****



**研 究 生 毕 业 论 文**

**（申请工程硕士学位）**

|  |  |
| --- | --- |
| **论文题目** | 基于Tensorflow的信息抽取系统的设计与实现 |
| **作者姓名** | 安 磊 |
| **学科、专业名称** | 工程硕士(软件工程方向) |
| **研究方向** | 软件工程 |
| **指导教师** | 伏晓　副教授 |

**2019年 04 月 11 日**

**学 号： MF1732001**

**论文答辩日期： 年 月 日**

**指 导 教 师：**

**基于Tensorflow的信息抽取系统的**

**设计与实现**

|  |  |
| --- | --- |
| **作 者:** | **安磊** |
| 指导教师: | **伏晓　副教授** |

|  |
| --- |
| **南京大学研究生毕业论文** |
| **(申请工程硕士学位)** |

|  |
| --- |
| **南京大学软件学院** |
| **2019年04月** |

**The Design and Implementation of Information Extraction System Based on Tensorflow**

**An, Lei**

**Submitted in partial fulfillment of the requirements for the degree of Master of Engineering**

Supervised by

Associate Professor **Fu, Xiao**

Software Institute

**NANJING UNIVERSITY**

Nanjing, China

Apr, 2019

# 摘 要

随着人工智能技术的快速发展，越来越多的算法得到应用。自然语言处理作为人工智能领域中的一个重要分支一直是许多人关注的焦点，而知识图谱是自然语言处理中一个重要的研究方向。构建知识图谱的数据来源包括结构化文本和非结构化文本，作为构建知识图谱的重要数据来源之一的非结构化文本，如何从中提取高质量的信息成为一个研究热点。

面对上文所述的问题，一个能够提供高质量数据的信息抽取系统就显得十分重要。本文基于从非结构化文本中进行信息抽取的背景，以构建知识图谱时所需的三元组作为目标

**关键词**：信息抽取、命名实体识别、关系抽取、知识图谱、Tensorflow

# Abstract

With the rapid development of Internet technology, online games have become one of the main entertainment methods of people. The traditional network game server architecture adopts the client-server pattern. It means that the client initiates a network request directly to the server and the server responds accordingly. With the continuous increase in the number of game players, a single game server can not deal with this problem, and the architectural pattern, meaning the client and server directly connected, is difficult to support the game server cluster and limit the game's load ceiling. On the other hand, according to the traditional server architecture pattern, servers are independent of each other without communication. It is difficult to meet business requirements, such as cross-server battlefields and overall-server battlefields.

This thesis is based on the background of the growing number of gamers and aim to remove the difficulty of achieving cross-server battlefields. On the basis of the current research status of game server architectures, it addresses their shortcomings and considers the needs of game company server architectures, and proposes a gateway server architecture from the perspective of design and set-up. The main work of this thesis is as follows:

(1) In response to the problem of game server load caused by increasing game players, this thesis proposes to introduce a gateway server in the directly connected architecture pattern. The gateway server balances load for game server clusters, allowing them to on-line or off-line game servers dynamically based on actual game load. It supports more players to attend online competition at the same time, without a load ceiling.

(2) In response to the demand for cross-server battlefields, this thesis proposes to define communication messages as three types: unicast, multicast and broadcast. Unicast messages are used for client-to-server communication; multicast messages for cross-server battlefield communications; and broadcast messages for full-server battlefield communications. Gateway servers and game servers will respond differently for different types of messages, effectively solve cross-server battlefield issues, and support more complex business requirements.

(3) For the problem of message serialization, this thesis proposes a custom binary protocol, which implements the coding and decoding of the message protocol. It improves the transmission efficiency of the message and solves the problems of packetization and sticky packets.The binary protocol supports any language to implement the game client.

(4) For the problem of service management, this thesis proposes to introduce a service registration center in the server architecture. The service registration center based on ZooKeeper is able to register, discover and monitor game services. At the same time, the service registration center realizes the management of the service by dividing the nodes to support the game server to go online and offline dynamically. The developer does not need to manually maintain the server configuration table, which simplifies the development process.

At present, a game company in Shanghai has organized a team for developing game gateway server architecture. The use of this architecture has completed the partial reconstruction of the game server logic, laying the foundation for the evolution of the game server architecture.

**Keywords**：Game Server, Gateway Server, Binary Protocol, Service Registration Center

# 

**目 录**

[摘 要 I](#_Toc514163438)

[Abstract III](#_Toc514163439)

[图目录 VIII](#_Toc514163440)

[表目录 IX](#_Toc514163441)

[第一章 引言 1](#_Toc514163442)

[1.1 项目背景 1](#_Toc514163443)

[1.2 国内外研发概况 2](#_Toc514163444)

[1.3 本文主要研究的工作 4](#_Toc514163445)

[1.4 本文的组织结构 4](#_Toc514163446)

[第二章 技术综述 6](#_Toc514163447)

[2.1 Socket 6](#_Toc514163448)

[2.2 Netty 7](#_Toc514163449)

[2.3 ZooKeeper 9](#_Toc514163450)

[2.4 本章小结 11](#_Toc514163451)

[第三章 网关服务器架构的分析与设计 12](#_Toc514163452)

[3.1 项目总体规划 12](#_Toc514163453)

[3.2 系统需求分析 12](#_Toc514163454)

[3.2.1 通信模块需求分析 13](#_Toc514163455)

[3.2.2 网关服务器模块需求分析 14](#_Toc514163456)

[3.2.3 服务注册中心模块需求分析 15](#_Toc514163457)

[3.2.4 游戏服务器模块需求分析 16](#_Toc514163458)

[3.2.5 非功能性需求分析 17](#_Toc514163459)

[3.3 系统总体设计 18](#_Toc514163460)

[3.3.1 总体结构 18](#_Toc514163461)

[3.3.2 总体流程 19](#_Toc514163462)

[3.4 通信模块设计 20](#_Toc514163463)

[3.4.1 自定义通信协议 20](#_Toc514163464)

[3.4.2 心跳检测设计 22](#_Toc514163465)

[3.5 网关服务器模块设计 23](#_Toc514163466)

[3.5.1 服务管理设计 23](#_Toc514163467)

[3.5.2 消息路由设计 24](#_Toc514163468)

[3.5.3 单播与多播设计 24](#_Toc514163469)

[3.6 服务注册中心模块设计 25](#_Toc514163470)

[3.6.1 服务注册设计 25](#_Toc514163471)

[3.6.2 服务发现设计 25](#_Toc514163472)

[3.6.3 服务监听设计 26](#_Toc514163473)

[3.6.4 节点管理设计 26](#_Toc514163474)

[3.6.5 负载均衡设计 27](#_Toc514163475)

[3.7 游戏服务器模块设计 27](#_Toc514163476)

[3.7.1 单播与多播设计 28](#_Toc514163477)

[3.7.2 扩展接口设计 28](#_Toc514163478)

[3.8 本章小结 29](#_Toc514163479)

[第四章 网关服务器架构的实现 30](#_Toc514163480)

[4.1 通信模块的实现 30](#_Toc514163481)

[4.1.1 通信协议的实现 30](#_Toc514163482)

[4.1.2 心跳检测的实现 33](#_Toc514163483)

[4.2网关服务器模块的实现 34](#_Toc514163484)

[4.2.1 服务管理的实现 34](#_Toc514163485)

[4.2.2 消息路由的实现 35](#_Toc514163486)

[4.2.3 单播与多播的实现 37](#_Toc514163487)

[4.3 服务注册中心模块的实现 38](#_Toc514163488)

[4.3.1 服务注册的实现 38](#_Toc514163489)

[4.3.2 服务发现的实现 39](#_Toc514163490)

[4.3.3 服务监听的实现 40](#_Toc514163491)

[4.3.4 节点管理的实现 41](#_Toc514163492)

[4.3.5 负载均衡的实现 42](#_Toc514163493)

[4.4 游戏服务器模块的实现 43](#_Toc514163494)

[4.4.1 单播与多播的实现 43](#_Toc514163495)

[4.4.2 扩展接口的实现 44](#_Toc514163496)

[4.5 系统测试 45](#_Toc514163497)

[4.6 本章小结 48](#_Toc514163498)

[第五章 总结与展望 49](#_Toc514163499)

[5.1 总结 49](#_Toc514163500)

[5.2 进一步工作展望 50](#_Toc514163501)

[参 考 文 献 51](#_Toc514163502)

[致 谢 54](#_Toc514163503)

[版权及论文原创性说明 55](#_Toc514163504)

# 图目录

[图2.1 Socket模型 6](#_Toc514163394)

[图2.2单线程模型 8](#_Toc514163395)

[图2.3多线程模型 8](#_Toc514163396)

[图2.4 ZooKeeper系统角色 10](#_Toc514163397)

[图3.1 通信模块用例图 13](#_Toc514163398)

[图3.2 网关服务器用例图 14](#_Toc514163399)

[图3.3 服务注册中心用例图 15](#_Toc514163400)

[图3.4 游戏服务器用例图 17](#_Toc514163401)

[图3.5 系统总体结构 18](#_Toc514163402)

[图3.6 系统时序图 19](#_Toc514163403)

[图3.7 游戏协议 20](#_Toc514163404)

[图3.8 HTTP协议 21](#_Toc514163405)

[图3.9 二进制格式协议 22](#_Toc514163406)

[图3.10 节点管理 23](#_Toc514163407)

[图3.11 节点命名空间 26](#_Toc514163408)

[图4.1 自定义二进制协议 30](#_Toc514163409)

[图4.2 消息编码代码 31](#_Toc514163410)

[图4.3 消息解码代码 32](#_Toc514163411)

[图4.4 消息解包代码 33](#_Toc514163412)

[图4.5心跳检测代码 34](#_Toc514163413)

[图4.6管理游戏服务器代码 35](#_Toc514163414)

[图4.7随机路由算法代码 36](#_Toc514163415)

[图4.8指定路由算法代码 37](#_Toc514163416)

[图4.9单播与多播代码 38](#_Toc514163417)

[图4.10服务注册代码 39](#_Toc514163418)

[图4.11服务发现代码 40](#_Toc514163419)

[图4.12服务监听代码 41](#_Toc514163420)

[图4.13节点管理代码 41](#_Toc514163421)

[图4.14随机算法代码 42](#_Toc514163422)

[图4.15一致性哈希算法代码 42](#_Toc514163423)

[图4.16单播、组播与广播代码 43](#_Toc514163424)

[图4.17扩展接口代码 44](#_Toc514163425)

[图4.18测试部署图 45](#_Toc514163426)

[图4.19 ZooKeeper服务器1启动 46](#_Toc514163427)

[图4.20 ZooKeeper服务器2启动 46](#_Toc514163428)

[图4.21 ZooKeeper服务器3启动 46](#_Toc514163429)

[图4.22 客户端1发送单播消息 47](#_Toc514163430)

[图4.23 客户端1发送分组消息 47](#_Toc514163431)

[图4.24 客户端2发送分组消息 47](#_Toc514163432)

[图4.25 客户端3发送广播消息 47](#_Toc514163433)

# 表目录

[表 3.1通信模块用例表 13](#_Toc514163434)

[表 3.2网关服务器用例表 15](#_Toc514163435)

[表 3.3服务注册中心用例表 16](#_Toc514163436)

[表 3.4游戏服务器用例表 17](#_Toc514163437)

# 第一章 引言

## 1.1 项目背景

网络游戏诞生于二十世纪七十年代，最初是由计算机爱好者为了娱乐而开发出来的，这些游戏实现了简单的远程联机功能，但是无法模拟一个持续发展的游戏世界。从七十年代末到九十年代中期，部分专业的游戏开发商开始涉足网络游戏这个领域，网络游戏的类型和玩法也逐渐丰富了起来，在技术上也实现了网络游戏的可持续发展。到了九十年代后期，随着互联网热潮的兴起，游戏从最早期的单机游戏逐步发展成为大型多玩家在线游戏[许登元等, 2014]，越来越多的游戏开发商和游戏发行商进入了网络游戏产业，最终形成了一个分工明确、规模庞大网络游戏产业。从2005年起，网络游戏产业进入了一个新的发展时期，网络游戏玩家的数量也在持续增长[刘海超, 2015]，网络游戏成为了人们娱乐消遣的主要活动之一。到了2017年，中国游戏产业整体保持稳健的发展态势，移动游戏开始进入存量市场，政府政策有所回落，对整个游戏行业的发展仍有较大带动作用。

2017年中国游戏市场实际销售收入达到2036.1亿元，同比增长23.0%。其中，移动游戏市场实际销售收入1161.1亿元，份额继续增加，占57.0%；客户端游戏市场实际销售收入648.6亿元，份额减少，占31.9%；网页游戏市场实际销售收入156.0亿元，份额大幅减少，占7.6%；家庭游戏市场实际销售收入13.7亿元，份额有所增加，占0.7%。报告认为移动游戏市场增速放缓，市场实际销售收入向优质产品集中，两极化趋势明显。一方面，优质产品收入持续大幅增长；另一方面，大量中游甚至中上游的游戏市场实际销售收入锐减。部分游戏公司为避免与优质产品直接竞争，将研发与运营中心转向细分市场，2017年发布的新游戏已经明显具备细分市场的特点，加上用户对游戏的个性化需求等因素的共同作用，移动游戏行业开始注重细分化，差异化经营策略。

网络游戏的核心是游戏服务器，游戏服务器需要处理数据通信、业务逻辑、数据存储等问题，一个优秀的服务器架构不仅能够支撑游戏业务，为玩家提供高性能的游戏服务，而且具备良好的通用性和扩展性。通用性使得游戏服务器架构能够满足不同类型的业务需求，同时，在面对迭代速度非常快的业务需求时，只需要对服务器架构做少量修改。扩展性使得服务器架构在面临游戏业务扩展的时候，方便支持二次开发而不需要对现有架构进行重构，所以一个优秀的服务器架构不仅可以承载更多的玩家同时在线游戏，减少企业的硬件投入成本[江岸, 2014]**，**还可以缩短游戏开发周期，节省游戏研发成本。

## 1.2 国内外研发概况

在国内外游戏服务器架构的发展过程中，主要经历了三个时期：单服务器模式、多服务器模式和分布式服务器集群模式[武雪芳等, 2012]。

（1）单服务器模式

单服务器模式是将游戏的业务逻辑处理和数据存储都放在一台服务器中处理，单台服务器的负载上限决定了系统的性能瓶颈，随着游戏玩家数量的增加，单台游戏服务器并发连接数和请求连接数达到上限，可能会导致游戏崩溃。

（2）多服务器模式

多服务器模式是根据用户进行分流，在这种模式下，可以使用多个游戏服务器组成一个游戏服务器群组来支撑整个游戏业务，但是需要有一个负载均衡器[毛仁伟, 2014]来支持负载均衡，可以在客户端实现负载均衡，也可以单独实现一个负载均衡器来提供负载均衡服务。

（3）分布式服务器集群模式

分布式服务器集群模式比单服务器模式和多服务器模式在架构设计上有着本质的提升。单服务器模式和多服务器模式将游戏业务看作一个整体，游戏服务器需要承担业务逻辑处理和数据存储的工作，而分布式服务器集群模式将游戏业务按照业务逻辑进行了模块划分。分布式服务器集群模式的每个模块处理单独的业务逻辑，模块与模块之间相互独立，某个模块出现问题并不影响其他模块的业务处理，保证了系统的可靠性和稳定性。同时某个模块也可以由一个服务器集群组成，并为集群增加一个负载均衡器实现对该模块的负载均衡，集群中某些机器宕机的时候，该模块仍然可以对外提供服务。

目前很多mmorpg游戏公司仍然采用客户端与服务器直连的架构模式，这种直连的架构模式存在着以下问题：

（1）直连的架构模式中游戏服务器直接暴露在外，容易被第三方拦截抓包，对游戏服务器进行恶意攻击。

（2）直连架构模式中如果使用服务器组来对某一个游戏区服提供服务，需要额外实现一个负载均衡器，或者在客户实现负载均衡，但是此时客户端需要维护一份服务器地址配置表，一旦服务器组有服务器上线或下线，客户端都需要更新服务器配置表，无法实现服务器动态上线和动态下线。

（3）直连架构模式中服务器区服与区服之间是互不通信的，无法实现跨服战场、全服战场等业务需求。

游戏网关服务器架构针对以上问题，对直连架构模式进行了改进，在客户端与游戏服务器之间引入了网关服务器，作为客户端与游戏服务器之间的通信中介，同时还引入了服务注册中心，实现对服务的动态注册、发现和监听。改进后的服务器架构有以下优点：

（1）在游戏客户端和游戏服务器之间增加了网关服务器，网关服务器实现对请求的安全检查和过滤，可以防止游戏服务器被恶意攻击。

（2）单台游戏服务器的系统负载有限，使用游戏服务器组来对外提供服务，网关服务器可以实现一个负载均衡器，对游戏服务器组进行负载均衡，使得游戏没有负载上限。

（3）客户端与服务器直连的架构模式下，客户端需要维护一个服务器地址配置表，服务器上线、下线或者发生地址变更都需要更新该配置表，改进后的服务器架构引入了服务注册中心，网关服务器在服务注册中心设置对游戏服务器的监听后，游戏服务器上线、下线和服务器地址变更都会被推送到网关服务器，网关服务器进行相应的逻辑处理，而这些变化都屏蔽了客户端，客户端不再需要维护服务器地址配置表。

（4）客户端与服务器直连的架构模式下，游戏服务器之间是互不通信的，因此无法实现跨服战场和全服战场，改进后的架构将通信消息定义为单播、组播和广播三种消息类型，单播消息用来做客户端到服务器的通信，组播消息用来做跨服战场通信，广播消息用来做全服战场通信，网关服务器和游戏服务器根据消息类型做不同的处理，有效地解决了跨服战场的问题，也支持更复杂的业务需求。

## 1.3 本文主要研究的工作

本文所研究的游戏网关服务器架构是对传统游戏服务器架构的改进，在游戏客户端和游戏服务器之间增加一个网关服务器和一个服务注册中心。网关服务器作为游戏客户端与游戏服务器之间的通信中介，对客户端消息进行过滤，并将客户端消息路由转发至游戏服务器。服务注册中心对游戏服务进行服务注册、服务发现和服务监听，网关服务器对游戏服务器进行管理。

游戏网关服务器架构使用Netty网络库进行网络通信，为游戏客户端与网关服务器之间、网关服务器和游戏服务器之间的消息传输提供了基础。网关服务器架构通信采用自定义二进制协议，使得游戏客户端和游戏服务器支持任意语言进行二次开发。使用ZooKeeper服务来构建服务注册中心，保证了分布式服务的可靠性，服务注册中心提供服务注册、服务发现和服务监听等功能。游戏服务器上线、下线或者服务器地址变更的时候，服务注册中心将服务器地址信息推送至网关服务器，由网关服务器进行逻辑处理。同时网关服务器作为消息中转站，将内网与外网隔开，使外部无法直接访问内部服务器，从而保证了内部服务器的安全[杨玲, 2010]。

## 1.4 本文的组织结构

本文的组织结构如下：

第一章 引言。本章介绍了论文的研究背景，游戏服务器架构在国内外的研究进展，论文的主要研究工作和论文的组织结构。

第二章 技术综述。本章详细介绍了论文涉及的相关技术，包括Socket通信、Netty框架、Zookeeper服务。

第三章 网关服务器架构的分析与设计。本章对网关服务器架构进行了需求分析，给出了架构的总体设计，为架构划分了通信模块、网关服务器模块、服务注册中心模块和游戏服务器模块，并给出了每个模块的详细设计。

第四章 网关服务器架构的实现。本章在网关服务器架构设计的基础上，给出了通信模块、网关服务器模块、服务注册中心模块和游戏服务器模块的具体实现。

第五章 总结与展望。总结论文期间所做的工作，列举出了网关服务器架构的不足之处，并且对该架构的未来扩展做了进一步展望。

# 第二章 技术综述

本系统以Java作为开发语言，系统各个模块之间基于Socket通信，采用Netty作为通信框架，基于ZooKeeper框架实现了一个服务注册中心，可以提供服务注册、服务发现和服务监听等功能，保证了服务的可靠性和稳定性。

## 2.1 Socket

通过双向的通信连接，网络上的两个程序可以实现数据交换，通信连接的每一端都是一个socket[夏玲等, 2009]。每一端socket都需要一个应用端口号，socket通过这两个端口实现客户端到服务器之间的数据传递，具有数据传输时间短、传输效率高等特点[姚晓芳等, 2017]。



图2.1 Socket模型

如图2.1所示，socket套接字的启动连接有三个步骤[严谦等, 2016]：服务器端监听、客户端请求和服务器端连接确认。

1. 服务器端监听：服务器端套接字启动之后，绑定一个端口然后监听网络，等待客户端的连接请求，直到有客户端发起连接请求。
2. 客户端请求：客户端套接字启动之后，对网络上某一个服务器的某一个端口发出连接请求[何咏明, 2012]，该请求需要指定一个具体的服务器地址和端口号，然后等待服务器端套接字的响应。
3. 服务器端连接确认：服务器端套接字在未接收到客户端的连接请求之前一直处于监听网络状态，服务器端接收请求后创建连接[王智印等, 2017]，开启一个新的线程来维持与当前客户端套接字的连接，建立连接之后，服务器端套接字继续处于监听状态，等待下一个客户端套接字的连接。

## 2.2 Netty

Netty是一个由JBOSS提供的Java开源框架。Netty提供异步的、事件驱动的网络应用程序框架和工具[金志国等, 2014]，用来快速开发高可靠性、高性能的网络应用服务器和客户端程序。Netty是一个基于NIO的客户端/服务器框架[鲁兆硕, 2017]，支持快速、简单地开发网络应用，如协议服务器和客户端**[Hitchens, 2002]**，其中异步客户端参考了很多开源框架的设计原理[Gourley et al., 2012]。

Netty是目前业界最流行的NIO框架[李林锋, 2015]，得益于它的功能、性能、健壮性、可定制性和可扩展性。目前，Netty已经得到成千上万商业项目的实践应用，例如Hadoop[Shvachko et al., 2010]的Avro框架就使用Netty作为底层通信框架，业界很多其他主流RPC框架[张艳军等, 2016] 也使用Netty来构建具有高性能的通信能力。

Netty因其内部Reactor模型[李攀登, 2012]而展现出高性能。Netty的Reactor模型主要由多路复用器（Acceptor）、事件分发器（Dispatcher）和事件处理器（Handler）组成，主要分为单线程模型、多线程模型和主从线程模型。

单线程模型：一个线程完成所有的I/O操作，即I/O多路复用、I/O事件分发和I/O事件处理都是由一个Reactor线程完成的。

图2.2描述了Netty的Reactor单线程模型，多个客户端同时连接Reactor，Acceptor实现客户端的IO多路复用[罗振兴等, 2007]，Dispatcher分发器实现I/O事件分发，Handler实现I/O读写事件处理。

图2.2单线程模型

单线程模型适用于低负载、低并发的应用场景，但是并不适合高负载、高并发的场景，主要原因如下：

* 单个Reactor线程无法同时处理成千上万的客户端连接，单线程即使CPU负荷达到100%，也无法满足数以万计消息的编码、解码、读取和发送。
* 单个Reactor线程同时处理多路复用、事件分发和事件处理，一旦该线程进行进入死循环或者意外终止，就会导致整个系统通信陷入瘫痪，造成单点故障。

多线程模型：为了解决单线程模型的性能和单点故障问题，引入了Reactor多线程模型。

图2.3多线程模型

图2.3描述了Netty的Reactor多线程模型，多个客户端同时连接Reactor，Acceptor线程实现客户端的多路复用功能，Dispatcher分发器将I/O事件分发到专门处理I/O事件的线程池中，该线程池完成事件分发和事件处理。多线程模型具有如下特点：

* NIO[吴高阳, 2014]线程池来负责I/O事件分发和事件处理，负责对所有消息的编码、解码、读取和发送。
* NIO线程池中的一个NIO线程可以同时处理多条消息链路，但是一条链路只能对应一个NIO线程，这样可以避免并发问题。

## 2.3 ZooKeeper

ZooKeeper**[Hunt, 2010]**是一个开放源码的分布式应用程序协调服务，是Google的Chubby一个开源实现，是Hadoop和Hbase的重要组件。ZooKeeper由若干个服务器节点组成，用户和ZooKeeper服务器之间的交互是通过ZooKeeper客户端和ZooKeeper服务器之间的会话进行的[谭玉靖, 2014]。Zookeeper实现了分布式环境中的数据管理，保证了分布式环境中数据的强一致性。ZooKeeper服务被广泛应用于分布式系统的开发，ZooKeeper的数据保存在内存中，因此数据的读写速度非常快，其中数据读操作比数据写操作更快。

ZooKeeper由两部分组成，ZooKeeper服务器和ZooKeeper客户端。如果ZooKeeper服务器采用集群模式，则不会出现单点失效的问题[Skeirik et al., 2013]。ZooKeeper集群服务在集群节点之间进行数据信息和状态的同步，当集群中任意一台服务器发生故障时，其他服务器并不受到影响，整个集群对外仍然提供正常服务。ZooKeeper客户端可以连接到服务器，创建数据节点、删除数据节点、修改节点信息等，创建后的数据节点会在数据更新后得到通知**[Ding et al., 2012]**。

ZooKeeper服务的集群工作模式[Shi et al., 2014]启动后，会从众多的服务器之间选择一个作为Leader服务器，其余作为Follower服务器。Leader服务器主要负责处理写请求和更新系统的状态。Follower主要用于接收客户端的读写请求，并将结果返回客户端。如果集群服务器的某一台机器宕机，客户端会选择另一台服务器去连接，从而保证连续和稳定的服务[Guo, 2013]。

如图2.4所示，ZooKeeper主要有以下三类角色，分别是领导者（Leader）[吴宪国等, 2015]、学习者（Learner）和客户端（Client）。领导者主要负责发起投票和对投票进行决议，并且更新系统的状态。学习者包含两种角色，其中跟随者（Follower）负责接收客户端请求并向客户端返回相应的处理结果，在选举的过程中主要负责参与投票，观察者（Observer）负责接收客户端的连接，并将写请求转发给领导者，但是观察者不参与投票，只同步领导者的状态。客户端主要负责发起请求。



图2.4 ZooKeeper系统角色

ZooKeeper的视图结构和标准Unix文件系统非常相似，ZooKeeper使用了自己特有的节点概念，称为Znode。Znode是ZooKeeper中最小的数据单元，每个Znode都可以保存数据，同时也可以挂载子节点，形成了一个树形的命名空间。在ZooKeeper中，节点类型分为持久节点、临时节点和时序节点，实际使用过程中，可以组合生成4种节点类型：

（1）持久节点：持久节点创建后就一直存在，直到客户端主动删除这个节点，该节点才会消失。

（2）持久顺序节点：持久顺序节点的基本特性和持久节点一致，额外的特性是，在ZooKeeper中，每个父节点为其一级子节点维护一份时序，记录着子节点创建的先后顺序。在创建子节点的时候，ZooKeeper会自动为节点名后加一个数字后缀作为新的节点名。

（3）临时节点：临时节点的生命周期与客户端的会话绑定，客户端会话一旦失效，节点就会被自动清除掉。

（4）临时顺序节点：临时顺序节点的基本特性和临时节点一致，与持久顺序节点相同的是，创建子节点的时候，ZooKeeper会自动为节点名后加一个数字后缀作为新的节点名。

## 2.4 本章小结

本章主要介绍了网关服务器架构开发过程中所使用的技术，并且简要地介绍了相关技术内容、技术背景、技术优势等，使用到的技术包括Socket编程、Netty框架和ZooKeeper服务等。

# 

# 第三章 网关服务器架构的分析与设计

本章详细描述了游戏网关服务器架构的需求分析，按照需求分析、总体设计、详细设计的顺序，对网关服务器架构进行了功能模块划分，并给出了每一个子模块的详细设计。

## 3.1 项目总体规划

游戏网关服务器架构主要包括通信模块、网关服务器模块、服务注册中心模块和游戏服务器模块。通信模块负责客户端与网关服务器之间的通信，网关服务器与游戏服务器之间的通信，通信模块需要实现模块之间的通信编解码和网络连接的心跳检测。网关服务器作为客户端与游戏服务器的通信中介，同时保持与客户端和游戏服务器的网络连接，对客户端和游戏服务器的消息进行转发，网关服务器模块需要实现游戏服务管理、消息路由、消息单播与多播等功能。服务注册中心提供网关服务和游戏服务的注册监听功能，实现对服务节点变化的监听，服务注册中心模块主要需要服务注册、服务发现、服务监听、节点管理和负载均衡等功能。游戏服务器与网关服务器保持网络连接，客户端的所有消息都由网关服务器转发至游戏服务器，游戏服务器处理后的消息发送至网关服务器，再由网关服务器转发至相应的客户端，游戏服务器模块需要实现消息单播与多播、扩展开发接口等功能。

## 3.2 系统需求分析

游戏网关服务器架构需求来自于对某游戏公司服务器架构的改进，当前的游戏服务器架构是游戏客户端与游戏服务器直接通信，改进后的架构是在游戏客户端和服务器之间增加一个网关服务器，网关服务器对客户端消息进行转发和对游戏服务器进行管理，还增加了一个服务注册中心提供服务注册、服务发现和服务监听等功能。将整个服务器架构划分为通信模块、网关服务器模块、服务注册中心模块和游戏服务器模块。

### 3.2.1 通信模块需求分析

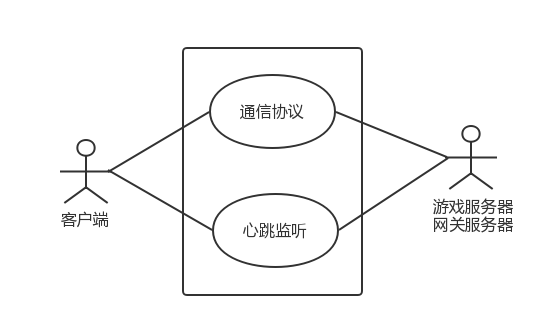


图3.1 通信模块用例图

如图3.1所示，通信模块是系统的核心模块，为客户端、网关服务器和游戏服务器提供通信协议、心跳监听等功能。

在网关服务器架构设计中，客户端与网关服务器之间、网关服务器与游戏服务器之间都需要进行socket通信，socket通道传输的是二进制字节流，因此需要定义一个通信协议来进行消息的传输，协议规定可读消息与二进制字节流的转化关系，因此通信模块需要实现对通信协议的编码和解码。协议编码是将可读消息转化为二进制字节流在socket通道中传输，供消息发送方使用，协议解码是将socket通道中读取到的二进制字节流转化为可读消息，供消息接收方使用。协议编码和解码提供给服务器架构的客户端模块、网关服务器模块和游戏服务器模块使用，客户端如果使用java实现，可以复用通信协议编解码，如果使用其他语言实现，需要根据协议重新实现协议编码和解码方法。

通信模块还需要实现的一个重要功能是心跳监听，客户端与网关服务器、网关服务器与游戏服务器之间的网络连接如果一段时间没有消息通信，网络连接可能会自动断开。服务器监听到网络通道一段时间没有消息传输后，向网络通道发送ping心跳包，接收到ping心跳包时向网络通道发送pong心跳响应包，可以保证网络通道保持连接不断开。

表 3.1通信模块用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | 名称 | 描述 | 优先级 |
| R1 | 通信协议 | 客户端与网关服务器、网关服务器与游戏服务器之间通信消息的编码和解码 | 高 |
| R2 | 心跳监听 | 网络连接的心跳监听 | 中 |

### 3.2.2 网关服务器模块需求分析

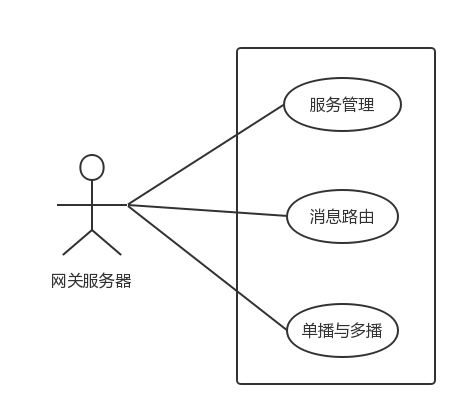


图3.2 网关服务器用例图

如图3.2所示，网关服务器模块是整个网关服务器架构的核心模块，网关服务器作为客户端与游戏服务器的通信中介，主要提供服务管理、消息路由、消息单播与多播等功能。

游戏服务器启动之后需要向服务注册中心注册服务器信息，网关服务器启动后向服务注册中心请求游戏服务器节点列表信息，然后根据节点信息向游戏服务器发起连接，如果连接成功，网关服务器将该服务器通道保存在内存集合中，对游戏服务器进行管理。

网关服务器作为客户端与游戏服务器的通信中介，需要对客户端消息进行消息路由，通过路由算法将客户端消息转发至相应的游戏服务器。这里的消息路由既可以是对服务器集群进行负载均衡路由，也可以根据客户端携带的游戏服务器信息路由到指定的游戏服务器，网关服务器需要实现以上两种路由算法，根据实际业务需求进行调用。

网关服务器需要实现的不仅仅是转发客户端消息，还需要作为游戏服务器的消息中转器。不同的游戏服务器消息在网关服务器架构中可以通过网关服务器转发消息，实现玩家跨服通信。将服务器消息分为单播、组播与广播几种类型，网关服务器需要根据消息类型做相应的处理，将服务器消息转发至对应的客户端。网关服务器从单播消息中解析出客户端Id，发送给对应的客户端，从组播消息中解析出分组Id，发送给分组中的所有客户端，广播消息不需要解析，直接发送给网关服务器连接的所有客户端。

表 3.2网关服务器用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | 名称 | 描述 | 优先级 |
| R1 | 服务管理 | 从服务注册中心获取游戏服务器节点列表，并向游戏服务器发起连接 | 高 |
| R2 | 消息路由 | 根据路由算法将客户端消息转发至相应的游戏服务器 | 高 |
| R3 | 单播与多播 | 网关服务器根据消息类型将服务器消息转发至对应的客户端，实现消息单播、组播与广播 | 中 |

### 3.2.3 服务注册中心模块需求分析

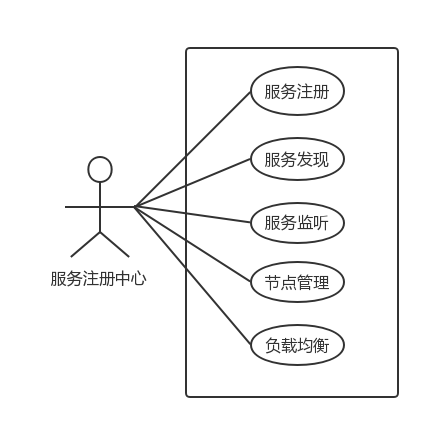


图3.3 服务注册中心用例图

如图3.3所示，服务注册中心模块主要为网关服务器与游戏服务器提供服务注册、服务发现、服务监听、节点管理、负载均衡等功能。

服务注册中心需要对外提供服务注册功能。网关服务器在成功启动之后，需要将网关服务器地址信息注册到服务注册中心，客户端可以通过服务发现获取网关服务器节点信息列表，然后向网关服务器发起连接。游戏服务器在成功启动之后，需要将游戏服务器地址信息注册到服务注册中心，网关服务器可以通过服务发现获取游戏服务器节点信息列表，然后向游戏服务器发起连接。为了区分网关服务和游戏服务，需要将网关服务和游戏服务划分到不同的服务节点下。

服务注册中心需要对外提供服务发现功能。网关服务和游戏服务被注册到服务注册中心之后，客户端可以通过服务发现获取网关服务器节点信息列表，网关服务器可以通过服务发现获取游戏服务器节点信息列表，然后向游戏服务器发起连接。

服务注册中心需要对外提供服务监听功能。网关服务器在服务注册中心设置对游戏服务器的监听，游戏服务器上线、服务器下线和服务器地址变更时，服务注册中心会将服务器变更信息推送给网关服务器，网关服务器做相应的逻辑处理。

服务注册中心需要提供节点管理功能。不仅仅游戏服务器和网关服务器信息会被注册到服务注册中心，其他服务也可能向服务注册中心注册服务，所以需要对服务注册中心进行节点管理。为每一种服务单独划分节点，这样不管是服务注册、服务发现还是服务监听，都会根据注册服务类型对该服务范围内的节点进行相应的操作，而不会影响到其他服务节点。

服务注册中心需要对外提供负载均衡功能。网关服务与游戏服务需要划分在不同的节点下，方便对服务进行管理。为保证服务注册中心服务稳定性，需要使用集群部署。其他模块在调用服务注册中心服务的时候，需要调用负载均衡算法从服务注册中心集群中选择一个服务器来服务，负载均衡有效地保证了服务注册中心的可用性和稳定性。

表 3.3服务注册中心用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | 名称 | 描述 | 优先级 |
| R1 | 服务注册 | 网关服务器和游戏服务器向服务注册中心注册服务 | 高 |
| R2 | 服务发现 | 客户端和网关服务器从服务注册中心发现网关服务和游戏服务节点信息列表 | 高 |
| R3 | 服务监听 | 服务注册中心将节点信息变化推送至客户端和网关服务器 | 中 |
| R4 | 节点管理 | 服务注册中心为各种服务划分独立节点空间 | 中 |
| R5 | 负载均衡 | 服务注册中心集群使用负载均衡算法来保证集群对外可用 | 高 |

### 3.2.4 游戏服务器模块需求分析

如图3.4所示，游戏服务器模块作为系统的核心模块，主要为游戏服务器提供消息单播与多播、扩展接口等功能。

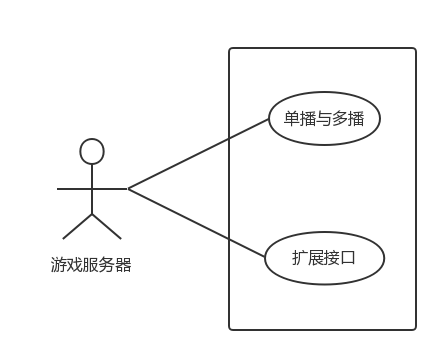


图3.4 游戏服务器用例图

网关服务器架构中，可以方便地实现消息单播与多播。将消息分为单播消息、组播消息、广播消息几种类型，游戏服务器根据服务器消息类型将消息发送至哪些网关服务器，单播消息只需要发送至消息来时的网关服务器即可，组播消息因为无法确定分组中的客户端与哪个网关服务器连接，所以和广播消息一样，都需要将服务器消息转发至所有的网关服务器。

游戏服务器模块作为一个需要二次开发的模块，需要抽象出扩展接口供开发者二次开发使用。该扩展接口需要封装服务器的底层通信实现，只对开发者提供消息响应接口，并提供系统底层功能的封装api供开发者使用。

表 3.4游戏服务器用例表

|  |  |  |  |
| --- | --- | --- | --- |
| 用例ID | 名称 | 描述 | 优先级 |
| R1 | 单播与多播 | 游戏服务器根据消息类型实现对消息的单播、组播和广播 | 高 |
| R2 | 扩展接口 | 游戏服务器抽象扩展开发接口供开发者二次开发 | 中 |

### 3.2.5 非功能性需求分析

本节主要从可靠性、可扩展性、易用性、安全性等方面描述网关服务器架构的非功能性需求。

可靠性。网关服务器架构具有较高的可靠性，服务注册中心采用集群部署，注册中心服务器半数以上存活就可以保证服务的可用性，具有较好的容灾，可用性达到99.9%。

可扩展性。网关服务器架构提供通用模块和众多底层api的封装，游戏服务器模块设计了扩展接口，系统架构扩展开发非常方便。

易用性。网关服务器架构提供简洁实用的api接口，开发者可以方便地进行方法调用。

安全性。网关服务器架构模块之间通信采用自定义二进制协议，可以使用加密算法和加密key对消息进行加密，保证消息传输的安全性。

## 3.3 系统总体设计

### 3.3.1 总体结构



图3.5 系统总体结构

如图3.5所示，游戏网关服务器架构主要由通信模块、网关服务器模块、服务注册中心模块和游戏服务器模块组成。通信模块由通信协议子模块和心跳检测子模块组成。网关服务器模块由服务管理子模块、消息路由子模块和单播与多播子模块组成。服务注册中心由服务注册子模块、服务发现子模块、服务监听子模块、节点管理子模块和负载均衡子模块组成。游戏服务器模块由单播与多播子模块和扩展接口子模块组成。

### 3.3.2 总体流程

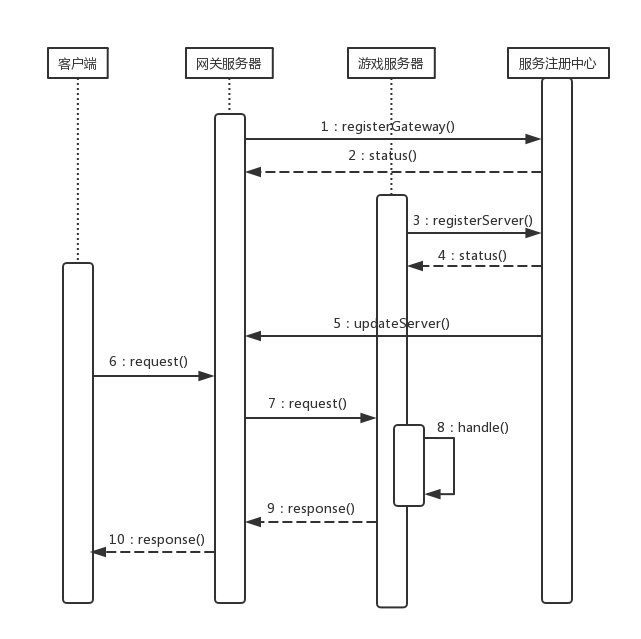


图3.6 系统时序图

网关服务器架构所有模块的启动顺序如图3.6所示，步骤如下：

（1）服务注册中心启动，开始对外提供服务注册、服务发现和服务监听。网关服务器启动，启动后将网关服务器地址信息注册到服务注册中心。

（2）服务注册中心向网关服务器返回网关注册状态。

（3）游戏服务器启动，将游戏服务器地址信息注册到服务注册中心。

（4）服务注册中心向游戏服务器返回注册状态。

（5）服务注册中心向网关服务器主动推送游戏服务器节点信息，网关服务器根据节点信息向游戏服务器发起连接。

（6）客户端启动，向网关服务器发起连接，并向网关服务器发送消息。

（7）网关服务器接收到客户端消息后，根据路由算法将客户端消息转发至对应的游戏服务器。

（8）游戏服务器接收到消息后，解析出消息内容并对其进行业务逻辑处理得到结果消息。

（9）游戏服务器根据消息类型将消息转发至网关服务器。

（10）网关服务器根据消息类型将消息转发至对应的客户端。

## 3.4 通信模块设计

通信模块是网关服务器架构的核心模块，为客户端、网关服务器和游戏服务器提供通信协议、心跳监听等功能。本架构没有采用现有的序列化和反序列化框架，而是采用了自定义二进制协议，自定义二进制协议不仅传输效率高，还支持任意语言来实现游戏客户端，同时还可以有效解决tcp分包与黏包的问题。网关服务器架构使用Netty网络库提供的心跳监听类监听网络状态，监听到网络一段时间无消息通信时发送心跳包，可以防止网络连接被断开。

### 3.4.1 自定义通信协议

网络协议定义了通信双方传递信息的语法、语义和时序[符军, 2009]。游戏通信协议比较特殊，要求通信流量少、安全性高和支持多平台等，所以游戏协议的设计需要具备简单、通用和代码层上易扩展等特点。

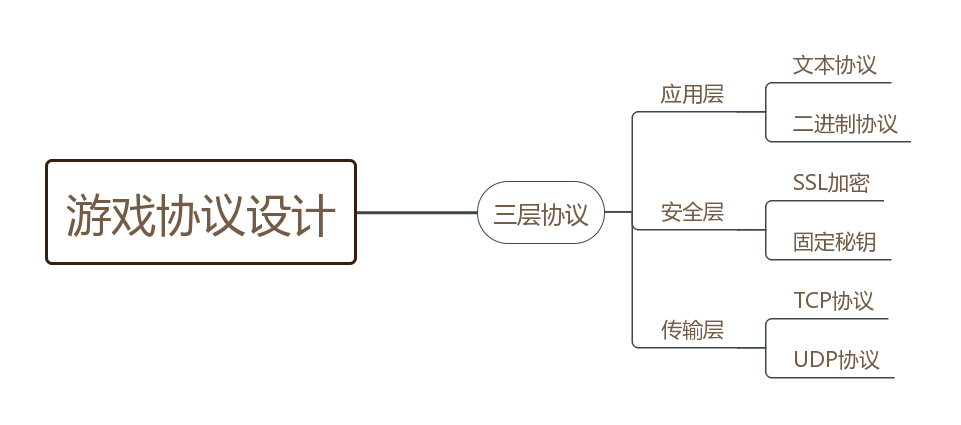


图3.7 游戏协议

如图3.7所示，游戏协议设计包含三个层次，分别为应用层、安全层和传输层。应用层主要是常用解析方式的定义，选型需要符合系统的实际需求，常用的应用层协议主要有文本协议和二进制协议。

文本协议设计的目的是方便理解，如常见的HTTP协议[晓涵, 2017]，一个典型的HTTP协议如图3.8所示：

|  |
| --- |
| GET/sample.Jsp HTTP/1.1  Accept:image/gif.image/jpeg,\*/\*  Accept-Language:zh-ch  Connection:Keep-Alive  Host:localhost  User-Agent:Mozila/4.0(compatible;MSIE5.01;Window NT5.0)  Accept-Encoding:gzip,deflate  Username=test&password=1234 |

图3.8 HTTP协议

二进制协议[杨建禹, 2013]是一串字节流，通常由消息头header和消息体body两部分组成。其中消息头header的长度是固定的，里面存储着固定字段属性，消息体body长度不固定，其中包含着消息内容的主体。通常消息头中会有一个字段表示消息体的长度，可以基于该字段从二进制流中解析得到消息体长度，进而解析出消息内容。

文本协议具有良好的可读性和扩展性，但是文本协议的解析效率比较低。二进制协议的可读性差，但是解析效率高。网关服务器架构实现的自定义协议不仅需要实现消息传输，还需要定义消息命令，所以在应用层选用二进制协议。二进制协议不仅支持自定义消息类型，而且可以有效解决流传输的黏包与分包问题。

安全层主要有SSL加密和固定秘钥加密两种方式。SSL证书管理比较复杂，代价比较高，不适合游戏通信协议。固定秘钥加密指客户端与服务器端约定好加密与解密算法，消息发送方在发送消息前用秘钥和加密算法对消息进行加密，消息接收方在接收消息后用秘钥和解密算法对消息进行解密。因此，网关服务器架构在安全层采用固定秘钥加密。

传输层主要有tcp协议和udp协议两种传输方式。tcp协议提供可靠的消息传输，udp传输效率高但是可靠性低，传输层的协议选择需要视业务而定。网关服务器架构要保证传输消息的可靠性，所以在传输层选用tcp协议。

综上所述，网关服务器架构在传输层采用tcp协议，安全层采用固定秘钥加密，应用层采用二进制协议，协议包括消息头和消息体，具体的消息格式如下：

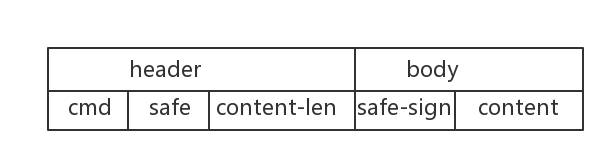


图3.9 二进制格式协议

如图3.9所示，header部分包含cmd、safe和content-len，而body包含safe-sign和content。

cmd表示命令字段，表示该消息的命令类型。其中0x01表示向所有客户端发送消息；0x02表示向某一个客户端发送消息；0x03表示向某一个用户发送消息，包含该用户的所有客户端设备；0x04表示向某一个分组发送消息，该分组的所有客户端都会收到该消息；0x05表示获取所有在线客户端数量；0x06表示获取某个用户的所有在线客户端设备数量；0x07表示获取某个分组的所有在线客户端数量；0x08表示客户端绑定某用户；0x09表示客户端解绑某用户；0x0A表示客户端加入某分组；0x0B表示客户端离开某分组；0x65表示该消息的消息源是客户端；0x66表示该消息的消息源是游戏服务器。

safe表示安全字段，0x00表示该协议不加密，body消息体中就不含验签字符串，0x01表示该协议加密，在body消息体中就含有16位的验签字符串，对消息体中的内容进行加密，接收消息的时候用解密算法对消息进行解密。

content-len字段表示body消息体的二进制流长度，发送方需要计算对应body消息体的长度并设置到content-len字段，接收方根据该字段将从二进制流解析出body消息。

safe-sign字段表示加密验签字段，当safe为0x00时，该字段为空字符串；当safe为0x01时，该字段为长度为16的验签字符串。

content字段表示协议的消息内容，如果safe字段是0x00时，该字段为原始消息，如果safe字段是0x01时，该字段为加密后的消息，需要对消息进行解密。

### 3.4.2 心跳检测设计

客户端与网关服务器、网关服务器与游戏服务器之间的网络连接如果一段时间没有消息通信，网络连接可能就会断开，心跳监听会在网络通道一段时间没有消息传输时，做相应的处理来避免网络通道被关闭。

本系统心跳监听机制使用Netty库的IdleStateHandler类来实现。IdleStateHandler可以实现对网络通道进行三种心跳监听检测，分别是指定空闲时间网络未发生读事件、指定空闲时间网络未发生写事件和指定空闲时间网络未发生读写事件。系统监听到这三种心跳事件，会触发userEventTriggered方法，在该方法中，可以根据心跳事件类型来做相应的处理，发出ping心跳包或者pong响应心跳包。

## 3.5 网关服务器模块设计

本节主要为网关服务器模块划分了服务管理、消息路由和单播与多播子模块，并且描述了每个子模块的详细设计思路。

### 3.5.1 服务管理设计

在网关服务器架构中，通常会存在多个网关服务器。一个网关的I/O通信负载有限，当游戏服务器数量增加的时候，需要引入多个网关服务器，每个网关服务器管理多个游戏服务器，这样可以解决网关服务器的负载压力问题。在这种情况下，需要对服务器节点进行管理，每个网关服务器管理一定数量的游戏服务器节点。

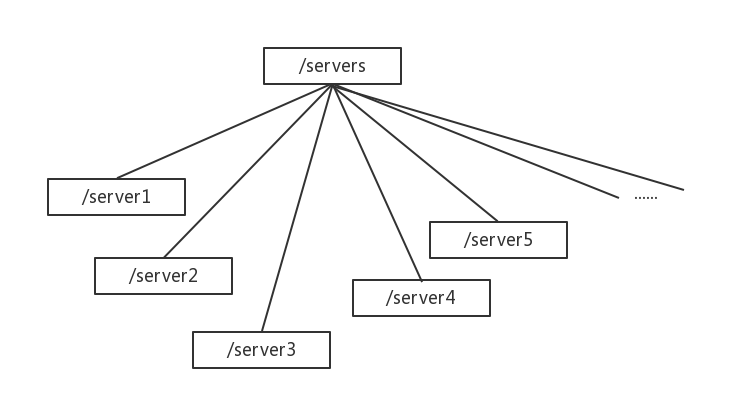


图3.10 节点管理

如图3.10所示，游戏服务器节点/servers下面有众多服务器子节点，一个网关服务器无法管理全部游戏服务器，需要引入多个网关服务器。例如每个网关服务器管理4个游戏服务器，网关服务器与游戏服务器可以建立这样一种映射关系，将网关服务器和游戏服务器按照顺序编号，每四个游戏服务器一组划分到一个网关服务器下。当游戏服务器增加的时候，只要按上述规则分配网关服务器，如果网关服务器不够，增加网关服务器即可解决。

### 3.5.2 消息路由设计

路由算法是网络传输的核心，主要负责将数据报从指定源节点送达指定目的节点。路由算法主要包括两个功能，第一是为源节点和目的节点之间选择一条正确的传输路径，第二是正确地指引数据报消息从数据源节点通过网络到达数据目的节点。

网关服务器作为客户端与游戏服务器的通信中介，需要对客户端的消息进行消息路由，通过路由算法将客户端消息转发至指定的游戏服务器，这里的消息路由既可以是对服务器集群进行负载均衡路由，也可以根据客户端携带的服务器信息路由到指定的游戏服务器，具体由路由算法来实现。

网关服务器需要考虑以下两种场景，第一种场景是单台游戏服务器不足以支撑游戏业务，需要使用服务器集群对外提供服务，网关服务器需要实现路由算法来对服务器集群进行负载均衡。第二种场景是不同的游戏服务器对外提供不同的游戏服务，客户端发送消息时，网关服务器根据客户端携带的服务器消息找到对应的游戏服务器，并将客户端消息转发过去。

### 3.5.3 单播与多播设计

网关服务器作为客户端与游戏服务器之间的通信中介，同时保持与多个客户端和游戏服务器之间的连接，可以方便地实现服务器与服务器之间的通信。将通信消息分为单播消息、组播消息和广播消息三种类型，单播消息指客户端与服务器一对一通信，网关服务器收到游戏服务器的单播消息后，从消息中解析出客户端Id，然后根据该客户端Id找到对应客户端通道，将消息发送至该客户端通道中；组播消息指客户端与游戏服务器多对一通信，网关服务器收到游戏服务器的组播消息后，先从该消息的cmd字段中解析出命令，如果是加入分组命令，将消息中的客户端Id绑定到分组Id集合中，如果是离开分组命令，将消息中的客户端Id从分组Id中解绑，如果是向分组发送消息，网关服务器从保存的对应关系中找到该分组Id对应的所有客户端通道，然后将消息发送到这些通道中。广播消息指客户端与游戏服务器多对一通信，网关服务器在收到游戏服务器的广播消息时，将消息转发至所有与该网关服务器保持连接的客户端通道中。

## 3.6 服务注册中心模块设计

本节主要介绍服务注册中心模块的详细设计，将服务注册中心划分成了服务注册、服务发现、服务监听、节点管理和负载均衡子模块，并给出了每个子模块的详细设计。

### 3.6.1 服务注册设计

服务注册中心需要提供服务注册功能，网关服务器可以向服务注册中心注册网关服务器的服务器地址信息，游戏服务器可以向服务器注册中心注册游戏服务器的服务器地址信息，甚至其他应用也可以向服务注册中心注册应用服务。

在进行服务注册时，首先确定当前注册节点的命名空间，然后确定当前节点的应用节点路径，服务注册中心会判断当前注册的节点路径是否已经被注册过，如果被注册过则服务注册失败，如果没有被注册过则注册成功。

### 3.6.2 服务发现设计

服务注册中心需要提供服务发现功能，客户端可以向服务注册中心获取网关服务器节点信息列表，然后根据节点中网关服务器地址信息向网关服务器发起连接。网关服务器可以向服务注册中心获取游戏服务器节点信息列表，然后根据节点中游戏服务器地址信息向游戏服务器发起连接。

在进行服务发现的时候，首先确定当前注册节点的命名空间，然后确定需要获取的应用节点路径，如果当前获取的节点是叶子节点，服务注册中心返回当前节点的节点信息，如果当前获取的节点非叶子节点，服务注册中心返回该节点的所有子节点的节点信息列表。

### 3.6.3 服务监听设计

服务注册中心需要对外提供服务监听功能，网关服务器和游戏服务器被注册到服务注册中心之后，客户端可以通过服务发现获取网关服务器节点信息列表，网关服务器可以通过服务发现获取游戏服务器节点信息列表。客户端和网关服务器可以为指定节点设置节点监听，设置监听之后，当该节点被删除、节点信息变化、新增子节点、删除子节点的时候，都会将发生变化的节点信息主动推送到客户端与网关服务器，监听方设置回调事件句柄，对不同的节点变化进行不同的处理，实现对服务节点的监听处理。

### 3.6.4 节点管理设计

在网关服务器架构中，游戏服务器作为注册到服务注册中心的服务，可以由网关服务器来设置监听，服务注册中心主动推送服务器节点信息到网关服务器中。如图3.11所示，首先设计顶级命名空间，第一个节点为gproxy，表示该节点下面的所有子节点信息都是网关服务器架构所注册的节点信息。如果服务注册中心还需要服务于其他应用，再设计一个应用空间，将该应用需要注册的服务都注册到该命名空间节点下，再在该节点下面设计节点划分方案。

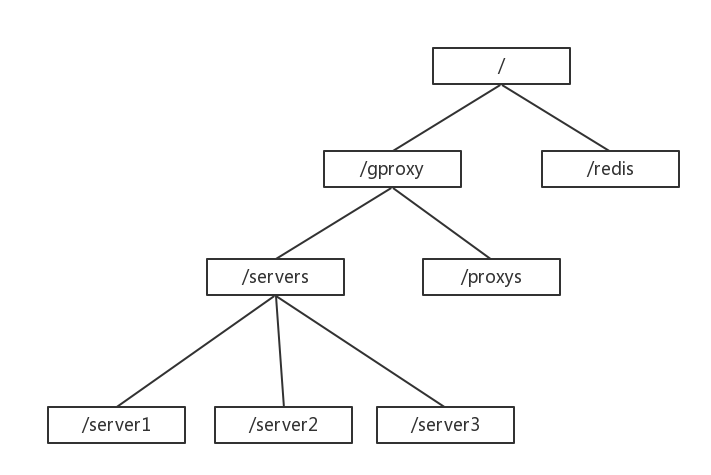


图3.11 节点命名空间

在gproxy节点下面，需要设计两个子节点，第一个子节点是proxy，该节点本身并不存储有实际含义的信息，但是该节点下面的子节点存储着所有网关服务器的信息，例如网关1在该节点下面存储的绝对路径是/gproxy/proxy/proxy1，内容是网关服务器1的服务器地址，网关2在该节点下面的存储的绝对路径是/gproxy/proxy/proxy2，内容是网关服务器2的服务器地址。第二个子节点是server，该节点下面的子节点存储着所有游戏服务器的信息，例如游戏服务器1在该节点下面存储的绝对路径是/gproxy/server/server1，内容是游戏服务器1的服务器地址，游戏服务器2在该节点下面存储的绝对路径是/gproxy/server/server2，内容是游戏服务器2的服务器地址。

### 3.6.5 负载均衡设计

ZooKeeper可以使用单机提供服务，也可以使用ZooKeeper服务器集群提供分布式服务，服务器集群保证了ZooKeeper服务的可靠性。使用ZooKeeper服务器集群对外提供服务，客户端请求ZooKeeper服务时需要使用负载均衡算法。常见的负载均衡算法如下：

1. 轮询法：将所有的客户端请求按照顺序轮流分配到ZooKeeper服务器上，不考虑服务器的当前连接数和系统负载，每台服务器拥有相同的权重，客户端平等地看待每台服务器。
2. 随机法：某个客户端调用ZooKeeper服务时，调用系统的随机算法生成一个序号索引，从ZooKeeper服务器集群列表中选择该序号对应的服务器，使用该服务器来为当前客户端提供ZooKeeper服务。
3. 源地址哈希法：客户端在调用ZooKeeper服务时，将客户端IP地址通过哈希算法计算得到一个序号索引，从ZooKeeper服务器集群列表中选择该序号对应的服务器，使用该服务器来为当前客户端提供ZooKeeper服务。如果客户端IP地址和服务器集群列表都没有发生变化，源地址哈希法会将该客户端的每次请求都映射到相同的ZooKeeper服务器。

## 3.7 游戏服务器模块设计

本节主要介绍游戏服务器模块的详细设计，将游戏服务器模块划分成了单播与多播和扩展接口设计子模块，并给出了每个子模块的详细设计。

### 3.7.1 单播与多播设计

单播表示一个游戏客户端与游戏服务器一对一通信，客户端发送的消息，经过网关服务器转发至指定的游戏服务器之后，游戏服务器进行逻辑处理，然后将处理后的结果顺着消息转发来的路径转发至源客户端。为了实现单播，需要为消息cmd中定义一种类型作为单播消息，游戏服务器需要保存客户端id与网关服务器id之间的一对一关系，以便从对应关系中找到客户端的网络通道。

组播表示一组游戏客户端与游戏服务器实现多对一通信，客户端发送的消息，经过网关服务器转发至指定的游戏服务器之后，游戏服务器进行相应的逻辑处理，如果是组播消息，会从消息中解析出对应的分组id，然后根据分组id找到该分组下的所有网关通道，向这些网关通道中发送服务器消息。

在组播服务中，分组是一个重要概念，每个分组拥有一个唯一的分组id。在通信消息协议的cmd字段，需要定义几种消息类型来实现组播功能，分别是加入分组、离开分组和消息组播。加入分组表示某一个客户端加入某个分组，离开分组表示客户端离开某个分组，消息组播表示向某个分组中所有的客户端发送消息。此外，游戏服务器需要保存分组与网关服务器之间的一对多关系，当消息是组播类型的时候，游戏服务器从对应关系中找到分组下所有的网关通道，将消息发送至这些网关通道中。

广播表示所有游戏客户端与游戏服务器多对一通信，客户端发送的消息，经过网关服务器转发至指定的游戏服务器之后，游戏服务器进行逻辑处理，然后将处理后的消息转发至所有的网关服务器通道中，再由网关服务器转发至所有连接的客户端通道中，这样可以实现广播功能。

### 3.7.2 扩展接口设计

网关服务器架构中的游戏服务器模块提供很多二次开发的组件，但是这样还不能保证服务器架构的可扩展性，需要将游戏服务器模块划分为基本模块和开发模块，基本模块实现服务器模块的基本功能，包括消息处理、消息组播等，开发模块抽象出开发者接口，开发者实现抽象接口的方法即可实现新的业务逻辑，并且该接口可以调用基本模块的方法。

游戏服务器启动之后是一个实例对象，需要为开发者提供一个二次开发接口，而这个接口需要调用实例对象中提供的方法，这个时候比较适合的类关系是组合关系，让游戏服务器拥有一个抽象接口的实例引用，这样抽象接口的实现类反过来也可以调用实例对象所提供的方法。

## 3.8 本章小结

本章首先对网关服务器架构进行了需求分析，然后将架构划分为通信模块、网关服务器模块、服务注册中心模块和游戏服务器模块，对每个模块的需求进行了描述和分析，最后对每个模块进行了详细设计，并给出了每个模块主要功能的设计思路。

# 第四章 网关服务器架构的实现

本章主要介绍网关服务器架构的实现过程，并对其中的主要功能模块通用模块、网关服务器模块、服务注册中心模块和游戏服务器模块的实现进行了详细的描述，最后给出了测试。

## 4.1 通信模块的实现

通信模块是网关服务器架构的各个模块之间通信的核心，本节给出了通信协议子模块和心跳检测子模块的具体实现。

### 4.1.1 通信协议的实现

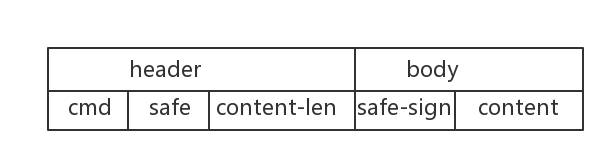


图4.1 自定义二进制协议

如图4.1所示，网关服务器架构实现的是自定义的二进制协议，包括协议头header和消息体body，具体每个字段的解释如下：

* cmd：8bit，可表示255种在协议命令。
* safe：8bit，安全位，默认为0表示协议不加密，同时body中safe-sign为空，如果为1，表示用safe-sign字符串对content内容进行加密。
* content-len：16bit，消息内容长度，如果safe安全位为1，需要分别读取safe-sign和content，然后用解密算法对content进行解密。
* safe-sign：16Byte，当safe安全为为0时，safe-sign字段为空字符串，当safe安全位为1时，safe-sign表示16位字符串，content为被safe-sign加密过的内容。
* content：长度未知，由content-len字段指定。

其中cmd字段8位，可以用1-255之间任意的整数来表示，总共可以表示的命令类型有255种。网关服务器架构已经定义了13种命令，其中1表示向所有客户端发送消息；2表示向某一个客户端发送消息；3表示向某一个用户发送消息，包含该用户的所有客户端设备；4表示向某一个分组发送消息，该分组的所有客户端都会收到该消息；5表示获取所有在线客户端数量；6表示获取某个用户的所有在线客户端设备数量；7表示获取某个分组的所有在线客户端数量；8表示客户端绑定某用户；9表示客户端解绑某用户；10表示客户端加入某分组；11表示客户端离开某分组；101表示该消息的消息源是客户端；102表示该消息的消息源是服务器。

网关服务器架构的通信协议是自定义二进制协议，没有引入其他序列化与反序列化框架，这样可以保证客户端可以使用任意语言来实现，但是需要实现自定义协议的编码和解码方法，在发送消息的时候需要将消息对象编码成二进制数据流，在接收消息的时候需要将二进制消息流解码成消息对象。

|  |
| --- |
| public byte[] encode(GproxyProtocol protocol) {  byte[] identifier = ByteUtil.int2Byte(protocol.getIdentifier(), 1);  byt byte[] cmd = ByteUtil.int2Byte(protocol.getHeader().getCmd(), 1);  byte[] safe = ByteUtil.int2Byte(protocol.getHeader().getSafe(), 1);  byte[] safeSign = protocol.getBody().getSafeSign().getBytes();  byte[] content = protocol.getBody().getContent().getBytes();  byte[] contentLen = ByteUtil.int2Byte(content.length, 2);  byte[] length = null;  if (header.getSafe() == 0) {  length = ByteUtil.int2Byte(content.length + 4, 2);  } else {  length = ByteUtil.int2Byte(content.length + 20, 2);  }  return ByteUtil.mergeArrays(identifier, length, cmd, safe,  contentLen,safeSign, content);  } |

图4.2 消息编码代码

图4.2是消息编码代码，其中GproxyProtocol类是系统变量类，byte[]数组是字节数组， encode方法将GproxyProtocol类编码为字节数组。identifier字段为消息开始标志符，当socket通道读取到该标志时即开始读取字节流并解析为对应的消息，占一个字节长度；cmd是消息命令字段，标识着这条消息的类型，与业务逻辑相关，占一个字节；safe是消息安全标识符，当该字段为0时，表示这条消息不加密，消息体全部填充为消息内容，当该字段为1时，表示该条消息加密，消息体前16个字节为加密字符串，之后的字节流是加密后的消息内容，接收方可根据该加密串使用解密算法解密消息内容，占一个字节；safeSign字段为加密字串默认16个字节，如果该消息不加密则该字段为空；content字段为消息内容，占用字节数根据实际内容而定；contentLen为消息体长度，表示消息内容的长度，占用两个字节；length为消息长度，表示消息总长度，占两个字节。以上各字段的字节组成的字节流可以在socket通道中传输。

|  |
| --- |
| public GproxyProtocol decode(byte[] bytes) {  int cmd = ByteUtil.byte2Int(ArrayUtils.subarray(bytes, 0, 1));  int safe = ByteUtil.byte2Int(ArrayUtils.subarray(bytes, 1, 2));  int contentLen = ByteUtil.byte2Int(ArrayUtils.subarray(bytes, 2, 4));  if (contentLen > 0) {  if (safe == 0) {  content = new String(ArrayUtils.subarray(bytes, 4, bytes.length),"UTF-8");  } else {  safeSign = new String(ArrayUtils.subarray(bytes, 4, 20));  content = new String(ArrayUtils.subarray(bytes,20,bytes.length),"UTF-8");  }  }  return new GproxyProtocol(new GproxyHeader(cmd, safe, contentLen),  new GproxyBody(safeSign, content));  } |

图4.3 消息解码代码

图4.3是消息解码代码，其中GproxyProtocol类是系统变量类，byte[]数组是字节数组，decode方法是将字节数组根据通信协议解码为GproxyProtocl类变量。字节数组的第一个字节解析为cmd变量，第二个字节解析为safe变量，第3到4字节解析为contentLen变量，再根据safe和safeSign字段解析出content变量，最后将GproxyProtocol变量返回。

由于TCP数据传输是无边界的数据流传输形式，接收方接收到的数据流可能是半个数据包，也可能是多个数据包黏在一起，为了解决这个问题，通信协议采用自定义消息协议，并且在消息协议中定义消息头header和消息体body，消息头header中携带消息体长度，所以可以通过程序来读取一个完整的数据包。首先需要定义一个协议开始标识符，当读取到该开始标识符的时候，表示接下来可以根据消息头header的长度来读取协议消息头，并且从消息头中读取消息体的长度，将消息头和消息体总共长度的字节流传递到解码器中进行解码。

图4.4是消息解包代码，其中decode方法利用底层缓冲区来缓冲字节数组，根据自定义二进制协议，来实现对消息的解包，解决了消息分包黏包的问题。BASE\_LENGTH表示数据包的最短长度，当缓冲区中缓冲的字节数超过该数值时，才开始从缓冲区中读取数据并解析为消息。MAX\_LENGTH定义为数据包的最大长度，当数据包长度大于该数值时直接丢弃，判定该数据是socket字节流攻击。当读到消息开始标志符时，才开始将后面的数据解析为一个数据包，接着从缓冲区中读取两个字节，表示该数据包的长度，再根据数据包的长度从缓冲区中读取字节流，对该字节数组调用decode方法即可得到GproxyProtocol类变量，之后便可以进行业务逻辑处理。

|  |
| --- |
| protected void decode(ChannelHandlerContext channelHandlerContext, ByteBuf byteBuf, List<Object> list) throws Exception {  if (byteBuf.readableBytes() > BASE\_LENGTH) {  if (byteBuf.readableBytes() > MAX\_LENGTH) {  byteBuf.skipBytes(byteBuf.readableBytes());  }  while (true) {  beginReader = byteBuf.readerIndex();  byteBuf.markReaderIndex();  int readByte = ByteUtil.byte2Int(byteBuf.readByte());  if (readByte == Const.HEAD\_DATA) { break; }  byteBuf.readByte();  }  byteBuf.readBytes(new byte[byteBuf.readShort()]);  list.add(GproxyCoder.getInstance().decode(bytes));  }  } |

图4.4 消息解包代码

### 4.1.2 心跳检测的实现

在网关服务器架构中，网关服务器需要同时维持与客户端和游戏服务器的连接，为此需要为通信连接实现心跳检测，使用Netty框架提供的IdleStateHandler类可以实现心跳检测功能。

图4.5代码是心跳检测代码，其中baseInit方法负责初始化一系列Handler句柄，其中IdleStateHandler是Netty框架提供给开发者的心跳检测Handler，有四个构造参数，第一个参数是读空闲超时时间，在该时间内无读操作就触发超时事件调用userEventTriggered()方法；第二个参数是写空闲超时时间，在该时间内无写操作就触发超时事件调用userEventTriggered()方法；第三个参数是读写空闲超时时间，在指定时间内无读写操作就触发超时事件调用userEventTriggered()方法；第四个参数是时间单位，一般设置为秒。当心跳检测触发事件调用userEventTriggered方法时，在该方法内分别对读超时事件、写超时事件、读写超时事件分别进行业务逻辑处理。

|  |
| --- |
| public static SocketChannel baseInit(SocketChannel ch) {  ch.pipeline().addLast(new IdleStateHandler(30, 30, 30，TimeUnit.SECONDS));  }  public void userEventTriggered(ChannelHandlerContext ctx, Object evt) {  if (evt instanceof IdleStateEvent) {  if (evt.state().equals(IdleState.READER\_IDLE)) { ctx.close();}  else if (event.state().equals(IdleState.WRITER\_IDLE)) { ctx.close(); }  else if (event.state().equals(IdleState.ALL\_IDLE)) {  MsgHandleService.getInstance().sendMsgUtil.sendHeartMessage(ctx);  }  }  } |

图4.5心跳检测代码

## 4.2网关服务器模块的实现

网关服务器模块是网关服务器架构的核心模块，本节给出了服务管理子模块、消息路由子模块和单播与多播子模块的具体实现。

### 4.2.1 服务管理的实现

在网关服务器架构中，一个网关服务器的I/O通信负载有限，当游戏服务器数量增加的时候，需要引入多个网关服务器，每个网关服务器管理多个游戏服务器，这样可以解决网关服务器的负载压力问题。在这种情况下，需要对服务器节点进行管理，每个网关服务器管理指定数量的游戏服务器节点。

图4.6代码为网关服务器管理游戏服务器代码，ProxyNodeEventHandler类是网关服务器节点管理事件的类，网关服务器注册了游戏服务器节点监听事件，节点增加的时候会调用addNode方法，addNode方法有两个参数，第一个参数serverName表示服务器注册名称，第二个参数uri表示游戏服务器地址。在addNode方法中，首先调用NodeEventFilter实例的filter方法，该方法将serverName作为参数，如果该游戏服务器被划分到当前网关服务器，则不会执行下面的return语句，会继续执行下面的代码，向uri表示的游戏服务器发起连接，如果该游戏服务器没有被划分到当前网关服务器，直接执行return语句不再向游戏服务器发起连接。节点删除的时候会调用removeNode方法，该方法同样会调用NodeEventFilter实例的filter方法，filter方法作用和addNode中filter方法相同，removeServerAddress方法将网关服务器保存的游戏服务器连接通道删除，客户端消息不会再被路由到该游戏服务器。节点信息变化的时候会调用updateNode方法，同样会调用NodeEventFilter实例的filter方法，filter方法作用和addNode中的filter方法相同，需要将原先保存的客户端通道删除，再重新向新的游戏服务器发起连接。

|  |
| --- |
| public class ProxyNodeEventHandler implements NodeEventHandler {  private NodeEventFilter filter;  public void addNode(String serverName, URI uri) {  if (!filter.filter(serverName)) { return; }  proxy.addServerAddress(serverName, uri);  }  public void removeNode(String serverName) {  if (!filter.filter(serverName)) { return; }  proxy.removeServerAddress(serverName);  }  public void updateNode(String serverName, URI uri) {  if (!filter.filter(serverName)) { return; }  proxy.removeServerAddress(serverName);  proxy.addServerAddress(serverName, uri);  }  } |

图4.6管理游戏服务器代码

### 4.2.2 消息路由的实现

消息路由是网关服务器用来实现路由算法的模块，根据需求目前实现了两种路由算法，一种是随机路由算法，一种是指定路由算法。

图4.7代码为随机路由算法，serverChannels字典中存储着该网关服务器所连接的服务器channel通道，key为客户端Id，value为ServerChannel通道。当客户端第一次向服务器发送消息时，网关服务器调用该随机路由算法，生成一个随机数索引，根据该索引找到serverChannels中对应的ServerChannel通道，并将其与客户端Id绑定。当客户端再次向服务器发消息时，网关服务器根据客户端Id找到对应的ServerChannel通道，不需要再调用路由算法。

网关服务器在接收到客户端的消息时，调用路由算法获得channel通道。如果消息未到达服务器，重新发送一遍该消息，如果消息仍然未到达服务器，则该游戏服务器宕机或者网关服务器与该游戏服务器之间的网络无法连通，此时需要再次调用路由算法将该客户端重新绑定到另外一个游戏服务器上，该客户端的所有消息将会被网关服务器转发至新绑定的游戏服务器中去。

|  |
| --- |
| public void randRoute(String clientId, ConcurrentHashMap<String, ServerChannel> serverChannels, HashMap<String, ServerChannel> map) {  int randIndex = random.nextInt(length);  Iterator iterator = serverChannels.entrySet().iterator();  int index = 0;  while (iterator.hasNext()) {  if (randIndex == index) {  ServerChannel serverChannel = (ServerChannel) iterator.next().getValue();  map.put(clientId, serverChannel);  break;  }  index++;  }  } |

图4.7随机路由算法代码

图4.8代码为指定路由算法，serverChannels字典中存储着该网关服务器连接的所有服务器channel通道，key为客户端Id，value为ServerChannel通道。客户端发送消息时，网关服务器先从该客户端的消息中解析出指定的服务器Id，从serverChannels中查找有没有对应的服务器，如果有就将客户端Id与找到的服务器通道serverChannel绑定。网关服务器调用该算法之后，将客户端消息转发至对应的服务器，如果消息第一次发送失败，再重试一遍发送消息，如果消息仍然没有到达服务器，则游戏服务器宕机或者网关服务器与游戏服务器网络断开，不再向该服务器发送消息直到该服务器重新启动。

|  |
| --- |
| public void routeByServer(String clientId, String serverId, ConcurrentHashMap<String, ServerChannel> serverChannels, HashMap<String, ServerChannel> map) {  if (serverChannels.containsKey(serverId)) {  ServerChannel serverChannel = serverChannels.get(serverId);  map.put(clientId, serverChannel);  }  } |

图4.8指定路由算法代码

### 4.2.3 单播与多播的实现

单播是客户端与游戏服务器之间的一对一通信，组播和广播都是客户端与游戏服务器之间的多对一通信。客户端的消息经过网关服务器的转发到达游戏服务器，经过游戏服务器处理后决定发送的消息类型为单播、组播还是广播。

图4.9是网关服务器单播与多播代码，其中channelRead方法是网关服务器接收到游戏服务器消息时的响应方法，该方法的一个参数是GproxyProtocol类实例，首先从该protocol变量中读取消息命令字段，接着根据命令字段分别做相应的处理。服务器消息中常用的命令字段解释如下，SEND\_TO\_ALL表示该消息是一条广播消息，网关服务器将该消息转发至当前连接的所有客户端通道中；SEND\_TO\_CLIENT表示该消息是一条单播消息，网关服务器从消息中解析出clientId，然后根据clientId找到对应的客户端连接通道，将服务器消息转发至该客户端通道中；SEND\_TO\_GROUP表示该消息是一条组播消息，网关服务器从消息中解析出groupId，然后根据groupId找到绑定在该分组中的所有客户端连接通道，然后将服务器消息转发至这些客户端通道中；JOIN\_GROUP表示该消息是一条命令消息，网关服务器从消息中解析出clientId和groupId，然后将该clientId对应的客户端通道绑定到groupId对应的分组中，下次向该分组中发送组播消息的时候，clientId对应的客户端就能够接收到对应的组播消息；LEAVE\_GROUP表示该消息是一条命令消息，网关服务器从消息中解析出clientId和groupId，然后将该clientId对应的客户端通道从groupId对应的分组中解绑，下次向该分组中发送组播消息的时候，clientId对应的客户端就接收不到对应的组播消息。

|  |
| --- |
| public void channelRead(ChannelHandlerContext ctx, Object msg) throws Exception {  int cmd = ((GproxyProtocol) msg).getHeader().getCmd();  String clientId = GproxyJson.getClientId(content);  String groupId = GproxyJson.getGroupId(content);  String message = GproxyJson.getMessage(content);  switch (cmd) {  case GproxyCommand.SEND\_TO\_ALL:  proxy.sendToAll(protocol);  break;  case GproxyCommand.SEND\_TO\_CLIENT:  protocol = BaseEventConverter.converter(protocol, message);  proxy.sendToClient(protocol, clientId);  break;  case GproxyCommand.SEND\_TO\_GROUP:  protocol = BaseEventConverter.converter(protocol, message);  proxy.sendToGroup(protocol, groupId);  break;  case GproxyCommand.JOIN\_GROUP:  proxy.joinGroup(clientId, groupId);  break;  case GproxyCommand.LEAVE\_GROUP:  proxy.leaveGroup(clientId, groupId);  break;  }  } |

图4.9单播与多播代码

## 4.3 服务注册中心模块的实现

服务注册中心模块是网关服务器架构的核心模块，本节给出了服务注册子模块、服务发现子模块、服务监听子模块、节点管理子模块和负载均衡子模块的具体实现。

### 4.3.1 服务注册的实现

游戏服务器在启动之后，需要向服务注册中心注册该游戏服务器的服务器地址信息，网关服务器在启动之后，需要向服务注册中心注册该网关服务器的服务器地址信息。此外，还可能有其他应用使用服务注册中心注册应用服务，服务注册中心需要提供服务注册功能。

|  |
| --- |
| public void registerNode(String path, CreateMode mode, byte[] bytes, boolean is) {  if (is) {  curatorFramework.create().creatingParentsIfNeeded().withMode(mode)  .forPath(path, bytes);  } else { curatorFramework.create().withMode(mode).forPath(path, bytes); }  }  public void removeNode(String path, boolean is) {  if (is) { curatorFramework.delete().deletingChildrenIfNeeded().forPath(path);}  else { curatorFramework.delete().forPath(path);}  } |

图4.10服务注册代码

图4.10代码表示服务注册代码， 服务注册包括两个方法，registerNode方法表示向服务注册中心注册节点，该方法有四个参数，path表示需要注册的节点路径，mode表示该节点的类型，在ZooKeeper中有四种节点类型，PERSISTENT表示该节点是持久化节点；PERSISTENT\_SEQUENTIAL表示该节点是顺序编号持久化节点；EPHEMERAL表示该节点是临时节点；EPHEMERAL\_SEQUENTIAL表示该节点是顺序编号临时节点。bytes字节数组表示该节点需要注册的内容字节数组，is表示是否创建不存在的父节点，true表示自动创建不存在的父节点，false表示不创建不存在的父节点，registerNode方法将节点信息注册到服务注册中心的某个节点路径上。

removeNode方法表示向服务注册中心删除节点，该方法有两个参数，path表示需要删除的节点路径，is表示递归删除子节点，true表示将递归删除所有子节点，false表示不删除子节点，如果存在子节点则删除失败。removeNode方法将某节点路径上的节点从服务注册中心中删除。

### 4.3.2 服务发现的实现

客户端在启动之后，需要从服务注册中心获取网关服务器节点信息列表，然后向网关服务器发起连接，网关服务器启动之后，需要向服务注册中心获取游戏服务器节点信息，然后向游戏服务器发起连接。

图4.11表示服务注册中心服务发现代码，服务发现包括两个方法。getData方法表示获取指定节点路径下节点数据，有一个参数path表示节点路径，该方法从服务注册中心获取到的是字节数组，将其转化为字符串形式返回，获取到的节点数据是服务注册时注册到节点中的数据。

getChildNodes方法表示获取指定节点路径下所有子节点数据，有一个参数path表示节点路径，该方法从服务注册中心获取到list集合，list中存储所有子节点数据，将其转化为RemoteAddress形式数组，然后返回该数组。

|  |
| --- |
| public String getData(String path) {  byte[] forPath = curatorFramework.getData().forPath(path);  return new String(forPath);  }  public RemoteAddress[] getChildNodes(String path) {  List<String> forPath = curatorFramework.getChildren().forPath(path);  RemoteAddress[] addresses = new RemoteAddress[forPath.size()];  int num = 0;  for (String paths : forPath) {  RemoteAddress address = new RemoteAddress(data);  addresses[num++] = address;  }  return addresses;  } |

图4.11服务发现代码

### 4.3.3 服务监听的实现

网关服务器在服务启动后，需要向服务注册中心对已经注册过的服务进行拉取，并为已注册服务的子节点设置监听或者为一个管理众多子节点的父节点设置监听。一旦为某一个节点设置监听，当这个节点的内容或者该节点的子节点内容发生了变化，都需要调用handler处理器来进行相应的处理。

图4.12代码表示为服务注册监听，为每一个节点注册子节点监听事件，当该父节点有子节点变化的时候，分别为三种变化注册了事件处理handler，包括子节点增加、子节点内容更新、子节点删除事件，分别为每一种事件实现一个handler句柄。子节点增加的时候，需要读取该子节点的内容，并向该游戏服务器发起连接，连接完成后将其保存在对应关系中；子节点内容更新的时候，需要读取该子节点的内容，并向该游戏服务器发起连接，连接完成后将原先保存的对应关系更新；子节点删除的时候，需要将保存的对应关系删除。

|  |
| --- |
| public void setPathChildrenListener(String path) {  public void childEvent(CuratorFramework client, PathChildrenCacheEvent event) {  String path = event.getData().getPath();  switch (event.getType()) {  case CHILD\_ADDED:  eventHandler.addNode(path, uri);  break;  case CHILD\_REMOVED:  eventHandler.removeNode(path);  break;  case CHILD\_UPDATED:  eventHandler.updateNode(path, uri);  break;  }  }  } |

图4.12服务监听代码

### 4.3.4 节点管理的实现

在网关服务器架构中，所有的游戏服务注册到服务注册中心的时候，需要为游戏服务统一规划节点。首先为整个架构划分一个命名空间Gproxy，在该节点下面划分一个proxy节点存储所有网关服务节点信息，在该节点下面存储每一个具体的网关服务器节点信息；然后再Gproxy命名空间下面划分一个server节点存储所有网关服务节点信息，在该节点下面存储每一个具体的游戏服务器节点信息。

|  |
| --- |
| public void registerServer(String zkAddress, String space, String serverName, String serverAddress) {  ZookeeperService zookeeperService = new ZookeeperService();  zookeeperService.setPath(space);  zookeeperService.setZkAddress(zkAddress);  zookeeperService.afterPropertiesSet();  if (!zookeeperService.exists(serverName)) {  zookeeperService.registerNode(serverName, new URI(zkAddress),  CreateMode.EPHEMERAL, serverAddress.getBytes(), false);  }  } |

图4.13节点管理代码

图4.13是节点管理代码，registerServer方法可以向服务注册中心注册服务节点信息，其中zkAddress表示ZooKeeper服务器地址，space表示该节点在ZooKeeper中的应用空间，serverName表示需要注册的服务器名称，serverAddress表示需要注册的服务器地址。在向ZooKeeper中注册服务信息的时候，首先要初始化ZooKeeperService对象，然后为该对象初始化命名空间、ZooKeeper服务器地址和验证权限等信息，最后将服务器名称和服务器地址注册到ZooKeeper集群中去，由ZooKeeper对该节点进行监听，节点信息发生变化的时候，会主动将信息推送到网关服务器。通过对不同服务划分不同的节点空间，实现了对服务的管理。

### 4.3.5 负载均衡的实现

网关服务器架构的服务注册中心是基于ZooKeeper服务来实现服务注册、服务发现和服务监听的功能，为了保证注册服务中心的可用性，使用ZooKeeper集群对外提供统一服务，由于是集群服务，所以要使用负载均衡算法为服务器集群实现负载均衡。

|  |
| --- |
| public URI selectRandom(RemoteAddress[] uris) {  int length = uris.length;  int randIndex = new Random().nextInt(length);  return uris[randIndex].getUri();  } |

图4.14随机算法代码

图4.14是随机算法代码，实现了随机算法对ZooKeeper服务器集群的负载均衡。算法首先生成一个随机数索引序号，然后根据该索引序号从ZooKeeper服务器列表中找到对应的服务器，即可返回对外提供服务的服务器。

|  |
| --- |
| public URI selectRandom(RemoteAddress[] uris) {  String hostAddress = Inet4Address.getLocalHost().getHostAddress();  Long hash = MurMurHash.hash(hostAddress);  hash = Math.abs(hash);  int ipIndex = (int) (hash == null ? 0 : (hash % uris.length));  return uris[ipIndex].getUri();  } |

图4.15一致性哈希算法代码

图4.15代码实现了一致性哈希算法来实现对ZooKeeper服务器集群的负载均衡。一致性哈希算法是把所有服务器ip从小到大放到一个环中，在向ZooKeeper服务器集群发起请求的时候，会根据当前服务器ip调用一致性哈希算法找到最接近当前服务器ip并且大于当前服务器ip的ZooKeeper服务器，对外提供ZooKeeper服务。

## 4.4 游戏服务器模块的实现

游戏服务器模块是网关服务器架构的核心模块，本节将游戏服务器模块划分成了单播与多播子模块和扩展接口子模块，并给出了每个子模块的具体实现。

### 4.4.1 单播与多播的实现

单播是客户端与游戏服务器之间的一对一通信，组播和广播都是客户端与游戏服务器之间的多对一通信。客户端的消息经过网关服务器的转发到达游戏服务器，经过游戏服务器处理后根据消息类型来决定是单播、组播或者是广播。

|  |
| --- |
| public void send(GproxyProtocol protocol, String proxyId) {  if (!StringUtils.isBlank(proxyId)) {  ProxyChannel proxyChannel = proxyChannels.get(proxyId);  proxyChannel.getChannel().writeAndFlush(protocol);  } else {  Iterator iterator = proxyChannels.entrySet().iterator();  while (iterator.hasNext()) {  ProxyChannel proxyChannel = ((ProxyChannel) iterator.next().getValue());  proxyChannel.getChannel().writeAndFlush(protocol);  }  }  } |

图4.16单播、组播与广播代码

图4.16代码表示游戏服务器在发送消息的时候，根据消息类型对消息进行单播、组播或者广播。游戏服务器在发送消息到客户端之前，先判断当前消息调用是否携带通道Id，如果是单播消息，只需要将该消息转发至消息源网关服务器，该消息会携带通道Id，游戏服务器可以从proxyChannels集合中找到对应的网关服务器通道，将该消息转发至该通道中；如果是组播或广播消息，游戏服务器会将该消息转发至所有的网关通道中，网关服务器接收到消息后，再根据消息类型决定对该消息进行组播还是广播，如果是组播消息，网关服务器从消息中解析出分组Id，并将该消息发送至该分组的所有客户端通道中，如果是广播消息，则将该消息发送至连接的所有客户端通道中。

### 4.4.2 扩展接口的实现

网关服务器架构中的游戏服务器模块提供很多二次开发的组件，但是这样还不能保证服务器架构的可扩展性，需要将游戏服务器模块划分为基本模块和开发模块，基本模块实现服务器模块的基本功能，包括消息处理、消息组播等，开发模块抽象出开发者接口，开发者实现抽象接口的方法即可实现新的业务逻辑，并且该接口可以调用基本模块的方法。

|  |
| --- |
| public abstract class Callback {  public abstract void onMessage(GproxyProtocol protocol);  public void sendToAll(String message) {  server.sendToAll(message);  }  public void sendToClient(String message, String clientId) {  server.sendToClient(message, clientId);  }  public void sendToUser(String message, String userId) {  server.sendToUser(message, userId);  }  public void sendToGroup(String message, String groupId) {  server.sendToGroup(message, groupId);  }  public void bindUid(String clientId, String userId) {  server.bindUid(clientId, userId);  }  public void unBindUid(String clientId, String userId) {  server.unBindUid(clientId, userId);  }  public void joinGroup(String clientId, String groupId) {  server.joinGroup(clientId, groupId);  }  public void leaveGroup(String clientId, String groupId) {  server.leaveGroup(clientId, groupId);  }  } |

图4.17扩展接口代码

图4.17代码是供开发人员二次开发实现的抽象类，开发人员需要实现onMessage回调方法。onMessage方法是游戏服务器接收到消息的回调方法，方法中有一个GproxyProtocol对象，该对象是由缓冲区二进制消息经过协议解码后得到的。开发人员可以在该方法中实现业务逻辑的二次开发，并且还可以调用被封装好的方法，方法的解释如下：

* sendToAll：将消息发送至所有客户端，是一条广播消息。
* sendToClient：将消息发送至指定客户端，是一条单播消息。
* sendToUser：将消息发送至用户组中，该用户组包含一个用户的不同终端设 备，是一条组播消息。
* sendToGroup：将消息发送至自定义分组中，该分组包含多个用户，是一条 组播消息。
* bindUid：将客户端绑定到某个用户分组中。
* unBindUid：将某个客户端从指定用户分组中解绑。
* joinGroup：将客户端绑定到某个自定义分组中。
* leaveGroup：将某个客户端从指定自定义分组中解绑。

## 4.5 系统测试

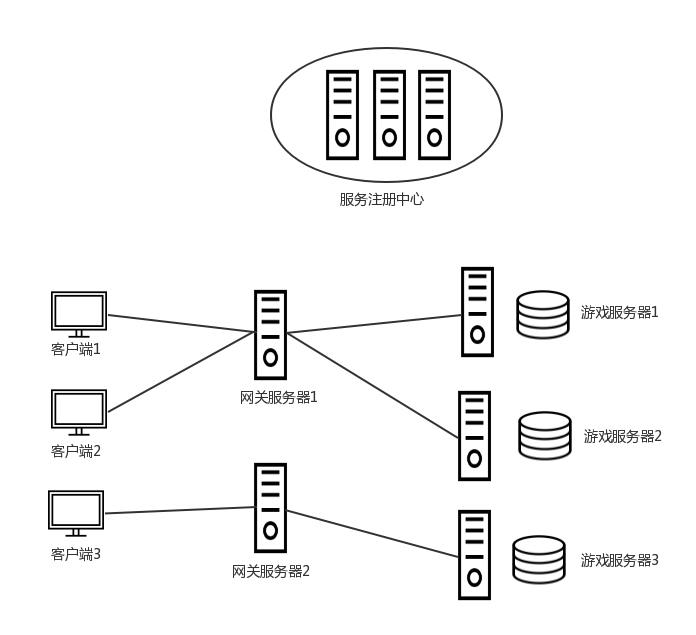


图4.18测试部署图

游戏网关服务器架构是一个服务器架构，可以直接使用该架构的网关服务器模块和服务注册中心模块，游戏服务器模块抽象了二次开发接口，客户端如果使用java语言开发，也可以使用系统的通信模块，所以本网关服务器架构不是一个完整意义的可运行系统，但是并不妨碍对该网关服务器架构已实现的功能进行测试。

为了测试网关服务器架构已实现的功能，在该系统架构上实现了一个测试系统，具体部署如图4.18所示。该测试系统包括三个游戏客户端，两个网关服务器，一个服务注册中心，三个游戏服务器。客户端1与客户端2连接网关服务器1，客户端2连接网关服务器2，网关服务器1管理游戏服务器1和游戏服务器2，网关服务器2管理游戏服务器3。

为了实现对系统已实现功能的测试，对系统进行如下的测试步骤：

（1）启动服务注册中心，如图4.19- 4.21所示，ZooKeeper服务注册中心集群启动成功，服务器1使用2181端口对外服务，服务器2使用2182端口对外服务，服务器3使用2183端口对外服务，三个服务器并通过选举算法，顺利达成一致，对外提供稳定的服务。

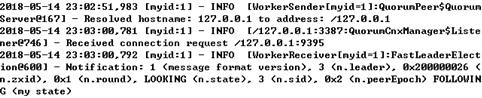


图4.19 ZooKeeper服务器1启动

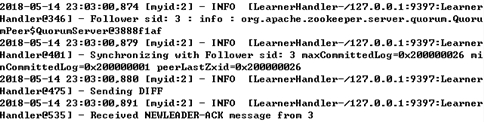


图4.20 ZooKeeper服务器2启动

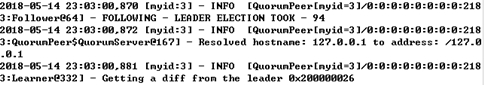


图4.21 ZooKeeper服务器3启动

（2）启动游戏服务器1，游戏服务器2和游戏服务器3。

（3）启动网关服务器1和网关服务器2。

（4）客户端1向网关服务器1发起连接，客户端2向网关服务器1发起连接，客户端3向网关服务器2发起连接。

（5）客户端1向游戏服务器1发送消息。如图4.22所示，客户端1向游戏服务器1发送消息，服务器1收到消息并将消息返回到客户端1，客户端2和客户端3接收不到这条消息。

wps1ED4

图4.22 客户端1发送单播消息

（6）客户端1加入分组1，并向分组1发送消息。如图4.23所示，客户端1向分组1发送消息，此时分组1中只有客户端1，所以只有客户端1能接收到消息，客户端2和客户端3接收不到消息。

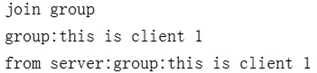


图4.23 客户端1发送分组消息

（7）客户端2加入分组1，并向分组1发送消息。如图4.24所示，客户端2向分组1发送消息，此时分组1中有客户端1和客户端2，都可以接收到分组消息，客户端3接收不到消息。

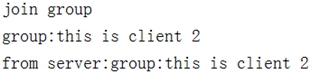


图4.24 客户端2发送分组消息

（8）客户端3发送广播消息。如图4.25所示，客户端3发送广播消息，客户端1、客户端2、客户端3都可以接收到该消息。

wps9695

图4.25 客户端3发送广播消息

经过以上八个步骤的验证，系统的通信模块、网关服务器模块、服务注册中心模块和游戏服务器模块的功能都得到了验证。

## 4.6 本章小结

本章对网关服务器架构的主要模块通信协议模块、网关服务器模块、服务注册中心模块和游戏服务器模块进行了详细的描述，并对每一个模块的具体功能给出了代码实现并进行了相应的解释，最后对网关服务器架构进行了测试，验证了已实现的功能。

# 第五章 总结与展望

## 5.1 总结

本文以上海某游戏公司服务器架构为背景，当前服务器架构客户端与游戏服务器直连，随着玩家数量的增加，无法支持对服务器组的负载均衡，而且服务器之间相互独立不通信，无法实现跨服战场。文中对现有服务器架构进行了改进，在客户端和游戏服务器之间增加了网关服务器，有效地解决了当前服务器架构的不足之处，并将整个系统架构划分为通信模块、网关服务器模块、服务注册中心模块和游戏服务器模块，对每个模块进行了需求分析和详细设计，并给出了每个模块的具体实现，最后对架构进行了测试，验证了网关服务器架构实现的功能。

本文阐述的网关服务器架构采用Java作为开发语言，以Netty网络库作为网络通信的基础，采用了Netty的主从多线程模型，保证了各模块之间通信的可靠性和稳定性，通信消息使用了自定义的二进制协议，实现了二进制消息的编码和解码，支持任意语言的客户端实现。使用ZooKeeper服务构建了服务注册中心，ZooKeeper集群实现了分布式一致性协议，只要集群中有半数以上机器存活，就可以保证服务注册中心的可用性。

本文详细介绍了网关服务器架构的需求分析、详细设计和具体实现，网关服务器作为客户端与游戏服务器之间的通信中介，不仅可以实现对消息的路由转发，还根据定义的消息类型实现了消息单播、消息组播和消息广播功能，有效地解决了跨服战场和全服战场的问题。服务注册中心对游戏服务进行管理和监听，实现了服务器动态上线、下线和地址变更。

游戏网关服务器架构目前已经开发完成，可以在该架构上进行二次开发。客户端可以使用系统现有的通信协议模块，网关服务器模块和服务注册中心模块不需要进行二次开发，游戏服务器模块已经抽象出用户开发接口，提供可用的api，开发人员可以对游戏服务器进行二次开发。目前公司已经为网关服务器架构成立开发小组，使用该架构已经完成了对游戏服务器逻辑的部分重构，为游戏服务器架构的演进奠定了基础。

## 5.2 进一步工作展望

目前游戏网关器架构已经开发基本完成，但是还存在一些不足之处，有很多需要完善的地方：

（1）目前网关服务器架构还没有实现数据库部分，可以在服务注册中心划分数据库节点信息，实现对数据库服务器的管理，支持数据库服务器动态上线和下线。

（2）游戏客户端与服务器之间通常需要Session来保存和验证玩家信息，后续需要实现Session模块，来为系统架构提供Session管理功能。

# 

# 参 考 文 献

**[Ding et al., 2012] H.** Ding, Y. Jin and Y. Cui, Distributed storage of network measurement data on hbase//Cloud Computing and Intelligent Systems (CCIS), *2012 IEEE 2nd International Conference on. IEEE*, 2012, 2: 716-720.

[Gourley et al., 2012] D. Gourley, B. Totty and M. Sayer, *HTTP 权威指南*, 2012.

[Guo, 2013] S. Guo, *Hadoop operations and cluster management cookbook*, Packt Publishing Ltd, 2013.

**[Hitchens, 2002]** R. Hitchens, *Java NIO\c Ron Hitchens*, " O'Reilly Media, Inc.", 2002.

**[Hunt, 2010] P.** Hunt, M. Konar and F. P. Junqueira, ZooKeeper: Wait-free Coordination for Internet-scale Systems//*USENIX annual technical conference*, 2010, 8(9).

[Shi et al., 2014] X. Shi, H. Lin and H. Jin, GIRAFFE: A scalable distributed coordination service for large-scale systems//Cluster Computing (CLUSTER), *2014 IEEE International Conference on. IEEE*, 2014: 38-47.

[Shvachko et al., 2010] K. Shvachko, H. Kuang and S. Radia, The Hadoop Distributed File System// *MASS Storage Systems and Technologies. IEEE*, 2010:1-10.

[Skeirik et al., 2013] S. Skeirik, R. B. Bobba and J. Meseguer, Formal analysis of fault-tolerant group key management using ZooKeeper//Cluster, Cloud and Grid Computing (CCGrid), *2013 13th IEEE/ACM International Symposium on. IEEE*, 2013: 636-641.

[符军, 2009] 符军，基于sniffer的TCP/IP协议分析，*科技风*， 2009(4X):74-74。

[何咏明, 2012] 何咏明，JAVA中使用Socket通讯的实现，*长江工程职业技术学院学报*，2012，29(4):35-37。

**[江岸, 2014]** 江岸，一种新型的网络游戏服务器架构设计，*计算机光盘软件与应用*，2014(3):296-297。

[金志国等, 2014] 金志国，李炜，基于Netty的HTTP客户端的设计与实现，*电信工程技术与标准化*，2014(1):84-88。

[李林锋, 2015] 李林锋，*Netty权威指南*，电子工业出版社，2015。

[李攀登, 2012] 李攀登，*全服单世界网页游戏服务器的研究与设计*，华南理工大学，2012。

[刘海超, 2015] 刘海超，手机网游服务器架构的研究，*通讯世界*，2015(13):92-93。

**[鲁兆硕, 2017]** 鲁兆硕，Java在高并发网络编程中的应用，*科学中国人*，2017(9)。

[罗振兴等, 2007] 罗振兴，努尔买买提，黑力力，徐大伟等，基于Java新IO的Web安全网关，*计算机工程*，2007， 33(6):107-109。

**[毛仁伟, 2014]** 毛仁伟，*大型多人在线游戏中负载均衡及相关技术的研究*，首都师范大学，2014。

[谭玉靖, 2014] 谭玉靖，*基于ZooKeeper的分布式处理框架的研究与实现*，北京邮电大学，2014。

[王智印等, 2017] 王智印，李丹，Java基于TCP/IP协议的Socket通信，*产业与科技论坛*，2017(21):41-42。

[吴高阳, 2014] 吴高阳，*基于NIO的Java高性能网络技术的研究与应用*，西安电子科技大学，2014。

**[吴宪国等, 2015]** 吴宪国，鲁洋均，*一种基于Zookeeper实现Leader选举的方法，*2015。

**[武雪芳等, 2012]** 武雪芳, 王英杰，网络游戏服务器并发架构的设计与研究，*华章*，2012(10)。

[夏玲等, 2009] 夏玲，王伟平，客户端与服务器端的Socket通信，*电脑知识与技术*，2009(4):49-51。

[晓涵, 2017] 晓涵，HTTP协议揭秘，*计算机与网络*，2017，43(z1)。

[许登元等, 2014] 许登元，王鹏，分布式实时游戏服务器集群架构设计，*微电子学与计算机*，2014(7):105-108。

**[严谦等,2016]** 严谦，阳泳，网络编程tcp/ip协议与socket论述，*电子世界*，2016(8):68-68。

[杨建禹, 2013] 杨建禹，*一种封装与解析二进制协议数据的方法和装置: CN*，2013。

**[杨玲, 2010]** 杨玲，一种高性能网络游戏服务器架构设计，*网络安全技术与应用*，2010(4):59-61。

**[姚晓芳等, 2017]** 姚晓芳，舒小松，基于Java Socket网络编程的基础性应用研究，*无线互联科技*，2017(22):32-33。

**[张艳军等, 2016]** 张艳军，王剑，叶晓平等，基于 Netty 框架的高性能 RPC 通信系统的设计与实现，*工业控制计算机*，2016 (5): 11-12。

# 致 谢

转眼之间我的研究生生涯就要结束了，对南大有一丝丝不舍，对未来有一丝丝憧憬和期待，在此我要向南大的研究生老师们和同学们表达最真挚的感谢。

首先我要感谢我的毕业论文指导老师申富饶教授，感谢申老师对我毕业论文的指导和修正。毕业论文开题前，申老师与我就毕业论文事宜进行了详谈，给我毕业论文的选题和方向做出了指导和建议，在论文的写作过程中，申老师一直耐心地给我写作意见，申老师对毕业论文负责任的态度，给了我莫大的帮助。

其次我要感谢南京大学软件学院的所有老师们，感谢老师们严谨的教学态度和学术态度，帮助我培养了良好的学习习惯和学习态度。还要感谢周围同学们在我学习生活中给予的帮助，给了我继续前进的动力。

再次我要感谢我的家人，他们在我最困难的时候默默地支持和鼓励着我，让我战胜了困难，最终进入南京大学就读研究生，实现了自己的梦想。

最后，由衷地感谢在百忙之中评阅论文的各位专家和教授。

# 版权及论文原创性说明

任何收存和保管本论文的单位和个人，未经作者本人授权，不得将本论文转借他人并复印、抄录、拍照或以任何方式传播，否则，引起有碍作者著作权益的问题，将可能承担法律责任。

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含其他个人或集体已经发表或撰写的作品成果。本文所引用的重要文献，均已在文中以明确方式标明。本声明的法律结果由本人承担。

作者签名：

日期： 年 月