上海交通大学

硕士学位论文

棋牌游戏平台服务端的设计与实现

姓名：吴兆定

申请学位级别：硕士

专业：信号与信息处理

指导教师：郑世宝

20070101

棋牌游戏平台服务端的设计与实现

摘 要

棋牌游戏平台是网络游戏中比较重要的一类，国内棋牌游戏开展广泛，有很多公司在棋牌游戏平台领域都取得了比较好的业绩，如腾讯和联众。但出于商业因素考虑，这些公司并不愿意公开其研究成果。服务端在棋牌游戏平台中占有比较重要的地位，本文致力于设计并实现一款棋牌游戏平台的服务端系统。

论文首先介绍了网络游戏的发展历程以及发展趋势，并介绍了网络游戏的分类，研究了网络游戏服务端所采用的主要技术及其发展趋势，分析了棋牌游戏平台的研究现状。

为了设计出一款棋牌游戏平台的服务端系统，论文对相关技术进行了研究，这些技术主要包括分布式技术、网络通信技术、多线程技术以及动态链接库技术。论文还对市面上最具代表性的 QQ 游戏和联众游戏进行了分析，这些平台的分析是我们设计的参考。

本文主要完成了一款棋牌游戏平台服务端的设计。文章从服务端架构、通信协议、多线程工作模型以及第三方游戏开发接口等方面分别进行了分析和设计，在此基础上给出了服务器的软件架构设计。

通过对服务端的测试和分析可知，本文提出的棋牌游戏平台的服务端系统，在实现了棋牌游戏平台基本功能的基础上，还提供了支持区域性运营的功能。另外，基于 DLL 的第三方游戏开发接口，方便了其他游戏开发者开发的游戏能接入本平台，增强了平台的开放性和

通用性。

论文在最后对全文进行了总结，并对未来棋牌游戏平台的发展趋势作了展望。

棋牌游戏健康益智，老少皆宜，玩家数量可观，本文提出的棋牌游戏平台的服务端，在理论和商业应用上都有一定的价值。

关键词：网络游戏，服务端，棋牌游戏平台，架构

DESIGN AND IMPLEMENTATION OF A SERVER SIDE IN PLATFORM FOR CHESS/CARD ONLINE GAMES

ABSTRACT

Platform for chess/card games is one important kind of online games. Especially in China, chess/card games are very popular, and several companies such as Tencent and Lianzhong are professional in developing platform for chess/card games, but they regard the technologies as commercial secret. Server side is very important for platform for chess/card games. A design of a server side of platform for chess/card games is described in this paper.

First, the development history and trend of online games is summarized, and also the sorts of online games are introduced. The main technologies of online games and the status quo of platform for chess/card games are discussed.

On purpose to design a server side of platform for chess/card games, we first do some research of the related technologies, containing distributed system, network, multiple thread and dynamic link library. We use “commview” tool to analyze QQ game and Lianzhong game, then we know the architecture of these platforms.

A design of server side of platform for chess/card games is described

in this paper. After designing the architecture, communication protocol, multi-thread model and third-part development interface of the server, we give the software architecture design.

We tested and analyzed the server side. From the result, we know that the server side provides local-area management, and the third-part development interface allows other developers to develop games for the platform.

Platform for chess/card games is an important kind of online games and it is very popular in China. The server side designee discussed in this paper has significant academic and commercial essentiality.

KEY WORDS: online game, server，platform for chess/card games, architecture

缩略语

B/S	Browser/Server	浏览器/服务器
C	Client	客户端
CN	Center Node	中心节点
CORBA	Common Object Request Broker Architecture	公用对象请求代理程序结构
C/S	Client/Server	客户端/服务端
DB	Database	数据库
DCOM	Distributed Component Object Model	分布式组件对象模型
DLL	Dynamic Link Library	动态链接库
DNS	Domain Name Service	域名解析服务
FPS	First Person Shoot Game	第一人称射击游戏
FTP	File Transfer Protocol	文件传输协议
GS	Game Server	游戏服务器
HS	Hall Server	大厅服务器
HTTP	Hyper Text Transfer Protocol	超文本传输协议
IOCP	I/O Completion Port	I/O 完成端口
IS	Information Server	信息服务器
LN	Leaf Node	叶节点
LS	Login Server	登录服务器
Main DB	Main Database	主数据库
MMOACT	Massive Multiplayer Online Action Game	大型多人在线动作游戏
MMOAVG	Massive Multiplayer Online Adventure Game	大型多人在线冒险游戏
MMOG	Massive Multiplayer Online Game	大型多人在线游戏
MMORCG	Massive Multiplayer Online Race Game	大型多人在线赛车游戏
MMORPG	Massive Multiplayer Online RPG	大型多人在线角色扮演游戏
MMOSLG	Massive Multiplayer Online Simulation Game	大型多人在线策略游戏
MMOSPT	Massive Multiplayer Online Sport Game	大型多人在线运动类游戏
MS	Main Server	中央服务器
MUD	Multi-User Dungeon	多人地下城
NPC	Non Player Character	非玩家控制角色
P2P	Peer-to-Peer	对等网络
PE	Processing Element	自处理单元
PLATO	Programmed Logic for Automatic Teaching Operations	用于自动教学的可编程逻辑
RPG	Role Playing Game	角色扮演游戏
RSG	Region Server Group	区域服务器组
SSA	Scalable Star Architecture	扩展星型架构
SDCP	Star Distributed Communication Protocol	星型分布式通信协议
TCP	Transmission Control Protocol	传输控制协议

TDI	Third-part Development Interface	第三方游戏开发接口
UDP	User Data Protocol	用户数据报协议
UI	User Interface	用户界面
US	Update Server	更新服务器

上海交通大学
学位论文原创性声明

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，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：吴兆定

日期： 2007 年　2 月 27 日

上海交通大学
学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在___年解密后适用本授权书。

本学位论文属于

不保密□√。

（请在以上方框内打“√”）

学位论文作者签名：吴兆定　　指导教师签名：郑世宝

日期：2007 年　2 月 27 日　　日期：2007 年　2 月 27 日

第 1 章 综述

1.1 网络游戏概述

近年来，互联网的发展速度惊人，与之相关的技术也得到了飞速的发展。网络内容变得越来越丰富多彩，而且正成为人们日常生活中所不可或缺的一个组成部分。学习、工作、资讯、交友、娱乐等等，这些我们都可以在互联网上找到相应的内容。说到娱乐，就不得不谈到网络游戏。网络游戏，也就是人们常说的在线游戏，通过互联网，可以让身处各地的数以万计的玩家在同一个游戏世界里游戏和交流。自 2000 年中国引入第一款网络游戏《万王之王》以来，网络游戏在国内得到了飞速的发展。据有关机构统计，2005 年，中国的网络游戏产业规模已经突破 60 亿人民币，网络游戏运营商达到 300 个之多，网络游戏玩家更是达到 2550 万，占到中国网民数量的 20%[1]。据新浪游戏统计，至 2006 年 11 月，国内网络上正在运营的和已经运营失败的网络游戏已经达到 310 款（其中 59 款停止运营）[2]。而且，这些数字仍然在迅速增长。正因为如此，很多人都瞄准了中国网络游戏这个巨大市场，数以千计的游戏开发团队，数以百计的网络游戏运营公司，在中国这块肥沃的土地上正迅速生根。

1.1.1 网络游戏的发展历史[3]

从 1969 年诞生第一款真正意义上的网络游戏以来，网络游戏已经发展了近 40 年，在这 40 年的历程中，网络游戏的发展经历了三个重要的阶段。

1.1.1.1 第一代网络游戏（1969-1977）

第一代网络游戏实际上是网络游戏起初的试验阶段，主要存在于欧美高等院校的大型主机上。这一阶段的网络游戏通常都不能模拟一个持续发展的游戏世界，往往在服务器重启后，相关游戏信息就丢失了。而且，由于当时的计算机软硬件标准都没有统一，各个网络游戏平台使用的操作系统和编程语言也不一样，因而游戏无法进行跨系统运行。

第一款真正意义上的网络游戏是瑞克·布罗米在 1969 年为 PLATO（Programmed Logic for Automatic Teaching Operations）系统编写的《太空大战》。实际上，PLATO 原本只是一套远程教学系统，由于 PLATO 是第一套分时共享系统，而且一般都运行在大型主机上，在当时具有很强的处理能力和储存能力，因而之后的几年内，PLATO 平台上出现了好几款类似的网络游戏，PLATO 成了早期网络游戏的温床。

1.1.1.2 第二代网络游戏（1978-1995）

第二代网络游戏的标志是，专业的游戏开发商和发行商开始涉足网络游戏，他们将网络游戏从试验平台带入了商业运作。第二代网络游戏出现了“可持续性”的概念，玩家的角色可以持续不断地在游戏世界中发展，而不像第一代网络游戏中那样会丢失游戏信息。另外，这一代的网络游戏实现了跨系统运行，只要有兼容机和入网设备，玩家就能接入网络上的任何一款网络游戏。由于从试验平台进入了商业运作，因而在这一代网络游戏中，正式出现了收费运营的概念，网络游戏进入了收费时代。

第二代网络游戏的标志性作品是英国埃塞克斯大学的罗伊·特鲁布肖在 1978 年编写的《MUD1》，这是世界上第一款 MUD（Multi-User Dungeon，多人地下城）游戏，同时也是世界上第一款真正意义上的实时多人交互网络游戏。1991 年，诞生了世界上第一个网络游戏的服务平台“The Sierra Network”，这实际上就是目前国内流行的棋牌游戏平台的前身。随后，MPG-Net、TEN、Engage 和 Mplayer 等一批网络游戏专用平台相继出现。

1.1.1.3 第三代网络游戏（1996 至今）

第三代网络游戏的特点是，越来越多的开发商和运营商进入网络游戏，形成了一个规模庞大的产业。同时，MMOG（Massive Multi-player Online Game，大型在线游戏）的概念开始出现，人们并不满足于单一的服务平台，网络游戏开始朝大规模、全球性的方向发展。

第三代网络游戏的标志性作品是 1997 年的《网络创世纪》，这部作品一经推出后，在线人数很快就突破了 10 万大关。这部经典作品也被视作网络游戏的真正奠基石，因为从游戏内容上来讲，随后的网络游戏作品，充其量也就是在《网络创世纪》的同一层面上发展，并没有取得实质性的突破。《网络创世纪》的成功加速了网络游戏产业链的形成，一些传统的顶级专业单机游戏公司也开始加入网络游戏的开发，网络游戏市场迅速膨胀。这些顶级公司带来了很多经典作品，最著名的莫过于暴雪娱乐的《魔兽世界》。另外，在这一阶段，涌现出了大量中小游戏开发团队，他们为网络游戏市场创造了更丰富的游戏作品。

1.1.1.4 网络游戏发展趋势

网络游戏的发展，包括了游戏内容和游戏技术两方面的发展。游戏内容方面，将朝着多样化的方向发展。除了休闲游戏、MMORPG（Massive Multiplayer Online Role Playing Game，大型角色扮演网络游戏），将会出现更多的游戏形式，不过，提供更真实的游戏环境将是一个趋势。在游戏世界中，将会加入大量的虚拟社区的内容，将来的游戏将更加强调交互方式的多样化。网络游戏技术的发展，包括

两个方面的发展，一方面，客户端做得更加逼真和绚丽，另一方面，服务端将提供更多用户、更大规模、更稳定高效的服务。

近来很多新兴行业得到了异常迅猛的发展，比如 IPTV 和移动通信。IPTV 的发展，为电视开展更多增值业务提供了契机，相信将来 IPTV 上的网络游戏会再现当年红白机时代的疯狂。随着 3G 移动通信网络的发展，手机应用也变得越来越丰富，手机游戏已经不仅仅停留在单机形式上，由于手机的便携性，网络游戏在手机上的应用，必将是一个趋势。另外，一些专业的游戏机，也开始加入了网络功能，例如微软的 XBOX、Sony 的 Play Station、任天堂的 Wii，网络游戏必然也会在这些平台上大放异彩。

1.1.2 网络游戏的分类

1.1.2.1 网络游戏的类别

网络游戏的分类往往有很多种标准，比如可以按照单机游戏的分类来对网络游戏进行分类：动作类（MMOACT，Massive Multiplayer Online Action Game）、冒险类（MMOAVG，Massive Multiplayer Online Adventure Game）、运动类（MMOSPT，Massive Multiplayer Online Sport Game）、赛车类（MMORCG，Massive Multiplayer Online Race Game）、角色扮演类（MMORPG）、策略类（MMOSLG，Massive Multiplayer Online Simulation Game）、棋牌类等等[4]。

实际上，就目前国内运营的网络游戏来讲，市面上流行的网络游戏大致可分为三类。第一种是最常见的 MMORPG，即多人在线角色扮演游戏。这类游戏以《传奇》系列、《魔兽世界》等为代表，具有很高的人气和相当的粘着力。第二种是休闲对战游戏，以《泡泡堂》、《疯狂坦克》等为代表。这类游戏形式上比较类似，都是几个玩家（一般少于 8 个）在小房间内分组游戏，一分高下，但各种游戏的游戏方式存在着较大差异，有回合制，也有及时制。第三种是棋牌游戏平台，以《联众游戏》、《QQ 游戏》等为代表。棋牌游戏平台将民间流行的棋牌游戏集成到平台之上，一般一个平台上都有几十个不同的小游戏，玩家登录平台后，可以选择平台上的任何游戏进行娱乐，常见的有斗地主、拖拉机、麻将、象棋、围棋等等。由于这些棋牌游戏在中国有着悠久的历史且在民间相当流行，而且老少皆宜，健康益智，因而这类游戏在国内外华人中拥有相当的受众群。

1.1.2.2 各类网络游戏服务端的特点

对于 MMORPG、休闲类游戏和棋牌游戏平台这三种主流网络游戏种类，从技术角度来看，MMORPG 的服务端架构最为复杂。MMORPG 由于游戏世界相当之大，游戏人数相当之多，往往需要对服务端进行地区划分和功能划分。地区划分是指根据游戏地图，将游戏世界分成多个区，不同的游戏服务器组负责不同

地区的游戏逻辑处理。功能划分是指设置不同功能的游戏服务器，比如专门负责NPC 行动的 NPC 服务器，专门负责聊天的聊天服务器，专门负责商品交易的交易服务器，等等。要将这些服务器集合起来并相互交互，是相当复杂的。因而一般来讲，MMORPG 的一套服务器组所承受的玩家数量并不是很多，人数通常在2、3 千人。所以 MMORPG 通常会存在着多个服务器组，而且每个服务器组之间是相互独立的。

休闲类游戏的架构相对而言是最简单的。一方面，休闲类游戏的游戏逻辑规则一般都比较简单，不像 MMORPG 那样复杂，而且也不存在庞大的游戏世界；另一方面，休闲游戏的在线人数相对于棋牌游戏平台来说比较少。运营很火的休闲游戏，可以通过增加服务器组来解决负载问题，而且这些服务器组之间不像棋牌游戏平台的服务器组那样需要相互通信。

棋牌游戏平台由于需要支持大用户量和多种不同游戏，往往架构上会比休闲游戏复杂一点。但从另一方面来讲，棋牌游戏平台中的单个游戏，逻辑往往都比较简单，而且“大厅-房间-游戏桌”的结构设计也比较规范和成熟，这在一定程度上降低了开发难度。

1.2 网络游戏服务端的相关技术概况

网络游戏通常由 4 部分组成[5]：客户端、网络层、服务端和账号管理。账号管理已经非常统一化了，通常都是通过 Web 页面来进行账号的注册和管理，而且和整个游戏内容的关系并不是很大。网络游戏的客户端，最重要的技术应该是画面显示了，从文字到 2D 再到 3D，客户端的发展令人振奋。而对于网络层和服务端，要想让网络游戏承载更多的玩家，那么这两个方面都是非常重要的。

1.2.1 网络游戏的常用架构

网络游戏的架构是指网络游戏的客户端和服务端连接的一种拓扑结构。在网络游戏中经常采用的架构有：C/S 架构、B/S 架构和 P2P 架构。

C/S 架构即客户端/服务端架构，是最常用的网络应用架构之一，这种架构将网络游戏分为客户端和服务端两个部分，在客户端需要有专门的客户端游戏程序，在服务端也需要有专门的服务端游戏程序，服务端通常由高性能的工作站来担任。目前市面上绝大部分网络游戏采用的都是 C/S 架构，C/S 架构也是目前研究最为成熟的网络应用架构，不但在网络游戏行业得到了应用，在其他各行各业中，只要使用网络服务，绝大部分采用的都是 C/S 架构。

所谓 B/S 架构，即浏览器/服务器架构，是随着网络浏览器的发展而产生的一种新型的网络应用架构。这种架构与 C/S 架构最主要的区别就是客户端是通过

网络浏览器实现的，不需要专门的游戏程序。目前市面上也存在着好几款 B/S 架构的网络游戏，这类游戏的客户端程序通常都比较简单，而且通常都是通过 Flash 和 Java 直接在浏览器中实现。有关 B/S 架构和 C/S 架构的研究已经非常多了，具体可以参考文献[6]。

P2P 是 Peer-to-Peer 的简写，亦即对等网络。P2P 是目前互联网上比较流行的一项技术，著名的 BT，以及诸多在线视频直播服务，都是通过 P2P 技术实现的。P2P 技术最大的特点就是使得任何网络设备都可以为其它网络设备服务，淡化了服务端的概念，解决了网络应用中服务端瓶颈的问题。目前已经有相关基于 P2P 技术的网络游戏的研究，如文献[7][8][9]。

由于网络游戏是一个专用的网络应用程序，客户端技术复杂，服务端技术更加需要分布式来实现，因而绝大部分网络游戏采用了 C/S 架构，如参考文献[10][11][12][13]中所提到的架构，基本上都是基于 C/S 架构。采用分布式技术实现的服务端，需要有优秀的服务端架构，各个功能服务器的负载要均衡。分布式系统的负载均衡研究也是一个比较热门的研究课题。参考文献[14]就对网络应用服务器的负载均衡进行了详细说明。

1.2.2 网络游戏的通信技术

网络游戏是一种超大用户量的网络应用程序，一般网络游戏的在线人数通常都达到十万数量级。面对这种超大用户量访问，除了要提供出色的服务端性能外，优秀的网络通信技术也是一个必不可少的部分。目前的服务端通常都是基于两种操作系统——Windows 和 Linux。Windows NT 平台为大用户量网络通信专门提供了高性能的 I/O 模型——IOCP（I/O Completion Port，I/O 完成端口）。一般架设在 Windows 环境下的网络游戏服务端都采用了这种 I/O 模型。参考文献[15]详细介绍了 IOCP 的相关内容。Linux 从 2.6 内核开始，也提供了一种高效的 I/O 模型——Epoll。参考文献[16][17]详细介绍了 Epoll 的有关内容及其使用方法，而参考文献[18]则介绍了 Epoll 技术在网络游戏中的使用实例。

为了进一步优化网络游戏的网络性能，业界在网络性能优化方面做了很多研究工作。比如文献[19]就通过分析几款流行的网络游戏，提出了网络游戏的网络流量是可以预测的观点，并就此提出了一些优化方案。参考文献[20]通过对《Quake 3》服务器的长时间实验，提出了网络延时在 FPS（First Person Shoot Game，第一人称射击游戏）网络游戏中的重要性。另外，文献[21][22][23][24]都对网络游戏的网络性能优化进行了讨论。

在 P2P 架构中，往往会用到多播技术。多播是一种把信息同时传递给一组目的地址的技术。对于一对多的数据传输，多播要比单播更为高效。与单播不同，多播仅发送数据的一个副本[25]。在网络游戏中，经常需要进行一对多的广播通信，

因而多播技术在网络游戏中也得到了应用，比如文献[10][26]中提到的网络游戏就使用了多播技术。

1.2.3 网络游戏服务端的多线程模型

网络游戏的服务端程序，通常都由多个线程组成。因为网络游戏的服务端不但要不停地接收和发送网络消息，还需要对这些网络消息进行处理。所以，一个基本的网络游戏服务端，通常都包含 2 个基本的线程，一个用来收发网络消息，另一个用来处理网络消息。

为了进一步提高网络游戏服务端的性能，很多网络游戏的服务端都采用了多线程技术来处理网络消息。比如，为每一个玩家都建立一个专门的线程，由这个线程来处理来自此玩家的网络消息。采取多线程处理网络消息，可以充分利用 CPU 资源，从而达到提高服务端性能的目的。不过，使用多线程同样带来了同步和互斥的问题，这就需要做出额外的工作来实现多线程同步与互斥的功能。因而，应该在多线程和线程工作复杂度之间找到一个平衡点，从而更有效地提高服务端的性能。

由于网络游戏承担的用户数往往都比较多，因而当使用多线程来处理各个玩家时，为了避免频繁地创建和销毁线程，往往会采用线程池技术[27]。线程池的设计思想如下：在系统启动的时候初始化一定数量的线程，并将它们设置在空闲状态，以等待程序分配任务。当系统其他部分程序产生了需要执行的任务时，系统将任务封装为线程能运行的一个对象，并向线程池传递这个对象。线程池取得某个空闲的线程并运行这个任务，当线程完成这个任务后，返回线程池中，恢复为空闲状态，等待执行其他新任务[27]。

1.2.4 网络游戏服务端的技术发展趋势

网络游戏正朝着规模更大、稳定性更高、实时性更好的方向发展，因而网络游戏服务端的发展趋势将是扩展性更好、可靠性更高、更加高效实时。业界在网络游戏服务端方面的研究，主要也就是为了实现这样的目标。

随着网络游戏的规模越来越大，单一的服务器已经不能满足网络游戏服务端的要求，网络游戏服务端正向着集群化的方向发展。例如文献[13]就通过将游戏世界划分为多个区域，从而通过分布式技术来提高服务端的承载量，文献[29]则更加明确地阐述了集群化将是网络游戏服务端的发展趋势的观点。

网络通信方面，为了更好地提升网络游戏的网络性能，有关研究者已经提出了采用混合式的网络模型。例如文献[30]就在基于 P2P 的网络游戏中使用了 IOCP 技术，进一步加强了 P2P 网络游戏对大用户量的支持。而文献[10]中的网络游戏模型，则在 C/S 架构的基础上，在客户端之间加入了 P2P 架构，从而减轻了服务

端的负载。实际上，C/S 架构中服务端的分布式化，也要求服务端之间采取另外某种网络通信架构。网络通信方面，多种架构的混合使用将是网络游戏技术发展的另一个趋势。

实时性在网络游戏中是一项非常重要的指标。目前市面上的 MMOG 中，很少见到动作类的，这就是因为大用户量网络游戏往往实时性比较差，而动作类的网络游戏对实时性要求又非常高。因此，业界很多关于实时性的研究，使用的都是动作类游戏。比如文献[19]就研究了著名的 FPS 游戏《反恐精英》，文献[20]研究的也是著名的 FPS 游戏《Quake 3》。FPS 游戏对实时性要求应该说是最高的，如果能在 FPS 游戏中实现超大规模运营，那么这种实时性应该能满足绝大部分网络游戏的需求。

网络游戏服务端还有一个发展趋势就是更通用化。前面也曾提到，现在网络游戏已经不仅仅局限在 PC 机平台了，IPTV、手机、专用游戏机等都开始涉足网络游戏，照着这种趋势发展下去，极有可能出现多种客户端平台在同一个游戏服务端进行游戏的情况，这就要求网络游戏的服务端能够通用地支持多种客户端平台。参考文献[31]就提出了一套在 UBI 环境下用于多客户端平台的网络游戏的 3D 引擎、服务端以及中间件。

1.3 棋牌游戏平台的研究现状

由于棋牌游戏平台在国外比较少见，因而国外基本没有棋牌游戏平台技术的相关研究和应用。现在国外网络游戏的研究热点在于 MMOG，特别是超大规模的 MMORPG 和 FPS 游戏。国内就不一样了，棋牌游戏平台在国内运营得非常成功，特别是腾讯和联众这两家公司，他们在棋牌游戏平台方面取得了比较好的业绩，但可惜的是，他们并不公布有关棋牌游戏平台的研究成果，这对整个棋牌游戏平台的发展都是不利的。参考文献[32][33]是国内有关棋牌游戏平台方面不多的研究论文，这两篇论文的作者虽然各自实现了一两款棋牌类网络游戏，但并没有实现真正意义上的棋牌游戏平台。

棋牌游戏平台有一种向通用游戏平台演变的趋势，也就是说，以后可能会出现一个通用的游戏平台，上面除了棋牌游戏外，还有别的网络游戏。目前 QQ 游戏平台上，已经集成了棋牌游戏之外的网络游戏，如 MMORPG《QQ 幻想》、《凯旋》等，休闲游戏《QQ 音速》、《飘飘》等[34]。相信棋牌游戏平台还会集成越来越多的非棋牌类网络游戏，从而成为一个功能强大的网络游戏平台。

1.4 主要研究内容和论文组织

本课题是基于上海高清数字科技有限公司数字娱乐部的棋牌游戏平台项目，

该项目的目标是研究并开发一款棋牌游戏平台，要求此棋牌游戏平台能够有较好的扩展性、较高的性能，能实现区域性运营的功能，同时能提供一套通用的棋牌游戏开发接口，以支持第三方开发的棋牌游戏。

在其他棋牌游戏平台公司没有公开相关技术的前提下，我们自主研发了一款棋牌游戏平台并成功投入商业运作。作者在该项目组中主要负责了棋牌游戏平台服务端系统的设计与实现，解决了该服务端的架构设计、多线程工作模型设计、软件模块设计等问题，并参与解决了通信协议设计、第三方游戏开发接口设计等相关问题，同时负责绝大部分服务端功能的编码实现。

本文主要创新工作是完成一款棋牌游戏平台服务端的设计与实现。本文提出的棋牌游戏平台，与市面上已有棋牌游戏平台最大的不同点是提供了区域性运营的功能。所谓区域性运营，是指在不同区域架设多个服务器组，而且不同服务器组可以具有自己的运营策略。之所以提供区域性运营的功能，是因为棋牌游戏规则往往具有很强的本地化特点，如果能做到当地服务器组面向当地玩家，就能充分利用网络资源，提升游戏性能；另一方面，区域性运营可以方便运营商对不同区域设置不同的运营策略，从而实现运营多样化。本文还将游戏房间和其他功能服务器设计成独立的进程，不但增强了服务器配置的灵活性以及各功能服务器之间的独立性，还利用了多进程工作时效率高的特点，提高了棋牌游戏平台服务端的整体性能。本文还提供了基于 DLL 的第三方游戏开发接口，方便其他游戏开发者开发的棋牌游戏能接入平台，增强了平台的开放性和通用性。

棋牌游戏平台是网络游戏中比较重要的一类，特别对于国内来讲，棋牌游戏开展广泛，健康益智，老少皆宜，玩家数量相当可观，因而本文提出的棋牌游戏平台的服务端，在商业上也有一定的应用价值。

本文将在第 2 章介绍设计棋牌游戏平台服务端的相关理论基础，同时对市面上流行的棋牌游戏平台进行了实例分析。第 3 章给出本棋牌游戏平台服务端的详细设计与实现。第 4 章对本文提出的棋牌游戏平台的服务端进行了实际测试和分析。第 5 章对整篇文章进行了总结并对未来进行了展望。

第 2 章 棋牌游戏平台服务端设计关键技术与应用实例研究

2.1 棋牌游戏平台服务端设计关键技术研究

2.1.1 分布式技术[35]

随着对计算速度、系统可靠性和成本实效性要求的增加，传统的冯·诺依曼型结构的计算机已经不能满足需求，而计算机网络的出现，使得一个新的梦想成为现实——分布式计算。当用户需要完成任何任务时，分布式计算提供对尽可能多的计算机能力和数据的透明访问，同时实现高性能与高可靠性的目标。

目前，还没有对分布式的准确定义，但下面这段话可以描述出分布式的含义：如果一个系统的部件局限在一个地方，它就是集中式的；如果它的部件在不同地方，部件之间要么不存在或仅存在有限的合作，要么存在紧密的合作，它是分散式的。当一个分散式系统不存在或仅存在有限的合作时，它就被称做网络的；否则它就被称做分布式的，表示在不同地方的部件之间存在紧密的合作。

文献[35]中是这样定义分布式系统的：一个分布式系统是一个对用户看起来像普通系统，然而运行在一系列自治处理单元（PE，Processing Element）上的系统，每个 PE 有各自的物理存储器空间并且信息传输延迟不能忽略不计。在这些 P E 间有紧密的合作。系统必须支持任意数量的进程和 PE 的动态扩展。

简言之，分布式系统就是一个由许多独立单元紧密合作而成的、对外界透明的可动态扩展的系统。分布式系统最重要的一个概念就是分布式系统的静态网络拓扑结构。一般来讲，分布式的静态网络拓扑结构可以分为以下几种：

- 全连接网络：每个处理单元和系统中的其他所有处理单元直接相连，如图 2-1g。
- 直线型和环型：所有的处理单元都排列在一条直线上且相邻节点相连的就是直线型（图 2-1a），如果两个边界节点相连，就形成了环型（图 2-1b）。
- 网格和圆环：一个 k 元，n 维，有 N=kn 个节点的网格，其内部节点度数为 2n，网络直径为 k(n-1)。每个节点有一个地址（un，un-1，□，u1），其中 1≤ui≤k。如果两个节点（un，un-1，□，u1）和（vn，vn-1，□，v1）的地址中有且仅有一个元（维）不同，比如说第 i 维，并且|ui－vi |=1，则它们相互连接。基本上，沿着每一维的节点连成直线型。如果沿着每一维的节点连成环型，则网络形成一个 n 维的圆环。2 维和 3 维网格是最流行的，许多并行计算机都是基于它们构造的。圆环是带回绕连接的网格。图 2-1e 和图 2-1f 分别表示 2 维圆环和 2 维网格。
- 树型和星型：任意两个顶点间只存在唯一通路的图就是树，树中的节点

可以按层次排列（图 2-1c）。星型是两层的树，有一个高的节点度数（图 2-1d）。

- 超立方型：一个 n 维的超立方包含 N=2n 个节点，节点度数为 n，直径为 n。每个节点有一个 n 位的二进制地址。两个节点，当有且仅有一个位不同时，它们相互连接。二进制超立方在 8 0 年代得到普遍研究和发展。一个 n 维立方的节点度数和网络直径都是 n。实际上，节点度数随着维数的增加线性增大，使得超立方很难是一个可伸缩的结构。由于难以伸缩和包装更高维数的超立方，超立方结构逐渐被其他结构所取代了。但是，在超立方上设计的算法介于网络相关的算法和网络无关的算法之间。超立方算法的研究对于弥补使用 PRAM（并行随机访问机）的网络无关设计和使用网络模型的网络相关设计之间的差距仍然十分重要。图 2-1h 表示了一个 3 维立方。

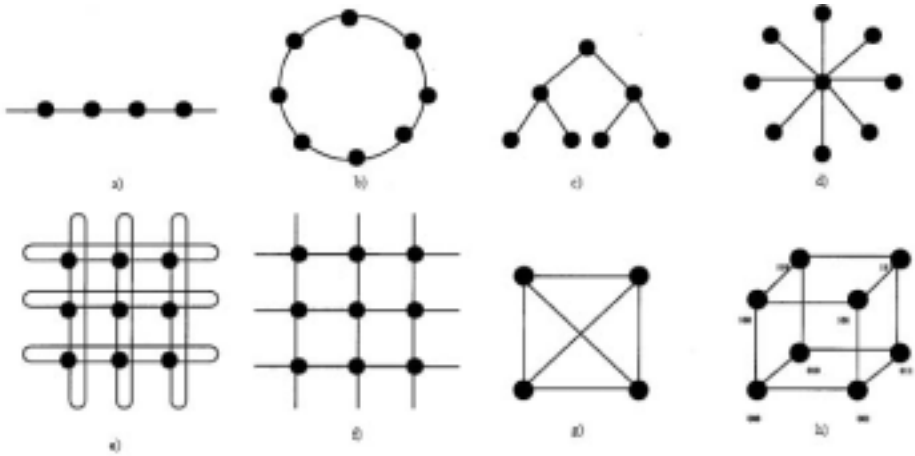

图 2-1 分布式系统的静态网络拓扑结构

Figure 2-1 Static architecture of distributed system

2.1.2 网络通信技术[15]

2.1.2.1 Winsock 基础

Winsock 是 Windows 平台下提供的开放的、支持多种协议的网络编程接口，是 Windows 网络编程的标准。Winsock 支持相当多的协议，比如网际协议（IP）既支持面向连接的流服务，又支持无连接的数据报服务。通常网络连接存在两种连接方式，一种是 TCP 连接，还有一种是 UDP 连接。TCP（Transmission Control Protocol，传输控制协议）是面向连接的协议，TCP 协议要求在进行数据通信前，通信双方首先要建立可靠的连接。TCP 协议建立在通信双方可靠连接的基础之上，通信过程中几乎不会发生丢包的情况，因而对可靠性要求较高的数据传输，可以采用 TCP 协议。但 TCP 协议在建立连接时，通信双方会产生一些额外的通信量，影响到了通信效率。UDP（User Data Protocol，用户数据报协议）是面向非连接的协议，正好与 TCP 相对立。UDP 不要求通信双方建立连接就直接把数

据包发送给对方，可能会出现丢包的情况，因而数据传送的可靠性不是很高，适用于实时通信、对可靠性要求不高的通信。由于 UDP 不需要在通信双方直接建立连接，因而通信效率要比 TCP 协议高。TCP 协议和 UDP 协议的特点可以总结为下表：

表 2-1 TCP 和 UDP 的比较

Table 2-1 Comparison of TCP and UDP

协议	连接特点	可靠性	传输效率	适用场合
TCP	面向连接	可靠	低	对可靠性要求高的大量数据传输
UDP	面向非连接	不可靠	高	对可靠性要求不高的实时数据传输

Winsock 是建立在套接字基础之上的。所谓套接字，就是一个指向传输提供者的句柄。Win32 中，套接字不同于文件描述符，它是一个独立的类型——SOCKET。网络通信双方，都必须使用 SOCKET 来实现网络通信。服务端和客户端是这样来建立连接的：首先服务端会调用 API 函数“bind”将一个 SOCKET 与本地 IP 地址和一个端口绑定，然后将该套接字设置为监听模式，这是通过调用“listen”函数来完成的，这样服务端就可以等待客户端发起连接了。客户端通过一个套接字调用“connect”函数向服务端发起连接，服务端这时会调用“accept”函数来接受客户端的连接请求，这样服务端与客户端之间就建立了连接，接下来就可以进行通信了。服务端和客户端都可以通过“send”函数来向对方发送消息，而通过“recv”函数来接收对方的消息。通信进行完毕后，可以通过调用“shutdown”函数或者“closesocket”函数来中断连接并释放相应资源，两者的区别是，前者是从容中断，而后者会造成数据丢失。服务端与客户端之间的整个通信过程可以表示为下图：

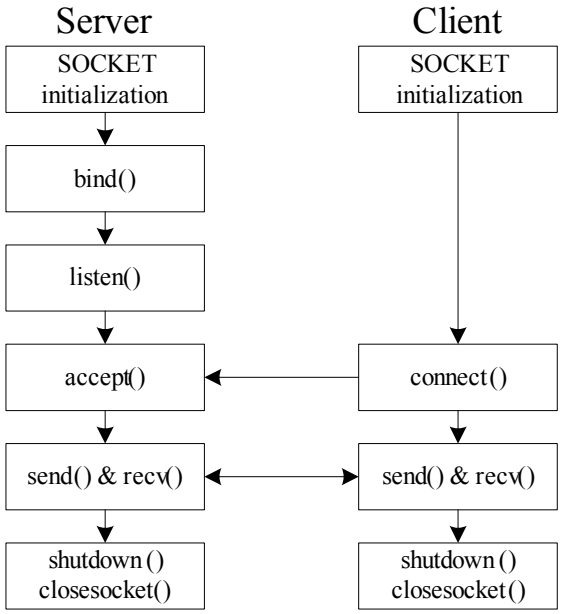

图 2-2 Winsock 通信过程

Figure 2-2 Process of communication using Winsock

2.1.2.2 Winsock I/O 模型

Winsock 主要是用来管理 Windows 应用程序中的 I/O 操作，I/O 模型则描述了一个应用程序如何对套接字上进行的 I/O 进行管理和处理。Winsock 提供了两种套接字模式——锁定和非锁定。在锁定模式下，在 I/O 操作完成前，执行操作的 Winsock 函数（比如 send 和 recv）会一直等候下去，不会立即返回程序（将控制权交还给程序）。而在非锁定模式下，Winsock 函数无论如何都会立即返回。锁定模式带来的问题是效率低下，因为 Winsock 函数在等待的时候系统不能做其他事情。非锁定模式存在的问题则是大多数情况下，请求的操作没法在规定的时间内完成，这样函数调用就会失败，并返回一个 WSAEWOULDBLOCK 错误。为了解决锁定模式和非锁定模式带来的问题，Winsock 提供了一些 I/O 模型来管理套接字上进行的通信。这些模型包括 select（选择）、WSAAsyncSelect（异步选择）、WSAEventSelect（事件选择）、Overlapped I/O（重叠式 I/O）以及 Completion port（完成端口）。

select（选择）模型是 Winsock 中最常见的 I/O 模型。之所以称其为“select 模型”，是由于它的“中心思想”便是利用 select 函数，实现对 I/O 的管理。它使那些想避免在套接字调用过程中被无辜“锁定”的应用程序，采取一种有序的方式，同时进行对多个套接字的管理。利用 select 函数，我们判断套接字上是否存在数据，或者能否向一个套接字写入数据。之所以要设计这个函数，唯一的目的便是防止应用程序在套接字处于锁定模式中时，在一次 I/O 绑定调用（如 send 或 recv）过程中，被迫进入“锁定”状态；同时防止在套接字处于非锁定模式中时，产生 WSAEWOULDBLOCK 错误。除非满足事先用参数规定的条件，否则 select 函数会在进行 I/O 操作时锁定。

Winsock 提供了一个有用的异步 I/O 模型。利用这个模型，应用程序可在一个套接字上，接收以 Windows 消息为基础的网络事件通知。具体的做法是在建好一个套接字后，调用 WSAAsyncSelect 函数。要想使用 WSAAsyncSelect 模型，在应用程序中，首先必须用 CreateWindow 函数创建一个窗口，再为该窗口提供一个窗口例程支持函数（Winproc）。亦可使用一个对话框，为其提供一个对话例程，而非窗口例程，因为对话框本质也是“窗口”。设置好窗口的框架后，便可开始创建套接字，并调用 WSAAsyncSelect 函数，打开窗口消息通知。

Winsock 提供了另一个有用的异步 I/O 模型。和 WSAAsyncSelect 模型类似的是，它也允许应用程序在一个或多个套接字上，接收以事件为基础的网络事件通知。该模型最主要的差别在于网络事件会投递至一个事件对象句柄，而非投递至一个窗口例程。

在 Winsock 中，相比我们迄今为止解释过的其他所有 I/O 模型，重叠 I/O（Overlapped I/O）模型使应用程序能达到更佳的系统性能。重叠模型的基本设

计原理便是让应用程序使用一个重叠的数据结构，一次投递一个或多个 Winsock I/O 请求。针对那些提交的请求，在它们完成之后，应用程序可为它们提供服务。

“完成端口”模型是迄今为止最为复杂的一种 I/O 模型。然而，假若一个应用程序同时需要管理为数众多的套接字，那么采用这种模型，往往可以达到最佳的系统性能。但不幸的是，该模型只适用于 Windows NT 和 Windows 2000 操作系统。因其设计的复杂性，只有在你的应用程序需要同时管理数百乃至上千个套接字的时候，而且希望随着系统内安装的 CPU 数量的增多，应用程序的性能也可以线性提升，才应考虑采用“完成端口”模型。要记住的一个基本准则是，假如要为 Windows NT 或 Windows 2000 开发高性能的服务器应用，同时希望为大量套接字 I/O 请求提供服务（Web 服务器便是这方面的典型例子），那么 I/O 完成端口模型便是最佳选择。从本质上说，完成端口模型要求我们创建一个 Win32 完成端口对象，通过指定数量的线程，对重叠 I/O 请求进行管理，以便为已经完成的重叠 I/O 请求提供服务。

有关 I/O 模型更详细的介绍，可以参考文献[15]。每种模型都有自己的优点和缺点，如何选用这些 I/O 模型呢？若打算开发一个客户端应用，令其同时管理一个或多个套接字，那么建议采用重叠 I/O 或 WSAEventSelect 模型，以便在一定程度上提升性能。然而，假如开发的是一个以 Windows 为基础的应用程序，要进行窗口消息的管理，那么 WSAAsyncSelect 模型恐怕是一种最好的选择，因为 WSAAsyncSelect 本身便是从 Windows 消息模型借鉴来的。若采用这种模型，我们的程序一开始便具备了处理消息的能力。若开发的是一个服务器应用，要在一个给定的时间，同时控制几个套接字，建议采用重叠 I/O 模型，这同样是从性能出发点考虑的。但是，如果预计到自己的服务器在任何给定的时间，都会为大量 I/O 请求提供服务，便应考虑使用 I/O 完成端口模型，从而获得更好的性能。

2.1.3 多线程技术[36]

进程是程序的执行状态，也就是说，一旦程序被装载到了内存中并准备执行时，它就是一个进程。进程通常由文本、数据和堆栈以及它自己的资源组成。资源可以是文件、对象句柄、设备、信号量、互斥量、管道等等。进程中一个重要的概念就是进程控制块。进程控制块用来保存大量与进程相关的信息，操作系统通过它来管理进程以及它们的资源。进程状态是进程某时某刻所处的模式和条件，进程的状态决定了将来的事件以及进程可能进入的状态。进程的状态通常包括就绪（Ready）、运行（Running）和阻塞（Blocked）。进程状态之间的转换可以表示为图 2-3。

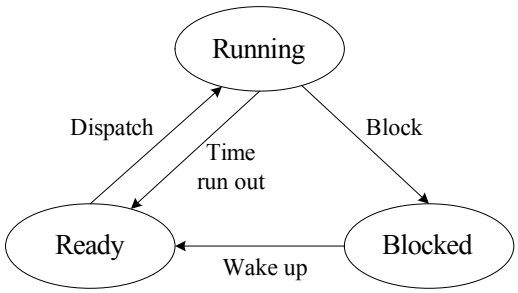

图 2-3 进程状态转换关系

Figure 2-3 Conversion between different states of process

线程是一种轻量级进程，与进程相比，线程给操作系统带来的创建、维护和管理负担要轻，因为与线程相关的信息非常少，因而线程的代价或开销要比进程少。进程中可以包含一个或者多个线程，而且进程与它的线程共享它的一切资源。进程是资源管理的单位，而线程是系统调度的单位，线程没有自己的地址空间。所有的线程都有一个线程 ID、定义线程状态的寄存器组、优先权以及堆栈。每个进程至少包含一个线程，称作主线程，实际上就是进程本身的一个控制流程。如果进程包含两个以上的线程，那么此进程称为多线程进程。无论多进程还是多线程，它们的目的都是用来实现并行处理，从而提供任务的异步执行，以提高效率。

与使用多进程相比，使用多线程来管理一个应用程序的子任务有各种不同的优点。当创建一个进程时，系统就创建了一个主线程。主线程为该进程的执行线程，这可能是实施进程功能的唯一所需线程。但是如果进程有许多并发子任务，多线程就可以提供子任务的异步执行。这种方式对上下文切换所花的开销较少。并发多进程需要上下文切换的开销则比较大。多线程之间并不需要特殊的通信机制，线程可以轻易地在任务间传递与接收数据，这也节省了系统资源。如果使用多进程，则需要使用额外的通信机制来实现进程间的通信，比如有消息通信机制、共享内存机制以及管道机制等等。

另一方面，多线程也存在着一些缺点。虽然线程间不需要特殊的机制来通信，但线程需要同步并发访问内存，这就带了了内存访问的同步与互斥问题。由于进程是孤立的，如果一个进程创建了不良数据，这一数据之限制与该进程，而多线程的话则可能产生影响其他线程的不良数据。线程导致的错误比进程产生的错误付出的代价更大。线程导致的访问违规可能导致整个进程的终止，线程可能产生影响所有线程整个内存空间的数据错误。

2.1.4 动态链接库技术[37]

动态链接库（Dynamic Link Library，DLL）技术是一种比传统静态链接更为现代的方法，它的优点是可执行文件的体积可以非常小。虽然由于运行时需要载

入函数库，致使运行的速度稍慢一些，但动态链接能够更加有效地利用磁盘空间，而且链接编辑阶段的时间也会缩短（因为链接器的有些工作被推迟到载入时）。

尽管单个可执行文件的启动速度稍受影响，但动态链接可以从两个方面提高性能：

1. 动态链接可执行文件比功能相同的静态链接可执行文件的体积小。它能够节省磁盘空间和虚拟内存，因为函数库只有在需要时才被映射到进程中。以前，避免把函数库的拷贝绑定到每个可知性问卷的唯一方法就是把服务置于内核中而不是函数库中，这就带来了可怕的“内核膨胀”问题。

2. 所有动态链接到某个特定函数库的可执行文件在运行时共享该函数库的一个单独拷贝。操作系统内核保证映射到内存中的函数库可以被所有使用它们的进程共享。这就提供了更好的 I/O 和交换空间利用率，节省了物理内存，从而提高了系统的整体性能。如果可执行文件是静态链接的，每个文件都将拥有一份函数库的拷贝，显然极为浪费。

动态链接使得函数库的版本升级更为容易。新的函数库可以随时发布，只要安装到系统中，旧的程序就能够自动获得新版本函数库的优点而无需重新链接。另外，动态链接还允许用户在运行时选择需要执行的函数库。这就是为了提高速度或提高内存使用效率或包含额外的调试信息而创建新版本的函数库是完全可能的，用户可以根据自己的喜好，在程序执行时用一个库文件取代另一个库文件。

2.2 棋牌游戏平台服务端应用实例研究

本小节将分别对市面最流行的两款棋牌游戏平台——QQ 游戏和联众游戏的服务端进行分析。QQ 游戏是目前最成功的棋牌游戏平台，而联众是运营时间最长的棋牌游戏平台。QQ 平台的游戏人数达到惊人的 300 万，而联众游戏的在线人数也有 50 万之多，没有优秀的服务端架构，不可能做到这样的成绩。由于无法获得这些棋牌游戏平台的相关技术文档，而我们的研究采用的是封包检测的方法，研究结论带有一定的主观判断，可能不是十分准确。很多软件都可以用来检测网络数据包，这里使用的是 CommView。该软件的界面如下图所示：

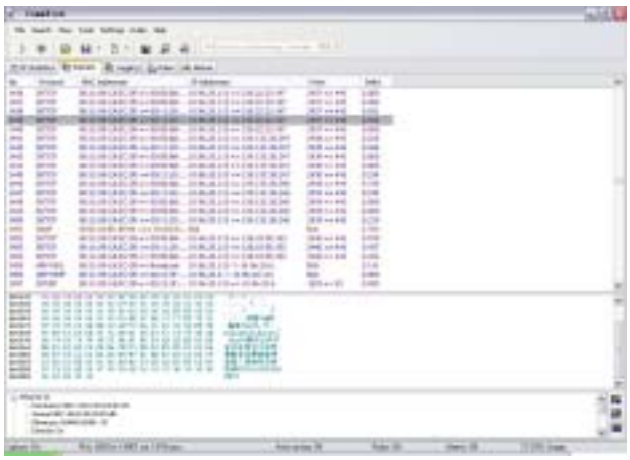

图 2-4 网络数据包检测软件 CommView 界面

Figure 2-4 UI of CommView

由图中可以看出，该软件能够检测到网络上传送的各种数据包，并能显示该数据包来自或者发送至哪个 IP 哪个端口，使用的是什么网络协议，甚至数据包的内容都可以查看。使用该软件，可以检测出游戏中某种特定的操作下，游戏客户端与游戏服务端之间会有怎样的通信，具体会与哪些服务器的哪些 IP 和端口通信。通过这样的分析，就能简单判断出服务端的架构，比如哪些服务器用于登录验证，哪些服务器用作游戏服务器，游戏服务器又怎样部署在物理服务器上，等等。

在下面的章节中，我们将借助对 QQ 游戏和联众游戏的封包检测，试图从服务端架构、网络通信协议、多线程模型和游戏接入形式等方面给出对这两款棋牌游戏平台的分析。

2.2.1 QQ 棋牌游戏平台分析

图 2-5 QQ 游戏大厅[34]

Figure 2-5 Hall of QQ game [34]

◆ **服务端架构分析**

为了分析 QQ 游戏的服务端架构，我们用 commview 监测了 QQ 游戏的整个游戏过程。典型棋牌游戏平台的游戏流程大致如下：

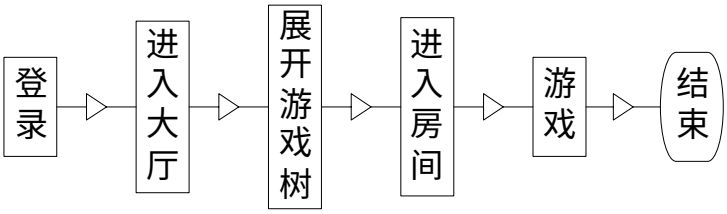

图 2-6 典型棋牌游戏平台的游戏流程

Figure 2-6 Typical gaming flow of chess/card game platform

在登录的时候，经多次试验，我们发现客户端会与某个固定的服务器进行通信，可以确定这个服务器是登录服务器，因为该服务器 IP 和游戏安装目录下的配置文件 Dirconfig.ini 中的 IP 是对应的，选择不同区域登录就会与不同的登录服务器进行通信。登录成功后，就会收到大量来自十几个不同 IP 的数据包，这些数据包应该是在向客户端传递一些大厅显示所必需的信息，比如大厅中有哪些游戏，每个游戏目前有多少玩家，当然还有很多广告信息。当展开大厅中的游戏树时，会收到服务端的数据包。这时应该是服务端将游戏服务器的一些信息传给客户端，比如此游戏下游戏服务器的连接信息、人数信息等等。经过几次试验，发现这些数据包来自不同的几个 IP，可见这些信息的提供是由多个服务器完成，一个服务器负责特定一部分游戏的信息统计。在游戏树中可以选择进入哪个房间，点击想要进入的房间时，客户端会与某个 IP 进行通信，这就是游戏服务器。检测中发现，并不是一个游戏房间就占用一台物理服务器，QQ 是根据游戏的负载情况来合理搭配房间数与物理服务器之间的关系的。比如，麻将这款游戏玩家比较多，在几乎每个房间都人满的情况下（400 人满），有 15 个房间的 IP 和端口是一样的，这说明这 15 个房间部署在同一台物理服务器上。每个房间 400 人，15 个房间就是 6000 人！据网上有关文章介绍，一般的游戏服务器的承载力基本是 2000 到 5000 人，可见 QQ 游戏做得是相当优秀的。再比如一款不是很热的游戏，即升级游戏的怀旧区。虽然客户端看来，怀旧区下面分了两个大区，每个大区下都有 25 个房间，但由于游戏人数少，总共只有 100 人左右，实际上这 50 个房间竟然被 QQ 同时部署在了同一台服务器上 !如果所有人都进入这 50 个房间，那么这台服务器肯定崩溃，因为性能再好的服务器也不能同时承受 20000 个玩家。

综合上面的监测可以推断，QQ 游戏中最少存在三类功能服务器，一种是用于登录验证的登录服务器，一种是用于提供游戏信息的信息服务器，还有一种就是提供游戏功能的游戏服务器。登录服务器被部署在多个区域，以方便不同地区

玩家快速登录。由于平台内游戏服务器很多，QQ 设置了多台信息服务器，每台信息服务器负责平台中的一部分游戏服务器的信息。对于游戏服务器，客户端体现出来的房间在服务端仅仅是一个逻辑上的划分，每个游戏服务器就是一个进程，进程中人为地进行了房间的逻辑分组划分，一个游戏服务器进程根据运营情况，包含了 15 个以上的房间划分。除了这三类功能服务器，平台中肯定还存在其他种类的服务器，有的用来提供广告信息，有的则用来维系这些服务器之间的通信，但我们很难再进一步获得这些服务器的信息。

◆ **通信协议分析**

在封包检测过程中我们发现，QQ 游戏采用了 TCP 协议和 UDP 协议混用的通信连接方式，但 TCP 协议数据包占了绝大多数。为了保证通信的可靠性，一般棋牌游戏平台都会采用 TCP 协议。至于 QQ 游戏平台中的少量 UDP 协议，我们认为是 QQ 游戏平台用来传送一些不是很重要的即时广告信息。UDP 协议的数据包在进入游戏大厅时出现比较多，在游戏过程中，数据包全部采用了 TCP 协议传输。

在处理和客户端之间的连接时，对棋牌游戏平台这类大用户量访问的服务端来讲，如果服务端是架设在 Windows 平台下，通常都会采用 IOCP 模型；如果服务端架设在 Linux 平台下，则通常会采用 Epoll。至于 QQ 游戏服务端分布式系统之间的通信协议，就没法去推测了。

◆ **多线程模型分析**

对于 QQ 游戏的各个功能服务器，从我们的监测结果很难看出它们到底采用了什么样的多线程模型。但从 QQ 游戏如此高效的性能来看，服务器应该采用了多线程而非单线程的方式来处理来自客户端的消息，特别对于游戏服务器来说，如果不采用多线程，很难达到 6000 人的单机承载量。QQ 游戏将每个房间的最大玩家数设置成了 400 或者 300 人，这样一台游戏服务器上就被划分为 15 个左右的游戏房间，每个游戏服务器中最起码有 15 个线程来分别处理各个房间的逻辑。为进一步提高效率，每个房间也可能不止一个线程在处理消息。如果有多于一个的线程处理一个房间的消息，那么就必须处理好数据访问的同步互斥问题，避免数据出错。

为什么要把每个房间的最大玩家数设置成 400 或者 300 人呢？一个房间能承载 6000 人多好！其实，不单 QQ 游戏是这样，所有的棋牌游戏平台都是这样。这样设计自然有它的合理性。游戏房间需要将房间内的信息及时通知给房间内的玩家，这些信息包括玩家信息和状态的改变、游戏桌状态的改变、玩家在房间里的聊天信息等等。这些信息的广播对于服务器来讲负载十分重，而且这种负载与

玩家数是 n 平方的关系，所以一个房间允许的最大玩家数应该有个限制，目前看来，400 人或者 300 人应该是一个比较合理的上限。

◆ **棋牌游戏接入形式分析**

为了便于对棋牌游戏进行维护和升级，一般的棋牌游戏平台的客户端，都采用了平台和游戏分离的形式。玩家开始安装的仅仅是一个不带任何游戏的平台，需要玩什么游戏时，再下载相应的游戏进行安装，这样避免了平台客户端过于庞大。在 QQ 游戏客户端的安装目录下可以发现，后来安装的棋牌游戏都以执行文件的形式存在，玩家玩哪个游戏时就动态地启动相应的执行文件，平台和游戏之间的关系是进程间的关系。采取这种方式，游戏和平台相对比较独立，方便对游戏的升级维护。

服务端的棋牌游戏程序和房间程序是绑定在一起，还是也采用了类似于客户端的形式？这点很难去判断。QQ 游戏中，同一种游戏基本都是设置在同一台物理服务器上，因而我们猜测 QQ 游戏中的游戏和房间程序是绑定在一起的，也就是没有采用类似于客户端游戏和平台分离的形式。

2.2.2 联众棋牌游戏平台分析

图 2-7 联众游戏大厅[38]

Figure 2-7 Hall of LianZhong game [38]

◆ **服务端架构分析**

在分析联众游戏时，我们仍然采用了 commview 来进行封包检测，检测的流程也如图 2-6 所示。首先来看登录。在联众客户端输入用户名和密码并给服务端发出去后，客户端就会与登录服务器进行通信。此时服务器的 IP 和游戏安装目录下的配置文件 LangRsc.ini 中的“RegisterIP”中的 IP 相对应。登录过程中，跟 QQ 游戏一样，同样会收到来自十个左右 IP 的数据包，这些应该也是大厅显示所

必需的信息。进入大厅后，这时的操作和 QQ 游戏有所不同。在 QQ 游戏大厅中，通过展开游戏树，可以看到最底层的游戏房间，而在联众大厅中，是不能到达最底层的游戏房间的，最低只能到达房间的上一层。对于具体的游戏信息，QQ 游戏是在玩家展开游戏树时，及时向服务器索取的，每次点击都会出现“正在下载目录信息，请稍候”的字样，而联众游戏在开始进入大厅的时候，就把所有的游戏信息，包括人数信息传给了客户端，点击展开的时候是不会和服务端产生通信的。由于在大厅中不能看到具体的游戏房间，因而要进入房间游戏，只有先点击房间的上一层，如“斗地主”游戏一副牌的北方网通 4。这时玩家被随机分配到北方网通 4 下若干房间中的一个房间里，如图所示：

图 2-8 联众游戏房间[38]

Figure 2-8 Room of LianZhong game [38]

玩家在房间界面里，可以通过右上角的游戏树，随时在此游戏的其他房间里进行切换。经试验发现，联众游戏中的“北方网通 X”实际上就对应着一台物理服务器。每个物理服务器上统一部署 8 个房间，每个房间的人数上限是 300 人，也有部分游戏是 400 人。这样，联众游戏的一台游戏服务器的承载量在 2000-3000 人，相比于 QQ 游戏的 6000 人，还是有一定差距的。当点击右上角的游戏树时，点击不同的“北方网通 X”就会与不同的 IP 进行通信来刷新房间人数信息，可见各个服务器都维护本服务器上游戏房间的人数信息，并负责向客户端提供，这点与 QQ 有专门的服务器负责房间人数信息是不同的。

从上述分析可以看出，联众游戏同样起码包含三类功能服务器：登录服务器、信息服务器和游戏服务器。与 QQ 游戏有所不同的是，联众游戏更明显地突出了游戏服务器，8 个房间就是一个游戏服务器，而且显式地标记了出来。同一个游戏服务器上的房间信息都由该游戏服务器负责像客户端提供。同样，除了这三类功能服务器，联众游戏的服务端应该也还存在其他类型的服务器用来维系各个服务器之间的协作，具体如何工作无从知晓。

◆ **通信协议分析**

联众游戏和客户端之间的通信和 QQ 游戏类似。绝大多数数据包都采用 TCP 协议，少量采用 UDP 协议的数据包，我们也认为是用于传输一些非重要信息。服务端采用了何种通信协议，同样没法分析。

联众游戏同样属于大用户量访问的网络应用程序，这类程序现在基本都采用同样的通信模型，即 Windows 平台下的 IOCP 以及 Linux 平台下的 Epoll，相信联众游戏也应该采用了其中之一。

◆ **多线程模型分析**

联众游戏的游戏服务器，也类似于 QQ 游戏的游戏服务器，逻辑上进行了房间的划分，因而游戏服务器中至少包含 8 个线程分别处理各个房间的游戏逻辑。联众游戏的单机服务器承载量不如 QQ 游戏，估计联众游戏的单个房间中，并没有采用多线程来进行逻辑处理，而 QQ 游戏的单个房间中，很有可能采用了多线程。当然，也有可能是物理服务器本身的硬件配置差别导致了它们之间的差距。

◆ **棋牌游戏接入形式分析**

在客户端，联众游戏采用了和 QQ 游戏类似的棋牌游戏接入形式，即游戏以进程的形式接入平台，游戏和平台之间通过进程间通信进行交互。服务端同样比较难于判断，我们猜测其服务端可能也和 QQ 游戏的服务端一样，将房间和游戏规则绑定在了一起。

2.2.3 分析小结

QQ 游戏和联众游戏这两款棋牌游戏平台做得都十分出色，而且它们在某些技术方面都采用了类似的方法。这两款棋牌游戏平台的在服务端架构、通信协议、多线程模型以及游戏接入形式方面的特点总结如下：

- 服务端架构

  两款棋牌游戏平台的服务端都起码具备三类功能服务器，即专门用于验证用户的登录服务器、用于提供各种信息的信息服务器和用于处理游戏逻辑的游戏服务器。对于登录服务器来讲，两款游戏平台都提供了多个登录服务器，从而实现登录的负载均衡。两款平台信息服务器的设置稍有不同，QQ 游戏的所有房间信息都由专门的信息服务器提供，而联众则由将此功能融合到了相应的游戏服务器上，这点差别大同小异。两款平台的游戏服务器上都进行了房间的逻辑划分，在服务器中将 300 或 400 个玩家划分为一个小组，这样布置有利于降低游戏服务器的负载。除了这三类基本的功能服务器，服务端还应该具备其他的服务器，以维持整

个服务端的有序工作，但这些服务器对客户端来讲是透明的，因而就比较难于分析。

- 通信协议

  两款游戏平台在通信协议方面比较类似。为了保证通信的可靠性，绝大多数通信都使用了 TCP 协议，存在的少量 UDP 协议应该都是用来传输一些不是很重要的广告信息。由于棋牌游戏平台属于大用户量访问的网络应用，因而两款平台应该使用了 IOCP 或者 Epoll 其中一种通信模型。

- 多线程模型

  由于两款棋牌游戏平台的游戏房间都是逻辑上的划分，因而游戏服务器上最起码应该为每个房间设置一个线程。至于房间中的逻辑处理是否又采用了多线程，这点无从知晓，但从 QQ 游戏如此高的单机承载量来看，QQ 游戏很可能还采用了多线程来处理房间内的游戏逻辑。

- 棋牌游戏接入形式

  为了方便维护棋牌游戏，同时也为了能动态地根据需要加载游戏程序，两款平台的客户端都选择进程作为棋牌游戏的接入形式，客户端平台和游戏之间通过进程通信进行交互。

通过以上的分析，了解了 QQ 游戏和联众游戏服务端的一些基本而简单的信息。虽然这些信息还不足以开发出一套棋牌游戏平台系统，但至少对设计起到了一定启发作用。在这样的分析基础之上，本文提出了自己的设计方案。

2.3 本章小结

本章是论文的技术与实例基础。本章第一部分介绍了棋牌游戏平台所涉及的相关技术，主要涉及分布式技术、网络通信技术、多线程技术以及动态链接库技术。本章第二部分通过封包检测的方法，分析了 QQ 游戏和联众游戏的服务端，为进一步的设计提供了参考。

第 3 章 一款棋牌游戏平台服务端的具体设计

3.1 设计中需要解决的问题

棋牌游戏平台发展到今天，其格式已经比较成熟。目前市面上存在的几款棋牌游戏平台，格式上基本相似，如 QQ 游戏、联众游戏等等，基本上都是由大厅、房间和游戏桌组成。玩家在大厅中可以看到平台上所有的游戏种类和游戏房间。玩家进入房间后，就可以看到房间里的所有游戏玩家和游戏桌。进入游戏桌就可以和桌上的玩家进行游戏。本文提出的棋牌游戏平台也将采用这样的格式。

棋牌游戏一个比较重要的特点就是具有很强的地区特色。中国地域广阔，流传于民间的棋牌游戏在各地都有不同的玩法，就拿麻将来说，流行于各地的就有四川麻将、武汉麻将、杭州麻将、长沙麻将、广东麻将等多达几十种的不同玩法。其他棋牌游戏同样存在这样的特点。正因为如此，有棋牌游戏平台的运营商提出了区域性运营这样一个概念。所谓区域性运营，是指在不同地区分别架设服务器组，比如架设上海服务器组、江苏服务器组等等，每个地区服务器组主要面向当地玩家提供具有当地特色的棋牌游戏，这样不但使得特色游戏相对集中化，也因为服务器的本地化避免了南北互连所带来的网络问题，提升了游戏性能。

为了给用户提供更多的增值服务，现在很多运营商提出了运营多样化的理念。所谓运营多样化，主要是指游戏的运营策略多样化。比如有些 VIP 用户可以到所有游戏服务器进行游戏，而普通用户却不能进入 VIP 区。我们认为，棋牌游戏平台的区域性运营，不仅仅局限于地理位置的划分，还可以用于运营策略的划分。棋牌游戏平台的运营区域化，可以使运营商很方便地对各个区域制定不同的运营策略，比如收费区和免费区、常规区和比赛区，等等。有了这样的概念，可以充分发挥运营商的各种运营手段，从而取得更好的运营效果。

棋牌游戏在国内非常流行，游戏种类繁多，游戏规则各异，这给棋牌游戏开发团队带来了困难。如果一个开发团队在完成了棋牌游戏平台的基础上，要开发出平台上的所有棋牌游戏，那可能要花上很长的一段时间。因而，由一家开发团队来开发平台上的所有棋牌游戏是不高效的。如果棋牌游戏平台能提供一套统一的开发接口，只要按照开发接口来开发棋牌游戏，这些棋牌游戏就能集成到平台之上，那么，将大大缩短游戏的开发周期。从另一方面来讲，如果有了统一的开发接口，也避免了游戏的重复开发。现在肯定已经存在很多现成的棋牌游戏模块，按照开发接口进行适当修改，就可以使用，避免了从头开发。

设计一款具有区域性运营功能，并提供第三方游戏开发接口的棋牌游戏平台，是本文的目的所在。为了设计出这样的棋牌游戏平台，我们必须解决如下问

题：

1. 采取什么样的服务端架构来实现区域性运营和服务端的扩展性要求？为了实现区域性运营、运营策略多样化，就必须采用某种分布式系统架构。这种架构不但要能实现区域性运营，还必须能实现跨区域游戏，也就是玩家可以到平台内的任何区域进行游戏而不需要重新登录。扩展性是评价网络应用的一个重要性能标准，我们提出的服务端架构也应当拥有较好的扩展性，而且各个功能服务器之间要尽量相互独立，在调整服务器时，尽可能不影响其他服务器的正常运行。

2. 采用什么样的通信模式来实现大规模用户访问以及服务端之间数据的高效传输？棋牌游戏平台需要支持大用户量访问，为了能高效处理大规模的网络连接，必须选用一种合适的通信模式。另外，服务端本身也是分布式系统，分布式系统间如何进行数据的有效传输？是采用现有的网络中间件还是另行设计？

3. 采用什么样的多线程模型在服务器工作效率与开发复杂度之间找到平衡点？多线程的高效性已经得到了广泛的认可，但其带来的种种问题也非常棘手，比如数据访问的同步互斥问题、局部线程对整个程序的危害性问题等等，如果解决不好这类问题，那么很难做到服务端的稳定，很难保证服务端数据的安全性，而要解决这些问题，在开发过程中不可避免地要做很多额外工作，加大了开发难度。是否因为这些就放弃多线程？还是有其他方法？

4. 如何实现第三方游戏开发接口？以什么样的形式来集成第三方开发的棋牌游戏？第三方开发的棋牌游戏可以以进程的方式和平台进行交互，它们之间只要协商好通信协议就行了；第三方棋牌游戏也可以以库的形式提供给平台，可以是静态库，也可以是动态库。那么到底采用哪种方式？第三方开发接口具体又如何定义？

在本章接下来的篇幅中，将根据这些问题，给出具体的服务端设计方案。

3.2 棋牌游戏平台服务端的具体设计

3.2.1 架构设计

3.2.1.1 需求分析

我们设计的架构，最主要的是要满足区域性运营的要求。从对区域性运营的说明可以看出，各个区域在网络架构中可以看成一个个的通信节点。而为了实现各个区域之间的通信，方便玩家能在所有区域中游戏，则需利用某种方式将这些通信节点连接起来。2.1.1 小节提到的分布式系统的 8 种静态拓扑结构中，星型结构正好符合我们的需求。将星型结构的叶节点用作区域节点，而中心节点则用来维护各区域之间的通信，就可以实现我们的需求。另一方面，星型结构具有较好的扩展性，还具有各叶节点之间不相关的特点，这些特点也符合我们对服务端

扩展性及功能服务器不相关性的要求。因而，本文设计的棋牌游戏平台服务端架构，采用了一种基于星型分布式系统的架构，称为扩展星型架构（SSA，Scalable Star Architecture）。SSA 充分利用了星型分布式系统扩展性好的优点，使得棋牌游戏平台具有较好的扩展性，而且比较方便地实现了平台的区域性运营。

为了实现棋牌游戏平台的基本功能，架构中需要设置一些功能服务器。通过 2.2 小节中的分析可以看到，一般棋牌游戏平台中需要有登录服务器用来进行登录验证，还需要有信息服务器来提供平台中的游戏信息，另外还需要专门的游戏服务器来负责棋牌游戏的逻辑处理。因而，在我们的区域服务器组中，应该至少有这三类功能服务器。

网络游戏具有用户量变化比较大的特点，运营初期和运营成熟期的用户数可能差距会很大，这就需要网络游戏的服务端能具备一定的扩展性，在用户数量增多时，可以比较方便地对服务器进行一定扩展，从而达到新的要求。针对这个需求，我们将区域服务器组也设计成了星型结构，位于星型结构叶节点的正是登录服务器、大厅服务器和游戏服务器这三类功能服务器，这样就能根据实际需要，来对这些功能服务器进行一定程度地扩展。

架构中还必须设计数据库服务器。对于棋牌游戏而言，数据库主要保存两类信息，一类是玩家用户名、密码、昵称等与具体游戏不相关的信息，还有一类就是与游戏相关的战绩信息，比如玩家在某种游戏中的输赢局数。我们在架构中设计了两种数据库，分别用来存放这两类信息。

3.2.1.2 具体设计

SSA 架构的中心是一个主服务器，用来维系各个区域服务器组（RSG，Region Server Group）之间的信息交互与同步，这里将主服务器称为 MS（Main Server）。由于棋牌游戏平台需要支持玩家经过一次登录就能在所有区域的服务器上进行游戏，因而中心节点 MS 是必不可少的，通过 MS 就可以实现各个区域服务器组之间的信息交互。叶节点就是区域服务器组了，区域服务器组的数量是可以根据需求来进行一定规模扩展的。总体结构如下图所示：

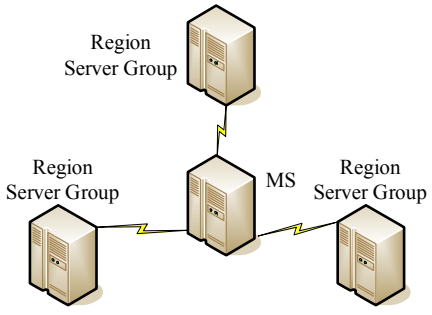

图 3-1 服务端整体架构示意图

Figure 3-1 Main architecture of server side

区域服务器组本身也是一个星型结构，位于中心的是区域中心服务器，而叶节点上则是三类功能服务器。区域中心服务器在区域服务器组中扮演一个十分重要的角色，因为各个区域分不同策略进行运营的功能，就是通过区域中心服务器来实现的。这里分别将区域中心服务器、登录服务器、大厅服务器和游戏服务器称为 IS（Information Server）、LS（Login Server）、HS（Hall Server）和 GS（Game Server）。区域服务器组的示意图如下图所示：

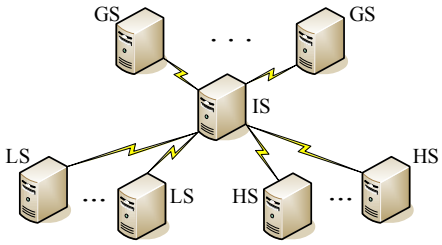

图 3-2 区域服务器组架构示意图

Figure 3-2 Architecture of region server group

架构中设计了两种数据库服务器。用于存储游戏不相关信息的数据库，称为 Main DB，整个系统中只设有唯一的一个 Main DB，MS 和所有的 LS 与之交互。LS 在玩家登录时，需要到数据库查询玩家的用户名和密码信息，因而需要与数据库通信。还存在其他一些与具体游戏不相关的数据，这些数据也要存储到 Main DB 中，例如游戏中的金币，这种数据需要通过 MS 来维护各个 IS 上的数据同步，因而由 MS 负责更新数据库比较合理。用于存储游戏相关信息的数据库，每个区域都设置一个，IS 负责与之交互。这种数据库用来存储玩家的战绩信息，只会与区域中的某个具体的游戏有关。这里规定，不同区域如果出现相同游戏，那么玩家在这两个游戏中的游戏战绩是相互独立的。这样，每个区域设置一个这样的数据库就可以了。由于一种游戏会存在很多个游戏房间，而且玩家也可以同时在多个房间游戏，因而在同一个游戏下的房间中，游戏战绩也存在着数据同步问题。因而，游戏战绩必须通过 IS 来同步，因而由 IS 来负责数据库中的战绩更新。

从玩家角度来看，整个游戏过程中，玩家只会和服务端的 LS、HS 和 GS 发生通信和连接关系，因而没必要对客户端提供 MS 和 IS 以及数据库的连接服务。MS、IS、数据库都是非常重要的关键性服务器，对玩家隐藏也有助于提高服务端的安全系数。棋牌游戏平台还需要提供自动更新的功能，因而在服务端架构中还必须设置一个专门提供自动更新功能的更新服务器（US，Update Server）。这样，我们给出含有一个区域服务器组情况下的完整服务端架构，多个区域服务器组作类似扩展即可，保证 IS 和 MS 连接。棋牌游戏平台的服务端完整架构如下图：

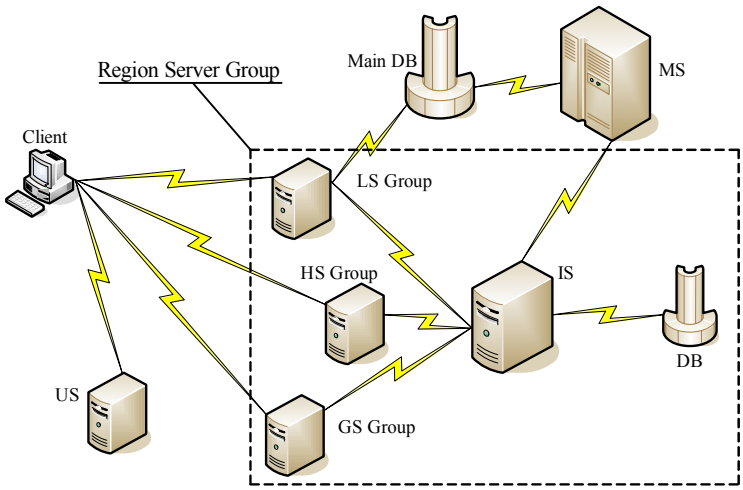

图 3-3 服务端完整架构（一个区域服务器组）

Figure 3-3 Server architecture (With 1 region group)

这里的各个 server 是一个个的“软服务器”[39]。所谓软服务器，是为区别于传统的物理硬件服务器而提出的。一个软服务器并不对应现实的一台物理服务器，一台物理服务器上可以部署多个软服务器。在本文的架构中，MS、IS、LS、HS、GS 和 US 都是软服务器。这些软服务器就是一个个的执行文件，软服务器之间通过 TCP/IP 进行通信，不同服务器通过不同的监听端口来进行区分，不管它们是否在同一台物理服务器上。采取这种方法，可以十分便利地进行服务器资源分配。极限情况下，可以将所有的软服务器都部署在同一台物理服务器上。当然，也很容易做到像 QQ 游戏和联众游戏那样，将几个游戏房间部署到同一台物理服务器上。说到游戏房间，这里的设计与 QQ 游戏和联众游戏是有所不同的。通过封包检测发现，无论是 QQ 游戏还是联众游戏，它们一台物理服务器上的所有游戏房间都是通过一个监听端口来与客户端进行通信的。而这里一个 GS 就对应着一个房间，不同房间对应着不同监听端口。采用这样的设计，不但能方便灵活地部署服务器，而且房间服务器之间没有相关性，一个房间服务器出现问题，不会影响到其他房间服务器的正常运行，另外房间的概念也更加清晰。

各个服务器的说明总结如下：

- MS（Main Server，主服务器）

  整个架构中包含唯一的一台 MS。MS 的功能主要有三个。功能之一，维护游戏平台上的全局登录信息。所有在平台上登录的玩家在 MS 上都有相应记录，便于玩家异地进行游戏，也可防止重复登录。功能之二，维护各区域 IS 的连接信息，维护所有游戏列表信息。功能之三，维护玩家的全局信息，如金币信息。

- IS（Information Server，区域信息服务器）

  IS 是每个区域服务器组的中心服务器，每个区域服务器组都有唯一一台

这样的服务器。IS 除了配合 MS 完成一些同步功能外，最重要的是提供该区域的运营策略设置。这些运营策略主要可能包括：进入该区域游戏房间的积分限制、金币数限制，输赢积分和金币的增减机制，玩家等级制的实现（如 VIP 用户和普通用户的权限有哪些不同），等等。

- LS（Login Server，登录服务器）

  LS 负责用户的登录验证。并不是所有区域服务器组都需要 LS。LS 可以根据需求，主要是网络状况，在全国设置多个登录服务器点。LS 需要与 IS 连接，且设有 LS 的服务器组中一定需要有 HS。每个登录服务器点，可以根据具体的负载情况设置多台 LS，通过最简单的 DNS 解析负载均衡算法[14]，可以实现登录时多个 LS 间的负载均衡。

- HS（Hall Server，大厅服务器）

  HS 负责实现玩家在大厅中的功能。HS 和 LS 应该是对应关系，只有配有 LS 的区域服务器组中，才应该部署 HS。为了能实时地更新客户端信息，这里将客户端和 HS 之间的连接设置为长连接。根据负载情况，HS 也可以设置多台。客户端与 HS 连接的负载均衡由相应的 IS 来实现。玩家通过验证后，IS 会将连接人数相对较少的 HS 的连接信息通过 LS 告诉客户端。

- GS（Game Server，游戏房间服务器）

  在客户端看来，一个 GS 就对应着一个游戏房间。区域服务器组中可以设置多个 GS。GS 主要负责游戏房间中的逻辑功能实现，包括游戏规则的实现。玩家进入某个房间，就会与相应的 GS 发起连接。

- US（Update Server，更新服务器）

  US 是一个功能比较独立的功能服务器，主要负责大厅以及游戏版本的验证与更新。玩家在每次打开大厅时，首先都会去 US 检测目前的大厅版本，如果需要更新，US 就会通知客户端，并将补丁文件的下载路径告诉客户端，随后客户端就可以根据路径来启动下载程序到相应的下载服务器下载补丁文件。游戏更新类似。US 根据负载也可以设置多个，使用 DNS 均衡算法同样能实现 US 访问的负载均衡[14]。

- DB（Data Base，数据库）

  本架构中，包括两种数据库。一种是唯一的 Main DB，存放所有玩家的账号信息，同时还存储与具体游戏无关的全局信息，比如金币数、全局道具、性别、所属公会等等，LS 和 MS 与这类数据库相连。另一种是区域 DB，存储玩家在本区域游戏中的数据信息，比如玩家在某个游戏中的胜负局数等等，区域的 IS 负责与区域 DB 通信。

由于采用了星型结构作为基本结构，因而此棋牌游戏平台的服务端具有较好的扩展性。就区域服务器组来讲，可以根据需求，设置多个区域服务器组，而且每个区域服务器组之间是相互独立的，各个区域服务器组可以设置自己的运营策略。就功能服务器而言，LS、HS、GS 都可以根据实际运营情况，动态地增减服务器数量，而且这些位于叶节点的功能服务器的动态开启、关闭或者是崩溃，并不会影响到其他服务器的正常运行。星型结构不但提升了服务端的扩展性，而且增强了服务端的稳定性。

可以为各个区域服务器组提供不同的运营策略，这是本文提出的棋牌游戏平台服务端的一个特点。区域服务器组中的 IS 提供该区域的运营策略设置。这些运营策略主要可能包括：进入该区域游戏房间的积分限制、金币数限制，输赢积分和金币的增减机制，玩家等级制的实现（如 VIP 用户和普通用户的权限有哪些不同），等等。当然，我们的棋牌游戏平台也可以做成 QQ 游戏或者联众游戏那样没有特殊运营策略的模式，只要搭建一个区域服务器组，或者所有区域服务器组采用相同的运营策略就可以了。

对运营商来讲，掌握用户群是至关重要的，掌握了用户就掌握了市场。区域性运营，必然会面临区域运营商和总运营商之间用户群的控制问题，总运营商不会让区域运营商完全控制区域用户。因而我们将所有玩家的注册信息都保存在 Main DB 中，要求所有玩家登录时都通过 Main DB 来进行合法性验证，而且登录成功后在 MS 上作相应记录。这样，总运营商通过 Main DB 就能掌握整个游戏平台的用户群，而且通过 MS 可以监控所有玩家的状态。

星型结构最大的问题就是随着规模的扩大，中心节点可能会成为系统的瓶颈。这里采用一种类似于镜像服务器的方法来解决这个问题。经分析，架构中 IS、MS 这些中心节点的主要负载来自维护各个区域房间信息的同步和玩家信息的同步，而维护房间信息的负载更大。比如某个玩家从一个区域登录，他需要知道其他区域的游戏服务器信息，这时就只有通过中心节点的周转去获取这些信息。为了降低这些中心服务器的负载，我们在所有 IS 上都放置了 MS 上的房间信息镜像，在 HS 上都放置了 IS 上所有房间信息的镜像，这样客户端直接向 HS 索要相关信息就可以了，而不需要频繁地去访问 IS、MS 等中心节点，中心节点只需要定时地去维护 HS 上信息的实时性。采用这种方法，中心节点的负载跟用户量没有关系，而 HS 作为叶节点可以根据用户量扩展，这样一定程度上就解决了中心节点的瓶颈问题。

3.2.1.3 软件实现流程

基于上面设计的服务端架构，本小节将通过流程图的形式给出详细的软件实现流程。流程图采用了 UML 中的序列图来绘制。

- 注册

用户注册通过 web 方式进行，web 服务器另行搭建，与主数据库进行通信。

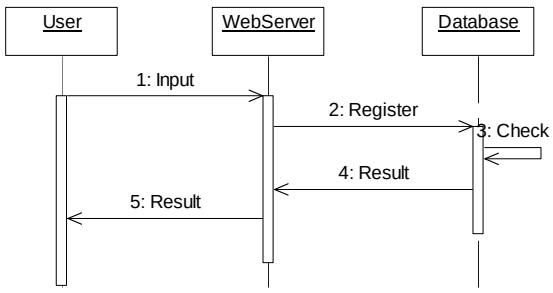

图 3-4 注册

Figure 3-4 Register

- 大厅更新

玩家打开大厅时，首先需要对大厅进行版本验证，如需更新则进行更新。

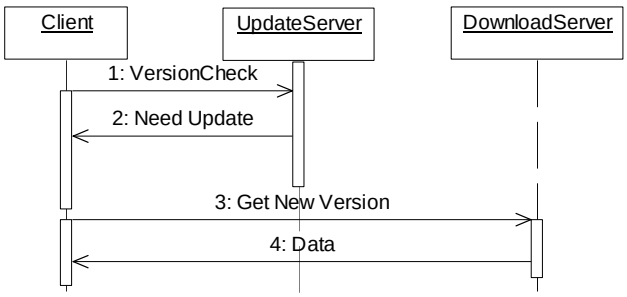

图 3-5 大厅更新

Figure 3-5 Update hall version

- 登录

客户端会与玩家选择的 LS 进行通信，LS 到 Main DB 检测用户名和密码是否正确。如果用户名和密码验证通过，则进一步验证玩家是不是重复登录。LS 可以通过 IS 到 MS 验证玩家是否重复登录。如果成功通过验证，IS 会将某个 HS 的连接信息通过 LS 返回给客户端。

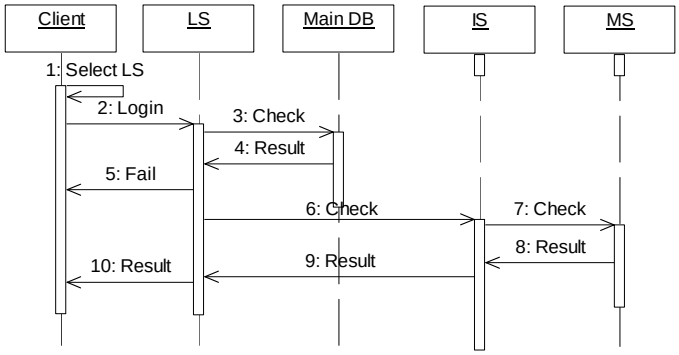

图 3-6 登录

Figure 3-6 Login

- 进入大厅

一旦玩家通过登录验证，客户端就会与 LS 断开连接，通过 LS 返回的 HS 的连接信息，客户端向该 HS 发起连接。HS 为了验证该客户端连接是否合法，会到 IS 上进行验证。通过 IS 的检验之后，玩家便成功与 HS 建立连接，此时 HS 将游戏列表信息传送给客户端。同时，HS 经过 IS，通过 MS 来读取玩家信息并将其返回给客户端。由于房间级之上的树结构相对比较固定，这样，这些结构信息就可以存储于客户端的配置文件中，而不需要从服务端获取。由于要维护游戏树上人数的实时性，这就需要隔一段时间（如 10 分钟）重复一次 10 和 11 步骤。

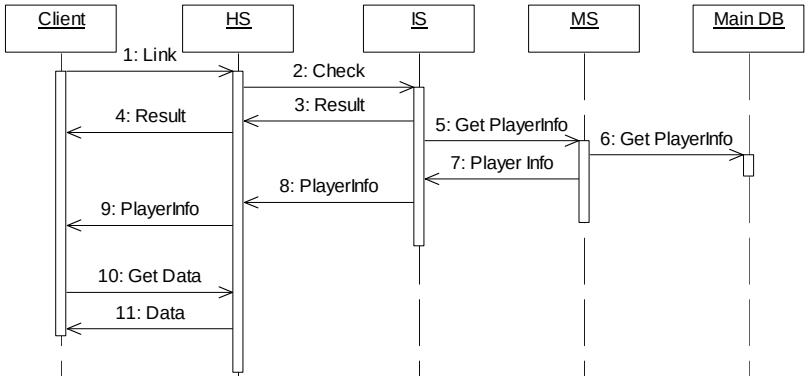

图 3-7 进入大厅

Figure 3-7 Enter hall

- 获取房间信息

由于玩家进入大厅时，只是将房间以上级的信息传给了客户端，这样玩家在展开某个游戏时，就需要到服务端获取房间信息。房间信息包括房间的连接信息和房间当前玩家人数信息。每个 HS 上都有包含所有房间信息的副本，客户端直接向 HS 索取即可。客户端只向服务端索要一次房间连接信息，而人数信息需要多次刷新。为尽量降低服务端的负载，这里并不是每次点击展开就向 HS 索要人数信息，客户端可以做个限制，距离上次索取人数信息达到一定时间以后，才可以向 HS 索要最新的人数信息。

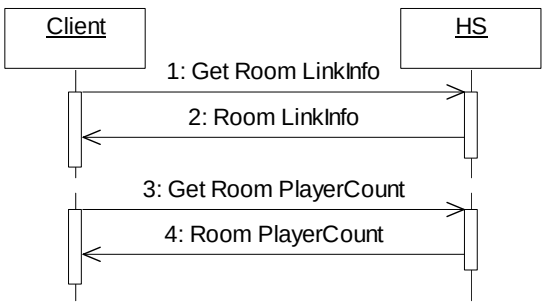

图 3-8 获取房间信息

Figure 3-8 Get rooms' information

- 服务端房间列表同步

GS 在向 IS 发起连接时，会将自身的连接信息告知 IS，IS 这时会将房间的连接信息通过 MS 广播给所有的 IS，IS 又会将这些连接信息告知与之相连的 HS，这样每个 HS 上就有所有 GS 的连接信息了。当 GS 关闭时，通过同样的过程即可在 IS、MS 和 HS 上删除相应 GS 的连接信息。增加房间连接信息的过程如下：

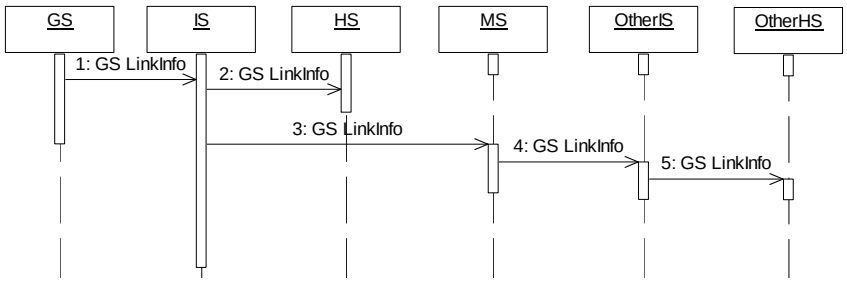

图 3-9 GS 连接信息的维护

Figure 3-9 Maintain link information of GS

每个 HS 上都有平台下所有房间列表信息的镜像。主要采取两个策略来维护镜像的实时性。第一，通过 MS 来维护各个 IS 上的人数信息同步；第二，维护 IS 和 HS 之间的人数信息同步。每隔一段时间，MS 逐个向 IS 索取此 IS 下房间的人数信息，并将最新信息广播给其他 IS。两个 IS 时的情况可以表示如下：

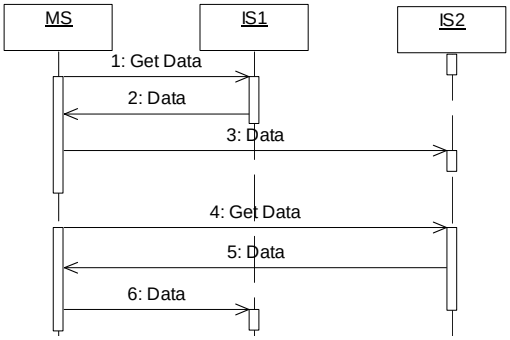

图 3-10 MS 同步 IS 上的房间信息（两个 IS 时的情况）

Figure 3-10 Synchronization of rooms' information on IS (With 2 IS)

通过 MS 来同步 IS，这样每个 IS 上就有了平台上所有游戏房间的人数信息，接下来只要维护 IS 和其下 HS 之间的人数信息同步就可以了。IS 和 HS 之间的人数信息是通过 HS 每隔一段时间向 IS 获取一次信息来实现同步的。具体如下：

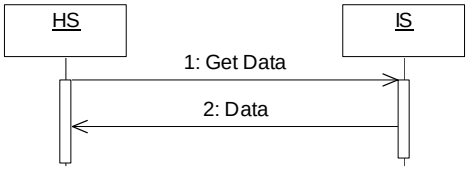

图 3-11 HS 上房间信息的同步

Figure 3-11 Synchronization of rooms' information on HS

经过上述步骤，玩家就能从 HS 获得比较实时的人数信息了。但在维护这些数据同步时，需要注意一个问题，就是在刚启动 IS 时，MS 必须将已经开启的 IS 下的房间信息发送给新开启的 IS。

- 游戏更新

在用户点击某个房间之后，首先要做的就是检测游戏的版本，如果版本过低或者根本还没安装游戏，则提示到相应的下载服务器下载。整个过程跟大厅的更新类似，这里就不重复给出流程图了。

- 进入房间

对游戏进行过版本检测之后，就可以根据所选 GS 的连接信息向相应的 GS 发起连接了,只有通过平台认证的玩家才能进入房间。进入房间可能会存在两种情况，一种是进入本地房间，一种是进入异地房间。如果一个玩家从某地的 LS 登录，他进入该地的 GS，就叫做进入本地房间，进入别的地区的 GS，则称为进入异地房间。玩家从某个地区登录，会在该地区的 IS 上作相应记录，因而进入本地房间时，只需要到该地区的 IS 上进行合法性验证就可以了，如下图所示：

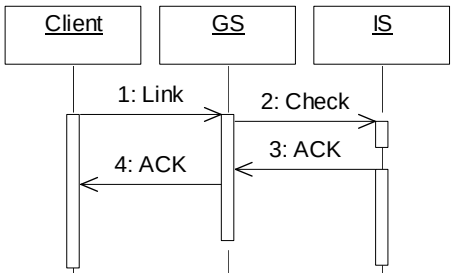

图 3-12 进入本地房间

Figure 3-12 Enter local room

玩家向异地 GS 发起连接，GS 到相应 IS 上验证，这时该 IS 上肯定没有玩家信息，就需要去 MS 上进一步验证，这样才能确认该玩家是否合法。一旦玩家通过验证进入异地房间，那么在异地 IS 上也要创建相应的玩家信息。过程见下图：

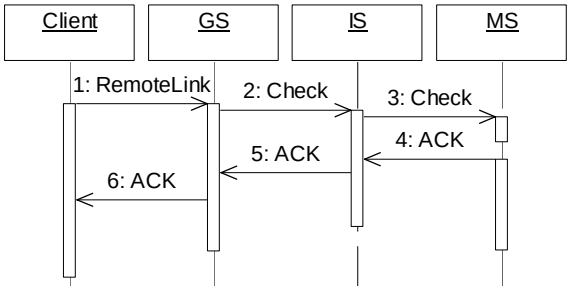

图 3-13 进入异地房间

Figure 3-13 Enter other room

- 房间内信息刷新

房间的一个重要功能就是将房间内的信息广播给玩家。一般的棋牌游戏平台，房间都会提供房间内玩家信息，这些信息包括玩家昵称、等级，在此游戏中的积分、输赢局数，此时在房间中的状态等等。另外，房间还要实时地将桌子的状态告诉给房间里的玩家，比如是不是正在游戏，没有游戏的话目前桌上有几个人，等等。玩家状态信息和桌子状态信息变化会非常频繁，要将这些信息广播给房间内的所有玩家，会对 GS 造成比较大的负载。为了尽量降低服务器的负载，这里我们设计了一个算法，就是"定时广播差量法"。我们在每个 GS 上设置一个特殊的列表，用来存储状态发生改变的玩家。一旦玩家的状态发生改变，就会将其添加到列表中。每隔一段时间，GS 就将这个列表广播给房间内的玩家，广播完毕后清空列表，等待下次的存储和广播。一般这样的时间间隔设置为 1 至 2 秒，这样的误差在客户端是允许的。客户端根据收到的信息，对玩家状态进行相应修改，这样就维护了玩家状态的实时性。至于桌子的状态，服务端没有必要另外广播桌子信息给客户端，通过玩家的状态，客户端就可以判断出此时各个桌子的状态。通过这种算法，没有必要在玩家状态或者桌子状态发生变化时每次都广播，这样大大减少了服务端的负载，还节约了网络资源。整个过程可以表示如下：

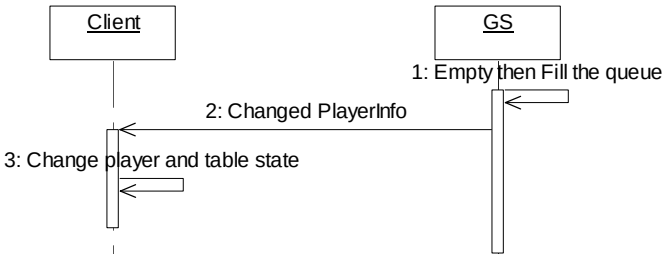

图 3-14 房间内信息刷新

Figure 3-14 Update information in room

- 聊天

聊天包括公聊和私聊，公聊的内容房间里所有玩家都能看到，私聊的内容只有相应玩家可以看到。公聊的话，GS 只要将玩家发来的聊天信息在房间内广播即可，私聊的内容就发送给相应的玩家。聊天过程比较简单，如下：

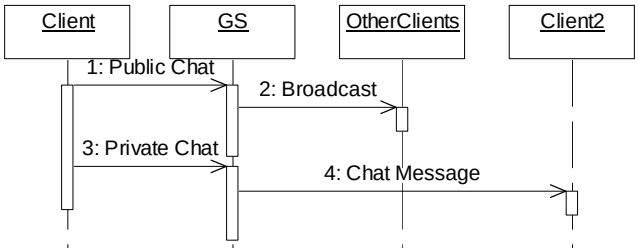

图 3-15 聊天

Figure 3-15 Chat

- 游戏

当玩家进入房间内的桌子后，我们就认为他进入了游戏状态，这时客户端就会弹出相应的游戏界面。玩家的进出桌子、在游戏桌上的活动，包括之后游戏过程中的动作，都需要告诉桌上的其他人，都需要在桌子上进行小范围的广播。这些就是游戏过程，表示如下：

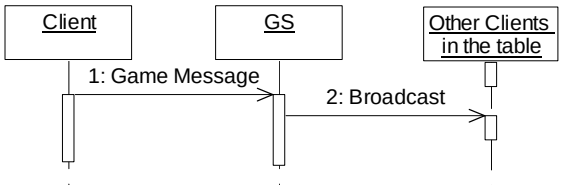

图 3-16 游戏

Figure 3-16 Gaming

- 游戏结果同步

由于玩家可以同时在多个游戏房间里游戏，这样就存在游戏结果同步的问题。比如玩家在房间 A 中赢了一局游戏，增加了 2 个积分，而同时玩家又在相同游戏的房间 B 中，这时就要将玩家在房间 A 中的变化结果通知到房间 B。由于游戏结果的同步只会发生在相同游戏的不同房间，那么这种同步只要通过 IS 来做就可以了。游戏结束时，GS 先不修改玩家数据，而是先将游戏结果传给 IS，IS 到数据库作相应修改，之后 IS 给玩家所在的所有 GS（包括发来消息的 GS）都发送游戏结果，GS 在收到此消息时对玩家数据进行修改并将这些变化发送给房间内的玩家。

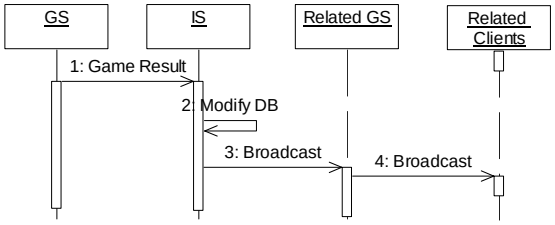

图 3-17 游戏结果同步

Figure 3-17 Synchronization of game result

- 全局信息同步

游戏过程中可能还会涉及到玩家的一些全局信息的变化，比如全局道具的使用，以及最典型的金币变化。全局信息变化同样也需要进行同步。全局信息同步会发生在不同地区之间，比如某玩家在 A 地的房间 B 中赢了 20 个金币，此时该玩家还在 C 地的房间 D 中，这时就需要将 A 地房间 B 中的金币变化也通知给 C 地的房间 D。不同地区的同步，只能通过 MS 来实现了。玩家在某个 GS 发生全局信息变化，这时 GS 不作数据修改，GS 将这一变化通知 IS，IS 通知 MS，MS

到 Main DB 进行数据存储，随后 MS 将变化信息通知给其他的 IS（那些该玩家在的 IS），IS 再将此信息通知给相关 GS（那些该玩家在的 GS），GS 收到消息时对玩家数据进行修改并将修改信息通知给客户端。

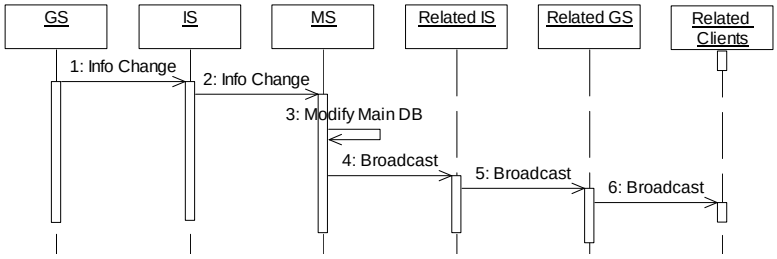

图 3-18 全局信息同步

Figure 3-18 Synchronization of global information

- 退出房间

房间中的玩家不想继续游戏时，可以选择退出房间。退出游戏房间，除了要向房间内的玩家广播外，还要将玩家退出的消息告诉 IS。IS 判断这个玩家是不是从此地登录，如果是，则修改房间的人数信息；如果不是，除了要修改人数信息之外，还需要看看此玩家还有没有在此地的其他房间里游戏，如果不在此地的任何房间里，则删除该玩家信息，同时将删除该玩家这件事告知 MS，让 MS 也知道该玩家不在此 IS 上。

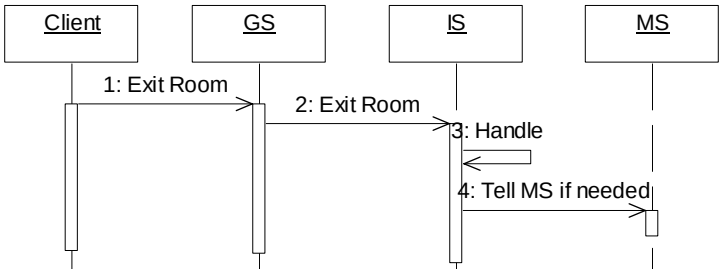

图 3-19 退出房间

Figure 3-19 Exit room

- 退出大厅

最后，玩家要退出大厅了。退出大厅之前，客户端应该做个限制，如果玩家还在某个房间中，则提示玩家首先退出房间。玩家不在任何房间之后，就可以关闭大厅了。关闭大厅时，客户端会发送消息告诉 HS，HS 通知 IS，IS 删除玩家信息同时告诉 MS，MS 将玩家信息删除，玩家成功退出。

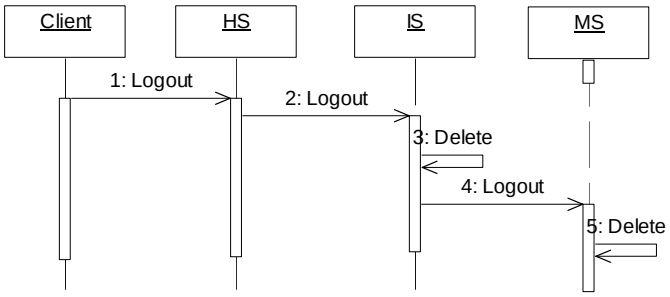

图 3-20 退出大厅

Figure 3-20 Exit hall

3.2.1.4 小结

通过上述分析可以看出，此服务端架构，不但满足了棋牌游戏平台中的一般功能需求，而且还提供了区域性运营的功能，可以针对不同区域设置不同的运营策略。玩家在总运营商的控制下，可以到平台上所有区域的游戏服务器进行游戏。基于软服务器的星型结构的采用，也大大提升了服务端的扩展性和稳定性。

3.2.2 通信协议设计

3.2.2.1 需求分析

网络游戏最重要的特征就是存在网络通信，这也是网络游戏区别于其他游戏的最大不同之处。在本文的棋牌游戏平台中，由于服务端是由多个服务器协同合作而成，因而除了客户端与各服务器之间存在网络通信之外，各个服务器之间也存在着网络通信。

网络游戏中，客户端与服务器之间的通信属于大用户量类型的通信，通常一台服务器要处理几千个玩家的连接。在这种情况下，I/O 完成端口模型是比较适合的。“要记住的一个基本准则是，假如要为 Windows NT 或 Windows 2000 开发高性能的服务器应用，同时希望为大量套接字 I/O 请求提供服务（Web 服务器便是这方面的典型例子），那么 I/O 完成端口模型便是最佳选择！”[15]本文的服务器都是在搭建在 Windows 平台下的，因而在处理服务端和客户端之间的通信时，本文采用了 IOCP（I/O Completion Port，I/O 完成端口）模型。

服务端之间的连接数目并不是很多，因而也就没有必要采用 IOCP 了。服务端实际上是一个分布式系统，目前市面上存在着多种现成的分布式系统之间的通信解决方案，如 DCOM、CORBA 等。这里并没有采用这些网络中间件，一方面是考虑到开发成本问题，另一方面，这些网络中间件的很多功能并不完全适合本设计。本文根据自己的需求，提出了一种通信协议，即 SDCP（Star Distributed Communication Protocol，星型分布式通信协议）[39]。本文各个功能服务器都是以软服务器的形式提供，多台软服务器可以架设在同一台物理服务器上，区别这

些软服务器的重要方法，就是网络通信中的套接字。不管服务器是不是在一台物理服务器上，它们之间的通信都是套接字通信。对于同一台物理服务器上的软服务器，只要设置不同的端口就可以区分它们了。

无论是客户端与服务端之间的通信，还是服务端相互间的通信，我们都采用了 TCP 协议。因为对于网络游戏服务端来讲，玩家数据的安全性和游戏逻辑的正确性是相当重要的，TCP 协议保证了通信的可靠性。

为了实现消息的传送，还需要设计一套消息管理机制，要求发送方能对特定的消息进行封包发送，而接收方能将收到的消息包中包含的信息解析出来。我们自己定义了消息的格式，并提供了一套封包解包机制，满足了这方面的需求。

3.2.2.2 具体设计

◆ **客户端与服务端的通信协议**

客户端与服务端之间的通信模型，主要采用了 IOCP 模型。IOCP 的具体使用方法，可以参阅参考文献[15]。客户端与服务端之间的通信模型可以表示如下：

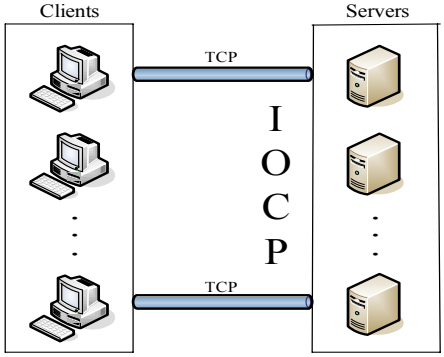

图 3-21 客户端与服务端通信模型

Figure 3-21 Communication model between client and server

客户端与服务端之间的通信模式，是一种典型的 C/S 模式，通常服务端会启动一个监听端口，各个客户端通过此监听端口与服务端建立连接，建立连接之后就可以进行数据通信了。这里为 C/S 模型的客户端和服务端分别设计了网络应用类。C/S 模型中双方的网络应用类的部分接口如下：

```
/// Client网络层类
class ClientNetBase
{
public:
  bool Connect(Target* t);        //< 向server发起连接
  bool Disconnect();              //< 与server断开连接
  Message* GetMsg();              //< 从消息接收队列中获取一条消息
  void SendMsg(Message* msg);     //< 发送一条消息
  ...
};

/// Server网络层类
class ServerNetBase
{
```

```
public:
  bool StartListen(int port);   //< 开始监听
  Message* GetMsg();              //< 从消息接收队列中获取一条消息
  void SendMsg(Message* msg);   //< 发送一条消息
  ...
};
```

◆ **服务端之间的通信协议**

整个服务端是以星型结构为基础架设的，MS 和 IS 之间是一种星型关系，而 IS 和它本地的其他功能服务器之间也是一种星型关系。（这里不用考虑 DB，因为采用的 Oracle 数据库本身提供了一套网络连接方法。）为了能更加灵活地部署服务端，在设计 SDCP 时，我们不考虑架构中的服务器是否位于同一物理服务器上，而将服务器之间的通信都看成 Socket 通信，服务器之间都使用 TCP 协议，这并不影响同一物理服务器上服务器之间的通信，只要为它们分配不同的 Socket 就可以了。SDCP 中将位于星型结构中心的通信节点称为 CN（Center Node，中心节点），将与 CN 通信的节点称为 LN（Leaf Node，叶节点）。在 CN 首先启动好的前提下，SDCP 允许 LN 随时向 CN 发起连接请求，也随时允许 LN 与 CN 断开连接，这样就实现了整个系统的可扩展性。对于区域服务器组而言，当觉得 LS、HS 或者 GS 的负载过重时，完全可以动态地增加相应的服务器，当然，人数减少时也可以动态地减少相应服务器；而对于运营商来说，要增加区域运营商，只要为 IS 分配合法的身份，IS 即可与 MS 建立连接而加入平台，减少区域服务器组时，IS 与 MS 断开连接即可[39]。SDCP 协议模型可以表示如下：

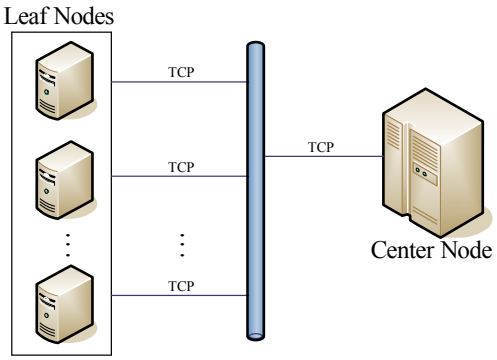

图 3-22 SDCP 协议模型

Figure 3-22 Model of SDCP

实际上，SDCP 仍然是一种 C/S 模式的通信协议，CN 作为 Server，而 LN 作为 Client。这样，CN 将提供监听端口，LN 向 CN 发起连接后进行数据通信。这里为 LN/CN 模型的叶节点和中心节点，也设计了相应的网络应用类，形式上与 C/S 模型的基本类似，所不同的仅仅是网络 I/O 模型，C/S 模型采用的是 IOCP，而 LN/CN 模型采用的是重叠 I/O 机制（Overlapped I/O）[15]。

```
/// LeafNode网络层类
class LeafNetBase
{
```

```
public:
  bool Connect(Target* t);        //< 向CN发起连接
  bool Disconnect();              //< 与CN断开连接
  Message* GetMsg();              //< 从消息接收队列中获取一条消息
  void SendMsg(Message* msg);     //< 发送一条消息
  ...
};

/// CenterNode网络层类
class CenterNetBase
{
public:
  bool StartListen(int port);     //< 开始监听
  Message* GetMsg();              //< 从消息接收队列中获取一条消息
  void SendMsg(Message* msg);     //< 发送一条消息
  ...
};
```

◆ 消息封包解包机制

对底层网络通信来讲，在发送端，网络负责将一串连续的字符发送出去，在接收端，网络负责接收来自发送端的一串连续字符。所以，要将数据发送出去，在发送端，必须将要发送出去的信息打包成一个消息包，而在接收端，必须将消息包中携带的信息解析出来。本文为所有的消息定义了一个消息基类，该消息基类定义了消息的具体格式，同时提供了一套方法用来打包和解析消息，通过调用 Add 方法，就可以将消息打包，而通过调用相应的 Get 方法，就可以将消息解包。消息基类定义如下：

```
class Message
{
public:
  int size;          //< 消息字节数
  int senderID;      //< 该消息发送方的标识
  int type;          //< 消息类型
  char *body;        //< 存放消息内容
  Message();
  virtual ~Message();
  Message(char* bdy);   //< 将接收到的字符串解析成Message

  void Add(const char* str);                      //< 在body中增加一个字符串
  void Add(const string str);                     //< 在body中增加一个string
  void Add(const int len, const char* mem);       //< 在body中增加一段连续内存内容
  void Add(const void* strct,const int len); //< 在body中增加一个结构体
  void Add(const int i);                          //< 在body中增加一个整数
  void Add(const unsigned int ui);                //< 在body中增加一个无符号整数
  void Add(const char c);                         //< 在body中增加一个char类型数
  void Add(const unsigned char c);                //< 在body中增加一个unsigned char类型数
  void Add(const bool b);                         //< 在body中增加一个布尔数
  void Add(const float f);                        //< 在body中增加一个float数
  void Add(const short s);                        //< 在body中增加一个short数
  void Add(const unsigned short s);               //< 在body中增加一个无符号的short数

  char* GetStr();                    //< 获得body中的一个字符串
  string GetString();                //< 获得body中的string
  char* GetMem(int len);             //< 获得body中长为len的一段字符
  void* GetStruct(int len);          //< 获得body中的一个结构体
  int GetInt();                      //< 获得body中的一个整数
```

```
  unsigned int GetUnsignedInt();   //< 获得body中的一个无符号整数
  char GetChar();                  //< 获得body中的一个char类型数
  unsigned char GetUnsignedChar();//< 获得body中的一个unsigned char类型数
  bool GetBool();                  //< 获得body中的一个布尔数
  float GetFloat();                //< 获得body中的一个float数
  short GetShort();                //< 获得body中的一个short数
  unsigned short GetUShort();      //< 获得body中的一个无符号short数
  int current;   //< 指向body中的位置，用于增加和读取信息
};
```

Message 基类规定了消息结构，消息结构如图 3-6 所示。消息的前 12 个字节定义为消息头，分别由消息总长度 size、发送方标识 senderID 和消息类型 type 组成。消息后面的（size-12）个字节就用来存储具体消息的具体信息。

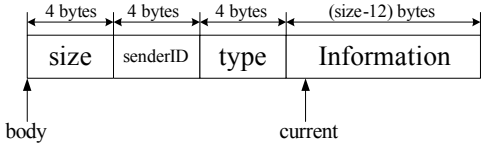

图 3-23 Message 结构

Figure 3-23 Structure of Message

下面通过一个例子来说明消息是怎么样在通信双方传递信息的。这里就用客户端给 LS 发送的登录消息来作为例子。首先来看登录消息的定义：

```
class LoginMsg : public Message
{
public:
  char* account;     //< 用户名
  char* password;    //< 密码

  LoginMsg(char* nAccount, char* nPassword); //< 用于打包消息
  LoginMsg(Message* nMsg);                   //< 用于解包消息
};

LoginMsg::LoginMsg(char* nAccount, char* nPassword)
{
  size = strlen(nAccount)+1 + strlen(nPassword)+1; //< 消息长度里加上信息长度
  size += 12;                  //< 加上消息头长度
  senderID = -1;               //< 发送方标识由服务端分配，登录消息还未获得标识，设为-1
  type = MSG_LOGIN;            //< 消息类型
  body = new char[size]; //< 分配消息体空间
  current = 0;                 //< 游标设为0，从消息体开头加入信息
  Add(size);                   //< 加入size
  Add(senderID);               //< 加入senderID
  Add(type);                   //< 加入type
  Add(nAccount);               //< 加入用户名
  Add(nPassword);              //< 加入密码
}

LoginMsg::LoginMsg(Message* nMsg)
{
  size = nMsg->size;               //< 获得size
  senderID = nMsg->senderID;   //< 获得senderID
  type = nMsg->type;               //< 获得type
  nMsg->current = 12;              //< 游标设到消息头后，准备读取后续信息
  account = nMsg->GetStr();    //< 读取用户名
  password = nMsg->GetStr();   //< 读取密码
}
```

客户端构造并发送登录消息的流程如下：

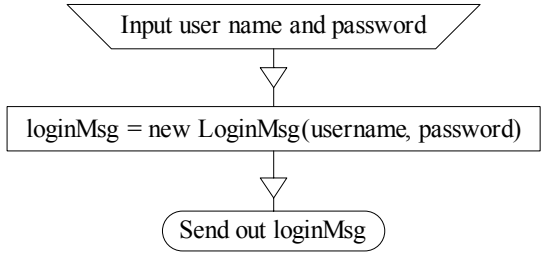

图 3-24 客户端构造登录消息

Figure 3-24 Construction of login message on the client side

首先，客户端接收用户输入用户名和密码，然后用来构造登录消息：

```
LoginMsg* loginMsg = new LoginMsg(username, password);     //< 打包登录消息
```

消息构造好之后，就可以调用网络层将消息发送出去。由于消息类中的指针 body 就是指向整个消息的开头，而前四个字节 size 就是整个消息的长度，有了这两个信息，网络层就能正确地将消息发送给 LS。

LS 解析消息的流程如下：

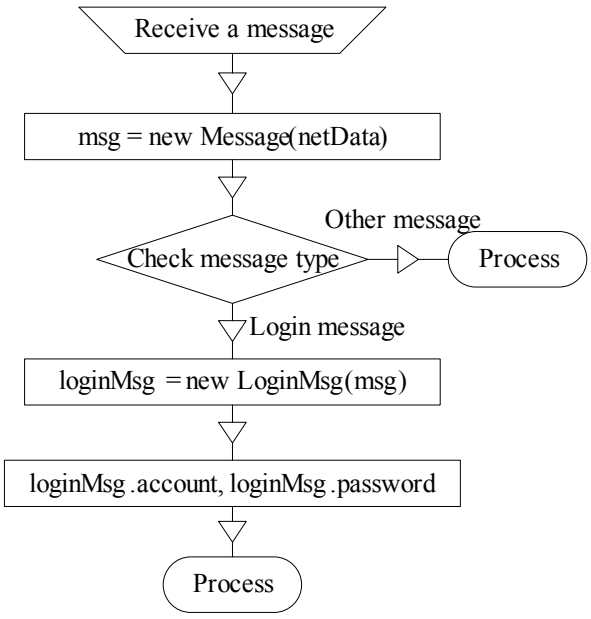

图 3-25 LS 解析消息流程

Figure 3-25 Destruction of login message on login server

LS 的网络层将接收到一条消息，但此时 LS 并不知道这是什么消息。LS 在收到这串字符之后，将调用 Message 的构造函数来生成一个 Message 对象：

```
Message* msg = new Message(netData);     //< 从网络层收到的数据生成 Message 对象
```

将网络层接收到的数据解析成 Message 之后，就可以取得该消息的 size、senderID 和 type 了。LS 根据 type 判断出这是一条请求登录的消息，于是就调用登录消息的解包函数来对消息进行解包：

```
LoginMsg loginMsg(msg);    //< 从 Message 对象生成 LoginMsg 对象
```

经过解包之后，LS 就获得了登录消息中的用户名和密码，即 loginMsg.account 和 loginMsg.password，接下来 LS 就能进行相应的验证处理了。

其他消息的定义和使用，基本上和登录消息一样。通过这套消息系统，就能比较方便地对消息进行打包和解包，进而可以方便地在客户端和服务端，以及服务端之间传送信息。平台中使用的消息类型，具体参见附录一。

3.2.2.3 小结

这里采用了完成端口模型来处理大用户量客户端的访问，这种模型是目前 Windows 平台下最高效的处理大用户量访问的模型，保证了服务端在通信处理方面的高效性。我们为服务器之间的通信专门设计了 SDCP，而没用采用现成的网络中间件，一方面节省了成本，另一方面自行设计的协议也更符合我们的需求。在设计了通信模型的基础上，我们还专门设计了通信消息机制，采用这套消息机制，能比较方便地完成消息的封包解包，从而实现了信息的传递。

3.2.3 多线程模型设计

3.2.3.1 需求分析

总体上来看，服务器的主要功能就是处理消息，有的处理来自客户端的消息，有的处理来自其他服务器的消息，有的两类消息都需要处理。可以采用单线程来处理这些消息，也可以采用多线程来处理这种消息。

对于这种存在多类网络消息（有的消息来自客户端，有的来自其他服务器），而且是来自多个不同玩家的消息，很自然会想到采用多线程来处理消息。采用多线程的最大好处是能充分利用 CPU 资源，从而提高服务器的效率。特别是如今很多服务器都采用了多核处理器，使用多线程更能体现其效率上的优势。

如果采用多线程，自然就会面临数据访问的同步互斥问题，如果处理不当，便会对数据的安全性造成隐患。网络游戏服务端，对游戏数据的安全性要求比较高，一旦核心数据出现错误，就会造成不可忽视的影响。从这个角度来看，单线程没有必要考虑数据访问的同步互斥问题，单线程在实现时更加简便和安全。

我们设计的功能服务器全是一个个进程，这样就能实现同一物理服务器上运行多个服务器，从而形成多进程工作的情形，就能更充分地利用 CPU 资源。多进程虽然在调度效率上不如多线程，但多进程不需要考虑复杂的同步互斥问题。有了多进程工作的效率保证，我们将服务器处理消息的线程设计成了单线程。单线程保证了服务器数据的安全性，降低了开发难度。

3.2.3.2 具体设计

在本文的通信系统中，存在着两种网络通信模式，一种是 C/S 模式，比如客户端和 LS 之间的通信；另一种是 LN/CN 模式，即各个叶节点与中心节点之间的通信。在此棋牌游戏平台架构中，为不同网络模式的不同通信方都设计了相应的网络应用类，即 C/S 模型对应的 ClientNetBase 和 ServerNetBase，以及 LN/CN 模型对应的 LeafNetBase 和 CenterNetBase。对于服务端来讲，各个功能服务器在网络架构中所处的地位不同，它们处理的消息来源也不同。比如对于 LS 来讲，它需要处理来自客户端的消息，同时也需要处理来自 IS 的消息，这样，LS 上就必须存在 ServerNetBase 对象来和客户进行通信，也必须存在 LeafNetBase 对象来和 CN 进行通信。而对于 IS 来讲，它需要和 LS、HS 等叶节点进行通信，这样 IS 就需要有 CenterNetBase 对象，IS 还需要和中心节点 MS 进行通信，那么 IS 就需要有 LeafNetBase 对象。同样的方法，可以分析出其他服务器上所应有的对象，总结如下表：

表 3-1 网络层对象分布情况

Table 3-1 Distribution of NetBase

	Client	MS	IS	LS	HS	GS	US
ClientNetBase	√						
ServerNetBase				√	√	√	√
LeafNetBase			√	√	√	√	
CenterNetBase		√	√				

对于底层网络来讲，每个网络层都有两个消息队列，一个用来存放接收到的消息，称为 inQueue；另一个用来存放需要发送出去的消息，称为 outQueue。每个网络层还分别启动了两个线程，一个用于不停地发送 outQueue 中的数据，另一个用于不停地将网络上接收到的数据存储于 inQueue 中。对于各个功能服务器来讲，所要做的就是不停地将网络层 inQueue 中的消息取出来进行处理，需要发消息时将消息扔给网络层发出去。本文采用了单线程来处理所有网络层的消息。后来在开发过程中也证实了这个选择是正确的。如果采用多线程，不但要面临多线程访问数据的问题，还需要对消息处理进行同步，因为有时处理一个消息时，需要处理完另一个网络层收到的某条消息，这种情况需要做消息同步，就比较复杂了。这样，当服务器上存在两个网络层时，服务器的工作模式可以表示为：

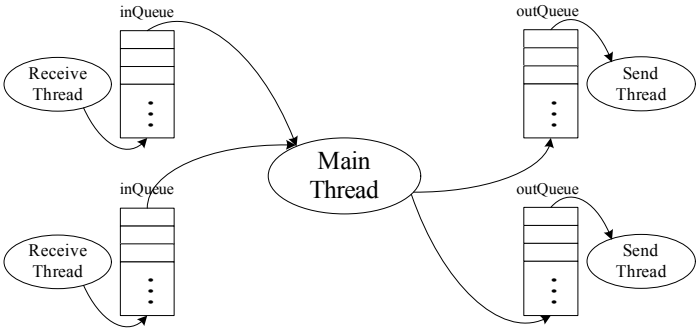

图 3-26 服务器工作模式（两个网络层时的情况）

Figure 3-26 Working pattern of server (With 2 NetBase)

虽然在处理接收和发送网络消息时采用了多线程，但由于这些线程之间没有任何关系，因而没有必要做数据同步互斥处理。服务器可以这样来安排各个线程的工作：服务器启动时就启动网络层的消息接收线程和发送线程，另外再启动唯一的消息处理线程。网络层的线程各司其职，管理好各自的消息队列即可。消息处理线程逐个查询每个网络层的 inQueue，有消息就取出处理，没消息时就查询下一个网络层，如此循环即可。

3.2.3.3 小结

在选择单线程还是多线程来处理网络消息时，为了保证服务端数据的安全性，同时也为了方便开发，我们采用了单线程来处理网络消息。虽然单线程在效率上不如多线程，但服务端的多进程工作模式，一定程度上弥补了单线程的这点不足。

3.2.4 第三方游戏开发接口设计

3.2.4.1 需求分析

棋牌游戏平台最重要的功能自然是为玩家提供各种各样的棋牌游戏。这些棋牌游戏的种类繁多，就算 QQ 游戏和联众游戏，它们平台上现有的游戏种类也仅仅是众多棋牌游戏中的一部分。棋牌游戏流传于民间，不同地区有不同玩法，本文的棋牌游戏平台设计成可区域运营，就是为了能让各区域的玩家能玩到真正的本地游戏。这些本地特色游戏种类繁多，规则复杂，全由一个开发团队来开发显然不是很合适。因而本文提供了一套开放的第三方游戏开发接口，只要游戏开发者按照这个接口来开发游戏，那么游戏就可以成功地接入到此棋牌游戏平台中来。有了第三方开发接口，就可以让不同的游戏开发者来开发游戏，为那些了解特色游戏具体规则的开发团队提供了便利，大大提高了开发效率。

第三方开发的棋牌游戏，应该通过何种方式和平台交互？常用的方法是，第

三方开发的棋牌游戏以进程的方式接入平台，它和平台之间的通信可以通过进程间的通信来实现，QQ 游戏和联众游戏的客户端都采用了这种方式。这种方法的优点是，棋牌游戏和平台之间的独立性比较强，可以独立对棋牌游戏进行相关维护和更新，而不会影响到平台的正常运行。这种方法也存在着缺点，就是进程间的通信效率相对来讲并不高。为了提高相互之间的通信效率，可以考虑第三方以函数库的形式提供给平台。函数库通常分为静态库和动态库两种，它们之间的优劣我们在第 2 章已经进行了说明，很显然，DLL 的优点更适合用在这里。例如，采用了 DLL 后，可以十分方便地对棋牌游戏进行版本更新，而不需要对客户端或者服务端重新编译发布；如果不采用 DLL，那么客户端需要加载所有的棋牌游戏代码，客户端将变得十分庞大。本文的服务端和客户端的第三方功能，都以 DLL 的形式提供给平台。

3.2.4.2 具体设计

第三方游戏开发接口分为两部分，即服务端的第三方开发接口和客户端的第三方开发接口。既然第三方开发接口是为开发相关棋牌游戏设定的，那么就将功能作如下划分：第三方负责玩家进入桌子之后的逻辑处理，而玩家在桌子外的逻辑由平台处理。也就是说，玩家给服务端发送了请求进入桌子的消息，这时对于服务端来讲，从判断能不能进入桌子开始，接下来的逻辑就由第三方的服务端来实现了。对客户端来说，如果成功加入桌子，这时客户端会跳出游戏界面，从这时开始，客户端的逻辑就由第三方的客户端来负责。玩家退出桌子后，平台接着处理相关逻辑。由于所有棋牌游戏在进入桌子、准备游戏以及退出游戏的模式都类似，因而这类功能全部由平台来实现，通过接口函数供第三方调用。其他与游戏相关的功能，由第三方实现。

以服务端为例，我们介绍一下第三方是如何以 DLL 的形式接入平台的。服务端 DLL 初始化流程可以表示如下：

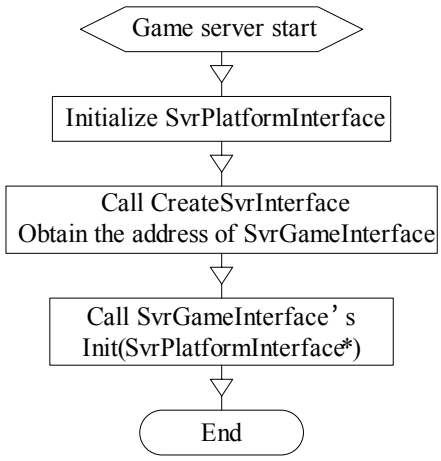

图 3-27 服务端 DLL 初始化流程

Figure 3-27 Process of initialization of DLL on server side

第三方 DLL 的导出函数为 SvrGameInterface* CreateSvrInterface()，该导出函数的作用是返回第三方提供的接口类 SvrGameInterface 的地址。SvrGameInterface 中提供了多个接口函数供平台调用。GS 在初始化时，会调用 DLL 的导出函数以获得第三方提供的接口类，随后调用接口类的初始化函数 Init(SvrPlatformInterface*)，通过初始化函数将平台的接口类地址传给第三方，这样第三方就能调用 SvrPlatformInterface 中提供的平台接口了。通过平台接口和第三方接口之间的交互，就能成功地将第三方游戏接入服务端平台了。对于客户端来说，原理基本上和服务端一样，所不同的是，服务端是在 GS 启动的时候就将第三方 DLL 初始化好了，而客户端是在玩家成功进入桌子以后才初始化客户端第三方 DLL 的。

第三方游戏是一款网络游戏，这样第三方在开发游戏时就不可避免地需要在第三方客户端和第三方服务端之间传递消息。这里规定，所有第三方之间传递的消息都通过平台来实现传送，所有这类消息都封装成 MSG_GAMEMSG 类型的 Message，通过平台客户端和服务端提供的发送消息接口，第三方就可以实现发送消息功能。例如第三方客户端需要给第三方服务端发送一条消息，第三方客户端在封装好要发送的消息之后（第三方自行定义第三方使用的消息），调用客户端平台的发送消息接口，平台就将消息封装成 MSG_GAMEMSG 格式的 Message 发送给服务端。服务端平台收到此消息后，经解析发现是第三方消息，于是就将消息内容转发给第三方服务端，这样就实现了第三方客户端和服务端之间的消息传送。

根据上面的分析，可以把第三方接口和平台之间的关系表示为下图：

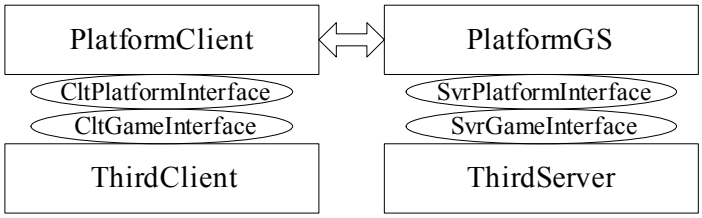

图 3-28 第三方游戏接入平台

Figure 3-28 Interface for third-part

附录二中给出了第三方游戏开发接口的详细描述。下面我们就参照附录二，说明在一些游戏主要的逻辑处理过程中，第三方和平台是怎样配合来实现具体功能的。

首先从玩家加入桌子开始。玩家请求加入桌子，客户端平台发送加入桌子消息给服务端平台，服务端平台收到此消息后，进行判断，若玩家可以进入桌子，那么服务端平台就会调用 SvrGameInterface 中的 OnTableEnter 接口告诉第三方服务端有玩家进入桌子了。同时，服务端平台会返回加入成功的消息给客户端平台，

告诉玩家加入桌子成功。于是客户端平台调用 CltGameInterface 中的 OnSelfEnterTable 接口，第三方客户端启动游戏桌界面，玩家便成功进入桌子。如果这张桌子上原来还有其他的玩家，那么服务端平台在判断玩家成功加入桌子的时候，会给桌上的其他玩家广播一条有玩家进入的消息。其他玩家的客户端平台收到此消息时，就会调用 CltGameInterface 的 OnTableEnter 接口，告诉第三方客户端有玩家进入桌子。

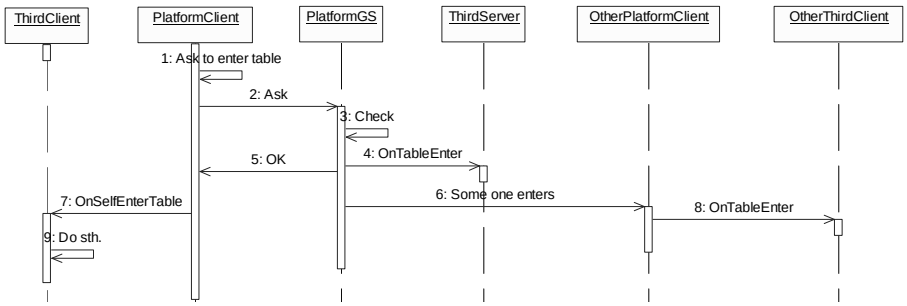

图 3-29 玩家进入桌子时，平台与第三方的交互流程

Figure 3-29 Process of entering table

玩家进入桌子之后，点击游戏界面上的“准备”按钮，第三方客户端调用 CltPlatformInterface 中的 UserReady 接口，客户端平台于是向服务端平台发送准备消息。服务端平台收到此消息后，改变玩家状态，同时调用 SvrGameInterface 的 OnPlayerReady 接口。如果此时桌子上还存在其他玩家，那么服务端平台会向相应客户端平台发送有玩家准备的消息，客户端平台收到此消息后，调用 CltGameInterface 的 OnPlayerReady 接口。

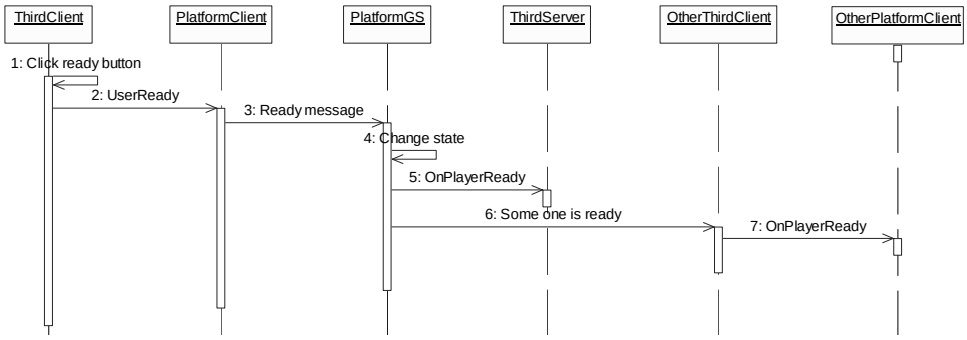

图 3-30 玩家准备时，平台与第三方的交互流程

Figure 3-30 Process of being ready

玩家进入准备状态后，如果此时游戏还没有开始，那么玩家可以选择取消准备。第三方客户端调用 CltPlatformInterface 中的 UserCancelReady 接口，客户端平台于是向服务端平台发送取消准备的消息。服务端平台收到此消息后，改变玩家状态，同时调用 SvrGameInterface 的 OnPlayerCancelReady 接口。如果此时桌上还存在其他玩家，那么服务的平台会向相应的客户端平台发送有玩家取消准备

的消息，客户端平台随后调用 CltGameInterface 的 OnPlayerCancelReady 接口。取消准备的流程图和准备流程图类似，不重复给出。

当桌上的玩家状态达到游戏开始的要求时，第三方服务端判定游戏开始，这时调用 SvrPlatformInterface 中的 NotifyGameStart 接口，服务端平台此时会发送游戏开始的消息给桌上所有玩家的客户端平台。客户端平台收到此消息时，调用 CltGameInterface 的 OnGameStart 接口，游戏开始。

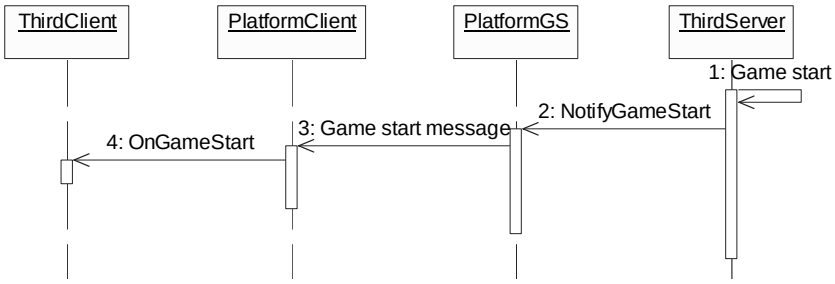

图 3-31 游戏开始时，平台与第三方的交互流程

Figure 3-31 Process of starting game

游戏开始后，玩家在游戏中会进行一系列操作，导致第三方服务端和第三方客户端之间需要相互进行通信。第三方客户端向第三方服务端发送消息时，调用 CltPlatformInterface 的 SendGameMsg 接口，客户端平台会将游戏消息发送给服务端平台。服务端平台收到 MSG_GAMEMSG 类型的消息时，知道是第三方客户端发来的游戏消息，于是就调用 SvrGameInterface 的 OnGameMessage 接口，将游戏消息交给第三方服务端处理。第三方服务端向第三方客户端发送消息的流程类似。

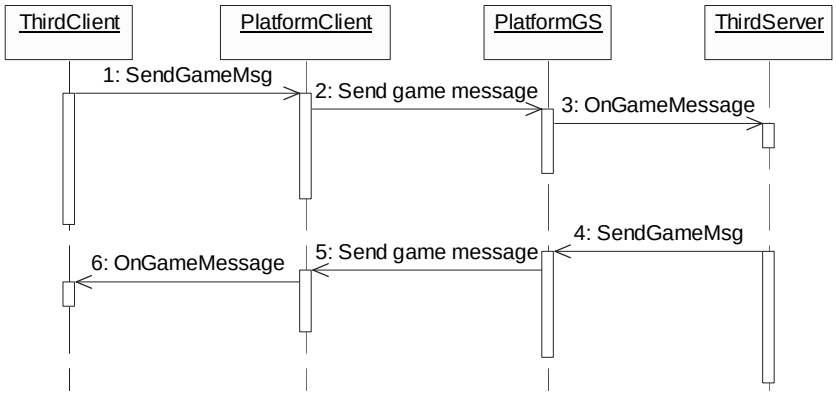

图 3-32 游戏过程中，平台与第三方的交互流程

Figure 3-32 Process of gaming

第三方服务端判定游戏结束时，调用 SvrPlatformInterface 中的 NotifyGameEnd 接口，服务端平台此时会发送游戏结束消息给桌上所有玩家的客户端平台。客户端平台收到此消息时，调用 CltGameInterface 的 OnGameEnd 接口，游戏结束，显示结果。具体流程图和游戏开始的流程图类似，从略。

玩家点击第三方游戏界面上的“退出”按钮时，第三方客户端调用 CltPlatformInterface 的 UserExit 接口，客户端平台发送退出桌子消息给服务端平台时，服务端认为玩家可以正常退出桌子，于是调用 SvrGameInterface 的 OnTableLeave 接口。如果此时桌上还有其他玩家，服务端平台会向其他玩家的客户端平台发送有玩家退出桌子的消息。其他玩家的客户端平台收到此消息后，会调用 CltGameInterface 的 OnTableLeave 接口。

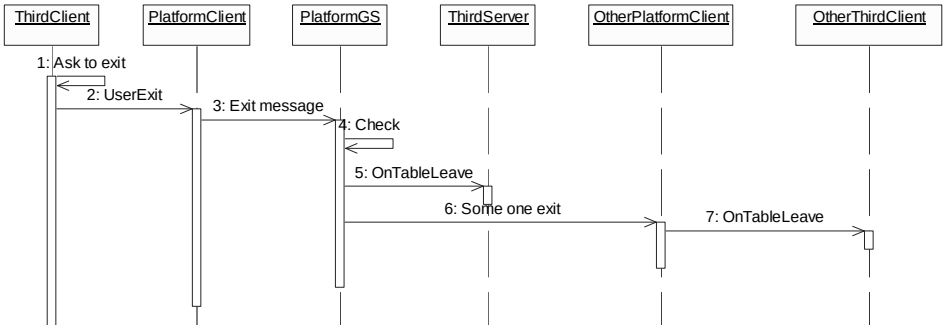

图 3-33 玩家退出桌子时，平台与第三方的交互流程

Figure 3-33 Process of exiting table

如果玩家在游戏过程中就要求退出，服务端会将玩家这种行为判断为逃跑，那么在上面流程中的第 5 步，就调用 SvrGameInterface 的 OnPlayerRunAway 接口。第三方服务端决定扣除玩家多少积分进行惩罚，再回调 SvrPlatformInterface 中的 NotifyPlayerRunAway 将惩罚结果告诉服务端平台。这时，服务端平台给桌上的其他玩家的客户端发送玩家逃跑消息。其他玩家的客户端平台收到此消息时，调用 CltGameInterface 的 OnPlayerRunAway 接口。

玩家在游戏中会发生意外断线的情况。服务端平台发现玩家意外断线时，会调用 SvrGameInterface 中的 OnUserDown 接口，同时服务端平台向桌上的其他玩家的客户端平台发送 MSG_USERDOWN 消息。其他玩家的客户端平台收到此消息后，就会调用 CltGameInterface 中的 OnUserDown 接口。玩家发生意外断线后，在规定的时间内回来了。服务端平台在收到玩家进入桌子的消息时，发现玩家是属于断线重入，于是就调用 SvrGameInterface 的 OnPlayerReCon，同时，服务端平台会返回加入成功的消息给客户端平台，告诉玩家加入桌子成功。于是客户端平台调用 CltGameInterface 中的 OnSelfEnterTable 接口，第三方客户端启动游戏桌界面，玩家便成功断线重入。此时，服务端平台还会给桌上其他玩家的客户端平台发送玩家重新进入的消息。客户端平台收到此消息后，就会调用 CltGameInterface 中的 OnPlayerReCon 接口。

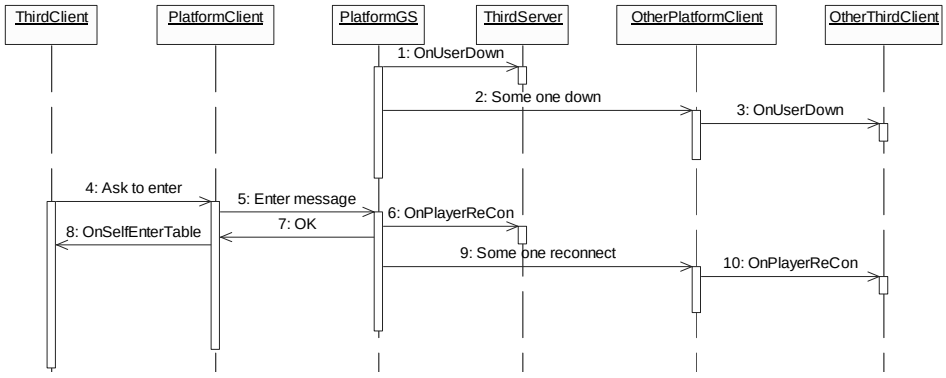

图 3-34 断线重入过程中，平台与第三方的交互流程

Figure 3-34 Process of reconnection

游戏结束时，玩家的战绩信息会发生变化。这时如果玩家还在同一游戏的其他游戏房间中游戏，服务端平台会通过同步机制来进行相应更新。其他 GS 会收到对应消息，于是调用 SvrGameInterface 的 OnUserScoreChange 接口。同时 GS 还会给相应客户端平台发送游戏结果消息。客户端平台收到此消息时，就会调用 CltGameInterface 的 OnUserScoreChange 接口。

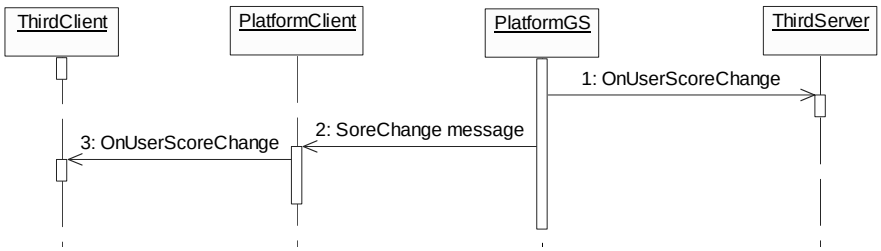

图 3-35 战绩信息同步时，平台与第三方的交互流程

Figure 3-35 Process of score synchronization

3.2.4.3 小结

通过上面的分析可以看出，这套第三方游戏开发接口，满足了一般棋牌游戏的基本功能要求，根据此开发接口开发出来的棋牌游戏，可以很好地接入到平台中来。第三方棋牌游戏，以 DLL 的形式提供给平台，不但方便第三方对棋牌游戏进行升级维护，还节省了应用程序所占用的空间，较之进程形式具有更高的效率，可以作为传统进程形式的一种替代形式。

3.2.5 服务端软件架构设计

3.2.5.1 需求分析

UML 是在软件工程中占主导地位的标准建模语言，特别在 OOA 和 OOD 领域，UML 更是成为了事实上的工业标准。UML 广泛应用于软件工程的各个阶段，

从需求分析、设计、开发到测试维护，UML 规范了软件工程，提高了软件开发的效率。我们在对服务端进行软件架构设计时，也使用了 UML。主要使用了 UML 的序列图来进行需求分析，如 3.2.1.3 中就使用了序列图来进行用例分析和流程设计，3.2.5.2 中使用了序列图来说明平台和第三方的交互流程；还使用了 UML 的静态类图来进行服务端的软件架构设计，本节就采用 UML 来进行服务端的类图设计。

面向对象设计最重要的特点是模块的可复用性，我们在进行设计时，也充分考虑了这个特点。我们设计了一个服务器基类，所有的服务器类都从该基类派生。服务器基类具备一般功能服务器的基本功能。我们在基类中设计了基于 MsgHandler 的消息处理模块，该模块可根据需要在服务器类中设置多个，满足了服务器处理来自多方消息的要求。服务器基类具有很强的可复用性，可应用于其他网络服务应用程序。

面向对象设计不像传统的软件设计那样抽象，面向对象设计中的“对象”可以和实际的事物关联起来，例如这里一个功能服务器就可以看成一个对象，服务器上的玩家信息、游戏信息等又可以看成其他种类的对象。通过使用面向对象设计中的聚合、组合等方法，可以很方便地设计出各个功能服务器。

3.2.5.2 具体设计

鉴于各个功能服务器的功能基本一致，这里设计一个服务器基类，对服务器的工作模式进行了封装和抽象，规定所有功能服务器都从此类派生。服务器基类部分定义如下：

```
class Server
{
public:
  Server();
  virtual ~Server();
  void Start();                          //< 开启服务器
  virtual void Stop();                   //< 停止服务器
  ...
protected:
  virtual bool Initialize() = 0;         //< 初始化函数
  virtual void LoopFunc(){}              //< 定时循环功能
  void MainLoop();                       //< 主线程
  void AddHandler(MsgHandler* handler); //< 添加handler
  vector<MsgHandler*> handlers;          //< 存放handler的向量表
  ...
};
```

基类中最重要的是 MsgHandler。服务器需要处理来自具体网络层的消息，这就需要抽象一个类来完成这个工作。这个类应该提供取消息、发消息以及处理消息等接口，因而将 MsgHandler 定义为：

```
class MsgHandler
{
public:
  virtual bool Initialize() = 0;                    //< 初始化函数
```

```
    virtual Message* GetMsg() = 0;                          //< 从inQueue中取出一条消息
    virtual void SendMsg(Message* msg) = 0;                 //< 发送消息
    virtual void HandleMessage(Message* msg) = 0; //< 处理消息
};
```

具体的网络底层都会有一个具体的 MsgHandler 派生类与之对应。派生类中应该持有具体的 NetBase，然后通过 NetBase 提供的相关接口来实现基类 MsgHandler 中的 GetMsg 和 SendMsg，而 HandlerMessage 则根据处理的具体消息来实现。

Server 基类中有一个用来保存 MsgHandler 对象的向量表，通过 AddHandler 就可以将相应的 MsgHandler 对象加入其中。各个服务器在初始化时，根据表 3-1，生成相应的 MsgHandler，再将其加入到向量表中，这样服务器就能处理来自不同网络层的消息了。具体的功能主要由 Server 基类中的主循环 MainLoop 实现，MainLoop 代码如下：

```
void Server::MainLoop()
{
    Message* msg = NULL;
    bool isSleep;
    while(1){
        isSleep = true;
        for(unsigned int i=0;i<handlers.size();i++){      //< 逐个处理handler
            msg = handlers[i]->GetMsg();                  //< 取消息
            if(msg != NULL){
                isSleep = false;
                handlers[i]->HandleMessage(msg);          //< 处理消息
                delete msg;
                msg = NULL;
            }
        }
        LoopFunc();             //< 附加功能
        if (isSleep) {
            Sleep(interval);
        }
    }
}
```

从上面的代码可以看出，服务器会不停地将各个网络层中的消息取出来，随后进行处理。代码中提供了 LoopFunc()函数，此函数用来完成一些额外的需要定时激活的功能。比如 GS 每隔 1、2 秒向客户端广播房间内玩家信息的变化，就是在此函数里实现的。

这样，整个 server 软件架构设计的类图可以表示如下：

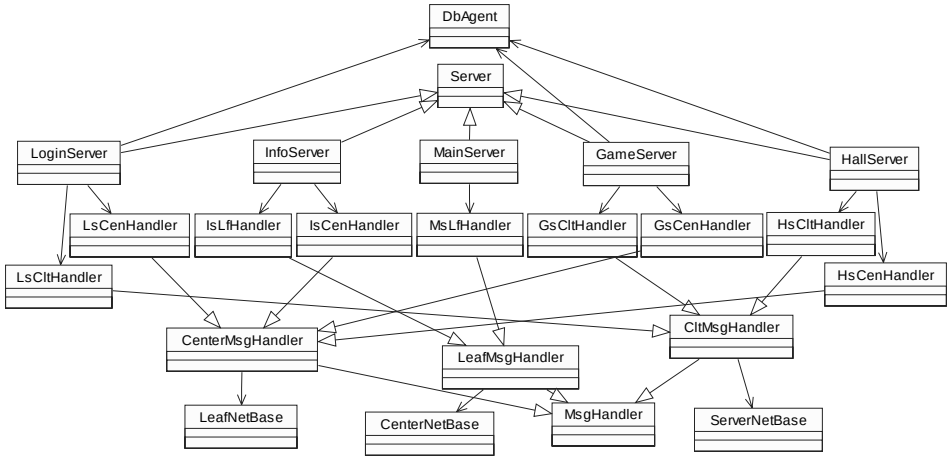

图 3-36 服务端软件架构总体设计类图

Figure 3-36 Class structure of server

下面以 LS 为例，看一下功能服务器是怎么设计的。首先，LS 作为一个功能服务器，必须从 Server 基类继承。其次，根据表 3-1 可知，LS 存在 ServerNetBase 和 LeafNetBase 两个网络层，这样就必须为 LS 从 MsgHandler 派生两个类来分别处理来自客户端的消息和来自中心节点的消息。主类 LoginServer 从 Server 派生，除了持有数据库服务类 DbAgent 外，还持有两个 MsgHandler 对象——LsCltHandler 和 LsCenHandler。LsCltHandler 从 CltMsgHandler 派生，持有 ServerNetBase 对象，用于和客户端进行通信；LsCenHandler 从 CenterMsgHandler 派生，持有 LeafNetBase 对象，用于和中心节点进行通信。这样，实现 LS，基本上就是实现两个 MsgHandler 里的 HandleMessage 函数。其他功能服务器的实现基本上和 LS 一样。各个服务器之间的区别，就在于消息处理函数的功能不同。

各个功能服务器上除了存在 MsgHandler 之外，还有一些数据结构，下面就介绍各个服务器上的数据结构设计。

◆ MS 数据结构设计

根据前面的分析我们知道，MS 主要是用来维护平台内的游戏房间列表信息，以及维护所有登录玩家的信息，由此，我们将 MS 上的数据设计如下：

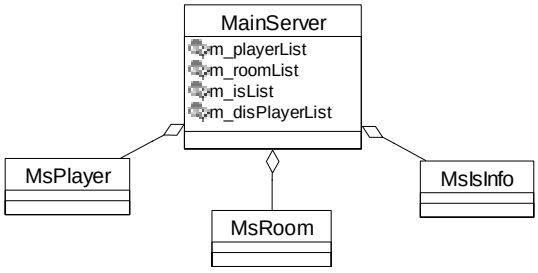

图 3-37 MS 上的数据

Figure 3-37 Data on MS

MS 上存在四个信息列表，这四张表的主要功能是：

- m_playerList
  玩家成功登录游戏平台，在该列表中添加此玩家的相应信息。玩家登录时，会到 MS 来验证是否重复登录，MS 查询此列表，若已存在该玩家，则判为重复登录，否则就在列表中加入该玩家的信息，通过登录验证。玩家在退出平台时，将玩家信息从此表中删除。
- m_disPlayerList
  用于存放游戏过程中意外断线的玩家。玩家意外断线时将玩家从正式列表中移出并加入此表，同时开始计时，规定时间内玩家回来了，则返回至正式表，否则移除。
- m_roomList
  这个列表用来保存平台内所有开启的房间信息。GS 在向 IS 发起连接时，IS 会将这个信息告知 MS，MS 将房间信息加入列表。GS 与 IS 断开连接时，IS 同样会通知 MS，从列表中删除该房间信息。IS 与 MS 断开连接时，删除该 IS 下的所有房间信息。
- m_isList
  维护已经连接的 IS。IS 与 MS 建立连接后，就将 IS 的有关信息加入到该列表中。IS 与 MS 断开连接时，将相应 IS 的信息从该列表中删除。

◆ IS 数据结构设计

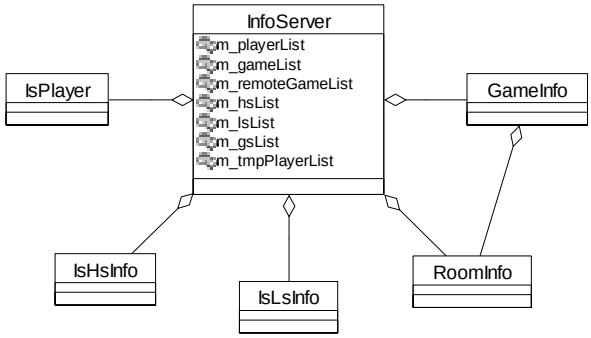

图 3-38 IS 上的数据

Figure 3-38 Data on IS

- m_playerList
  玩家与该 IS 下的 HS 成功连接后，或者在该 IS 下的某个房间里，则将该玩家的信息加入此列表中。若玩家没有与该 IS 下的任何 HS 连接，则玩家退出该 IS 下所有房间后，将该玩家信息从队列中删除；若玩家与此 IS 下的某个 HS 连接，则在玩家退出游戏与 HS 断开连接时，将该玩家信息

从队列删除。

- m_tmpPlayerList
  玩家在通过 LS 的验证之后，会向相应的 HS 发起连接，这时 HS 需要到 IS 上查询玩家是否已经通过登录验证。在这个过程中，IS 上必须保存玩家的相关信息以方便 HS 的查询。在玩家通过 LS 验证之后、与 HS 建立连接之前，玩家的相关信息是保存在这个临时列表中的。与 HS 建立连接之后，转入正式列表中。临时列表会对其中的玩家进行计时，超过一定时间就会删除，同时告诉 MS 删除相关登录信息。
- m_hsList
  维护已连接 HS 的信息。HS 与 IS 建立连接后，就将 HS 的有关信息加入到该列表中。HS 与 IS 断开连接时，将相应 HS 的信息从该列表中删除。
- m_lsList
  功能同上。
- m_gsList
  功能同上。但 IS 需要记录各个 GS 上的人数情况。
- m_gameList
  单纯地将 IS 下的 GS 罗列在一个列表中是不够的，因为有的时候需要对某类游戏进行某种特殊的处理，因而有必要对 GS 按照游戏类别分类。这个列表就是这个功能。GS 在向 IS 发起连接的同时，不但将 GS 加入到 m_gsList 中，还根据 GS 的游戏类别，将 GS 的指针存放到此列表的相应类别中。此列表中存放的 GS 指针指向 m_gsList 表中的相应 GS。GS 与 IS 断开连接时，删除表中相应内容。
- m_remoteGameList
  这个列表的功能和 m_gameList 功能类似，所不同的是这个列表中保存的是其他 IS 下的游戏房间。该列表由 MS 负责维护更新。IS 在刚启动时，MS 会将其他 IS 的房间列表发给此 IS，随即 IS 将房间按照类别存入到该列表中。其他 IS 增加房间时，通过 MS，IS 将房间加入列表中。其他 IS 减少房间的情况类似。每隔一段时间，MS 会将其他 IS 上的最新玩家数目传过来，IS 作相应更新。

◆ LS 数据结构设计

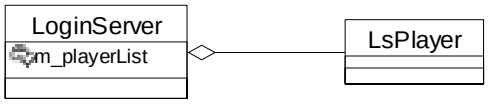

图 3-39 LS 上的数据

Figure 3-39 Data on LS

- m_playerList
  玩家在登录过程中，会在 LS 上的本列表中保留相关信息；登录完成后，玩家与 LS 断开连接，LS 删除列表中的相应玩家信息。

◆ HS 数据结构设计

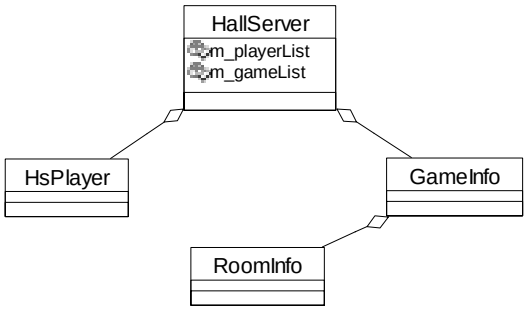

图 3-40 HS 上的数据

Figure 3-40 Data on HS

- m_playerList
  玩家与 HS 建立连接之后，将玩家加入该列表中；玩家退出游戏，与 HS 断开连接，HS 删除列表中的相应玩家信息。
- m_gameList
  此列表的功能与 IS 上的 m_gameList 功能类似，所不同的是，这个列表中保存的不仅仅是本 IS 下的房间信息，而是平台下所有房间的信息。平台在增减房间时，会通过 MS、IS 通知 HS 来进行相应的列表修改。列表中房间的人数信息通过 HS 和 IS 之间的交互定期更新。客户端游戏列表的信息，都是从这个列表中获得。

◆ GS 数据结构设计

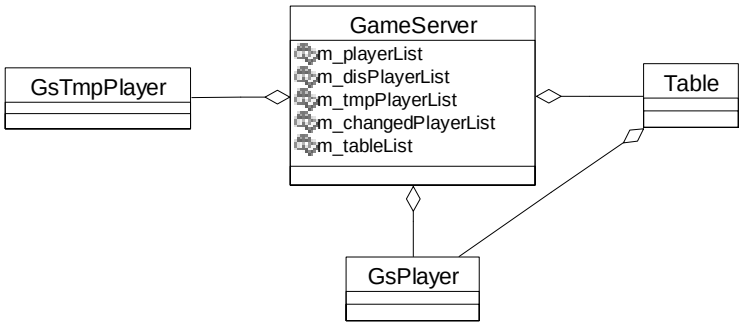

图 3-41 GS 上的数据

Figure 3-41 Data on GS

- m_playerList
  玩家成功与 GS 建立连接之后，就将玩家的信息加入此列表中；玩家退出房间时，将玩家从该列表中删除。
- m_disPlayerList
  此列表用于玩家的意外断线处理。玩家发生意外断线后，将玩家从正式列表移至该列表中，玩家在规定时间内回来，则从此列表移至正式列表中；超过规定时间，将玩家删除。
- m_tmpPlayerList
  玩家在与 GS 发起连接时，需要到 IS 上作合法性验证，连接异地 GS 时，更是要通过 MS 的验证。在这个过程中，我们将玩家保存在这个临时列表中。通过验证后，将玩家移至正式列表。
- m_changedPlayerList
  我们在分析“房间内信息刷新”这个用例时，曾经提到过“定时广播差量法”，这个算法里的特殊列表就是本列表。
- m_tableList
  此列表用来保存房间内桌子信息。每个桌子中有相应的玩家列表，用来保存在桌上游戏的玩家。当玩家进入桌子时，将玩家信息的指针加入此玩家列表中，该指针指向 m_playerList 中的相应玩家信息。玩家退出桌子时，删除指针即可。

3.2.5.3 小结

在对服务端进行软件架构设计时，我们充分利用了面向对象设计的特点，不但详细设计了各个功能服务器上的数据结构，还为整个 server 设计了一套复用性很强的软件架构。这套软件架构可以用于其他网络应用程序，具有很强的通用性。UML 工具的使用，也大大方便了类图设计，提高了软件设计效率。

3.3 本章小结

本章主要给出了一款棋牌游戏平台服务端的详细设计。在章节开始，我们首先分析了在设计过程中需要解决的问题，随后根据这些问题，从架构、通信协议、多线程模型和第三方开发接口等方面一一进行了设计。在解决了这些问题的基础上，我们利用 UML 设计了整个服务端的软件架构，并给出了各个服务器上具体的软件设计。

第 4 章 实现结果及其性能分析

4.1 实现结果及其特点

根据本文的设计方案，我们成功设计出了一款棋牌游戏平台的服务端。下面就通过客户端的相关界面，来给出本文的实现结果。本文的棋牌游戏平台，采用了和其他棋牌游戏平台相似的格式，游戏由大厅、房间和游戏桌构成。图 4-1 至图 4-4 给出了我们棋牌游戏平台的实际游戏界面。图 4-1 是客户端的登录界面，玩家可以手动选择某区域的登录服务器；图 4-2 就是成功登录以后的大厅界面；图 4-3 是房间内的界面，图中的玩家是用来进行压力测试的机器人；图 4-4 是一款游戏的游戏界面，这里选用的是斗地主游戏。

图 4-1 登录界面

Figure 4-1 Login UI

图 4-2 游戏大厅

Figure 4-2 Hall of game

图 4-3 游戏房间（机器人压力测试中）

Figure 4-3 Room of game (Being tested by robots)

图 4-4 游戏桌（斗地主）

Figure 4-4 Table of game (Game DouDiZhu)

本文提出的棋牌游戏平台服务端的设计，采用了星型结构作为基本的分布式系统结构。星型结构具有结构简单、便于管理、扩展性好等优点。利用星型结构的这些优点，我们的平台实现了区域性运营的功能。星型结构使得我们的服务端具备了较好的扩展性，达到了网络应用程序的基本要求。不但可以对区域服务器组进行扩展，还可以对区域服务器组中的各个功能服务器进行扩展。另外，位于星型结构中的叶节点之间的相关性小，对某个功能服务器进行调整不会影响到其他服务器的运行，位于非中心节点的服务器崩溃，也不会影响到其他服务器的正常运行，这些也是星型结构所带来的好处。

星型结构最大的缺点就是存在中心节点的瓶颈问题。我们在设计过程中充分考虑了这一点，采用一种类似于镜像服务器的方法一定程度上解决了中心节点的瓶颈问题。具体的实验数据也证明了我们设计中所采用方法的合理性。在下一小节中，我们将通过具体实验数据来说明我们比较好地解决了星型结构中心节点的瓶颈问题。

通信协议方面，对客户端的通信处理，我们采用了大规模网络应用中比较常用的完成端口模型。完成端口模型是 Windows 平台中一种常用的 Winsock I/O 模型，主要适用于大规模网络通信访问的应用。完成端口特别适用于需要管理上千个套接字的情况，而棋牌游戏平台正好符合这样的特征。通过使用完成端口，提升了服务端的性能。我们为服务端之间的通信专门设计了一套通信协议。虽然目前市面上存在多种应用于分布式系统的网络中间件，但为了节省成本以及开发更适合自己的通信协议，我们设计了 SDCP。SDCP 很适合用于我们的星型系统架构。

在我们的架构中，服务器都是软服务器。软服务器不仅仅使得服务器的配置更加直观和灵活，更重要的是使服务端能运行在多进程的工作模式，提升了服务端的性能。为了使服务端的数据更加安全，同时也为了降低开发难度，我们在设计服务器的工作线程时，采取了单线程处理消息的模型。

为了能让更多的开发团队来为棋牌游戏平台开发棋牌游戏，我们设计了一套第三方游戏开发接口。只要按照这套接口来进行棋牌游戏的开发，那么开发出来的游戏就能加入到平台里。棋牌游戏种类繁多，有了第三方游戏开发接口，就能加快游戏的开发进度，同时还实现了棋牌游戏的模块复用。设计中规定第三方开发的游戏部分以 DLL 的形式提供给平台。使用 DLL 主要有两个优点，其一，方便对棋牌游戏的版本进行维护和升级，要对游戏进行维护，只要更新相应的 DLL 即可，没有必要对平台进行修改或是重新编译；其二，减小了客户端和服务端的占用空间，客户端只会在需要时才加载相应的客户端 DLL，而服务端更可以做到多个 GS 共享一个 DLL，大大节省了内存空间。

服务端的一些特点，总结如下：

- 高效性

  由于多个软服务器可以部署在同一台物理服务器上，这样不但实现了多进程工作的情况，提高了效率，还可以根据运营时的负载情况，合理地部署软服务器，充分利用物理服务器资源。我们将游戏房间设计成单独的可执行文件 GS，房间的概念比较清晰，这样运营商在对服务端进行部署时，可以十分快捷地增减房间。另外，我们开发时使用的是 C++这种效率比较高的编程语言，在开发过程中我们也充分利用了它的高效性。比如接收消息线程从网络上接收到消息以后，通过 C++强大的内存管理能力，我们就能直接对消息所在的内存进行一系列操作，而不用再将消息中的信息再次复制出来使用，尽量减小内存拷贝的次数，也极大地提高了系统的性能。[39]

- 可扩展性

  我们的服务器架构，以星型结构为基础，提出了 SDCP 通信协议。通过

SDCP 协议，可以方便地对各个叶节点的功能服务器进行扩展。需要增加区域服务器组时，可以在不影响其他区域服务器组的情况下，动态地增加区域服务器组。而对于区域服务器组内而言，也可以根据需要动态扩展 LS、HS 和 GS 等功能服务器。我们的系统中，IS、LS、HS 和 GS 都有非常大的扩展空间，这使得平台能够承受更多的用户量。[39]

- 安全性

  在我们的系统平台中，所有的逻辑处理都是在服务端完成的，保证了最起码的安全性。玩家游戏时只能够与 LS、HS 以及 GS 发起连接和通信，而不会与 IS、MS 以及数据库这些核心服务器发生交互。核心服务器对玩家的透明化，大大提高了系统的安全性。另外，在设计开发过程中，我们充分考虑了玩家的合法性验证，比如在玩家登录 LS、HS 和 GS 时，我们都做了相应的合法性检验；服务器之间的连接，我们也有相应的合法性验证，这些都提高了系统的安全性。当然，我们还可以对某些重要的通信消息进行加密，进一步提高系统的安全性。[39]

- 通用性

  在设计之初，我们就将这个游戏平台定位在了通用性平台这个目标上。为了实现平台的通用性，我们为客户端和服务端分别定义了一套开放的第三方游戏开发接口。如果第三方开发者能按照这套接口去开发游戏逻辑，那么这些游戏就能很好地集成到我们的平台中来。我们在 2006 年的 7、8 月份，已经先后开发了 7 款不同的棋牌小游戏，而且其中有两款是由别的开发团队开发的。这些游戏都按照这套第三方接口的标准开发，通过 DLL 的形式很好地跟平台集成了起来。[39]

- 容错性

  我们在平台的开发和测试过程中，不断地改进系统的容错性。对于客户端来讲，当客户端意外断线时，我们的服务端在 MS、GS 上都作了比较合理的处理。对服务端来说，在 MS 启动的前提下，无论什么时候，我们都能增加区域服务器组，动态增加所引起的变化，都通过 MS 来进行了同步处理。当区域服务器组出现问题需要关闭时，通过 SDCP 协议以及 MS 的同步处理，也不会影响到其他区域服务器组。区域服务器组内的情况也是一样，通过 SDCP 协议和 IS 以及 MS 的同步处理，区域服务器组内的 LS、HS、GS 在发生错误时，不会影响到整个服务器组的运行，例如某个 GS 发生问题崩溃了，这时影响的仅仅是该房间内的玩家，而其他房间的玩家不会受到任何影响。可靠的容错性，增强了游戏过程的流畅性，也提高了运营商使用的便利性。

- 负载均衡

对于网络游戏服务端来说，负载均衡是一个不可回避的问题。由于很多网络游戏服务器都采用了分布式的系统架构，通常会有多个服务器用来实现同一种功能，如何实现玩家访问这些服务器的均衡性，就是负载均衡需要解决的问题。我们的系统架构中，典型的需要负载均衡的服务器有 LS 和 HS。对于 LS 来讲，我们采用了 DNS 解析的方法来实现负载均衡。而对于 HS 来讲，它们的负载均衡是通过 IS 来决定的。这两种不同的方法虽然都比较简单，但都达到了比较好的负载均衡效果。另外，对于 GS 来讲，由于玩家选择进入哪个游戏房间都有比较明确的目的性，因而这里就不能通过某种算法来实现 GS 的负载均衡了。要实现 GS 的负载均衡，只能在实际运营的过程中，根据具体的负载情况，合理地部署 GS 来实现负载均衡。我们的 GS 是一个个的软服务器，这样就能很方便地部署 GS，从而实现 GS 的负载均衡。

- 工作模式可复用

  我们在设计中提出的服务端工作模式，具有网络应用程序的一般性，可以用于其他网络游戏以及网络应用程序的开发。在我们项目组内，我们用同样的服务端工作模式就正在开发一款 MMORPG。我们提出的单线程处理的工作模式，在开发过程中具有很好的便利性。服务器内的 MsgHandler 模块，可以随意地增加不同种类，这也进一步增强了这种工作模式的复用性，实际使用中，可以根据需求增加 MsgHandler，从而可以处理更多的网络层消息，方便服务端网络结构的重组和扩展。

4.2 性能测试及其分析

前面曾经提到，星型结构中心节点的瓶颈问题是本设计面临的一个比较棘手的问题。为了尽量降低中心节点的负载，我们在设计中运用了一种类似于镜像服务器的方法。我们将中心节点 IS、MS 的部分信息在各个 HS 上都作了镜像，这样，所有需要到 IS 和 MS 上取得的信息，在 HS 上就可以获得，很大一部分负载被分派到了叶节点上，而叶节点不存在瓶颈问题。通过这样的方法，一定程度上解决了中心节点的瓶颈问题。下面通过测试数据，来具体进行说明。

为了测试本文提出的棋牌游戏平台服务端的性能，我们专门开发了一个虚拟客户端来模拟大量用户的访问。由于服务端是消息驱动的，因而虚拟客户端只要不断给服务端发送不同的消息就可以达到模拟用户的目的。虚拟客户端采用多线程实现，一个线程模拟一个用户。每个线程每隔 5 秒向服务端发送一条消息，消息从登录消息开始发送，随后依次模拟获取房间信息、进入房间、进入桌子准备游戏、退出游戏和桌子、退出房间、退出客户端，之后不断重复上述过程。由于模拟游戏逻辑相对复杂，这里就不进行游戏逻辑的模拟测试了。

测试将通过模拟不同数量的用户进行不停地登录、进出房间、进出游戏桌，来测试各个功能服务器在大量用户访问时的性能表现。为了使结果具有可比性，这里将待测试的功能服务器放置在同一台 PC 机上，其他功能服务器则安排到其它 PC 机上，多次测试以获得不同服务器在同一 PC 机上的性能表现。该 PC 机的配置如下表：

表 4-1 测试用 PC 机配置信息

Table 4-1Hardware information of the PC used for test

CPU	AMD Athlon64 2800+
内存	512M
操作系统	Windows XP
网络环境	100MBit 局域网

经多次测试，MS、IS、LS、HS、GS 在此 PC 机上的性能表现如下图所示，各图中，左图为 200 用户测试结果，右图为 400 用户测试结果，图中横轴为时间，整个横轴约为 4 分 45 秒，纵轴为 CPU 占用率，最高为 100%。

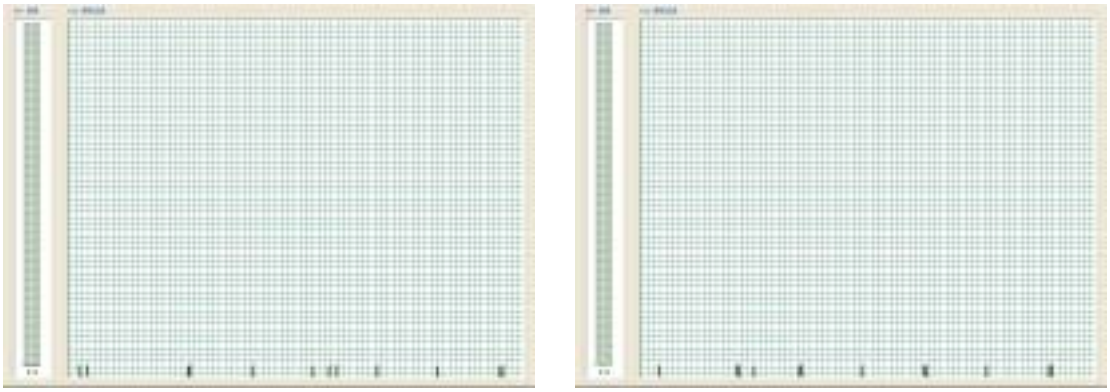

图 4-5 MS 测试结果

Figure 4-5 Test result of MS

图 4-6 异地登录时的 MS 测试结果（400 人）

Figure 4-6 Test result of MS when 400 users login from another region

图 4-7 IS 测试结果

Figure 4-7 Test result of IS

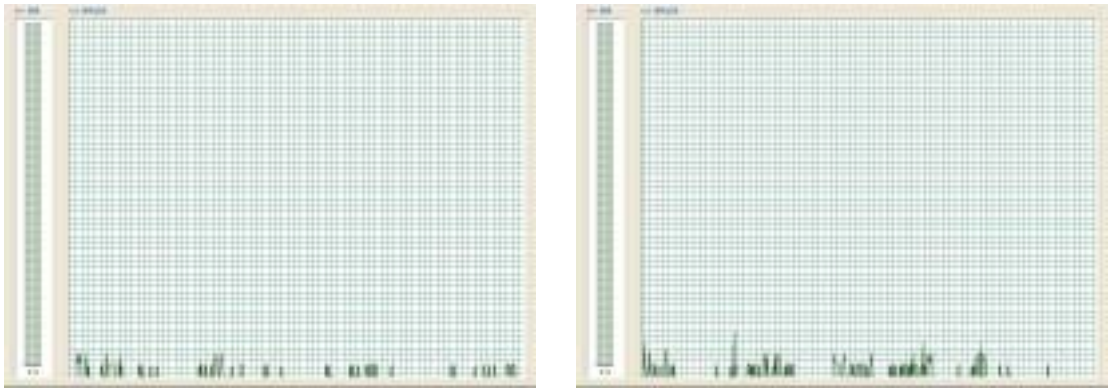

图 4-8 LS 测试结果

Figure 4-8 Test result of LS

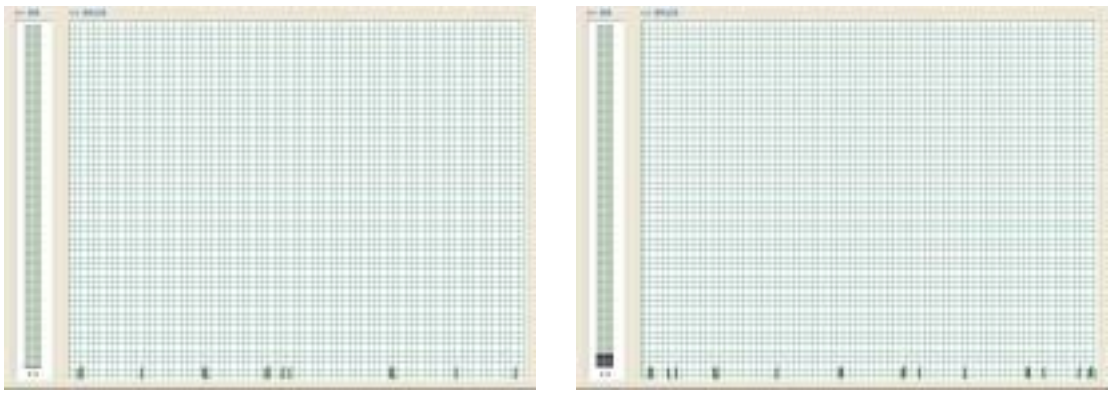

图 4-9 HS 测试结果

Figure 4-9Test result of HS

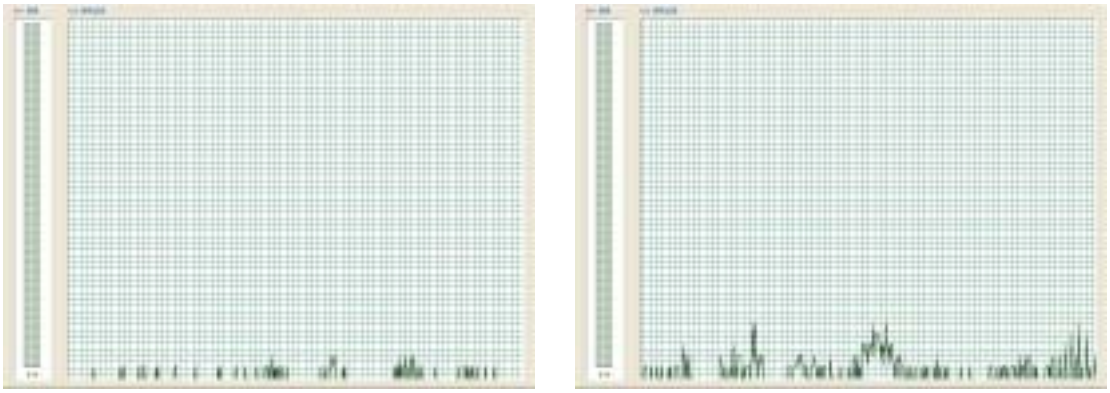

图 4-10 GS 测试结果
Figure 4-10 Test result of GS

由于计算机资源有限，而模拟客户端需要占用大量的计算机资源，因而并没有能够进行大规模的测试，仅仅采用了 200 人和 400 人两种测试用例。另外，由于模拟客户端采用的是阶段性向服务端发送消息的策略，比如，首先所有模拟用户发送登录消息，5 秒之后再集体发送进入大厅请求，5 秒后又发送别的消息，这样就导致服务器处理消息时会呈现周期性的特点，这也造成了测试结果中 CPU 响应带有明显的周期性和脉冲性。

MS 的负载相对比较小，CPU 占用率峰值在 4%左右，而且随着人数从 200 增加到 400，MS 的负载几乎没有变化。异地登录时，会一定程度地增加 MS 的负载，从测试结果来看，CPU 占用率峰值也在 4%左右，脉冲数略微增多，变化不是很明显。IS 的情况和 MS 比较类似，CPU 占用率的峰值也在 4%左右，而且人数增加后，IS 的负载也没有发生很明显的变化。LS 的负载稍大，200 人时的 CPU 占用率峰值在 7%左右，400 人时的负载增大比较明显，CPU 占用率峰值达到了 10%左右，而且脉冲数明显变多。HS 的负载比 LS 的负载要轻些，而且 200 人和 400 人时的变化不是很明显，CPU 占用率峰值在 4%左右。GS 的负载是最大的，脉冲数很多，而且人数增加时变化明显，200 人时 CPU 占用率峰值在 7%左右，400 人时则达到了 15%。

通过测试结果可以看出，本文提出的棋牌游戏平台的服务端架构虽然是基于星型结构的，但由于设计过程中充分考虑到了中心节点的瓶颈问题，并且采用了镜像的方法分派了中心节点的负载，MS 和 IS 等中心节点的负载得到了较好的控制，而且随着用户数的增加并没有发生很大的变化。LS 和 HS 的负载比中心节点高，这也证实了所用镜像策略的正确性。GS 的负载情况也在意料之中，负载随房间人数的变化比较明显，正如 2.2.3 小节中所分析的一样，房间必须有个最大人数限制，否则 GS 的负载将过大。

测试用例中的人数比较少，因而服务器的负载也很低。如果采用性能更好的服务器，而不是这里使用的 PC 机，相信服务端能承受一个可观的用户数。

4.3 和其他棋牌游戏平台的比较

和 QQ 游戏、联众游戏等棋牌游戏平台相比，本文提出的棋牌游戏平台和它们既有相似之处也有不同点。

功能方面，QQ 游戏和联众游戏提供了丰富的游戏功能。各种道具的设计给游戏带来了更多的乐趣。QQ 游戏还提供了纸娃娃系统，玩家不但可以在平台上进行游戏，还可以花一番心思装扮一下自己的形象。我们的棋牌游戏平台，跟它们比较起来，仅仅提供了基本的游戏功能，附加功能上差距还是相当大的，这需要在今后的运营过程中逐步改进。

性能方面，由于 QQ 游戏和联众游戏是由专业团队研发，而且已经在市面上成功运营了好几年，因而它们都取得了非常不错的业绩。QQ 游戏的在线用户数达到 300 万，而联众游戏也达到了 50 万。我们的棋牌游戏平台，研发时间有限，而且我们研发人员经验也非常有限，因而我们不奢望、也不可能达到它们的性能水平。遗憾的是，由于条件有限，这里并没有能够测试出我们棋牌游戏平台的最大承载量。

我们的棋牌游戏平台是在别人不公开相关技术的情况下完全自主设计和开发的，虽然在格式上和其他棋牌游戏平台基本相似，但我们在设计过程中，仍然提出了一些有别于它们的设计。

区域性运营就是我们平台具有的一个新特性。从图 4-2 的大厅界面可以看出，此棋牌游戏平台和普通棋牌游戏平台在界面上最大的不同在于，该大厅有左右两棵游戏树，左边的游戏树，跟普通的游戏平台一样，用来运营一些常规的游戏；右边的游戏树，就是区域性运营的体现，区域特色游戏以及需要有特殊运营策略的游戏，统一都放到右边的游戏树上。棋牌游戏具有很强的地区特色，因而运营商可以采用一种分地区运营的运营策略。这样不仅可以满足各个地区玩家对不同游戏规则的需求，本地化运营也有助于运营的便利。区域性运营不仅可以用于分地区运营，还可以用来实现运营策略的多样化。为了给玩家提供更多的增值业务，运营商往往会采用各种运营策略，比如 VIP 分区和普通分区，在 VIP 分区中可能会得到特殊的奖励，但进入 VIP 分区又有一定的限制。当然还有其他的运营策略。区域性运营完全可以做到这些。通过设置区域服务器组中 IS 上的运营策略，就可以实现对整个区域中游戏服务器的统一设置。而据我们分析，目前 QQ 游戏和联众游戏并没有这方面的功能，即使能对某些游戏服务器设置特殊规则，也不能像我们这样对某个区域进行统一批量设置。

软服务器不但使得服务器的配置更加灵活，更重要的是提升了系统的稳定性。从第 2 章对 QQ 游戏和联众游戏的分析可以看出，它们多个游戏房间在同一个进程里，如果有哪个房间突然崩溃，势必会影响到该进程内所有游戏房间的正常运行。当然，如果程序已经相当稳定，就不存在这种问题了。不过对于我们新

开发的程序来说，采用一个房间一个进程的方式能使服务端更加稳定，使配置更加灵活。

我们还提出了基于 DLL 的第三方游戏开发接口，方便其他游戏开发团队开发的棋牌游戏能接入我们的平台。市面上运营的棋牌游戏平台是否也提供了第三方游戏开发接口，我们无从知晓。不过从 QQ 游戏和联众游戏的客户端来看，它们都将棋牌游戏封装成了执行文件的形式，而服务端就不得而知了。相比于执行文件，DLL 具有更高的效率，而且更能节省运行时的占用空间。对服务端来讲，如果存在多个 GS 来运行同一款游戏，而这些 GS 又部署在同一台物理服务器上，那么这些 GS 只要使用一个 DLL 来处理游戏逻辑就可以了。

由于我们无法获得市面上相关棋牌游戏平台更详细的技术资料，所以和它们的比较很难再深入下去。

4.4 本章小结

本章主要对第 3 章提出的棋牌游戏平台的服务端进行了分析和实验测试，并和其他棋牌游戏平台进行了比较说明。通过分析、测试、比较可知，本文提出的棋牌游戏平台的服务端，在实现了棋牌游戏平台基本功能的基础上，还提供了区域性运营的功能，另外第三方游戏开发接口也方便了其他棋牌游戏开发团队开发的游戏能接入到平台中来。本文使用类似于镜像服务器的方法，一定程度上解决了服务端架构的中心节点瓶颈问题。本文的服务端虽然与市面上成熟的棋牌游戏平台有一定差距，但在高效性、可扩展性、安全性、容错性、通用性等方面都有自己的特点。

第 5 章 总结与展望

5.1 总结

5.1.1 本文的贡献

本文结合实际项目开发的需要，完成了一款棋牌游戏平台服务端的设计与实现。通观全文，论文的主要工作有：

1. 详细介绍了网络游戏的发展历程及其发展趋势，并介绍了网络游戏的分类；研究了网络游戏服务端采用的关键技术及其发展趋势；分析了棋牌游戏平台的研究现状，并指出存在的不足，从而提出本文的研究意义。

2. 学习了棋牌游戏服务端相关的基本理论技术，这些技术包括分布式技术、网络通信技术、多线程技术以及动态链接库技术，并将这些技术应用到了设计中去。还对市面上最具代表性的棋牌游戏平台 QQ 游戏和联众游戏进行了分析。通过对典型棋牌游戏平台的分析，为设计工作作了必要的准备。

3. 给出了一款棋牌游戏平台服务端的详细设计。综合运用相关技术和实例分析得出的结论，详细描述了棋牌游戏平台服务端的架构设计、通信协议设计、多线程工作模型以及第三方游戏开发接口，并在此基础之上，给出了服务器软件模块的详细设计。

4. 对实现的服务端进行了初步性能测试与分析，表明本案已基本达到项目提出的研制要求。另外，文章中提出的一些设计方法，在其他网络应用程序中也有一定的应用价值，例如单线程工作模式、handler 消息处理机制、SDCP 通信协议等等，这些方法都可以运用到别的网络游戏和网络应用程序中去。

本文的创新点主要包括以下三个方面：

1.提供了支持区域性运营的功能。本文提出的棋牌游戏平台服务端架构是基于星型结构的，星型结构的每个叶节点就是区域服务器组，不同区域服务器组可以面向不同地区实现区域性运营，另外，通过对区域服务器组中 IS 上运营策略的设置，就可以实现各个区域服务器组运营策略的多样化，从而方便运营商在同一平台中实施各种不同的运营策略。

2. 采用了软服务器的形式，特别的，一个游戏房间就是一个软服务器。这样可以方便地进行服务器的部署，从而充分地利用硬件服务器资源。多个服务器可以运行在同一物理服务器上，实现了多进程工作模式，可以提升整个服务端的效率。将各个游戏房间独立化，避免了某个房间出问题而影响到其他游戏房间的情况，增强了服务器的稳定性。

3. 提供了基于 DLL 的第三方游戏开发接口。第三方游戏开发接口的提出，方便更多的游戏开发团队来为棋牌游戏平台开发棋牌游戏，避免了棋牌游戏的重复开发，也提高了游戏开发效率。DLL 形式比进程间通信更加高效，也方便维护和升级，可以作为传统执行文件的替代形式。

5.1.2 后续工作

本文在介绍棋牌游戏平台服务端设计时，只对一些基本功能作了说明和分析，实际开发过程中要考虑的远远比文中介绍的要复杂得多，实际中还要考虑更多的附加功能，但由于篇幅所限，文章没有把一个完完全全的可用于实际运营的设计写出来。即便如此，文章已经将设计的精髓呈现了出来。文中的设计方案，肯定还存在着一些不足，需要在今后的工作中不断加以改进和完善，这些后续工作主要包括：

1. 中心节点的瓶颈问题。虽然本文通过镜像服务器的方法，一定程度上解决了中心节点的瓶颈问题，但这种解决方法也只是暂时的。随着服务端的规模不断扩大，中心节点必然还会成为瓶颈。为了彻底解决这个问题，在后续工作中，可以考虑采用集群来实现 MS、IS 等中心节点；或者考虑在客户端之间采用 P2P 的方法来共享中心节点的信息，从而减轻中心节点乃至整个服务端的负载。

2. 多线程问题。虽然单线程处理消息使得服务端数据更安全，也方便了开发，但如果想要进一步提高服务器性能，还必须考虑采用多线程来处理消息，弥补单线程的不足。

本文提出的设计方案，架构设计和多线程模式设计部分全部由作者负责完成，通信协议部分和第三方游戏开发接口的设计，由作者和项目组中其他成员合作完成。由于时间问题以及本人精力、能力有限，本文的设计方案还存在很多问题需要留给开发团队进一步完善。

5.2 对未来的展望

未来棋牌游戏平台的发展，必将是规模更大而且更加开放。要想支持更多的玩家在平台上游戏，就需要设计出更加优秀的服务端架构，需要更先进的技术。随着棋牌游戏平台规模的不断扩大，不久的将来可能会出现一个游戏平台标准，这个平台具有强大的性能，而且这个平台提供了统一的游戏开发接口，根据开发接口开发的游戏就能加入到平台中去，而且这些游戏不仅仅是棋牌游戏，还包括各种网络游戏类别。平台也不仅仅只支持 PC 机客户端，平台还将支持移动平台、电视平台、专用游戏机等等。有了这样的平台，玩家就能更方便地进行游戏，就能与更多的玩家进行游戏，从而实现网络游戏真正的开放性。

参考文献

[1] 孙迎新，2006 年中国网络游戏市场研究报告，http://www.eb-mag.com.cn/detail.asp?Unid=6796

[2] 新浪网络游戏专题，http://games.sina.com.cn/netgames/

[3] 赵挺，网络游戏发展史，http://www.blogchina.com/new/source/122.html

[4] 荣钦科技，游戏设计概论，北京科海电子出版社，2003

[5] Gao Huang，Meng Ye，Long Cheng，Modeling system performance in MMORPG，Global Telecommunications Conference Workshops，2004，GlobeCom Workshops 2004，IEEE，29 Nov.-3 Dec，2004 Page(s):512-518

[6] 黄文博，燕杨，C/S 结构与 B/S 结构的分析与比较，长春师范学院学报，2006，25(4)，56-58

[7] He Y，Zhang Y，Guo J，On mitigating network partitioning in peer-to-peer massively multiplayer games，NETWORKING AND MOBILE COMPUTING，PROCEEDINGS LECTURE NOTES IN COMPUTER SCIENCE 3619: 481-490，2005

[8] 缪小亮，万旺根，P2P 技术及其在网络游戏中的应用，微计算机信息，2006 第 18 期，15-17

[9] Rooney，S.Bauer，D.Deydier，R.A federated peer-to-peer network game architecture，Communications Magazine，IEEE，May 2004，42(5)，114-122

[10] C.Diot，L.Gautier，A distributed architecture for multiplayer interactive applications on the internet，IEEE Network Magazine，13(4)，July/August 1999

[11] Ghosh.P，Basu.K，Das.S.K，A cross-layer design to improve quality of service in online multiplayer wireless gaming networks，Broadband Networks，2005 2nd International Conference on 3-7 Oct. 2005 Page(s):813 - 822 Vol. 2

[12] Abdelkhalek.A.，Bilas.A.，Moshovos.A，Behavior and performance of interactive multi-player game servers，Performance Analysis of Systems and Software，2001，ISPASS. 2001 IEEE International Symposium on Nov. 4-6, 2001 Page(s):137 - 146

[13] Heitbrink.D.A，Makki.S.K，Distribution of a world space for real-time 3D applications，Cluster Computing and the Grid，2006，CCGRID 06，Sixth IEEE International Symposium on Volume 1，16-19 May 2006 Page(s):8pp

[14] 李文正，郭巧，王利，Internet 服务器负载均衡的研究与实现，计算机工程，2005，31(6)，98-99

[15] 杨合庆译，Windows 网络编程，北京，清华大学出版社，2002

[16] EPOLL，http://www.xmailserver.org/linux-patches/nio-improve.html，2004.
[17] 段翰聪，卢显良，宋杰，基于 EPOLL 的单进程事件驱动通信服务器设计与分析，计算机应用，2004，24(10)，36-39
[18] 崔滨，万旺根，余小清，楼顺天，基于 EPOLL 机制的 LINUX 网络游戏服务器实现方法，微计算机信息，2006，22 卷 7-3 期，64-66
[19] Wu-chang Feng，Chang.F，Wu-chi Feng，Walpole.J，A traffic characterization of popular on-line games，Networking，IEEE/ACM Transactions on Volume 13，Issue 3，June 2005 Page(s):488-500
[20] Armitage.G，An experimental estimation of latency sensitivity in multiplayer Quake 3，Networks，2003，The 11th IEEE International Conference on 28 Sept-1 Oct. 2003 Page(s):137-141
[21] Johannes Farber，Network Game Traffic Modelling，Proceedings of the first workshop on Network and system support for games，2002.
[22] Lothar Pantel，Lars C，Wolf On the impact of delay on real-time multiplayer games，NOSSDAV，2002
[23] R.A.Bangun，E.Dutkiewicz，G.J.Anido，An analysis of multiplayer network games traffic，In Proceedings of the 1999 International Workshop on Multimedia Signal Processing，pages 3-8，Copenhagen，Denmark，September 1999
[24] Pellegrino，Dovrolis，Bandwidth requirement and state consistency in three multiplayer game architectures，NetGame 2003
[25] IP 多播概述，http://www.microsoft.com/china/technet/community/columns/cableguy/cg0202.mspx
[26] Shi Xing-bin，Xing Yuan-sheng，An AOI node joining algorithm for P2P MMOG multicast，Mini-Micro Systems，2006，27(4)，623-626
[27] 苏羽，王媛媛，Visual C++网络游戏建模与实现，北京，北京科海电子出版社，2003
[28] 黄健，网络游戏服务平台基本架构设计与实现，[硕士学位论文]，广州，华南理工大学，2005
[29] 冯磊，Cluster 与网络游戏共舞，信息系统工程，2005.09，69-70
[30] 文俊浩，徐传运，温文聪，朱晓飞，IOCP 机制在 P2P 网络游戏中的应用，计算机工程与应用，2006.07，205-207
[31] JungHyun Han，In.H.P，Jong-Sik.Woo，Towards situation-aware cross-platform ubi-game development，Software Engineering Conference，2004.11th Asia-Pacific 30 Nov.-3 Dec.2004 Page(s):734-735
[32] 惠煌，网络跳棋类游戏分析与设计，[硕士学位论文]，大连，大连理工大学，

2005
[33] 郑炜旸，锄大地与五子棋网络游戏的设计与实现，[硕士学位论文]，广州，中山大学，2004
[34] QQ 游戏，http://game.qq.com/
[35] 高传善译，分布式系统设计，机械工业出版社，2001
[36] 周良忠译，C++面向对象多线程编程，人民邮电出版社，2003
[37] 徐波译，C 专家编程，人民邮电出版社，2002
[38] 联众游戏，http://www.ourgame.com/
[39] 吴兆定，袁江海，郑世宝，棋牌类网络游戏服务端的架构设计，计算机工程 ，将于 2007.8 刊登

附录一 平台中使用的通信消息类型

注：表中“ C ”是 Client 的简写，代表客户端。

- 大厅更新消息

消息类型	方向	说明
MSG_HALLVSNCHK	C>>US	客户端到US进行大厅版本验证，消息中应包含客户端大厅的版本号
MSG_HALLVSNACK	US>>C	US 将大厅版本验证结果告诉客户端，如果需要更新，则消息里应该包括下载路径信息

- 登录消息

消息类型	方向	说明
MSG_LOGIN	C>>LS	客户端发给LS的登录请求消息，消息中应包括玩家的用户名和密码
MSG_LSISLOGINCHK	LS>>IS	LS到IS上进行玩家重复登录验证，消息中应包括玩家的ID号
MSG_ISMSLOGINCHK	IS>>MS	IS到MS上进行玩家重复登录验证，消息中应包括玩家的ID号
MSG_MSISLOGINACK	MS>>IS	MS将重复登录的验证结果告诉IS
MSG_ISLSLOGINACK	IS>>LS	IS将重复登录的验证结果告诉LS
MSG_LOGINACK	LS>>C	LS 将玩家登录的验证结果告诉客户端

- 进入大厅消息

消息类型	方向	说明
MSG_LINKHS	C>>HS	客户端向HS发起连接，消息中包含玩家ID号
MSG_HSISLINKHS	HS>>IS	HS到IS进行验证，看玩家的连接是否合法，消息中包含玩家的ID号
MSG_ISHSLINKHSACK	IS>>HS	IS将验证结果告诉HS
MSG_LINKHSACK	HS>>C	HS将玩家请求连接的结果告诉客户端
MSG_ISMSGETPLYINFO	IS>>MS	如果通过IS的验证，那么IS就会像MS索取玩家的用户信息，消息中应包括玩家ID号
MSG_MSISPLYINFO	MS>>IS	MS将从数据库中读取的玩家用户信息传送给IS，消息中是玩家的用户信息

（续）

消息类型	方向	说明
MSG_ISHSPLYINFO	IS>>HS	IS将从MS获得的玩家用户信息传送给HS
MSG_PLYINFO	HS>>C	HS将玩家信息传送给客户端
MSG_GETGAMEINFO	C>>HS	客户端向HS请求游戏列表的信息，这里只请求房间以上级的当前玩家人数信息
MSG_GAMEINFO	HS>>C	HS 将游戏列表信息告诉客户端

- 获取房间信息消息

消息类型	方向	说明
MSG_GETRMLNKINFO	C>>HS	客户端向HS发送请求，获取某个游戏下的所有房间的连接信息
MSG_RMLNKINFO	HS>>C	HS将房间的连接信息告诉客户端，消息中包括某个游戏下的所有房间的连接IP地址和端口号
MSG_GETRMCOUNT	C>>HS	客户端向HS发送请求，获取某个游戏下的所有房间的当前玩家人数信息
MSG_RMCOUNT	HS>>C	HS 将房间人数信息告诉客户端，消息中包括某个游戏下的所有房间的当然玩家人数信息

- 服务端房间列表同步消息

消息类型	方向	说明
MSG_SVRCONNECTREQ	LN>>CN	叶节点服务器向中心节点服务器发送连接请求的消息，消息中应包括叶节点服务器的类型，以及验证信息
MSG_SVRCONNECTACK	CN>>LN	中心节点服务器将叶节点请求连接的结果告诉相应叶节点
MSG_SVRDISCONNECT	LN>>CN	叶节点服务器向中心节点服务器发送断开连接的请求
MSG_ISMSROOMADD	IS>>MS	在GS与IS成功建立连接后，IS将这个房间的连接信息告诉MS，以方便MS通知其他IS
MSG_MSISROOMADD	MS>>IS	MS将新增GS的连接信息通知给其他IS
MSG_ISHSROOMADD	IS>>HS	IS将新增GS的连接信息通知给与之相连接的所有HS
MSG_ROOMADD	HS>>C	HS将新增GS这件事情及时地告诉客户端，方便玩家进入新GS游戏
MSG_MSISROOMRMV	MS>>IS	MS将GS关闭的信息通知给其他IS

（续）

消息类型	方向	说明
MSG_ISMSROOMRMV	IS>>MS	GS和IS断开连接时，IS将这件事情告诉MS，以方便MS通知其他IS删除该房间信息
MSG_ISHSROOMRMV	IS>>HS	IS将GS关闭的信息通知给与之相连接的所有HS，以便HS删除该房间信息
MSG_ROOMREMOVE	HS>>C	HS将GS关闭的信息及时地告诉客户端，客户端从游戏列表上删除该游戏房间
MSG_MSISRQRMLIST	MS>>IS	定期地，MS向某个IS请求该IS下的GS的在线人数信息
MSG_ISMSRMLIST	IS>>MS	IS将该地区的GS当前人数信息告诉MS，以便MS告诉其他IS，进行人数信息同步
MSG_MSISRMLIST	MS>>IS	MS将某IS下的房间人数信息告诉其他IS，以便其他IS进行同步
MSG_HSISRQRMLIST	HS>>IS	定期地，HS向IS请求平台下所有房间的人数信息
MSG_ISHSRMLIST	IS>>HS	IS将平台下所有房间的人数信息告诉HS
MSG_MSISPRERMLIST	MS>>IS	在 IS 启动时，如果之前已经有 IS 启动，那么 MS 就通过此消息，将已经启动的 IS 下的房间信息告诉该 IS，房间信息包括连接信息和人数信息

- 游戏更新消息

消息类型	方向	说明
MSG_GAMEVSNCHK	C>>US	客户端到US进行游戏版本验证，消息中应包含游戏号以及游戏的版本号
MSG_GAMEVSNACK	US>>C	US 将游戏版本验证结果告诉客户端，如果需要更新，则消息里应该包括下载路径信息

- 进入房间消息

消息类型	方向	说明
MSG_ENTERROOM	C>>GS	客户端向GS发起进入房间请求，消息中应该包括玩家的ID号
MSG_GSISENTERRMCHK	GS>>IS	玩家请求进入房间时，GS到IS上进行验证，看IS上是否有该玩家的信息，如果有，则通过验证，没有的话继续到MS上进行验证
MSG_ISMSENTERRMCHK	IS>>MS	IS上没有要进入房间的玩家的信息，可能是异地登录，到MS上作进一步验证

（续）

消息类型	方向	说明
MSG_MSISENTERRMACK	MS>>IS	MS将验证结果告诉IS
MSG_ISGSENTERRMACK	IS>>GS	IS将验证结果告诉GS
MSG_ENTERROOMACK	GS>>C	GS 将验证结果告诉客户端，结果包含多种情况，除了不合法不能进入房间外，可能还会因为房间人数已满而不能进入房间

- 房间内信息刷新消息

消息类型	方向	说明
MSG_UPDATEPLAYER	GS>>C	GS每隔1、2秒，将房间中状态发生改变的玩家的信息广播给房间里的所有人，方便客户端进行玩家列表和桌子列表状态的更新
MSG_ADDPLAYER	GS>>C	房间里进来一个人，将这个人的信息广播给房间内的所有玩家
MSG_REMOVEPLAYER	GS>>C	房间里退出一个人，将这件事情告诉房间里的所有玩家

- 聊天消息

消息类型	方向	说明
MSG_PUBCHAT	C<>GS	公共聊天信息。客户端发给GS，以及GS广播给客户端，只要是公聊，都用此消息类型，消息中包括聊天内容
MSG_PRICHAT	C<>GS	私有聊天信息，客户端发给 GS，以及 GS 发给指定玩家，都用此消息类型，消息中包括接收玩家 ID 和聊天内容

- 游戏消息

消息类型	方向	说明
MSG_JOINTABLE	C>>GS	客户端向GS发起进入桌子的请求，消息中应该包括玩家ID和桌子号
MSG_JOINTABLEACK	GS>>C	GS将加入桌子的结果告诉客户端，可能因为人满或者已经开始游戏等情况而不能进入
MSG_USERENTERTABLE	GS>>C	某玩家成功进入桌子时，服务端发送此消息给该桌上的其他玩家，通知其他玩家有玩家加入房间
MSG_EXITTABLE	C>>GS	客户端请求退出游戏桌
MSG_USERLEAVETABLE	GS>>C	某玩家退出桌子，服务端将此消息广播给该桌上的其他玩家，通知其他玩家有玩家离开桌子了
MSG_READY	C>>GS	玩家做好准备

（续）

消息类型	方向	说明
MSG_USERRECON	GS>>C	GS将某个玩家意外断线后重入这件事通过此消息告诉桌上的其他玩家
MSG_USERREADY	GS>>C	GS将某个玩家做好准备这件事通过此消息告诉桌上的其他玩家
MSG_CANCELREADY	C>>GS	玩家取消准备
MSG_USERCANCELRDY	GS>>C	GS将某个玩家取消准备这件事通过此消息告诉桌上的其他玩家
MSG_GAMESTART	GS>>C	GS告诉桌上的玩家，本局游戏开始
MSG_GAMEEND	GS>>C	GS告诉桌上的玩家，本局游戏结束
MSG_USERRUNAWAY	GS>>C	GS将某个玩家逃跑这件事通过此消息告诉桌上的其他玩家
MSG_USERDOWN	GS>>C	GS将某个玩家意外断线这件事通过此消息告诉桌上的其他玩家
MSG_GAMEMSG	C<>GS	游戏过程中的消息。游戏过程中会有很多种类的消息，但具体消息只能由小游戏开发者自己定义。因而这里我们定义了一个大的游戏消息种类，对于游戏平台来讲，平台如果发现是这类消息，就会交给第三方去处理。有关这方面的内容在下面第三方相关章节将着重说明。

- 游戏结果同步消息

消息类型	方向	说明
MSG_GSISGAMERST	GS>>IS	游戏结束时，GS计算出游戏结果，GS将游戏结果告诉IS，由IS来更新数据库
MSG_ISGSGAMERST	IS>>GS	如果该玩家还在同一种游戏的其他房间中，IS就将游戏结果的变化告诉其他GS，以同步游戏结果
MSG_GAMERESULT	GS>>C	IS 更新完数据库之后，会将结果告诉各个相关 GS，GS修改数据，同时通过此消息告诉客户端

- 全局信息同步消息（以金币为例）

消息类型	方向	说明
MSG_GSISCOINCHG	GS>>IS	玩家在游戏过程中发生了金币的变化，GS将该变化告诉IS，以便IS告诉MS
MSG_ISMSCOINCHG	IS>>MS	IS将金币变化的信息告诉MS，由MS来更新数据库

（续）

消息类型	方向	说明
MSG_MSISCOINCHG	MS>>IS	MS更新完数据库之后，将结果告诉各个相关IS，IS进行相关处理
MSG_ISGSCOINCHG	IS>>GS	IS将金币变化告诉各个相关GS，GS修改数据，以同步游戏结果
MSG_COINCHG	GS>>C	GS 修改数据时，将金币的变化通知房间内的所有玩家，客户端及时修改信息

- 退出房间消息

消息类型	方向	说明
MSG_EXITROOM	C>>GS	客户端向GS发出退出房间的请求
MSG_GSISEXTROOM	GS>>IS	GS告诉IS，玩家退出某个房间了，消息中包括玩家ID和房间号
MSG_ISMSPLYEXTIS	IS>>MS	IS 在处理玩家退出房间的消息时，如果发现该玩家既不是从该地区登录，又不在该地区的其他 GS 上，那么 IS 就删除该玩家信息，同时通过此消息告诉 MS，该玩家已经离开这个 IS

- 退出大厅消息

消息类型	方向	说明
MSG_LOGOUT	C>>HS	客户端向HS发起logout请求
MSG_HSISLOGOUT	HS>>IS	HS将玩家logout事件告诉IS
MSG_ISMSLOGOUT	IS>>MS	IS 删除玩家信息后，将 logout 事件告诉 MS，MS 亦删除该玩家信息

附录二 第三方游戏开发接口明细

- SvrPlatformInterface

接口函数	说明
NotifyGameStart	第三方服务端发现桌面已经满足游戏开始的规则，于是就判定游戏开始，调用这个接口告诉服务端平台，平台随即将游戏开始的信息广播给相应玩家的客户端
NotifyGameEnd	第三方服务端判定一局游戏已经结束，这是玩家的积分和战绩记录会发生变化，具体积分变化是由第三方服务端决定的。第三方服务端在游戏结束时，计算出游戏结果后，调用此接口上传给服务端平台，平台随即作相应处理
NotifyPlayerRunAway	游戏过程中，第三方服务端发现有玩家逃跑，这时第三方会决定对逃跑的玩家进行扣分惩罚，通过此接口告诉服务端平台某个玩家逃跑，并将扣除的分数告知平台
SendGameMsg	第三方服务端需要向第三方客户端发送消息时调用这个接口函数，通过函数参数将封装好的消息传入

- SvrGameInterface

接口函数	说明
OnGameMessage	服务端平台收到来自客户端的MSG_GAMEMSG种类的消息时，就调用此接口将消息传给第三方服务端进行处理
OnTableEnter	客户端平台向服务端平台发送进入桌子的请求，通过服务端平台的验证后，服务端平台就调用此接口将玩家进入桌子的信息告诉第三方服务端
OnTableLeave	客户端平台向服务端平台请求退出桌子，此时需要由第三方服务端来作某些处理，于是服务端平台就调用此接口来通知第三方服务端
OnPlayerReady	客户端平台告诉服务端平台某个玩家准备好了，服务端平台调用此接口将玩家准备好这件事情告诉第三方服务端
OnPlayerCancelReady	客户端平台告诉服务端平台某个玩家取消准备了，服务端平台调用此接口将玩家取消准备这件事情告诉第三方服务端
OnPlayerRunAway	服务端平台在玩家请求退出桌子时，发现玩家属于逃跑的情况，于是就调用此接口，让第三方游戏决定如何惩罚
OnUserDown	服务端平台的网络层发现某个玩家意外断线了，需要将这件事通知给第三方服务端，以方便第三方服务端来处理玩家重连进来后的逻辑处理

（续）

接口函数	说明
OnPlayerReCon	玩家意外断线后，在规定时间又回来了，这时平台就调用此接口告诉第三方服务端，第三方服务端进行相应处理
OnUserScoreChange	玩家在同一种游戏的其他房间内游戏时，发生了积分变化，这时服务端平台会通过同步机制告诉到这个GS上，GS再调用此接口告诉第三方服务端，第三方服务端作相应更新

- CltPlatformInterface

接口函数	说明
UserReady	玩家在游戏界面上点击准备按钮，第三方客户端就调用此接口将玩家准备好这件事告诉平台客户端，客户端平台再将这个信息发送给服务端平台
UserCancelReady	玩家在游戏界面上点击取消准备按钮，第三方客户端就调用此接口将玩家取消准备这件事告诉平台客户端，客户端平台再将这个信息发送给服务端平台
UserExit	玩家点击游戏界面上的退出按钮，第三方客户端就调用此接口将玩家请求退出桌子这件事告诉平台客户端，客户端平台再将这个信息发送给服务端平台
SendGameMsg	第三方客户端需要向第三方服务端发送消息时调用这个接口函数，通过函数参数将封装好的消息传入

- CltGameInterface

接口函数	说明
OnGameMessage	服务端平台收到来自客户端的MSG_GAMEMSG种类的消息时，就调用此接口将消息传给第三方服务端进行处理
OnSelfEnterTable	服务端判断玩家成功加入桌子后，客户端平台收到相应信息，于是调用此接口告诉第三方客户端玩家成功进入了桌子，此时第三方启动游戏界面并完成相应初始化工作
OnTableEnter	其他玩家成功加入桌子，客户端平台收到服务端平台的信息，通过此接口告诉第三方客户端，有个玩家进入桌子了
OnTableLeave	其他玩家成功退出桌子，客户端平台收到服务端平台的信息，通过此接口告诉第三方客户端，有个玩家退出桌子了
OnPlayerReady	其他玩家准备好了，客户端平台收到服务端平台的信息，通过此接口告诉第三方客户端，有个玩家准备好了
OnPlayerCancelReady	其他玩家取消准备，客户端平台收到服务端平台的信息，通过此接口告诉第三方客户端，有个玩家取消准备了

（续）

接口函数	说明
OnGameStart	客户端平台收到服务端平台游戏开始的消息时，调用该接口告诉第三方客户端，第三方客户启动游戏，进入游戏程序
OnGameEnd	客户端平台收到服务的平台游戏结束的消息时，调用此接口告诉第三方客户端，第三方客户端结束游戏
OnPlayerRunAway	其他玩家逃跑了，客户端平台收到服务端平台的信息，通过此接口告诉第三方客户端，某玩家逃跑了
OnUserDown	其他玩家发生意外断线，客户端平台收到服务端平台的信息，通过此接口告诉第三方客户端，有个玩家发生意外断线了
OnPlayerReCon	其他玩家在发生意外断线后，在规定时间内又回来了，客户端平台收到服务端平台的信息，通过此接口告诉第三方客户端，有个断线的玩家又回来了
OnUserScoreChange	玩家（包括自己）的积分发生变化，客户端平台收到服务端平台的信息，通过此接口告诉第三方客户端，玩家的积分发生了变化

致 谢

首先，衷心感谢我的导师郑世宝教授。从我本科的毕业设计开始，到随后的两年半的研究生生涯中，郑老师在学习和生活上给予了我莫大的关怀和帮助。感谢郑老师对我的悉心培养，我将牢记老师的教诲！

本论文的研究工作，是和我们部门的工作密不可分的。感谢上海高清数字科技有限公司数字娱乐部的所有同事。感谢部门经理袁江海老师和项目经理印华老师对我的培养和指导，你们渊博的知识和忘我的工作热情，为我树立了榜样。感谢王小勇硕士、李前程硕士、李兴东硕士和吴永明硕士，和你们一起在公司实习的那段时间，是我研究生阶段最值得怀念的时光。感谢工程师陈云凌、郑晓蕾、朱嘉、谢飞、薛青斌、卢震宇、王冠宇，感谢美工毛异、张进、冯树、薛晓斌、黄福利、付雪，感谢策划舒丹、张占午、郭琦、王丽、王智，感谢那些所有和我共事过的同事，很幸运在我的第一份实习工作中认识你们，感谢你们给了我一段有别于学校的生活。

感谢上海交通大学图像通信与信号处理研究所。两年半的学习中，所里的老师和同学为我提供了很多帮助。

感谢两年半研究生生涯中，所有关心和帮助过我的人！

攻读学位期间发表的学术论文目录

[1] 吴兆定，袁江海，郑世宝，“ 棋牌类网络游戏服务端的架构设计 ”，发表于中文核心期刊《计算机工程》，将于 2007.8 刊登