照顺序添加的方式,将消息存储到日志文件 message.log 中。相比于其它访问文件的方式,本文采用顺序添加的方式具有更高的效率。由于将所有的消息都存储到一个日志文件中,为了区分不同主题的消息,本文通过引入一个索引文件 offset-topicName.log,来记录不同主题队列中的消息在 message.log 文件中的偏移量,每个主题队列对应一个索引文件。最后使用一个消费记录日志 consume-clusterId.log 记录每个消费者集群的消费情况,保存所消费的消息在索引日志中的序号,并采用顺序消费的方式,无需为每个消息单独记录消费状态,极大地降低了成本。存储模型如图 5 所示。

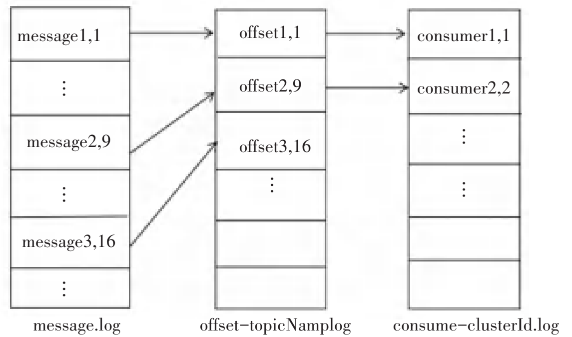


图 5　存储模型图

Fig. 5　Storage model diagram

2.1.5　幂等性处理

正常情况下,消费者在消费一条消息之后,都会给消息服务器回发一条确认消息。当消息服务器收到确认消息后,消费者才能拉取下一条消息。但是,由于网络环境的不确定性,消费者所发送的确认数据包在网络传输中可能会丢失,从而导致消息服务器误认为还没有消费者消费这条消息,继而对其进行重发,消费者就会重复消费这条消息,最终造成数据异常[18-19]。

对于系统幂等性处理,本文做法是首先保证每条消息都拥有一个全局唯一标识,然后在数据库中建立一张存储已处理的消息表,并将每条消息的全局唯一标识作为表的主键。每当消费者获取一个新消息时,先尝试向数据库中插入这条消息记录,如果插入失败,说明已经处理过该消息,并向消息服务器返回一条确认应答,以便获取下一条消息。

2.2　数据消息处理

消息生产者(终端设备)、消息中间件和消息消费者(智慧宿管平台服务端)3 者之间的数据交互主要是消息生产与消息消费这两个过程。其中,消息生产主要是消息生产者与中间件之间的数据交互,消息消费则是消息中间件与消息消费者之间的数据交互。终端设备发送数据消息时,会先向消息中间件发送建立连接的请求,中间件通过 Netty 通信模块与终端设备建立连接。连接建立后,终端设备便会将数据消息上传至消息中间件,中间件在接收到数据信息后,先将消息持久化存储到数据日志文件中,以保证消息的可靠性,然后再将数据消息根据其 Topic 放入到对应队列中,并通知消费者读取消息进行处理。整个消息处理流程如图 6 所示。

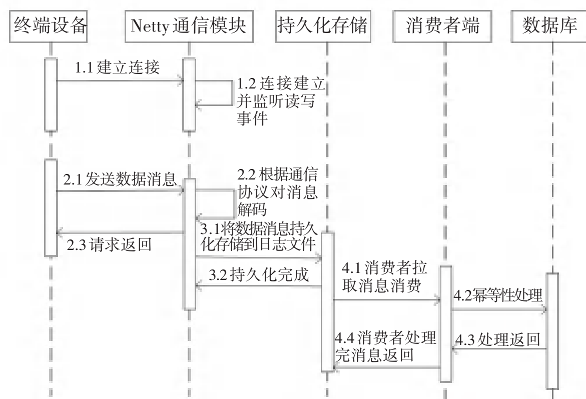


图 6　数据消息处理流程图

Fig. 6　Data message processing flow chart

3　实验结果与分析

3.1　实验环境

本文系统运行于一台四核高性能阿里云服务器上,服务器配置参数为:CPU 主频为 2.5 GHz、采用 Intel(R) Xeon(R) Platinum 8163(Skylake)处理器、8 G内存、40 G 高效云盘、操作系统版本为 CentOS 6.9;实验通过开源压力测试工具 Jmeter 模拟大量终端设备,同时唤醒业务场景,向系统发起大规模 TCP 连接请求;压测机器使用一台双核心四线程的计算机,其具体配置参数为:Intel(R) Core(TM) i5-4210M 处理器、CPU 主频为 2.60 GHz、8 G 内存、128 G固态硬盘、操作系统 Windows 10。

3.2　实验设计与结果分析

实验将通过对智慧宿管平台结合 NIO 通信进行压测,对智慧宿管平台结合高性能中间件 ActiveMQ 进行压测,以及对智慧宿管平台结合本文设计的中间件进行压测 3 种方案进行对比。从高并发情况下系统的总响应时间,以及数据准确率两方面来进行实验结果的对比分析。通过设置压测工具 Jmeter 的线程数,模拟实现不同数量的终端设备在 1 s内发送 TCP 请求,并设置请求参数为一串 16 进制数字,大小为 24 B,表示某一个门锁设备的开锁记录。

3 种方案的系统总响应时间如图 7 所示,并发量在 4 000 以下时,两种使用中间件的方案优势较为明显,其系统总响应时间明显少于传统 NIO 通信方案的总响应时间。由于传统 NIO 通信方案需要等待服务端将数据处理完毕后再返回,而数据处理是一个耗时操作,因此传统 NIO 通信方案的总响应时间最久。而当并发量达到 6 000 时,ActiveMQ 无法在运行过程中实现消费者集群的动态扩容,从而导致消息堆积并会阻塞生产者,响应时间随之增加。而本文设计的中间件方案通过动态改变消费者数量,加快了消息的处理,缩短了系统的响应时间,表现较为稳定。

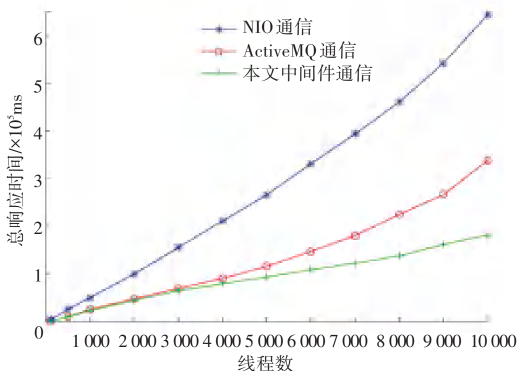


图 7　不同并发场景下系统总响应时间

Fig. 7　Total response time in different concurrent scenarios

在数据准确率方面,3 种方案的系统平均数据接收准确率如图 8 所示。当并发量小于 2 000 时,3 种方案的数据接收准确率表现相差不大;当并发量大于 5 000 时,NIO 通信以及 ActiveMQ 方案的数据接收准确率明显下降,出现较多数据丢失的情况,随着并发量上升,系统响应时间增长,导致了服务端响应异常。而本文中间件的表现则优于另外 2 种方案,数据丢失的情况较少,准确率稳定在 98%以上。

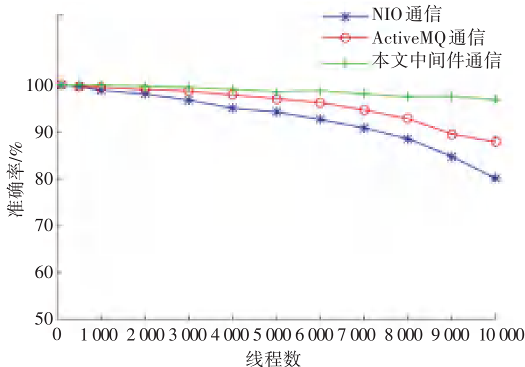


图 8　不同并发场景下系统接收数据的准确率

Fig. 8　The accuracy of receiving data in different concurrent scenarios

4　结束语

在当下大数据、大流量环境中,消息中间件的应用前景十分广阔。为解决智慧宿管平台在高并发情况出现的响应不及时、数据丢失的问题,本文设计的基于Netty框架的消息中间件和自定义协议进行通

(下转第 177 页)