

高性能 MMO 服务器架构与模式设计

李海燕

(广东外语艺术职业学院 信息技术系, 广东 广州 510640)

摘 要: MMO 服务器设计过程中, 需要关注整体结构、通讯模块和服务器的鲁棒性. 结合经典设计模式, 讨论 MMO 服务器构建的关键技术点. 采用模块化的构建方法组织服务, 通讯模块使用 IOCP 机制保证网络服务的并发性和高效性. 融合油漆工、代理、观察者等多种模式, 实现服务模块的低耦合结构. 加入共享内存机制和进程通信机制, 将服务数据与服务逻辑分离, 保证服务器的鲁棒性.

关键词: MMO; 设计模式; 软件体系结构; 完成端口

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2010)07-0097-05

High-performance MMO Server Architecture and Model Design

LI Hai-yan

(Department of Information Technology, Guangdong Teachers College of Foreign Language
and Arts, Guangzhou 510640, China)

Abstract: Process of MMO server design should focus on the overall structure, communication module and the server robustness. This paper presents MMO servers key technologies and modular organization based on design patterns. The communications module uses IOCP mechanism to guarantee network service concurrency and efficiency. With low coupling, the MMO service module structure is designed with decorator, proxy, observer and other patterns. To ensure the robustness of the server, the service data and service logic are separated with shared memory mechanism and process communication mechanism.

Key words: MMO; design pattern; software architecture; completion port

1 引言

计算机应用的发展过程经历了客户端-服务器(CS)架构到浏览器-服务器(BS)架构的变迁, 而随着通信技术的发展, 网络带宽迅速提高, 为了追求更好的用户体验和程序表现力, 越来越多的计算机应用回归到了富客户端模式, 例如大型客户端游戏、基于 Flash/Flex 技术的 SNS 应用^[1]等. 此类型的应用均需要构建能够支持大规模用户并发的高效服务器端解决方案, 本文将此类服务器软件设计统一归为 MMO(Massively Multiplayer Online)服务器范畴.

大量的开源项目, 提供统一化的 MMO 架构, 但针对具体应用, 开源项目难以满足个性化需求, 独立构建 MMO 服务器难于把握整体架构与关键技术选型. 本文

针对 MMO 应用的特点, 给出整体架构和模块划分方案, 并从设计模式的角度, 重点论述通讯模块的设计思路、服务并行控制方法及鲁棒性提高方案.

2 模块划分与组织

构建 MMO 服务器的目标是为大规模并发用户提供服务, 所以 MMO 服务器必然关联一系列业务逻辑, 同时, 服务提供过程需设计网络通讯和数据存储, 本文将 MMO 服务器的总体模块构成为图 1 所示结构.

MMO 服务器的主体模块由三个部分组成:

(1) 业务逻辑模块: 包含服务器提供的具体服务, 为更好地区分与扩展, 业务逻辑模块彼此独立, 以动态链接库的形式呈现, 供服务器模块调用;

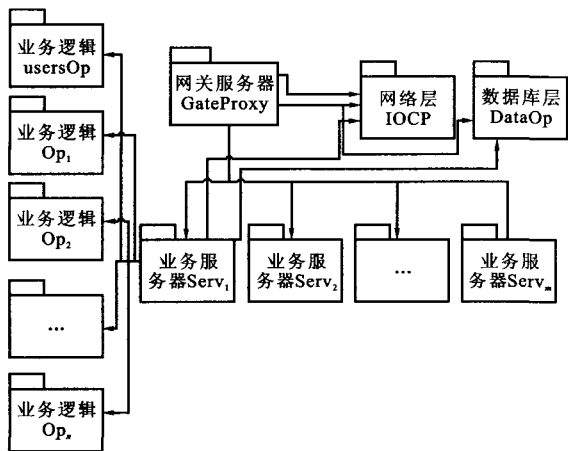


图 1 MMO 服务器总体架构图

(2) 服务器模块:包括网关服务器与业务服务器两类.网关服务器的职责是接收用户请求,并进行软件层面的负载均衡,同时负责用户的新建、删除、异常离线等业务,但不处理具体服务请求;业务服务器接受网关服务器的用户分派,并响应用户实际逻辑.一组服务器中,唯一的网关服务器可以关联若干业务服务器;

(3) 服务响应与处理模块:包括数据库操作模块与网络通讯控制模块,前者负责数据库连接的建立、增删改查操作以及事务、存储过程调用等;后者负责建立、保持、侦听网络连接,同时负责将接受到的网络报文转发响应模块处理.

3 关键模块设计

本节讨论构建 MMO 服务器的关键技术环节. MMO 服务器需要对大规模用户提供高效、稳定的服务,其核心重点解决三个问题:第一是构建稳定的通讯模块,处理服务器的数据收发;其二,合理组织服务种类,保证服务的响应;最后,对服务器的异常提供正确的处理方式,在发生致命错误时,尽快恢复服务器的正常运行,保证服务器的鲁棒性.

3.1 通讯模块

Windows 操作系统下使用 VC-C++ 开发通讯程序,可以使用标准 socket 或者 Windows 内核通讯机制,主要方法有 4 种^[2]:

(1) 标准 socket 通信:使用标准的 socket 构建基于 TCP 的程序,主要使用 ws2_32.dll 中包含的 socket、bind、listen、accept、send、recv 等函数构建服务器端的通讯架构.当接收到客户端的服务请求数据时,通过 select 机制选择处于活动状态的套接字,响应服务请求.使用标准的 socket 通讯机制可以处理少量的、可数的 TCP 用户连接,一旦连接数目过

多,则无法达到高性能的服务处理要求,故在 MMO 设计过程中不建议使用此通讯机制.

(2) 采用 MFC 中的异步套接字机制通讯:MFC 中的 CSocket 类型为程序开发者提供了异步套接字机制.其机理类似于标准 Win32 消息的处理流程:为每个窗体指定一个窗口函数 (WinProc),用来处理所接受到的消息.主要使用到的接口函数有 WSAConnect、WSAAccept、WSASend 等.一旦套接字建立且 TCP 握手成功后,使用 WSAAsyncSelect 函数侦听异步套接字请求,根据请求的类型进行判断处理.使用异步的 CSocket 通讯机制同样不能够满足 MMO 服务器的多用户、高效率的服务响应.

(3) 基于 Event 的事件驱动通讯机制:对于每一个套接字,可以调用 WSACreateEvent 方法为其创建一个侦听事件,当套接字发生回调时,可以通过侦听事件选择需要响应的 socket.侦听工作可以通过 WSAEventSelect 方法完成.同前两种方法一样,本机制仍旧是对每一个套接字的单独处理,在用户连接多、请求频繁时,无法构建高性能的服务机制.

(4) IOCP 通讯模式:完成端口模式 (CompletionPort IO) 是 Win32 系统为大规模并发套接字通讯提供的一种套接字管理与响应方法.适用于构建大规模并发的网络应用服务,本文采用 IOCP 方式构建 MMO 服务器的核心通讯模块.

核心通信类 CIocpNet 的成员组织关系如图 2 所示.其中,考虑高层抽象的设计原则,将网络模块行为抽象为接口 IServNet,供后继程序扩展使用. CIocpNet 的数据成员中包含两个关键变量: m_listenKey 与 m_SockArray [MAX_NET_CONNECT],均为 CSockKey 类型.前者表示侦听者套接字,是全局唯一的套接字实例;后者是一个容量为最大连接数的套接字数组,用于存储当前 MMO 服务器全部的活动套接字.根据 IOCP 机制原来,还需要一个服务线程与 CIocpNet 配合使用,本文将此线程命名为 ThreadIocpWorkerRoutine,其作用是侦听 IOCP 事件,并对接受、读取、发送和错误事件加以处理,其流程如图 3 所示.使用 GetQueuedCompletionStatus 方法可以获取 IOCP 上发生的通讯事件,对于接受连接事件的处理过程中,需要调用 CreateIoCompletionPort 方法将新产生的连接与 IOCP 进行关联.

CIocpNet 类的其他重要方法如下:

(1) StartNetService: 供主程序调用,初始化 IOCP.在此方法中,需要首次创建 IOCP,调用 Cre-

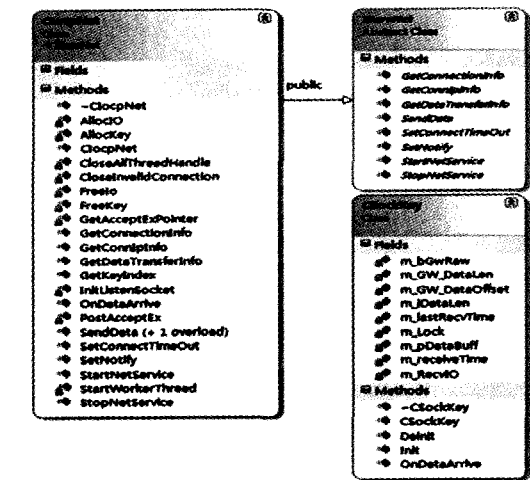


图 2 通讯模块主类型结构图

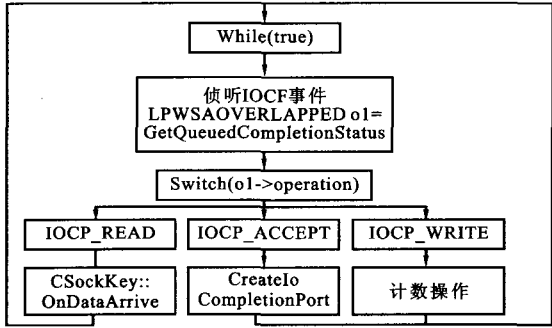


图 3 IOCP 主控线程流程图

ateIoComple tionPort 方法. 值得注意的是 IOCP 的工作原理规定, 需要首次调用此方法进行 IOCP 初始化, 以后每一个套接字的建立, 均需要重新调用此方法进行套接字与 IOCP 的关联.

(2) StartWorkerThread: 此方法在 StartNetService 内调用, 初始化 IOCP 之后, 开启服务者线程. 此时, 由于并未初始化侦听套接字, 固 IOCP 线程处于空等状态. 在方法实现过程中, 为了获取最优的 IOCP 效率, 需要根据 MMO 硬件服务器的 CPU 内和数创建工作线程, 可以通过 GetSystemInfo API 获取系统信息, 具体实现过程中, 可以创建较硬件系统 CPU 个数稍多的工作线程, 通常使用 CPU + 2 的方法.

(3) InitListenSocket: 此方法的作用是创建第一个侦听器套接字, 调用 WSASocket 方法获取新的 Socket 之后, 将其与 IOCP 进行绑定. MMO 实现过程中, 可以初始化多个侦听套接字, 以提高服务器响应性能, 通常 10~15 个侦听套接字为宜. 此方法依旧是在 StartNetService 方法中进行调用.

(4) SendData 方法的作用是发送数据至指定的连接, 其形参中包含连接的下标和发送的数据字节

流, 函数中由 m_SockKey 数组中获取套接字值, 并调用 WSASend 方法发送数据. 此方法由外部程序模块在业务流程需要时主动调用.

另外需注意, CIOcpNet 类型仅需要全局存在唯一对象, 因此, 需要使用单件模式处理类型的初始化过程.

3.2 服务组织模式

通讯模块正常接收到数据之后, 需要交给正确的逻辑处理实体进行数据处理. 在图 2 中可以获知, CIOcpNet 类型中和 CSockKey 类型中均含有 OnDataArrive 方法, 且通过图 3 可知, 一旦 IOCP 被触发且类型为数据读取请求, 则会调用 CSockKey 中的 OnDataArrive 方法. 实际上, CSockKey 中的 OnDataArrive 方法是一个“油漆工”方法, 文中采用 Decorate 模式^[3]实现数据的接收操作: CSockKey::OnDataArrive 方法用于校验数据的长度、格式是否符合预定义的传输协议, 对于符合要求的数据, 回调 CIOcpNet 类的 OnDataArrive 方法加以处理. 这样做的优势在于, 将数据校验与数据处理分离, 可以在不影响主体逻辑的方式下添加或者修改数据的校验逻辑.

实际的 OnDataArrive 方法的实现逻辑十分复杂, 针对不同类型的数据请求, 调用对应的逻辑行为模块加以响应. 考虑程序的可扩展性和可维护性, 本文提出如下的服务组织模式:

(1) 合法数据请求到达后, 并不加以判断, 而是将数据的判断工作交由单独的服务组织模块处理.

(2) 采用经典的 Observer 模式^[4], 在 CIOcpNet 类中设置观察者对象, 由其处理数据. 图 2 中可见, CIOcpNet 类中具有一个名为 SetNotify 的方法, 可以通过此方法设置观察者句柄.

(3) 所有的实际业务逻辑模块均有资格作为观察者对象存在, 因此, 为所有类型定义公共抽象类 IServNotify, 其中仅需定义唯一的纯虚方法 ProcessData, 所有实体逻辑类均继承自 IServNotify 且重写此虚函数.

观察者模式的应用, 将实体与控制相分离, 通过回调机制处理业务数据. 为了统一管理, 本文定义一个独立的业务管理模块, 将且命名为 DIIMain.

DIIMain 中包含类型 CMainMod, 其成员定义如图 4 所示. 其中, 对于不能够合成单独业务模块的操作进行了独立处理, 例如用户创建(DoCreatePlayer)、删除(DoDeletePlayer)等. 另外, CMainMod 类由 IServNotify 抽象类派生而来, 显然, CMainMod 并非

指代某一种特定业务模块,而是业务模块的集合,这里使用了经典的 Commander 模式完成统一的消息处理.在 ProcessData 方法中,根据数据类型的控制字,即可找到对应的模块进行处理.

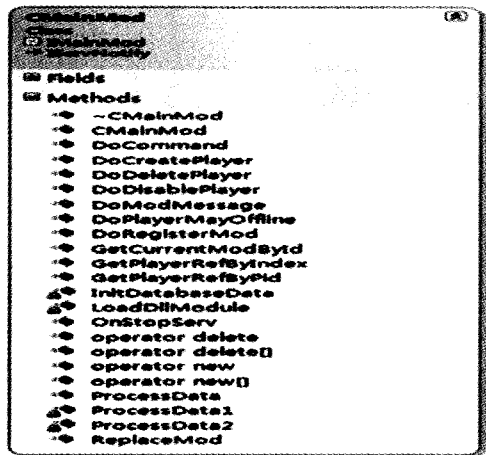


图 4 服务组织模块主类型结构图

CMainMod 中的另一个关键方法是 LoadDllModule,它负责加载 DLL 模块的至自身列表中.由于标准 C++ 不支持反射机制,故不能够靠模块名称直接产生对应对象.文中通过 LoadLibrary API 加载对应 DLL,通过 Proxy 模式^[3]激活相应类型.

LoadLibrary 是一个标准的 Windows API,其参数格式为字符串,指代需要加载的 DLL 名称,此方法恰好适用于本文讨论的情况.通过预定义的模块名称宏,可以顺利加载指定的动态库.动态库加载后,每个动态库均对应其主业务类型,创建类型过程又会遇到反射问题,因此,可以通过代理模式:为每个模块规定一个统一的命名入口,本文将其定义为 InitModule,每个业务逻辑模块均含有相同的入口函数命名,可以将加载模块代码构建成(未包含错误判断逻辑):

```
WIN32_FIND_DATA findData = {0};
char * ModFilter = ".\\Mod\\*.dll";
HANDLE hFind = FindFirstFile(ModFilter,
&findData);
bool nonefile = (findData.cFileName[0] == 0);
int i = 1;
char path[MAX_PATH];
while (FindNextFile(hFind, &findData)){
hLib = LoadLibrary(path);
m_HandMods[i - 1] = hLib;
PFN_InitModule pfnInitModule
= (PFN_InitModule)GetProcAddress(hLib, "InitModule");
if (pfnInitModule){
```

```
pfnInitModule(g_MainMod);
i++;}}
```

3.3 鲁棒保证策略

MMO 服务器运行过程中,可能出现两类错误:数据异常错误与程序异常错误.前者可以在错误异常抛出后,通过回滚数据库实现数据操作的终止,并提示用户重试操作;后者往往由于服务器硬件原因或者程序编写过程中的漏洞所引起.虽然经过长期的测试可以尽可能地发现并弥补程序漏洞,但考虑 C++ 中的指针操作的特殊性,对 MMO 服务器提供必要的逻辑级别的运行保护十分必要.

最为直观的解决方式是部署专门的监测程序,一旦发现服务器程序运行终止,则重启服务器.这种方案虽然易于实现,但重启服务器会造成先期回话中断,用户体验差.通过上述分析,在处理第二类错误时,可以忽略数据错误,因此,可以通过数据与行为分离的方式实现 MMO 鲁棒性增强,从而提供更好的用户体验——实现服务器逻辑的透明重启.具体做法分为 3 个关键步骤:

(1) MMO 服务器启动时,为数据服务器.仅加载各业务处理实体模块,初始化数据库连接,并不初始化网络连接,并通过 CreateEvent 构建信号量,表明数据服务器已经处于运行状态.而后通过 CreateFileMapping 共享自身内存

(2) 首次启动成功后,再次自运行,为逻辑处理服务器.启动网络模块,开始业务逻辑处理.首次读取共享内存数据,后通过循环线程定时写入步骤 1 的共享内存;

(3) 逻辑处理服务器中,通过 CreateEvent 方法创建心跳事件,数据服务器侦听心跳事件,一旦逻辑服务器异常则重启逻辑服务器.

上述步骤将服务逻辑与数据相分离,通过共享内存机制,逻辑服务器即使重启仍旧能够获取最近的数据镜像,保证服务的可持续性.

4 测试与评价

基于本文提出的 MMO 架构,实现 C++ 多人在线服务器.通过模拟客户端程序,构建虚拟用户进行测试.测试内容与环境情况如下:

(1) 考虑单台硬件服务器情况,网关于逻辑服务器、数据服务器运行于同一机器,分别考察在 1 000/2 000/3 000 并发用户情况下的 CPU 开销和服务程序的内存开销;

(2) 考虑两天服务器组成服务硬件组,两机器

分别运行逻辑服务与数据服务,网关服务器运行于一台机器上,考察 1 000/2 000/3 000 并发用户情况下的 CPU 和内存开销;

(3) 测试服务器为 PC 机,CPU 2.4GHz 双核,4GB 内存,Windows 2003 Server,除操作系统外无其他应用程序.

表 1 中给出了本文构建的 MMO 服务器的测试结果.1 000 并发用户时,CPU 开销很低,完全满足长期运行的 CPU 开销;用户量增至 3 000 后,单台服务器的 CPU 开销已经达到 35%,若长期运行,可满足需求,但不能够承载更多的并发用户,否则服务器将处于不稳定状态;多级群组运行服务器情况与单机情况差异不大,证明服务器的网关设计合理.考虑到测试环境采用性能较低的 PC 级,若使用运营级服务器,则可以承载更多的用户或维持更加稳定的运行状态.所以,本文设计的服务器架构在构建

表 1 服务器模拟测试结果

编号	主机	单机用户	CPU/%	内存/Byte
1	1	1 000	8	40 350
2	1	2 000	19	62 600
3	1	3 000	35	100 600
4	2	1 000	10	41 740
5	2	2 000	16	72 480
6	2	3 000	34	130 200

MMO 应用服务时切实可行.

5 结束语

本文归纳了 MMO 软件服务器的特点,并给出总体的设计思路.针对关键的技术问题,结合多种设计模式,详细论述了实现方法:通过 IOCP 机制构建网络通讯模块、通过代理模块实现模块加载的柔性配置、使用共享内存提高服务器运行鲁棒性.算法的优化以及服务器性能的提升将是下一步研究工作的关注点.

参考文献:

[1] 崔金红,王旭.能力共享架构及其在 IM 和 SNS 中的应用研究[J]. 计算机科学,2008,35(12):73-76.
[2] 黄超. Windows 网络编程[M]. 北京:人民邮电出版,2003.
[3] 刘海岩,锁志海,吕青,等.设计模式及其在软件设计中的应用研究[J]. 西安交通大学学报,2005,39(10):1043-1047.
[4] 钟金琴,辜丽川,张友华,等.设计模式的一种形式化描述方法[J]. 微电子学与计算机,2009,26(6):150-154.

作者简介:

李海燕 女,(1973-),硕士,讲师.研究方向为网络通讯及多媒体技术.

(上接第 96 页)

参考文献:

[1] Rashinkar P,Paterson P,Singh L. System on a chip verification methodology and techniques[M]. New York: Kluwer Academic Publishers,2001.
[2] 王继中,樊晓桢,黄小平.“龙腾”处理器 FPGA 验证平台的优化设计[J]. 计算机测量与控制,2009,17(1):170-172.
[3] Magnusson P S,Christenssonagnus,Eskilson J,et al. Simics:a full system simulation platform[EB/OL]. [2009-03-10]. IEEE Computer, <https://www.simics.net/simics/simics-computer-february-2002-wcover.peg>,2002-02.
[4] 吴松林,樊晓桢,安建峰.使用 Simics 模拟器辅助微处理器验证[J]. 科学技术与工程,2008,8(7):1847-1850.
[5] Virtutech AB. Simics/x86 Target Guide[M]. Simics3.0,

2006.

[6] Virtutech AB. Simics user guide for windows[M]. Simics3.0,2006.
[7] 黄小平,樊晓桢,贾琳,等.“龙腾”微处理器流水线的设计及优化[J]. 微电子学与计算机,2006,23(2):144-147.

作者简介:

赵君建 男,(1983-),硕士研究生.研究方向为计算机系统结构、专用集成电路设计.
张盛兵 男,(1968-),教授.研究方向为 VLSI 系统设计、计算机系统结构、电子设计自动化.
黄小平 男,(1979-),博士研究生.研究方向为计算机系统结构、专用集成电路设计.