基于 Netty 的 IoT 终端通信服务系统设计

顾振德[1,2] 刘子辰[2] 龙 隆[2] 牟林宏[1,2]

[1](重庆邮电大学通信与信息工程学院 重庆 400065)

[2](移动计算与新型终端北京市重点实验室(中国科学院计算技术研究所) 北京 100190)

摘 要 IoT 终端高并发连接通信服务系统,造成服务器数据交互 NIO 线程数飙升,导致服务器宕机。对此分析异步非堵塞 Netty 框架,提出一种通信服务系统的设计方案。方案包含优化 Netty 设计,提出自定义通信协议,并结合自定义线程池进行数据接收、session 管理、异常处理,日志记录,数据发送等主要模块的设计。经大于 2 000 并发测试表明,该方案的系统平均响应时间较 Java NIO 和 Netty 实现方案缩短了 97% 和 95%,吞吐量提高了 54% 和 33.2%。测试证明该设计具有较高的稳定性、可靠性以及高并发处理能力。

关键词 Netty 高并发 自定义线程池 Java NIO 吞吐量

中图分类号 TN915 TP3 **文献标识码** A DOI:10.3969/j.issn.1000-386x.2019.04.021

DESIGN OF IOT TERMINAL COMMUNICATION SERVICE SYSTEM BASED ON NETTY

Gu Zhende[1,2] Liu Zichen[2] Long Long[2] Mou Linhong[1,2]

[1](*School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing 400065, China*)

[2](*Beijing Key Laboratory of Mobile Computing and Pervasive Device (Institute of Computing Technology, Chinese Academy of Sciences), Beijing 100190, China*)

Abstract The high concurrency of IoT terminal communication service system connections leads to the soaring number of NIO threads in server data interaction, which eventually leads to server downtime. To solve this problem, we analyzed the asynchronous non-blocking Netty framework and proposed a design scheme of communication service system. The design scheme included optimizing Netty design, putting forward self-defined communication protocol and system design which was divided into data receiving, session management, exception handling, log recording, data sending and other main modules, combining with the custom thread pool. The concurrent tests over 2 000 show that the average response time of the design scheme is 97% and 95% shorter than that of Java NIO and Netty scheme, and the throughput is increased by 54% and 33.2%. The test proves that the system has high stability, reliability and concurrent processing ability.

Keywords Netty High concurrent Custom thread pool Java NIO Throughput

0 引 言

传统的 IoT 系统由三部分构成:传感器、网络传输、数据交互平台[1]。传感器技术已经取得了长足发展,市场上存在各种类型设备,网络传输的部署和应用都日趋成熟[2-4]。随着 IoT 的发展,终端的应用规模都在百万级别,终端向服务器并发地发送数据请求在某一时刻急剧增加,如何在短时间内提高服务器并发处理能力是数据通信服务系统开发面临的一个急需解决的问题[5]。传统的数据通信服务系统采用 Java 原生 NIO 技术方式实现,如果直接基于 NIO 类库和 API

收稿日期:2018-10-23。国家科技重大专项基金项目(2016ZX03002010-003)。**顾振德**,硕士生,主研领域:网络通信。**刘子辰**,博士生。**龙隆**,博士生。**牟林宏**,硕士生。

编程,会降低开发效率,同时出现 epoll bug,导致 Selector 空轮询,最终使 CPU 的占用率达到 100%。罗文韬[6]采用异步处理和基于事件驱动的机制来提高服务器高并发处理的效率。汪佳文等[7]提出一种动态负载均衡算法,并结合优化的 Pick-K 算法方案实现高并发传输。

本文提出一种基于 Netty 的通信服务系统的设计方案。该方案借助 Netty 的异步非堵塞、事件驱动等性能构建高性能网络通信程序[8],并通过结合自定义功能模块设计和自定义线程池进一步提高服务系统的并发处理能力。

1 Netty 简介与优化

1.1 Netty 介绍

Netty 是业界 NIO 框架中最流行的框架,它的健壮性、可扩展性、可定制性都是首屈一指的[9],可进行如 TCP、UDP 套接字服务器的开发。本系统因终端与服务器之间的数据通信要求实时性,因此终端与服务系统建立 TCP 长连接实现。基于 Netty 实现终端通信服务系统,可以不用过多关注连接的建立、数据的编解码等底层通信的实现,进而能够更好地关注业务模块的实现,极大地简化了网络编程。

1.2 Netty 优化设计

长链接需要维护每个链路自己消息接收和发送的缓冲区,而 JDK 原生 NIO 类库[10]无法动态扩容,从而给服务器带来沉重的内存负担。Netty 提供的 ByteBuf 支持容量的动态调整,选择 AdaptiveRecByteBufAllocatorv 在创建服务端时候指定 RecvByteBufAllocator,缓冲区的大小设置为消息的平均大小,避免额外的内存浪费。TCP 层面的接收和发送缓冲区的大小设置,对于长连接设置为 32 K。每个长连接就是一个会话,每个会话都有心跳等数据结构,给通信服务器带来沉重 GC(Garbage Collection)压力,同时消耗大量的内存[11]。本系统在设计通信服务端的时候采用 ByteBuffer 内存池技术来解决上述问题。

2 系统设计

2.1 协议设计

终端向服务器发送的数据是一连串的字节数据,为了让服务器识别这些字节数组,制定特殊的协议格式,服务器响应终端也要通过该协议。图 1 为协议的设计。

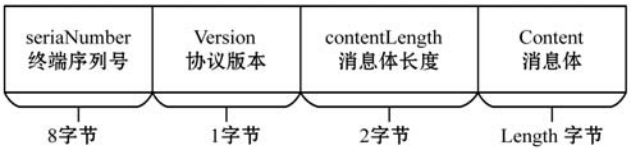

图 1 协议图

8 个字节的终端序列号,2 个字节的消息体长度和 1 个字节的协议版本,最后是消息体[12]。消息体设计代码如下:

```
public class ProtocolFrame{
private long seriaNumber;
private byte version;
private short contentLength;
private byte[ ] content
…
}
```

serialNumber 为终端的序列号,每一个终端将有唯一的序列号;version 表示协议的版本号;contentLength 为消息体的长度;content 表示消息体,消息体包含终端运行的各种参数等。

2.2 服务器设计

服务器设计划分为 6 模块,分别是异常处理、日志记录、数据接收、业务处理、数据发送、session 管理等模块。异常处理模块主要负责捕获 IoT 通信服务系统自身的异常,以及客户端的异常,从而提高系统的稳定性。处理异常日志记录模块主要满足性能测试和维护工作需要,设置输出内容,输出到控制台和文件等。数据接收模块首先对客户端的连接 IP 进行过滤,验证客户端的合法性。其次根据自定义通信协议解码接收的数据,对解码后的数据进行解密,验证数据的有效性和封装数据。同时添加空闲超时处理逻辑,设计失效时间为 180 s,若服务器 180 s 没有接受到数据包,则及时关闭超时的客户端连接。业务处理模块实现具体的业务逻辑,将来自客户端的数据分类存储到 MYSQL 数据库中,实现数据的持久化。数据发送模块主要负责将来自业务处理模块的数据进行处理并下发到终端,对数据加密,根据协议对发送数据编码,之后发送出去。session 管理模块在终端连接服务器时将 session 信息进行保存,从而管理和操作连接,定时清除非活跃的连接,释放内存,减轻服务器连接压力。终端通信服务系统的功能模块如图 2 所示。

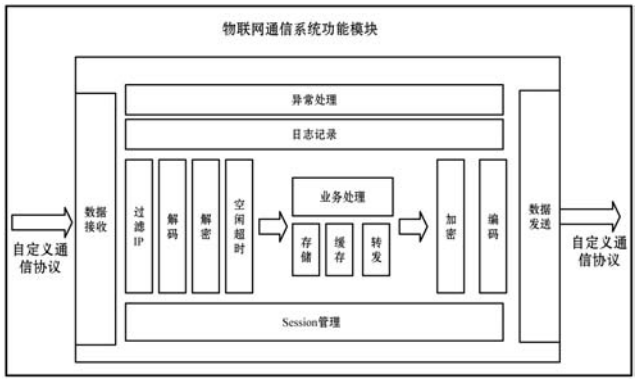

图 2　通信系统功能模块图

利用 Netty 框架构建服务器的大致流程:

1) 配置服务器的 NIO 线程组。

```
//该线程组用于处理服务器接收终端的连接;
master = isLinux ( )? new EpollEventLoopGroup ( DEFAULT_THREAD_NUM, threadFactory)
:new NioEventLoopGroup
(DEFAULT_THREAD_NUM, threadFactory);
//该线程组用于处理来自中端的网络读写;
worker = isLinux( )? new EpollEventLoopGroup
(DEFAULT_THREAD_NUM,threadFactory)
:new NioEventLoopGroup
(DEFAULT_THREAD_NUM, threadFactory);
```

2) 创建 ServerBootstrap 对象,传递 master、work 两个线程池。

```
bootstrap = new ServerBootstrap( );
```

3) 调用 ServerBootstrap 类的 childHandler 方法传入接口实现类处理具体的业务,绑定 I O 事件的处理类,监听端口 IP 地址,处理网络 IO 事件。

```
bootstrap. group( master, worker);
bootstrap. channel( serverChannelcls);
bootstrap. localAddress( ipaddress, port);
bootstrap. childHandler( handler);
```

网络 IO 事件处理类的部分关键代码块:

```
@ Override
protected void initChannel ( SocketChannel channel) throws Exception {
//IP 过滤,对黑名单中 IP 拒绝连接
channel. pipeline( ). addLast( filter);
//数据解码
channel. pipeline( ). addLast( getProtocolFrameDecoder( ));
//数据编码
channel. pipeline( ). addLast( protocalFrameDecrypt);
//数据解密
channel. pipeline( ). addLast( protocalFrameEncoder);
//数据加密
channel. pipeline( ). addLast( protocalFrameEncrypt);
//添加空闲超时的工具,关闭超时连接,删除
channel. pipeline ( ). addLast ( messageProcessofEventLoopGroup, new IdleStateHandler(180, 0, 0));
//具体业务处理
channel. pipeline ( ). addLast ( messageProcessofEventLoopGroup, messageProcessor);
//异常处理
channel. pipeline ( ). addLast ( messageProcessofEventLoopGroup, exceptionHandler);
}
```

4) 使用创建的 ServerBootstrap 对象绑定,开始监听,用于异步操作的回调通知。

```
ChannelFuture f = bootstrap. bind( ). sync( );
```

经过以上操作,服务系统成功启动,等待终端的连接请求,服务器接收到终端的数据后,将从过滤 IP 模块开始传递数据并处理。

2.3　业务处理线程池设计

业务处理需要访问数据库,如果直接使用 Netty 的 worker 线程进行业务处理,可能会因不确定的执行时间导致线程被阻塞,最终导致服务器宕机。因此采用自定义业务线程池来处理比较耗时的业务逻辑,进而提高通信系统性能。ThreadPoolExecutor 类从 JDK1. 5 开始被提供自定义线程池[13]。newFixedThreadPool 可创建固定大小的线程池和可控控制线程最大并发数,当线程池中的线程数达到其设定的核心线程大小时,新创建的线程会在无界队列中等待[14]。当线程池中的某个线程执行失败时新创建的线程会替代执行剩下任务。当线程池中创建的线程调用 shutdown 函数时会退出线程池[14]。线程池最大尺寸不要超过系统资源限制,算法公式如下:

$$N_s = N_i \times N_j \times \left(1 + \frac{T}{C}\right) \tag{1}$$

式中:N_s 为线程的设置数;N_i 为 CPU 核心数;N_j 为预期 CPU 核心利用率;T/C 为任务等待的时间与执行时间的比值[15]。

IoT 终端与服务器通信的数据交互主要包括数据的解析、访问缓存数据库、入库 MYSQL 数据库、发送数据的预处理等操作。其中数据的编解码、加解密等执行时间短操作交由 NIO 线程执行,而耗时比较长的业务处理交由自定义线程池执行完成,具体执行流程如图 3 所示。

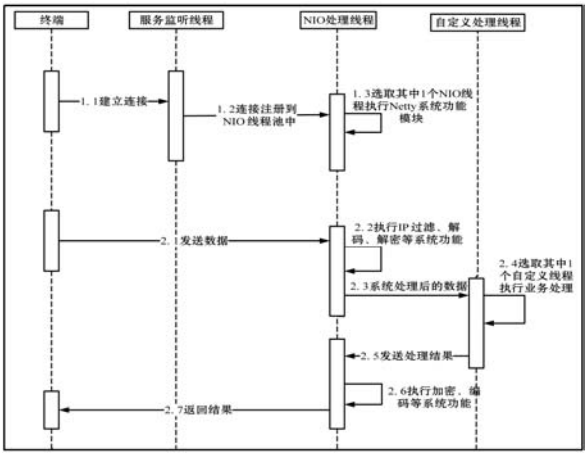

图 3 Netty 的 NIO 线程池与自定义线程池交互流程图

3 测试结果

采用开源压力测试工具 Jmeter 模拟 IoT 终端进行通信系统性能测试,从服务系统平均响应时间、系统 IO 吞吐量 2 个方面进行数据分析。因条件有限,性能测试在网络带宽 100 MB/S 的局域网中执行,服务器 CPU 四核 Intel(R) Core(TM) i5-3210M CPU @2.50 GHz 内存 8 GB, 操作系统 centos-release-7-5.1804.e17.centos.2.x86_64。

模拟 IoT 终端每 2 秒发送一个 180 字节的实时数据,由图 4 可知,当并发数小于 1 600 时,采用 Netty 框架结合自定义线程池方案并没有较大的优势,因为调用本地方法会有一定的系统开销。当并发数量达到 2 400 时,采用 Netty 框架结合自定义线程池方案优势较为明显。基于 Java NIO 的实现方案因系统消耗资源较大导致响应时间过长,同时基于 Netty 的实现方案因业务操作性能开销比较大,阻塞了 I/O 线程,导致响应时间过长。采用 Netty 框架结合自定义线程池方案系统平均响应时间仍在 100 ms 以下,且未达到瓶颈。较 Java NIO 和 Netty 实现方案平均响应时间缩短了 97% 和 95%。

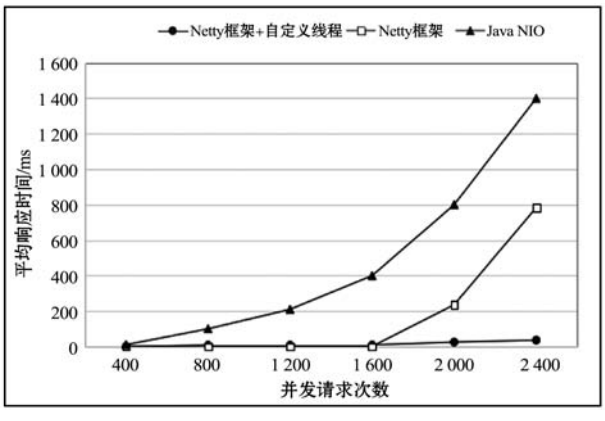

图 4 平均响应时间比较

在吞吐量方面,由图 5 可知,当并发达到 1 200 时,基于 Java NIO 的实现方案平均吞吐量达到最高值,随着并发数请求数增大,服务器出现异常吞吐量下降。当并发数达到 2 000 和 2 400 时,采用 Netty 框架结合自定义线程池方案比基于 Netty 框架平均吞吐量增加了 16.7% 和 33.2%。实验证明,当 4 000 并发连接系统时,系统仍然稳定运行,如图 6 所示。这说明采用 Netty 框架结合自定义线程池方案符合 IoT 终端通信服务系统处理高并发的设计要求。

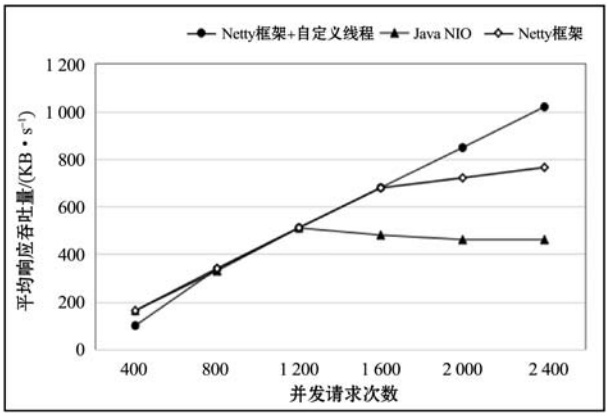

图 5 平均响应吞吐量比较

```
[root@gzd163 ~]# netstat -anp | grep 20007 | wc -l
4001
```

图 6 连接统计图

4 结 语

本文介绍了 IoT 移动终端与服务器数据交互系统,基于 Netty 框架和 Java 的 ThreadPoolExecutor 结合的设计方案,重点研究设计通信系统编解码、加解密等模块,提出一种自定义线程池处理耗时业务的设计方案。经验证采用 Netty 框架结合自定义线程池可提高系统的高并发处理能力。该设计已成功应用于某企业通信系统当中,并且与 30 000 多台 IoT 终端进行数据交互,未出现数据交互不稳定情况,且系统运行效果良好。实践证明,该设计是一种可参考的通信设计方案。

参 考 文 献

[1] 韦乐平. 物联网的特征、发展策略和挑战[J]. 现代电信科技,2011,27(4):1-5.

[2] Liu L, Zhou Y, Tian L, et al. Interference Aware CoMP for Macrocell-Based Heterogeneous Ultra Dense Cellular Networks[C]//2018 IEEE International Conference on Communications(ICC). IEEE, 2018: 1-6.

[3] Liu Z, Zhou Y, Han X, et al. Energy efficiency of CoMP-based cellular networks with guaranteed coverage [C]// Wireless Communications and NETWORKING Conference.

IEEE, 2013:2034 - 2039.
[4] Zhou Y, Liu H, Pan Z, et al. Cooperative Multicast with Location Aware Distributed Mobile Relay Selection: Performance Analysis and Optimized Design[J]. IEEE Transactions on Vehicular Technology,2017,66(9): 8291 - 8302.
[5] 蔡谊. 基于 MINA 框架的网络管理软件设计[J]. 通信技术,2013,46(3):115 - 117.
[6] 罗文韬. 大型电商网站服务系统关键改进的研究和实现[D]. 北京:中国科学院大学工程管理与信息技术学院, 2016.
[7] 汪佳文, 王书培, 徐立波, 等. 基于权重轮询负载均衡算法的优化[J]. 计算机系统应用, 2018, 27(4):138 - 144.
[8] 魏莹. 基于 Netty 框架的智能终端与服务器通信的研究[D]. 西安:西安电子科技大学,2017:19.
[9] 李林锋. Netty 权威指南[M]. 电子工业出版社,2015:60 - 63.
[10] Monet D G, Reid I N, Kirkpatrick J D, et al. New Neighbors from 2MASS: Activity and Kinematics at the Bottom of the Main Sequence[J]. Astronomical Journal, 2000, 120(2):1085 - 1099.
[11] 韩宏峰,冯石,罗羿隆. 基于 Java 与 Python 的面向对象编程的基本特征研究[J]. 软件工程,2017,20(8):1 - 3.
[12] 龚鹏, 曾兴斌. 基于 Netty 框架的数据通讯服务系统的设计[J]. 无线通信技术,2016,25(1):46 - 49.
[13] Pyarali I,Spivak M,Cytron R,et al. Evaluatingand optimizing thread pool strategies forreal-time CORBA[J]. ACM SIGPLAN Notices,2001,36(8):214 - 222.
[14] 吉利,潘林云,刘姚. 线程池技术在网络服务器中的应用[J]. 计算机技术与发展,2017,27(7):149 - 151.
[15] Peierls T,Goetz B,Bloch J,et al. Java Concurrency in Practice[M]. Addison-Wesley, 2006:1171 - 1177.

(上接第 102 页)

图 6　通信接口测试图

3) 效果对比测试。系统搭建好后,需对系统作出客观的综合测试评估,以得出使用该系统后驱鸟效果的提高率。为了能更好地观测,观察人员配备了高倍单筒望远镜、测距仪、摄像机、夜视仪等,在机场不同功能区安排对应的工作人员蹲点观察记录,得到如表 2 所示的观测结果。由表 2 可知使用了智能化驱鸟系统后对场内常见的涉场鸟类有不错的效果提升。

表 2　系统效果评估

驱鸟方法	鸟种				
	白鹭	牛背鹭	红隼	斑鸠	鹊鸲
	驱鸟效果				
原有驱鸟方法	57%	61%	45%	70%	64%
智能化驱鸟方法	89%	85%	82%	92%	93%

4　结　语

本文对机场智能驱鸟系统的上位机进行了设计与实现,并针对其以往在设备量增加后数据量急剧增加的问题上提出了几点改进方法。最后通过测试结果验证了这些方法的有效性,对今后系统挂载更多的驱鸟设备具有重要意义。在下一步工作中,将结合深度学习方法,探寻系统的智慧底线,帮助机场切实做好安全保障工作。

参　考　文　献

[1] 赵超. 基于物联网的机场驱鸟系统的研制[D]. 济南:山东师范大学,2014: 1 - 40.
[2] Ning H S, Chen W S, Mao X, et al. Bird-aircraft strike avoidance radar[J]. IEEE Aerospace and Electronic Systems Magazine,2010,25(1):19 - 28.
[3] 陈裕通,施泽荣,刘玉芬,等. 机场驱鸟设备的使用与维护技术[M]. 合肥:合肥工业大学出版社,2017: 1 - 272.
[4] 钟诗胜,张恩惠,王瑞,等. 一种机场空中驱鸟设备与系统的研究[J]. 南京理工大学学报:自然科学版,2004,28(3):238 - 242.
[5] Bradbeer D, Ryan E, Witter I. Wildlife hazard management handbook[M]. 2nd ed. Airports Council International,2013: 1 - 221.
[6] Stables E R, New N D. Birds and aircraft: the problems[C]//The Problems of Birds as Pests:Proceedings of a Symposium Held at the Royal Geographical Society, London, on 28 and 29 September 1967. Elsevier, 2013. 112 - 120.
[7] Rappaport T S, Annamalai A, Buehrer R M, et al. Wireless communications: past events and a future perspective[J]. IEEE Communications Magazine, 2002, 40(5): 148 - 161.
[8] Alliance Z B. ZigBee standards overview[OL]. 2017. www.zigbee.org.
[9] 陈裕通, 刘立程, 魏国廷,等. LTE-A 层 3 中继 RRC 层消息处理模块的设计与实现[J]. 计算机应用与软件,2015,32(6):149 - 151,159.