|  |  |
| --- | --- |
|  | 学校代码：10246 |
|  | 学 号：093053093 |



|  |
| --- |
| 硕 士 学 位 论 文 |

（专 业 学 位）

|  |
| --- |
| **类Unity3D引擎组件模式的网络游戏服务端架构** |

|  |  |
| --- | --- |
| 院 系： | 软件学院 |
| 专 业： | 软件工程 |
| 姓 名： | 梅久华 |
| 指 导 教 师： | 李旻 |
| 完 成 日 期： | 2012年8月28日 |

# 目录

[目录 I](#_Toc336157405)

[摘要 III](#_Toc336157406)

[ABSTRACT IV](#_Toc336157407)

[第一章 引言 1](#_Toc336157408)

[1.1 网络游戏的市场背景和研究意义 1](#_Toc336157409)

[1.2 论文的主要内容 2](#_Toc336157410)

[第二章 网络游戏服务端架构现状及未来 4](#_Toc336157411)

[2.1 游戏网络技术的发展 4](#_Toc336157412)

[2.2 主流的网络游戏服务端架构 4](#_Toc336157413)

[2.2.1 主流服务端架构及其特点 4](#_Toc336157414)

[2.2.2 主流服务端架构在应用中的困境 5](#_Toc336157415)

[2.2.3 服务端架构的需求与未来 6](#_Toc336157416)

[第三章 类Unity3d引擎组件架构探讨 9](#_Toc336157417)

[3.1 Unity3D引擎 9](#_Toc336157418)

[3.2组件设计模式 9](#_Toc336157419)

[3.2.1.组件模式的核心思想 9](#_Toc336157420)

[3.2.2.组件模式 10](#_Toc336157421)

[3.3 Unity3D引擎的组件设计 11](#_Toc336157422)

[3.3.1 Unity3D引擎世界的组成 11](#_Toc336157423)

[3.3.2.依赖注入模式 13](#_Toc336157424)

[3.3.3 商业UI组件——NGUI 14](#_Toc336157425)

[3.3.4 组件架构将是未来的发展方向 17](#_Toc336157426)

[3.4 组件模式引入服务端架构的可行性研究 18](#_Toc336157427)

[3.4.1 组件模式服务端架构的研究现状 18](#_Toc336157428)

[3.4.2 组件模式应用于服务端的优势 18](#_Toc336157429)

[3.4.3 组件模式应用于服务端的挑战及应对原则 19](#_Toc336157430)

[第四章 组件模式服务端架构的设计与实现 21](#_Toc336157431)

[4.1 组件模式服务端架构基础系统设计与实现 21](#_Toc336157432)

[4.1.1 详解Multi-Server架构 22](#_Toc336157433)

[4.1.2 基础系统 22](#_Toc336157434)

[4.2组件模式服务端架构核心系统设计与实现 26](#_Toc336157435)

[4.2.1 核心系统的构成 26](#_Toc336157436)

[4.2.2 游戏主循环 27](#_Toc336157437)

[4.2.3 组件架构整体设计 29](#_Toc336157438)

[4.2.4 组件架构网络与消息系统 33](#_Toc336157439)

[4.2.5 组件架构数据查询系统 39](#_Toc336157440)

[4.2.6 独立组件 41](#_Toc336157441)

[4.2.7 组件间的相互协作 42](#_Toc336157442)

[第五章 组件模式服务端架构的应用实例 44](#_Toc336157443)

[第六章 结论 49](#_Toc336157444)

[6.1 类Unity3D组件模式网络游戏服务端架构的系统特点 49](#_Toc336157445)

[6.2 不足与展望 50](#_Toc336157446)

[参考文献 51](#_Toc336157447)

[致谢 53](#_Toc336157448)

# 摘要

游戏行业的飞速发展，带来巨大经济效益的同时，也给游戏的研发技术带来了新的挑战和机遇。商业引擎的运用让游戏客户端的研发经历了一场“工业革命”。也正式进入工具化生产的成熟阶段。但随着信息化的普及和人们对游戏交互性需求的日益增长，纯粹的单机游戏已经无法满足市场，玩家对游戏的网络属性提出了更高的要求。而相对滞后、封闭的网络游戏服务端开发技术，在市场竞争日益激烈的今天，很难在开发速度和质量上两者兼顾。因此业界提出了网络游戏“服务端引擎”的构想，而要想实现这一构想，就必须要从服务端架构上寻求突破，打造一个开放式、易扩展、高性能、支持商业化协作的服务端架构平台。

基于这一思路，本文在分析和比较了当前已有网络游戏服务端开发架构的优缺点之后，借鉴Unity3D引擎中组建模式架构的成功经验，构建了一个类Unity3D引擎组件模式的网络游戏服务端架构。架构运用组件设计模式思想，将服务端面向对象的程序设计进一步抽象，引入组件，强调软件的可重用性、交互性和开放性。以组件的产生、装配和协作构成组件模式服务端架构的设计核心。文中对实现该架构的关键技术进行了分析和讨论，剖析Unity3D引擎中组件模式的设计和商业化组件的应用模式，详细介绍了支撑组件模式服务端架构的组件设计模式、网络系统、基础系统、消息系统以及核心组件系统的设计和实现。并给出了运用组件模式服务端架构设计实现的游戏项目实例。

关键词 网络游戏，Unity3D引擎，组件设计模式，服务端架构

# ABSTRACT

The game industry's rapid development has brought huge economic benefits,but also brings new challenges and opportunities to the game's development technology.The using of commercial engine let the client game development has experienced a "industrial revolution".also Officially entered the mature stage of tool production. But along with the popularity of information technology, the growing demand of people for interactive in game,The pure single game has been unable to meet the market,the players put forward higher requirements to the network properties of the game

.But the lagging behind and closed technology of the online game service,in the more and more competitive market today, it is difficult to balance between speed and quality. So the industry puts forward the network game "Server Engine" idea, and to realize this idea, Must seek a breakthrough from the current server Architecture,creating an open,easy extension,high performance,and support commercial cooperation’s server Architecture platform.

Based on this idea, after analysing and comparing the advantages and disadvantages of the current existing network game server architecture, and learning the successful experience from the Unity3D engine components mode architecture, this paper constructed a network game server architecture like the Unity3D engine component model. Using the component design model thinking, Abstract server object-oriented programming, adding components, emphasize the reusability,interactivity and open of the software. Generation of components, assembly and collaborate to form the core of the design of the component mode server architecture.In this paper, the key technology to realize this framework are analyzed and discussed, from the Unity3D engine component model architecture design and commercial application of the components, to the key technologies of achieve this architecture,Component design mode,network systems, infrastructure systems, messaging systems, and the core component system, from design to implementation,all have carried out a detailed introduction.The End ,this paper has given a game project example that achieved by the component mode server architecture.

Keywords The Network Game,Unity3D engine， Component Design Model, The Server Architecture

# 第一章 引言

## 1.1 网络游戏的市场背景和研究意义

游戏从最开始的简单卡片单机到今天的即时战略、角色扮演、休闲养成等大型多人在线游戏（MMOG），从2D 到3D，从PC到移动终端。游戏无论在画面表现力还是技术成熟度上，都以惊人的速度向前发展和突破。终端技术的发展和融合，带来了游戏跨平台发展机遇，人们通过个人电脑、大型游戏机、家用游戏机、手持终端、交互电视设备都可以进行互动交流。网络游戏走向“娱乐随时随地”正在梦想成真。

目前，游戏业已经成为全球最大的娱乐产业。中国市场预计到2016年网络游戏用户将达2亿，销售收入将达到858亿元，而网络游戏带动的相关产业收入增长更是超千亿元。目前，中国游戏产业已经成为新的经济增长点，极大地刺激了中国经济的增长[1]。

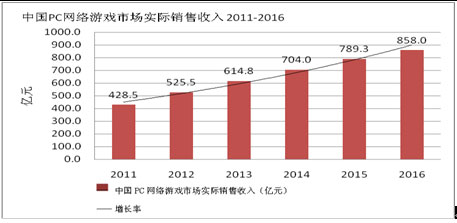


图1-1 中国PC网络游戏收入预计[1]

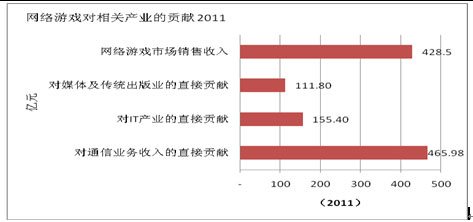


图1-2 2011年网络游戏对相关产业的贡献[1]

正是因为形成了如此庞大的产业经济和用户群体，那么对游戏行业的技术研究将具有充分的空间和市场价值。

随着信息化的普及和人们对游戏交互性需求的日益增长，纯粹的单机游戏已经无法满足市场，玩家对游戏的网络属性提出了更高的要求。而相对滞后、封闭的网络游戏服务端开发技术，在市场竞争日益激烈的今天，很难在开发速度和质量上两者兼顾。因此业界提出了网络游戏“服务端引擎”的构想，而要想实现这一构想，就必须要从服务端架构上寻求突破，打造一个开放式、易扩展、高性能、支持商业化协作的服务端架构平台。

## 1.2 论文的主要内容

本文在分析和比较了当前已有网络游戏服务端开发架构的优缺点之后，借鉴Unity3D引擎中组建模式架构的成功经验，构建了一个类Unity3D引擎组件模式的网络游戏服务端架构。架构运用组件设计模式思想，将服务端面向对象的程序设计进一步抽象，引入组件，强调软件的可重用性、交互性和开放性。以组件的产生、装配和协作构成组件模式服务端架构的设计核心。文中对实现该架构的关键技术进行了分析和讨论。本文的组织结构如下：

第一章 绪论，主要是对网络游戏的发展、市场规模以及研发技术发展进行了大体的介绍。

第二章 对当前主流的网络游戏服务端架构进行了剖析，并介绍了各自的利弊，以及总结了当前所有主流服务端在技术积累、复用，架构移植、可扩展、以及团队协作等方面存的共同缺点。以及探讨服务端架构的主要需求和解决方案。引入组件模式设计方案。

第三章 通过对Unity3D引擎中组件架构的应用研究，探索服务端开发工具化、开放式协作模式，以及将这一模式引入网络游戏服务端架构设计的可行性进行深入了研究。

第四章 详细介绍了类Unity3D引擎组件模式网络游戏服务端架构的详细设计和关键技术实现。包括基础架构、游戏主循环、网络系统、消息系统以及核心组件系统等。

第五章 介绍了一款使用Unity3D引擎以及组件模式服务端架构实现的3D休闲舞蹈类网络游戏《恋舞天使OL》的实现案例。

第六章 对类Unity3D引擎组件模式网络游戏服务端架构的特点进行了总结。



图1-3 论文的论述流程

# 第二章 网络游戏服务端架构现状及未来

## 2.1 游戏网络技术的发展

第一款真正意义上的网络游戏可追溯到1969年，瑞克-布罗米为PLATO系统编写的《太空大战》，它可以支持两人远程连线。这也是联机网络游戏的雏形。巧合的是，1969年也正是ARPAnet（美国国防部研制的世界上首个包交换网络）诞生的年份，ARPAnet的成功直接促成了互联网以及TCP/IP协议的诞生。在随后的四十余年时间里，一些专业的游戏开发商开始涉足网络游戏。游戏从最早的不可持续发展为可持续，玩家所扮演的角色可以在游戏世界中不断发展。而越来越多的开发商和发行商的介入，一个规模庞大、分工明确的产业生态环境最终形成。人们也开始思考网络游戏的设计方法，归纳系统的基础理论。随着网络时代的不断变迁和网络用户需求的不断高涨，大型网络游戏、web网页游戏、移动网络手机游戏等新型游戏形式不断涌现，网络游戏也迎来了鼎盛的发展时期[2]。

## 2.2 主流的网络游戏服务端架构

国内外网络游戏服务端架构技术的发展，也从另一个方面反映出网络游戏的发展历程。越来越复杂的网络游戏世界对游戏服务端架构也提出了更高的要求和挑战。目前在网络游戏行业，服务端普遍采用的架构主要有五种：Client/Server、Peer2Peer、Hybrid Client/Server、Multi-Server以及集群式架构[3]。

2.2.1 主流服务端架构及其特点

（1）Client/Server架构

多个Client连接至一个Server，Client只负责将玩家输入的信息发送给Server，Server处理大部分运算并将处理结果发回给Client。设计简单，玩家作弊情形不容易发生，但由于整个运算都是在Server端进行，所以Server的运算能力及网络的流量是真个系统的瓶颈，当Client没有收到Server的任何信息前，Client无法对玩家的输入做出任何反应，画面也无法及时更新，因此容易因Server运算延迟或网络延迟，造成游戏的不流畅，一旦Server达到上线或者Client增多时，则必须考虑使用功能强大的Server来取代。

（2）P2P架构

点对点构架最大的优势就是及时性，没有Server的介入，所有消息都是参与游戏的电脑之间的资料的传送。种构架避免了不必要的传送延迟，但是要在网络环境上建立点对点的架构，那么每台电脑必须对所哟的电脑先建立连线并做出传输的处理，因此电脑的运算能与连线的频宽会造成不小的负担。另外对于数据计算安全性要求较高的游戏类型，P2P架构风险较高。

（3）Hybrid Client/Server构架

此构架的特点在于Client可以自行推测目标的状态，并且可以立即针对玩家的输入做出反应。这种构架是对Client/Server架构和P2P架构进行的改进型整合。将部分计算分派的客户端，Server端进行适时核对。客户端可以依照资料对玩家的输入与玩家在游戏中的状态进行推测，并即时的反应给玩家。因此如果Client尚未收到Server信息，则Client端依旧可以进行游戏，但是最终数据的决定全仍然掌握在Server中，如果Client的自行计算结果与服务器的结果不相符合，则Server便会去修正Client的状态。此架构最大的问题在于网络延迟所带来的影响，若Client和Server之间传输延迟过大，则将会导致Client端所推测的资料库内容与Server端的资料库内容差距过大。

（4）Multi-Server架构

早起的mmorpg游戏是有单一的Server负责整个游戏的内容，由于是单一的Server，因此游戏中能够容纳的线上人数及玩家间的互动会受到限制。而在Multi-Server构架中，通过每一个Server负责一个部分的游戏的内容，但是在不同的Server上玩家长处于不同的游戏世界里，因此无法互动，为了要提高系统整体的效能有效利用系统的运算及频宽的资源，一半以空间切割的方式分配Server权限范围及适当划分Server负责的工作，是不同的Server负责不同区域间的玩家，因此能支持更多的线上玩家。在同时在线用户规模十万级的游戏中，Multi-Server架构被普遍采用。性能与用户规模在此可以达成一个较好的平衡[3]。

（5）集群式服务端架构

集群式服务端架构是大型网络游戏为了满足超大量用户同时在线需求，采用的Client->ConnServer->LogicServer方式，简单来说，其中ConnServer负责Client与Server的链接和负载均衡的分配，完成分配后ConnServer退出会话，Client与Server形成前面四种架构中的某种模式。此架构优点在于能同时承载海量用户同时在线，不足之处在于架构复杂，更新部署以及维护成本较高[4]。

2.2.2 主流服务端架构在应用中的困境

随着网络游戏架构的不断改进，发展到Multi-Server与集群式架构，单从某一个项目、某一款游戏的角度来说，已趋于成熟。但在网络游戏高速发展的今天，网络游戏的开发，已不仅仅是某一个项目或者某一款游戏的开发了，游戏研发公司，要想成功的研发出一款产品，可能需要从公司过往的项目中，同行业的成功作品中，去吸取经验并利用经验，才有可能在激烈的市场竞争中，找到一席之地。而单纯的依赖从零开始的研发，已很难在快速变化的市场中生存下来了。因此，项目积累、复用、移植以及支持工具化，这些极大提高开发效率的方式的采用，是如今一个网络游戏公司从粗狂走向成熟，并在激烈的竞争中立于不败之地的唯一办法。而这些，都需要建立一个支持研发成果高效积累和复用的基础平台。

而单一的服务端架构已无法解决这些问题，在多年的实践开发过程中，笔者总结了一下几点现有服务端架构存在的弊端。

（1）积累困难

游戏项目，特别是大型网络游戏，一般历时较长，投入较大，但网络游戏因其具有高风险的特征，因此失败的可能也很大。国内很多公司经常出现，由于项目失败，人员变动，造成公司大规模的投入变得颗粒无收。同一公司不同项目间，因标准不统一，架构移植困难，严重依赖熟悉架构的特定技术人员。以往项目的经验和成果，不宜被后续项目沿用。

（2）复用困难

无论是项目内部，还是不同项目间，虽有很多需求非常相同的模块，但往往因为接口和标准不同，难以复用，需要重复开发。

（3）协作困难

因项目成员过往经验不同，熟悉的架构也不尽相同，因此在开发中彼此需要互相学习和熟悉。沟通成本较大。而行业之间公司与公司、团队与团队的协作，更是天方夜谭。

如今的网络游戏行业，已不再是单一项目的竞争，而是一个不断积累、成熟、高效率立体式的竞争。现代网游企业，想要利于不败之地，必须要想办法解决难以积累、复用和协作等这些难题。

2.2.3 服务端架构的需求与未来

**（1）网络游戏服务端设计需求**

网络游戏服务端，作为网络游戏架构中的幕后支撑系统。在用户登录验证、负载均衡、用户管理、功能玩法、数据存储、玩家互动以及维护游戏世界秩序等等方面起着至关重要的作用。游戏服务端的开发包括基础架构系统和功能模块系统两大方面。而基础架构的设计往往决定着后期功能模块的开发效率和产品质量。

1．易积累、易重用、易扩展、易协作

网络游戏服务端，在网络游戏发展的前期，充当的角色大部分是“桥梁纽带”，但随着行业的发展，单单的这一角色已不能满足用户需求。特别是行业竞争日趋激烈的今天，如何提升用户细节体验将是开发团队必须面对的课题。这其中服务端就需要承担更多重任。这势必加大服务端的开发量，给开发进度造成压力。

而另一方面，一款网络游戏产品的研发周期动辄数年，更有甚者十年磨一剑。这在用户需求以及市场日益变化的今天，将给游戏产品带来巨大的风险。如何提升开发效率，缩短开发周期，将是网络游戏行业今天必须解决的问题。积累、重用，是解决这一瓶颈的有效办法。而扩展性，是在积累和重用的基础上，方便创新的源动力。良好的协作平台，是减少团队沟通成本，提高开发效率的基础保障。构建稳定的基础架构，支持项目的积累、重用、扩展，方便团队成员间的协作开发，将是服务端架构设计的重要需求。

2．高度交互

多用户交互,是网络游戏设计中的重点,也是网络游戏区别与其他游戏类型的主要特征。客户端与服务端的交互，玩家与玩家的交互，玩家群与玩家群的交互等等，构成了网游的核心价值。一款优秀的网络游戏,要运用多种手段为用户提供交互机会,并强化交互乐趣。多用户交互,主要通过用户间的比较、协作与对抗来体现。因此服务端架构是否能够良好的支撑高度的交互系统，将是评价未来网络游戏服务端架构优良的重要标准。

3．稳定、高效

而要想保障这一连串的互动性交互，服务端在其中充当了桥梁纽带的作用。因此服务端的稳定性，决定着所有在线玩家能否进行交互这一核心操作。稳定压倒一切，这一观点，在网络游戏服务端开发中，尤其重要！而要保障这些交互性操作顺畅如单机游戏，那服务端性能是否高效将起着关键作用。

**（2）服务端架构的未来**

单一架构有很多成熟技术和积累，它所存在的最大问题就是积累、复用和协作。而这一问题，在游戏发展的过程中，客户端的开发业曾经碰到过，并以游戏引擎的方式完美解决。游戏引擎是一套为了简化游戏开发，将诸多游戏开发工具整合到一个开发环境中的产物。这种介于硬件层面与用户界面之间的游戏中间层的封装模式，可通过其自身的通用化设计与强大的功能性，为游戏开发者提供更加简易、高效的开发空间，其通用的工具化思想，标准化的开发方法和易扩展的组件装配，从而产出更多的游戏产品，并为企业从日益激烈的游戏开发中获得更多的利益。

游戏引擎是游戏的核心组件，主要包括渲染引擎、物理引擎、网络引擎、碰撞检测系统、音效、脚本引擎、人工智能、电脑动画等等。由于引擎的开发和维护都具有相当的难度，因此开始出现专门开发和维护游戏引擎的厂商，这些厂商提供稳定且性能良好的引擎供游戏厂商开发游戏，如此一来游戏厂商便能将重心放在游戏的开发及设计上，而不再受底层的技术及工具设计烦恼[5]。

网络游戏服务端与客户端开发，在游戏世界的组成部分存在着诸多的相同点，在网络、管理与功能逻辑部分，又有着诸多的不同。因此，服务端的引擎化过程中，一方面需要借鉴客户端引擎的成功经验，另一方面需要解决两者之间的不同。服务端架构设计中，如何兼顾这两部分将是服务端架构的设计出路。因此接下来需要着重研究客户端引擎是如何解决开发中的积累、复用和协作的难题。以及将客户端引擎的成功设计引入服务端架构的可行性。

论文接下来将深入研究Unity3D引擎中组件架构，剖析其成功的设计思想以及组件化运作的协作机制。并通过商业UI组件NGUI的使用过程，论证组件模式方便快捷、易扩展、易装配的高效开发模式。

# 第三章 类Unity3d引擎组件架构探讨

## 3.1 Unity3D引擎

Unity3D引擎由于其完美的可移植性受到了全世界的瞩目。引擎全平台的支持PC，Mac，Web，IOS，Android，Wii，PS3，XBOX360，这种跨平台的能力，让人很难再挑剔。特别是Web, IOS和Android，这几个平台，在当前其重要性举足轻重。在游戏开发中，网页游戏、Android游戏以及苹果APPStore中数以万计的游戏产品都是使用Unity3D开发而来。

Unity3D引擎不仅移植性好，而且性能卓越，在开发效率上也是出类拔萃，并且具有世界上任何其他开发工具都不能与之媲美的性价比优势。特别是对于个人和小规模开发团队，Unity3D被视为最好的游戏引擎。在如此多的顶级商业游戏引擎中，能够脱颖而出，其难度可以想象是非常大的。计理念的前瞻性和技术上的创新是至关重要的两点。

本文着重关注的是在开发效率上起着至关重要作用的Unity3D引擎中组件模式的应用。并将这一成功模式借鉴到网络游戏服务端架构中。为网络游戏服务端开发向引擎化发展奠定良好的架构基础。在论述Unity3D组件模式之前，先简单介绍一下何为组件模式以及它的特点。

## 3.2组件设计模式

组件模式在游戏开发中的应用如此优秀，因此我们需要详细的研究这一模式的本质和原理。

3.2.1.组件模式的核心思想

组件模式的核心思想是树型结构，它是由n（n>=1）个有限结点组成一个具有层次关系的集合。

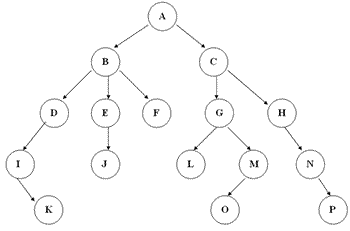


图3-1 树形结构

计算机科学领域中,数据结构被分为线性结构和非线性结构这两种类型。树形数据结构是一种常见的而且非常重要的非线性结构,它的应用非常广泛,也是层次模型的典型代表。在计算机领域中对树的研究主要针对于其不同的物理结构施加相应的算法来解决实际问题[6]。

3.2.2.组件模式

组件模式：允许你将对象组合成树型结构来表现“整体/部分”层次结构。模糊了对象和对象组合（简单元素和复杂元素）的概念，组件能让客户以一致的方式处理个别对象以及对象组合（简单元素和复杂元素）[7]。

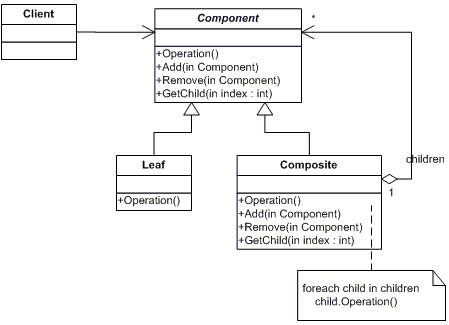


图3-2 组件模式UML图[7]

组件模式以不遵守单一责任原则换取透明性，让Client将对象和对象组合组合（简单元素和复杂元素）一视同仁。从而使得客户程序与组合对象（复杂元素）的内部结构解耦。

图3-6中，Component类：组件中的对象声明接口，在适当情况下，实现所有类共有接口的行为，声明一个接口用于访问和管理Component的子部件。

组件模式可以优化处理递归或分级数据结构。我们使用电脑时经常用到的文件系统就是其中之一。文件系统由目录和文件组成。每个目录都可以装内容。目录的内容可以是文件，也可以是目录。按照这种方式，计算机的文件系统就是以递归结构来组织的。在文件系统中，文件和目录都执行相同的接口，这是组件模式的关键。通过执行相同的接口，你就可以用相同的方式对待文件和目录，从而实现将文件或者目录储存为目录的子级元素[7]。

组件模式的适用性：

（1）你想表示对象的部分-整体层次结构。

（2）你希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象。

优点：

（1）组件模式可以很容易的增加新的组件。

（2）使用组件模式可以使客户端变的很容易设计，因为客户端可以对对象和对象组合（简单元素和复杂元素）一视同仁。

缺点：

（1）使用组件模式后，控制树枝组件的类型不太容易。

（2）用继承的方法来增加新的行为很困难[20]。

在网络游戏世界中，世界的构成有简单的物件，如一副只作装饰用的眼镜，也有复杂物件，如一个拥有种种技能以及点多钟服饰装扮的游戏角色。不管它们的复杂度如何，都是游戏世界中的一个物件。我们希望使用一副眼镜与使用一个角色有着同样的简单方式，而不会因为角色复杂性的增大，使得游戏的设计变得异常复杂。

因此，在网络游戏的开发技术架构中，创建层次结构，并可以在其中以相同的方式对待所有元素将是一个非常明确和现实的需求，因此组件模式不仅是游戏开发客户端架构最理想的选择，也同样是服务端架构最理想的选择。

## 3.3 Unity3D引擎的组件设计

3.3.1 Unity3D引擎世界的组成

Unity3D引擎中的游戏世界是一个完全面向对象的世界，引擎中所有的对象都源自都继承自根类Object。根类Object的下一级包括GameObject，Component，Mesh，Material，Texture，Shader，字面翻译为游戏物体，组件，模型，材质，贴图，着色器。分别承担着引擎中的不同分工任务，互相协作，完成游戏的制作和渲染呈现[8]。

这其中Mesh，Material，Texture，Shader与美术建模、绘画以及着色密切相关，属于游戏开发中美术相关类，本文不做深入讨论。GameObject与Component两种将是Unity3D引擎中组件模式的精髓，是本文重点研究对象，将会详细介绍将这一模式成功引入网络游戏服务端架构的过程。

简单一点来解释，开发工程师们利用物理引擎来搭建整个游戏对象即GameObject，用脚本引擎来驱动游戏对象完成特定功能，即Component。而这些特定功能，往往会将其实现成一个个组件。Unity3D引擎支持支持各类用户自定义组件，只需要该组件继承Unity3D提供的组件基类Component。

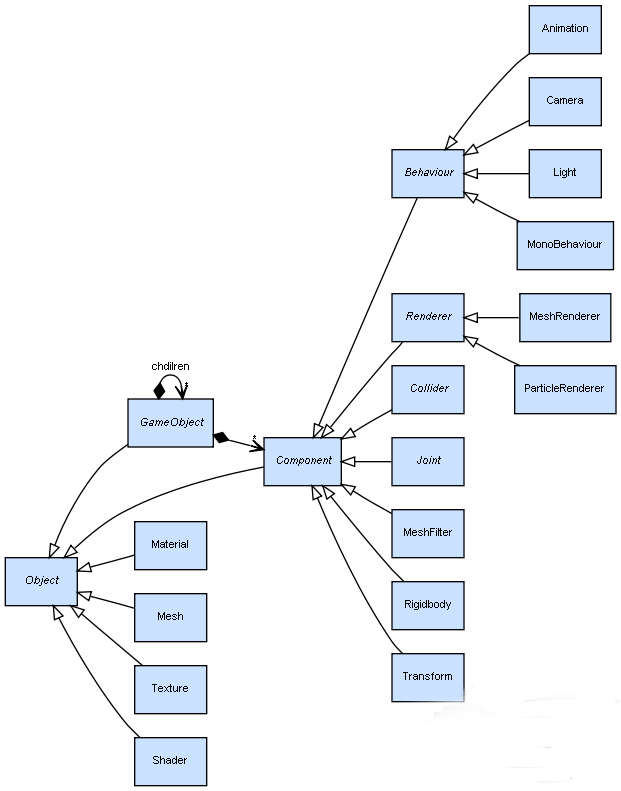


图3-3 Unity3D引擎世界的树结构[8]

游戏对象GameObject是所有其他组件的容器。在你的游戏中的所有物体都是包含不同组件的游戏对象。游戏对象被添加到游戏中时本身并不包含任何特性。当然，它们包含有已经被实现的控制组件。例如，一个灯光（Light）就是一个添加到游戏对象的组件。如果你想使用脚本创建一个组件，你需要创建一个空的游戏对象然后使用gameObject.AddComponent(ClassName)方法（函数）。你也可以在引擎的图形界面中创建游戏对象。并通过推拽的方式望其上面添加组件。从技术角度来讲，你可以不依赖游戏对象创建一个组件，但您将无法使用它，直到你将它应用到一个游戏对象。使用脚本，组件间可以轻易的通过发送消息或者使用GetComponent(TypeName)方法相互沟通。游戏对象除了被当做各个组件的容器外，它还拥有Tag（标签），Layer（层）和Name（名称）属性。Tag（标签）用途在于使用标签名来快速寻找物体，层可以用于投射射线，渲染或者对特定的一组物体单独应用光照[8]。

从图3-3中，我们可以看出游戏对象GameObject与组件Component之间的配合关系。游戏对象GameObject构成一个静态的游戏世界，而组件种类繁，且具有动态、可扩展特性。可以完成丰富的游戏功能，驱动静态的GameObject游戏世界变得生动有趣！如Transform组件可以控制游戏对象的位置、大小、缩放、旋转以及记录对象与对象的层次关系。我们可以通过GameObject创建一个人，将Transform组件添加到人身上，我们就可以通过Transform控制人的走动、跳跃等动作。将Rigidbody组件分别添加到两个人身上，则两个人相遇就可以实现碰撞效果。静态的游戏对象通过丰富的组件将整个游戏世界也变得丰富起来[21]。

3.3.2.依赖注入模式

Unity3D不仅支持引擎系统自带的以上各类组件，还支持用户自定义组件，只要该组件继承自Component组件基类。这一特性让Unity3D引擎变得无比强大。用户可以根据自己游戏的特点制作自己需要的组件，这些实现的组件库可以在任何游戏对象身上自由的装载复用。而更重要的是，专门的组件工作室或者公司，可以制作出通用的组件进行商业销售。各开发公司可以在整个组件市场，挑选自己需要的组件进行搭积木式的快速开发，不仅可以缩短软件产品的开发周期，同时也提高了系统的稳定性和可靠性。

而这种组件的生产和使用过程，有个生动的比喻。比如你想吃饺子，最原始的方式就是自己找面粉饺心，自己赶皮包心自己煮，这就是new的方式。后来卖饺子的餐馆出现了，你到餐馆去吃，餐馆负责做出他们擅长的饺子，这就是工厂模式。再后来，你只要自己坐在家里说，我要饺子，还是老面皮白菜馅的，然后饺子就出现在你面前了，这就是依赖注入模式。

依赖注入模式是继工厂模式之后又一思维方式的转变。他解决的是创建对象的方式问题。最原始的new方式，人们发现当项目规模大到一定程度的时候，当对象的创建者与对象的使用者不是同一人的时候，使用这种编码方式会带来比较紧密的耦合。比如业务层与数据层。业务层的编码者其实不关心你是以何者方式，向哪种数据库插入数据。他只关心你所能提供的功能。这时候就出现了工厂模式，工厂模式将对象的创建细节封装起来，实现了生产者与使用者的解耦。我们知道一个对象是由很多属性与方法组合起来的，工厂模式强调的是对象的批量生产，生产出来的都是事先规定好的规格。如果你想定制化生产，比如一部份属性由你来提供，工厂模式就不太好办到，这时依赖注入模式出现了。依赖注入不仅实现对象的自动化生产，还能实现个性化生产，把你所希望的元素加入进去[9]。

3.3.3 商业UI组件——NGUI

NGUI是Unity3D的一个插件，是一款商业的界面UI系统和事件框架，它提供了游戏开发中的界面系统的各类组件，使用它来制作你的游戏UI必定将事半功倍。为什么这么说呢？首先我们说说GUI与NGUI的区别，GUI是Unity自带的绘制界面工具，它的成像原理是基于表层的，所以执行效率非常的低，就算开发者硬着头皮写上去只能让UI的执行效率更低。然而NGUI 完全依赖与3D就好比在游戏世界中的摄像机直直的照射在一个平面中，在平面之上再去绘制自己的UI，所以它的执行效率会非常高。更重要的是GUI并且没有提供复杂的UI组件，如果要实现一个易用与美观兼容的界面系统，我们需要自己采用最原始的new的生产方式，开发出各类UI组件，这需要从最底层的绘制到上层管理，每一个环节都要涉及。而熟悉游戏界面开发的工程师们很清楚，开发出一套成熟的界面系统组件，将是一件多么庞大的工程。而NGUI为我们提供了一整套成熟的界面组件，很多复杂的UI界面，我们只需要简单的进行NGUI组件组装即可。由此可见，商业化的组件依赖注入模式，对于提高游戏行业的整体开发效率，有多么至关重要。

UI组件的配合使用方式如下例：

1. MonoBehaviour是Unity3D引擎自带的组件之一
2. UISprite是NGUI组件之一，作用是将图集的某张图片画出来
3. UIImageButton，通过MonoBehaviour与UISprite组件配合使用，产

生一个拥有有三种背景图片显示状态的按钮组件，

public class UIImageButton : MonoBehaviour{

//负责显示当前设置的图片

public UISprite target;

//三种状态下需要显示的图片名称

public string normalSprite;

public string hoverSprite;

public string pressedSprite;

//初始状态

void Start (){

if(target==null)

target=GetComponentInChildren<UISprite>();

}

//Hover状态

void OnHover (bool isOver){

if (target != null && enabled){

target.spriteName=isOver?hoverSprite:normalSprite;

target.MakePixelPerfect();

}

}

//Press状态

public void OnPress (bool pressed){

if (target != null && enabled){

target.spriteName=pressed?pressedSprite: normalSprite;

target.MakePixelPerfect();

}

}

}

上例中，通过两个简单组件，组合产生一个复杂组件的模式，让类似于NGUI这样的Unity3D引擎插件，能够方便的进行组件生产。而组件产生后的使用过程也是非常简单的。

如下是利用NGUI组件，创建一个UI界面的简单过程，而这一过程如果没有NGUI组件的帮忙，将会是另一番令人头疼的场景。

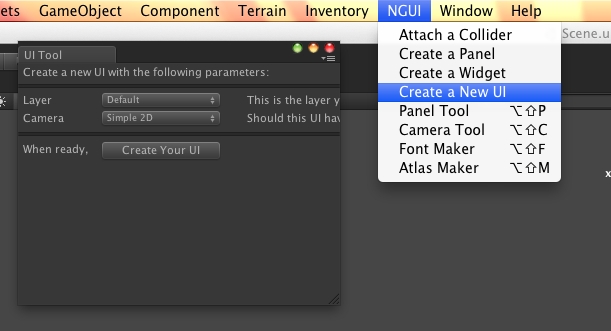


图3-4 NGUI创建界面UI过程一

在Unity3D导航栏菜单中选择NGUI->Create a New UI。如下图所示弹出UI Tool工具栏。选择好显示层和2D模式点击Create Your UI即可创建一个新的界面。接着NGUI会自动帮我们在Hierarchy视图中生成如下的对象关系。

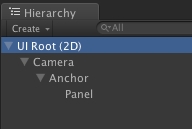


图3-5 NGUI创建界面UI过程二

接着我们开始创建界面组件，在Unity导航菜单栏中选择NGUI-> Create a Widget创建小部件，然后选择字体与图像集合，接着选择贴图的模板最后完成创建即可，这里我们创建一个简单的贴图。

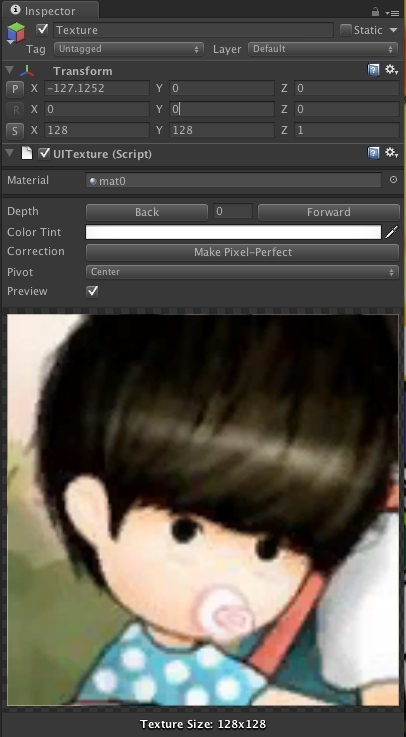


图3-6 NGUI创建界面UI过程三

OK下面我们将材质绑定在刚刚创建的Simple Texture身上，刚刚我们创建的模板为Simple Texture，NGUI会帮我们绑定UITexture脚本。界面创建就完成了[10]。

从上我们可以看出，组件模式的架构下的Unity3D引擎，对自定义组件的支持，将客户端的开发，变得更加开放。商业化环境下的各类优秀组件，让游戏的客户端开发变得越来越简单高效

3.3.4 组件架构将是未来的发展方向

面对网络游戏服务端日益复杂的需求和开发周期需要越来越短的矛盾，我们引入了重用思想，这与Unity3D引擎中的组建模式架构不谋而合。而组件化设计正是客户端解决开发效率的最有效方式

组件化程序设计方法继承并发展了面向对象的程序设计方法。它把对象技术应用于系统设计，对面向对象程序设计的实现过程作了进一步的抽象。我们可以把组件化程序设计方法用作构造系统的体系结构层次的方法，并且可以使用面向对象的方法很方便地实现组件[11]。

组件化程序设计强调软件可重用性和高度的互操作性。它侧重于组件的产生和装配，这两方面一起构成了组件化程序设计的核心。组件模型是一种规范，不管采用何种程序语言设计组件，都必须遵守这一规范。比如组装计算机的例子，只要各个厂商为我们提供的配件规格、接口符合统一的标准。这些配件组合起来就能协同工作。组件的装配使得软件产品可以采用类似于搭积木的方法快速地建立起来，不仅可以缩短软件产品的开发周期，同时也提高了系统的稳定性和可靠性[23]。组件程序设计的方法的透明性、可扩充性、可重用性与系统级的服务端架构非常吻合，具有强有力的基础设施，支撑系统一级的公共服务的能力[11]。

因为组件程序设计的特点，因此使用组件技术设计开发，具有以下一些优越性：

（1）降低应用软件的开发难度：由于采用了组件技术，开发设计人员可以更多的关注用户需求，在系统实现过程中采用合适的组件搭建框架，从而大大缩短开发周期，降低开发成本。

（2）可以灵活的定制产品：将开发需求中分离出变化和不变化的部分，并将变化部分粒度划分的更加精细，这样可以灵活的应对变化。

（3）可重用性强，组件一旦实现，就等于实现了一个标准化的服务，遵循标准化的输入，得到标准化的机构，组件内部甚至可以实现二进制级别的复用。

（4）有利于建设组件库，无论是对于单个企业还是整个行业，组件库的形成将是巨大的经验积累。对后续项目的开发将提供无可限量的帮助[22]。

网络游戏客户端组件模式的成熟应用，成就了诸多完美的游戏客户端引擎。极大的提高了客户端的开发效率。网络游戏的服务端面临前期客户端发展相同的需求压力，而组件模式的突出优点势必是助推网络游戏服务端向前发展的强力保障。

## 3.4 组件模式引入服务端架构的可行性研究

3.4.1 组件模式服务端架构的研究现状

组件模式在Unity3D引擎中应用成熟，但在网络游戏服务端的应用，目前国内外尚无成熟解决方案。主要原因是游戏作为一个网络产品时间还不长。研发的焦点还集中在客户端层面。依照单机游戏的成功模式，在向前探索。在一定时间内，服务端架构基本满足了市场需求，在网络游戏发展前期游戏产品供不应求的环境中，对于服务端的创新，经济效益远不及客户端。但随着客户端商业引擎的普及和开发技术成熟，以及资本市场的运作，网游行业竞争日益激烈，因此对产品的质量、稳定性，特别是产品的开发速度，提出了更高的要求。这时候没有引擎化成熟工具参与的服务端开发，就显得拙荆见肘了。因此，如何运用引擎工具化的思想，提升服务端开发速度和质量，就显得越来越迫切！

基于组件模式的服务端架构，正是服务端进行成熟工具化开发的必经过程，也是基础性架构。总体上说，程序设计的思想发展始终围绕着代码共享和标准化这两个方面进行着，组件是这种发展的新成果，组件化就是将其设计过程中基本稳定的部分抽取，固定下来，制作成软件中的标准件，也就是组件，从而将繁琐的设计工作进行简化。当服务端组件成熟运用，商业市场上的组件种类繁多，游戏开发团队能够进行自由的生产和装配的时候，服务端开发引擎也就真正走进网络游戏服务端的开发了[23]。

3.4.2 组件模式应用于服务端的优势

（1）与客户端相似的物理世界构成

网络游戏的开发中，由于无论是客户端，还是服务端，在游戏世界观中，两者有着非常相似的构成。某种程度上可以构建一个客户端与服务器端一一呼应的游戏世界，何为一一呼应，即客户端拥有一个A角色，对应的服务端也有一个A角色，客户端A角色身上拥有一件衣服c，对应服务端A角色拥有一件衣服c。这种物理的对应，让组件在服务端的应用将会得心应手。甚至可以完全模拟Unity3D引擎中GameObject与Component的配合模式。

（2）对服务端底层架构的需求非常相似

不同的游戏项目、游戏种类，虽然应用层和玩法上有许多的不同，但对底层支撑系统的需求非常的相似。比如对网络层的客户连接的维护，消息的接收、分发与发送，比如对登录系统、DB数据系统，几乎有着完全相同的工作流程。组件的引入将会使这些系统的复用变得异常简单。

（3）基础应用系统需求相似

虽然游戏种类不尽相同，但不同种类的游戏中，都存在着大量的相同基础系统。比如物品系统、角色系统、商城系统、好友系统、聊天系统、组团系统等等，这些基础系统功能和需求非常类似，可复用性强。比如物品系统，尽管游戏种类不同，但游戏中对物品的添加、管理、使用、删除等等操作需求，几乎一样。

3.4.3 组件模式应用于服务端的挑战及应对原则

（1）服务端逻辑代码多，不易工具化开发

服务端开发与客户端的一个最大不同就是，客户端偏重对实体对象的控制和管理，而服务端偏重于对这个实体对象对应的逻辑功能的实现。对实体对象的控制在游戏引擎中可以方便的进行所见即所得的方式开发，而服务端偏重与逻辑功能的开发则更偏重于代码的编写。

如果将逻辑也看作面向对象设计中的逻辑对象，那应对这一挑战的方式件很简单了，逻辑流程，对应于逻辑对象，一类逻辑对象完成一个特定功能。服务端的逻辑开发转变成标准化逻辑对象的开发，以及使用标准化逻辑进行复杂逻辑对象的组装工作。

（2）游戏应用层各不相同，不易复用

由于不同游戏，有着不同的应用形式，比如，舞蹈游戏，的核心应用是舞蹈，而策略类游戏，的核心应用是攻城拔寨。那这两类不同的游戏如何进行组件的开发。

当然，这类挑战，不仅仅存在于组件架构之中，所有的游戏开发架构都存在。不同种类游戏的应用逻辑开发，对于任何一款游戏而言，都无可避免。任何一种架构，都无法解决将舞蹈复用到攻城，但真正的问题是，如何利用项目积累以及行业积累，减少同类型游戏的重复性开发，最大限度的提高开发速度和效率才是问题的关键。在传统的服务端架构中，各个游戏项目组都在一个封闭的开发环境中，各自开发，面对不同的上层应用，需要从头从零开始。而组件架构，提供的是一种接口，一种标准。每一类应用，按照标准开发，特定组件提供特定服务。跳舞类游戏实现了一套舞蹈动作组件，那该动作就可以在任何基于组件模式架构的服务端进行装配使用。同样策略类游戏实现了一个战斗组件，那这个战斗组件也可以同样复用于其它基于组件架构的服务端。总之，解决这一问题的关键是，标准化组件接口，这样无论是同一公司不同项目或者行业间通过组件的商业合作，都将变得非常简单。

应对挑战，解决办法的宗旨是提取共同点、抽象、设计合理的组件化粒度，最后实现组件应遵循以下几个原则：

（1）独立性原则：所谓独立性原则是指封装的功能模块具有高内聚特点。可以是逻辑内聚，也可以是功能内聚或者数据内聚。做到组件与具体应用无关。

（2）基础性原则：所谓基础性原则指的是组件的粒度问题。如果组件包含太多功能则会影响组件的拼装的灵活性。功能太少则会导致组件数量增多，也会带来使用者组合的难度。作为游戏开发，组件化的第一步，就是基础性模块，将粒度定为基本的某块系统功能，如网络组件，只提供网络连接的维护和消息的接收发送工作。

（3）通用性原则：通用性指的是设计的组件，具备一定的普适性，因此，组件设计，不能过于依赖上层应用。因此需要进行抽象化的分析工作。

（4）简化原则：组件化的目标是使客户程序完成既定任务的过程更方便，因此不能将部分功能组件化后整个系统更加复杂。比如游戏中一些复杂的算法，可以做成算法组件[22]。

在组件模式应用于服务端的过程中，遵循组件模式的设计原则，将会最大限度的规避挑战，从而发挥组件模式的最大优势。接下来的章节，将进一步论述如何设计和实现网络游戏服务端的组件架构。

# 第四章 组件模式服务端架构的设计与实现

## 4.1 组件模式服务端架构基础系统设计与实现

遵循组件设计模式的独立性、基础性、通用性和简化设计原则，在网络游戏服务端基础架构中，借鉴Unity3D引擎中的组件设计架构，对目前服务端成熟的Multi-Server架构，对其进行组件化改造，形成类Unity3D引擎组件模式服务端架构。

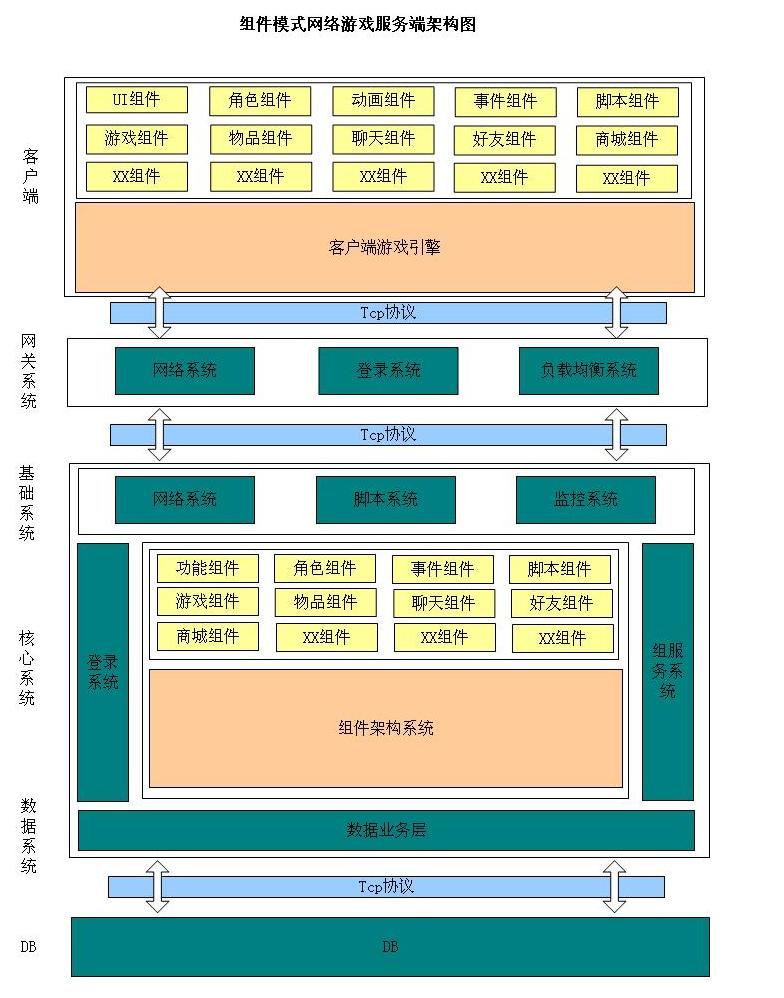


图4-1 组件模式网络游戏服务端架构图

将组件架构划分为基础系统、核心系统。基础系统为核心系统正常工作提供基础保障，核心系统主要包括游戏主循环、组件系统、网络系统、消息与数据查询系统以及独立组件，是组件化模式的核心保障，方便组件模式下各环节的工作。

4.1.1 详解Multi-Server架构

随着网络游戏行业的发展以及用户规模的急剧增长，单一的Server架构很难在性能上满足现代网游的需要。因此在Server端进行优化分工，也是大势所趋。而Multi-Server架构正是在这样的市场需求下应运而生。简单来说Multi-Server架构就是将原来承担所有任务的单Server，按照不同功能进行分配，形成各种按功能划分的Server，比如将维护用户网络连接部分划分出来，形成GateServer；将账号登陆验证部分划分出来，形成AccountServer；将数据库存储部分划分出来，形成DBServer；将游戏功能划分出来，形成GameServer；将GameServer与GameServer之间交互的部分划分出来，形成GroupServer等等。如此一来，游戏服务端资源将可以通过增加硬件服务器的方式得到极大的增强，不同的游戏服务器承担着不同的任务，用户体验得到较大的提升。

Multi-Server架构为了达到用户体验与性能最优，一般采用分区的策略。也就是在游戏运营中，将游戏人为的分成不同的独立游戏世界——游戏大区。每一个大区拥有自己独立的游戏数据库，同一大区内的所有玩家可以进行自由的游戏。但不同大区之间以各游戏以自身情况情况，支持度各不相同。同一大区支持的在线用规模，可达数万至数十万。而通过增加游戏大区，在线用户可以成倍增长。这对于绝大多数网络游戏来说，该架构一方面充分满足了大区内游戏玩家的高品质体验，另一方面每一个大区用户规模的和服务端承载状况都是可控可调整的，在实际运营中，方便维护和改进；其三，当用户量超过预期时可以通过增设大区来满足用户的需求，提高整款游戏的总在线人数和收入。因此这一架构在网络游戏的实际应用中被广泛采纳。

Multi-Server架构是网络游戏服务端架构从简单粗狂走向成熟的重要标志。Multi-Server架构也在国内外，不同的游戏公司之间，按照各自的需求，形成了不同的划分版本。但不管形式怎样划分，但其宗旨相同，也就是根据不同的游戏项目，对游戏性能、用户体验、网络状况等进行综合评估和考量，然后对服务端功能进行合理的优化分工协作，已达到用户对游戏的最佳体验。本文中组件模式服务架构也是建立在Multi-Server架构的基础之上。采用的是国内某游戏公司多个上线产品采用的架构。

4.1.2 基础系统

基础系统包括网关系统、脚本系统、数据系统和监控系统。网关服务器（GateServer）是网关系统的重点。负责维护客户端的所有连接与状态维护，客户端登录，各游戏Server的负载均衡，以及网络消息的检查与传送。是客户端与服务端的安全关卡和网络“信使”。脚本系统引入小巧、灵活、易扩展的lua脚本，监控系统及监控服务器（Agent），负责监控服务器所有运行进程以及硬件资源情况，出现宕机情况监控服务器会再次启动该Server，并向监控客户端发送各服务详细信息。

（1）网关系统

网关服务器的网络系统、登录系统和负载均衡系统按照组件架构要求，开发成对应的网络组件、登录组件和负载均衡组件。网关服务器工作时序图如下：

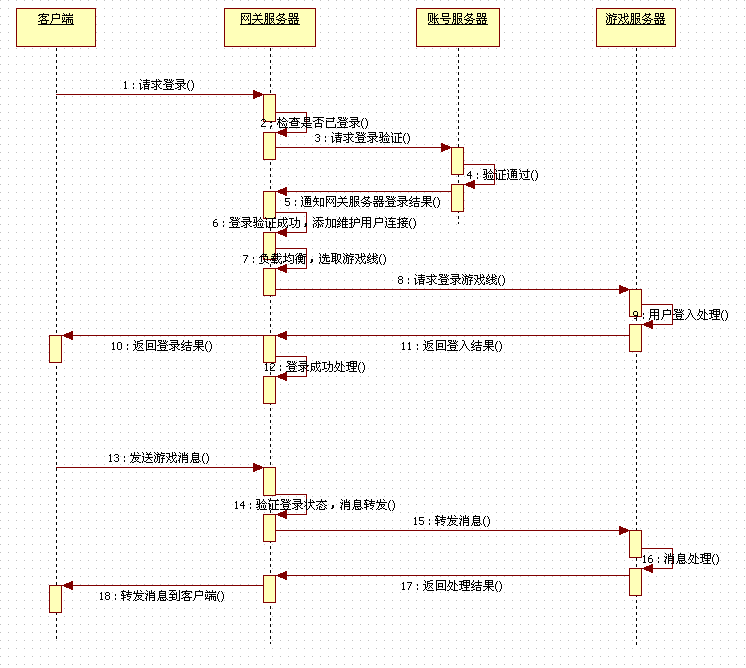


图4-2 网关服务器工作时序图

（2）脚本系统，Lua组件设计

Lua脚本可以很容易的被C/C++代码调用，也可以反过来调用C/C++的函

数，这使得Lua在游戏开发中可以被广泛应用。不仅仅作为扩展脚本，也可以作为普通的配置文件，代替XML,Ini等文件格式，并且更容易理解和维护。基于其无需编译易扩展的特点，lua主要使用在经常变化或经常需要扩展的部分。如物品系统，物品在游戏运营过程中会经常更新，为了减少常规版本更新给用户带来的下载不便，使用lua开发物品的功能逻辑，这样添加一个物品，则只需添加一个对用lua脚本文件即可，而无需更新游戏的可执行文件。

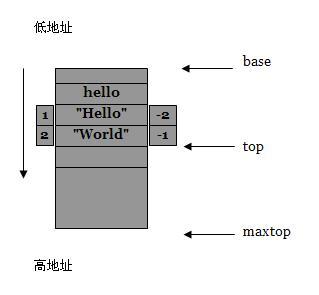


图4-3 lua栈

为了让lua在游戏开发中的使用更加简单，脚本系统需要封装lua与c++的函数接口，以及需要统一函数调用、参数与返回值传递的方法。并将lua部分组件化，方便所有的lua功能通过组件接口对外开放，方便其他系统重用该lua组件。Lua调用C函数的时候，使用和C调用Lua相同类型的栈来交互。C函数从栈中获取参数，调用结束后将返回结果放到栈中。为了区分返回结果和栈中的其他的值，每个C函数还会返回结果的个数[12]。

（3）数据系统

数据系统采用独立查询线程运行，主要部分包括查询管理器QueryManager与数据库接口CBDInterface。数据存储使用开源免费的MySQL5.1，MySQL在游戏行业中被普遍使用，一方面是由于MySQL的性能特点完全适合游戏数据库的规模和特点，在性能上可以完全胜任，另一方面MySQL是开源免费软件，在商业环境中应用，可以节省费用不低的数据库支出费用。

数据系统工作流程图如下：

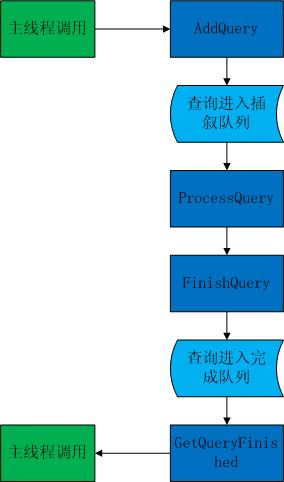


图4-4数据库查询流程图

（4）监控系统

监控服务器（Agent）采用游戏服务器（GameServer）相同的底层框架，只是上层服务相对简单很多。监控服务器由一个主循环负责连接和接收各个服务器的消息，搜集硬件及软件运行信息，根据运行信息及时对各游戏服务器状况进行处理。并将信息汇总到监控客户端，供游戏运维人员实时了解查看。监控服务器框架设计同后续章节中将要介绍的游戏服务器，因此此处不作更深层次介绍。

## 4.2组件模式服务端架构核心系统设计与实现

4.2.1 核心系统的构成

核心系统包括游戏主循环、组件系统、网络系统、消息系统、数据查询系统以及独立组件，围绕并使用这些核心系统，进行不同功能的装配，网络游戏服务端构建了三大游戏服务器，账号服务器（AccountServer），游戏服务器（GameServer），组服务器（GroupServer）三大游戏服务器。三大服务器互相协作运行，账号服务器负责账号的登录验证，并将验证结果通过网关服务器传递到游戏服务器，游戏服务器进而完成角色的信息读取和创建。游戏服务器是整个游戏的核心，几乎所有的游戏功能系统都在游戏服务器运行，为了缓解游戏服务器的压力，在一组游戏服中，一般都会设有多台游戏服务器，并增加一台组服务器，负责多台游戏服务器之间的相关通信及全局数据管理[14]。一组服务器的部署图如下：

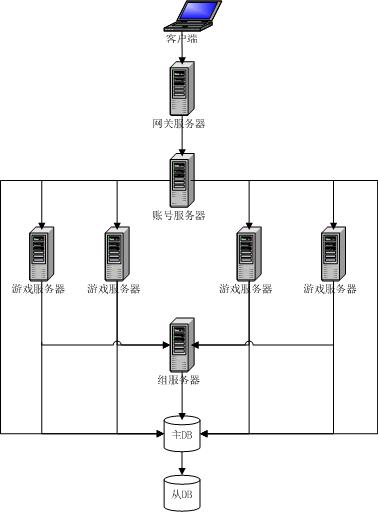


图4-5 服务端部署架构图

4.2.2 游戏主循环

游戏主循环是每一个游戏Server运行的主线程，其重要任务就是完成游戏Server的初始化、创建查询与网络子线程，定时获取子线程中接收到的数据库查询和网络消息，维护各游戏系统心跳频率。

读取全局配置，完成游戏Server的初始化，是游戏主循环（主线程）的第一项任务，通过ini文件系统，我们可以很方便的进行文件配置和读取，另外，由于框架底层引入的lua系统，我们也可以采用lua配置文件方式。分别使用DB数据库查询框架以及网络框架提供的接口，创建查询线程和网络线程[15]。

游戏主循环（主线程）的工作任务和实现非常的简明扼要。其控制全局的地位在代码实现中尽可一览无余。而这样的主循环，不仅仅适合于游戏服务器（GameServer），对所有的服务端服务器如账号服务器（AccountServer）与组服务器（GroupServer），主循环都能适用。因此将主循环封装成GameComponent类型组件，可使用组件模式架构快速开发其余各游戏服务器。组件模式架构在多类型网络游戏服务器中可得到进一步扩展。

主循环初始化过程，包括读取配置文件，创建数据库查询线程和创建网络线程。实现如下：

bool CCirculatorBase::Initialize(){

//通过ini读取配置文件，设置心跳间隔

m\_nHeartBeat = g\_Config.GetInt("interval","heartbeat",25);

//创建数据库查询线程

m\_pQueryMgr->Initialize(&m\_condNotify, &m\_mutexNotify)

//创建网络线程

bRes = InitSocketMgr(m\_pSocketMgr, nMaxClient, nPort);

}

主循环的启动过程，读取配置文件成功后，视情况创建新的主线程，还是直接将运行线程转换为主线程。然后将游戏控制权移交主循环，也就是主线程。

bool CCirculatorBase::Start(bool bCreateThread){

if(!Initialize()){

return false;

}

//创建线程，启动主循环

m\_bRunning = true;

if(bCreateThread){

#ifdef WIN32

m\_hThread = CreateThread(NULL,0,ThreadProc,this,0,NULL);

int nRes = m\_hThread == NULL ? 1 : 0;

#else

int nRes = pthread\_create(&m\_thread,NULL,ThreadProc,this);

#endif

if (nRes != 0){

m\_bRunning = false;

}

}else{

ThreadProc(this);

}

return m\_bRunning;

}

Windows版本主线程，主要工作为阻塞式监听其余子线程事件，如查询线程或网络线程有事件，主线程立即进入主循环任务处理环节。

DWORD CCirculatorBase::ThreadProc(void\* pParam){

CCirculatorBase\* pCirculator = (CCirculatorBase\*)pParam;

pCirculator->m\_dwTickLast = GetTickCount();

while(pCirculator->m\_bRunning){

//主线程阻塞，直到收到m\_hEventNotify事件，或者心跳时间到

WaitForSingleObject(pCirculator->m\_hEventNotify,

pCirculator->m\_nElapseHeartBeat);

if(!pCirculator->m\_bRunning)break;

//处理游戏主循环需要处理的任务

pCirculator->PumpProc();

}

TRACE("Game thread exited!\n");

return 0;

}

主线程需要完成的任务，主要包括处理查询线程中的查询返回和网络线程中的接收到的网络消息。

void CCirculatorBase::PumpProc(){

LPGameMsg\_Base pMsg = NULL;

unsigned short nSlot;

int nCount = 0;

//处理数据库查询

if (m\_pQueryMgr){

CQuery\* pQueryFinished = NULL;

while((pQueryFinished=m\_pQueryMgr->GetQueryFinished())){

ProcessQueryFinished(pQueryFinished);

delete pQueryFinished;

}

}

//处理网络消息

if(m\_pSocketMgr){

while(m\_pSocketMgr->GetGameMsg(pMsg, nSlot)){

m\_SlotPeer->SetSlot(nSlot);

GameMsg\_Map::Instantce().ProcessMsg(\*pMsg,\*m\_SlotPeer);

delete pMsg;

}

}

}

由此可见，组件模式运用得当，在网络游戏服务端开发中，不仅可以应用于游戏服务器中功能玩法，角色、物品、NPC等游戏对象的组件开发，同样可以应用于不同的游戏服务器开发，包括但不限于网关服务器、账号服务器、组服务器等。

组件模式架构当发展到能涵盖游戏服务端的所有开发内容的时候，也就形成了建立在组件模式架构之上的集成化开发工具——服务端引擎了！

4.2.3 组件架构整体设计

服务端的开发主要集中在游戏服务器（GameServer）上。这也是本文研究的意义所在。游戏系统中引入组件架构，将会使这一系统处于开放状态，标准化组件接口，引入商业组件，是解决游戏系统开发瓶颈的有效办法。因此我们引入组件架构设计。

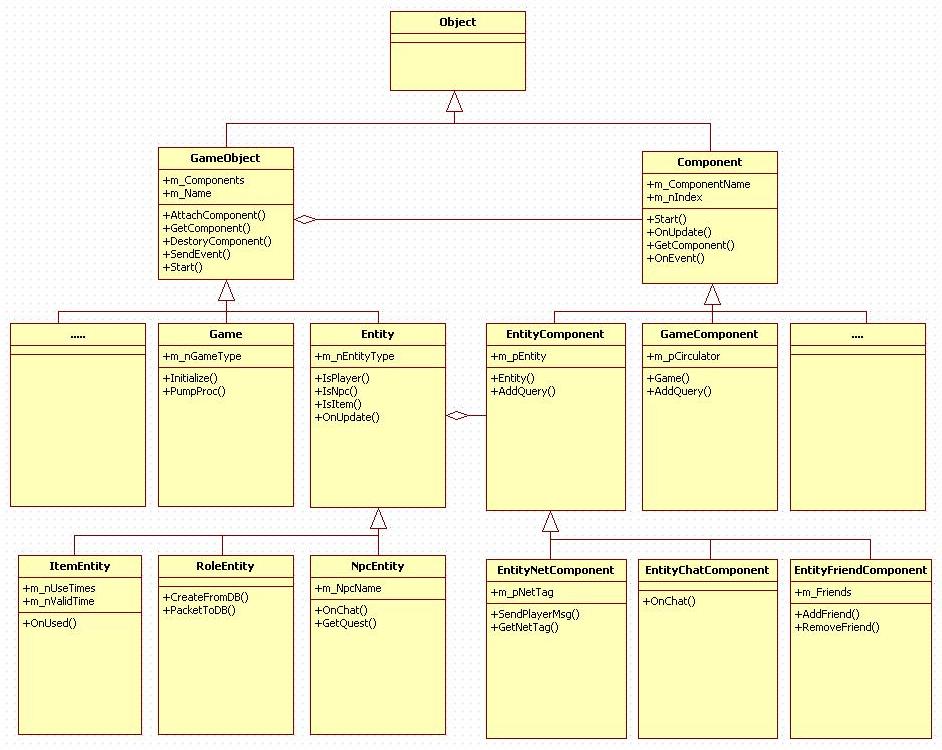
在前面章节中，我们介绍过Unity3D引擎中组件模式的应用，客户端GameObject与Component协作的工作方式，让客户端组件的商业化开发趋于成熟，我们将这一设计理念引入服务端架构。将服务端的游戏对象定义为GameObject，如一个角色，一个物品，游戏对象GameObject是所有其他组件的容器，将服务端的各类功能定义为组件，如聊天功能定义为聊天组件，好友功能定义为好友等等。将聊天组件添加到角色身上时，角色即可通过聊天组件提供的接口进行聊天功能操作。组件间可以轻易的通过发送消息或者使用GetComponent(TypeName)方法相互沟通。设计类图如下：  


图4-6 组件架构设计类图

上图中Object是所有对象的基类，GameObject为所有游戏对象的基类，Component为所有组件的基类，它们分别提供标准化的使用接口，供继承类定义实现。

Game类和Entity类为继承GameObject的游戏对象，Game类为创建不同类型的游戏服务器设计，如创建账号服务器，我们只需要创建一个Game对象，然后往Game对象上添加账号服务器相关的功能组件即可，如网络组件，数据库组件，登录验证组件，然后账号服务器就通过组件合作的方式简单快速的开发出来。Entity类为游戏服务器中的实体对象基类，如继承它的ItemEntity为一个物品，RoleEntity为一个角色，NpcEntity为一个NPC对象，通过往RoleEntity对象添加EntityNetComponent组件，RoleEntity就具有了网络属性，可以通过网络组件进行网络消息的收发。

EntityComponent类和GameComponent组件，分别是针对Entity与Game开发的组件基类，如EntityNetComponent网络组件，EntityChatComponent聊天组件，EntityFriendComponent好友组件，它们可以自由的装配到Entity对象身上，如将EntityChatComponent组件装配到NpcEntity对象身上，那这个NPC就可以对话啦！

类Unity3D组件模式的服务端架构设计，同样可以让网络游戏服务端的开发像客户端引擎化开发一样，变得简单高效。标准化不同的Component组件接口，即可实现服务端组件的商业化开发。

组件模式架构的服务端架构不仅具备组件设计模式的所有优点，如件可以很容易的增加新的组件。而且这些配件组合起来就能协同工作。简单组件和复杂组件一视同仁，方便统一管理。组件的自由装配，不仅可以缩短软件产品的开发周期，同时也提高了系统的稳定性和可靠性。还有更重要的一点，那就是服务端的组件可以做到与客户端组件一一对应，实现组件之间点对点的通信（可参见图4-1中的客户端与服务端一一对应的黄色组件部分）。如此一来，游戏组件则无需关注游戏的内部逻辑结构，只需要关注组件自身，逻辑复杂度大大降低。

在客户端与服务端组件一一对应的情况下，用户交互将会变得更加高效。如EntityChatComponent组件的实现，用户的聊天交互，将无需经历复杂的中间过程，只需在交互双方的EntityChatComponent组件之间进行，如下图。

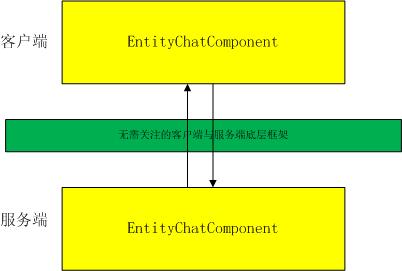


图4-7 组件一一对应交互

为了实现组件架构中组件之间如此高效交互需求，关键是网络消息流的设计，底层架构中消息流始终是一个重要的课题，而在网络游戏中，以用户与服务端的交互贯穿始终，所以，在网络游戏服务端开发中，以消息驱动为主的逻辑系统中，消息的接受、处理、分发则显得尤为重要。

组件模式服务端架构中，所有的基础系统都是为组件系统服务。而组件系统的设计中，我们采用类Unity3D引擎中的组件模式架构。即游戏对象（GameObject）与组件（Component）互相配合，相辅相成构成整个游戏的开发模式。游戏对象组成游戏主体，组件形成标准化，功能强大的组件必然丰富多样，游戏开发将会变成简单的组件装配及少量的集成开发。

因此在组件系统的实现中，要进一步强调组件接口的标准化，以及组件如何才能简易的装配到游戏对象。

在组件的设计类图中，我们定义了所有组件树形结构的根节点Component，也就是所有派生组件的基类，在基类中，我们定义了所有组件具有共性的重要属性和接口，如组件自身的名称、组件在游戏对象中的索引值，获得宿主、兄弟组件的方法，为游戏角色定制的Component的派生类EntityComponent组件同时继承了GameMsg\_Processor，因此具有网络层和DB层的通用属性和接口。可以注册自身关注的网络消息和数据库查询[15]。

//由于组件装载时已经存贮了宿主对象，因此获取宿主对象的实现非常简单

GameObject \* GetGameObject(){

return m\_pGameObject;

}

//通过组件的名称，可以很方便的通过宿主对象找到兄弟组件

template< class T > T\* GetComponent(){

T\* pResult = NULL;

if(m\_pGameObject != NULL){

pResult = m\_pGameObject->GetComponent<T>();

}

return pResult;

}

另一方面，作为组件的宿主，游戏对象树形结构中的基类GameObject，在组件模式架构中，同样起着非常重要的作用。游戏对象也同样拥有自身的名称，通过该名称可以在游戏世界中检索到该对象。游戏对象拥有保存注册组件的双重数据接口，线性数组以及非线性的映射表，线性数组方便顺序遍历各注册组件，非线性映射表方便快速的通过索引值找到对应组件。

组件装载接口实现的关键技术：在组件的装载过程中，m\_Components为存储组件的非线性映射表，可以快速查找，映射表通过组件类型名作为主键，m\_ComponentArray为线性数组，按注册的先后顺序存储组件。装载完成记录下自身在宿主对象身上的索引值。最后还有一件很重要的任务需要完成，那就是启动Start（），这一步的作用是关联所有需求明确的需要交互的各类组件。方便这些组件之间的沟通协作。

Void AttachComponent(Component & pComponent){

const type\_info &info\_type = typeid(pComponent);

if(m\_Components.find(info\_type.name())== m\_Components.end()){

m\_Components[info\_type.name()] = &pComponent;

m\_ComponentArray.push\_back(&pComponent);

pComponent.SetIndexInObject(m\_ComponentArray.size()-1);

pComponent.AttachToEntity(this);

if( m\_bStart ){

pComponent.Start();

}

}

}

为了与Unity3D的使用习惯保持一致，基类还提供了装载组件的模版接口

template< class T > T\* AttachComponent(){

T \* pt = new T;

AttachComponent( \*pt );

return pt;

}

组件检索接口实现的关键技术：由于在组件装载实现中，已经充分考虑到了组件检索的需求，所以在组件检索的实现中，我们就非常方便了。通过装载时的模版类型T，我们直接可以进行检索。

template<class T> T\* GetComponent() const{

T\* pResult = NULL;

const type\_info& info\_type = typeid(T);

ComponentCollection::const\_iterator it =

m\_Components.find(info\_type.name());

if(it != m\_Components.end()){

pResult = (T\*)it->second;

}

return pResult;

}

另外，基类还提供了直接通过组件索引进行检索的接口实现。

Component \* GetComponent(int Index) const{

return m\_ComponentArray[Index];

}

4.2.4 组件架构网络与消息系统

**（1）网络系统Socket库**

基础系统中网络Socket库是最基础也是最重要的系统，主要负责各个服务器之间的连接以及消息收发。所有的游戏服务器均采用同样的底层Socket库。

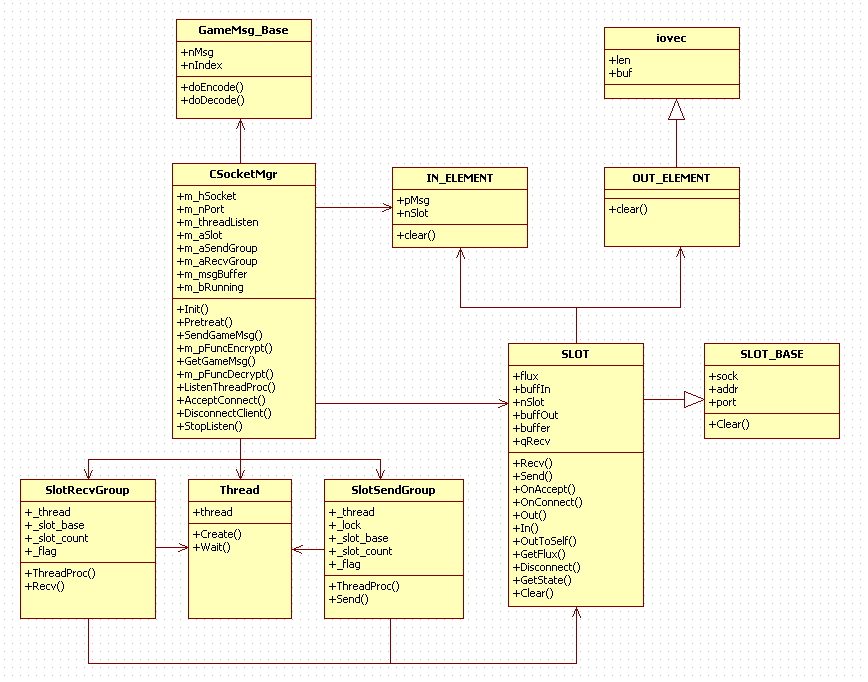


图4-8 网络socket设计类图

整个socket采用分组多线程与环形缓冲区技术，适合大量客户同时连接，这也是为了适应网络游戏的特点专门设计。CSocketMgr负责所有客户连接与管理工作。在每一个连接SLOT产生时，均衡的分配到Socket的组管理线程中，每一个组管理线程尽量达到负载均衡，并接管该SLOT的网络消息监听、接受以及发送工作。并在发送消息环节，支持缓冲区内的多条消息同时发送。在实际项目测试中，单条游戏服务器同时承载2000玩家在线，网络Socket毫无任何压力。这套专门为网络游戏行业服务端开发的Socket库表现出了优异的性能和稳定性，这也为组建模式底层消息框架打下坚实的基础。为上层组件之间的高效交互提供了保障[19]。

**（2）消息系统设计**

为了高效支撑组件之间的消息交互，底层架构必须实现组件的快速查找以实现消息的快速分发。为此，组件架构将采用消息定位与组件注册双重技术保障。

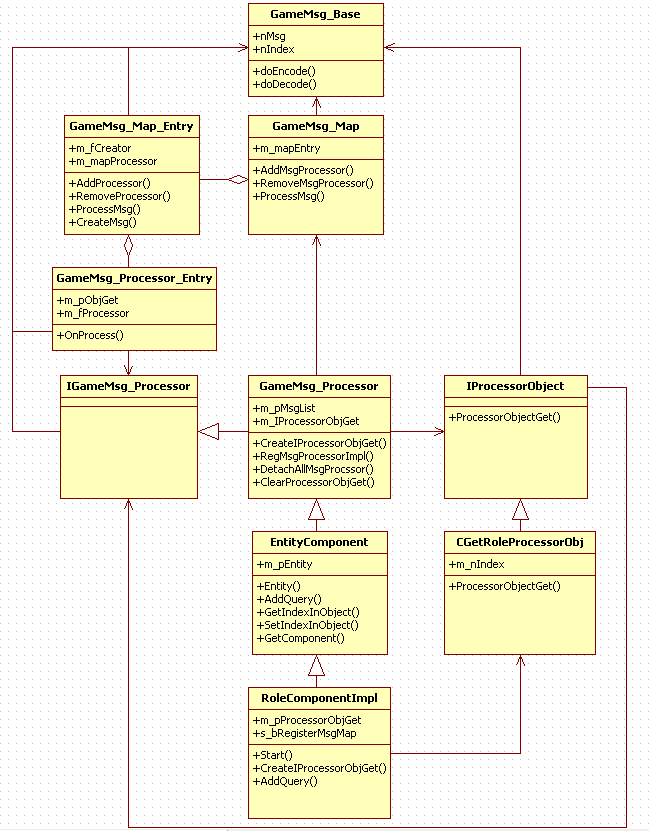


图4-9 组件架构消息框架设计类图

消息定位技术，我们在消息基类GameMsg\_Base中，引入索引字段Index记录游戏角色在服务端管理数据结构中的索引，为了不暴露内部数据结构，客户端发送的消息中并不包含该值，只当消息到达网关服务器时，该值由网关服务器填充。游戏服务器使用该索引值可以以查找复杂度为1的最快速度，找到角色对象。接下来消息快速分发的任务就交给组件注册技术了。

组件注册技术，正如设计模式中的观察者模式，组件关注什么消息，则向底层框架注册什么消息，这样底层架构收到消息，则会明确知道哪些组件关注此消息，并实现分发。而为了实现底层框架能够快速找到组件以及组件中对应的消息处理入口，我们在组件的实现架构中，引入了组件注册相关的支持模块GameMsg\_Processor\_Entry，并实现查找算法复杂度为1的最优算法。

图4-6为组件架构中底层消息框架的设计类图。图中GameMsg\_Base为所有网络消息的基类，其中的nMsg为消息类型，nIndex为角色的服务端索引，以实现消息快速定位。

GameMsg\_Map为整个消息框架中的核心。其中包含以消息ID,也就是GameMsg\_Base中的nMsg为主键的消息处理映射表。网络消息到达时，可以快速的找到该类型消息对应的处理单元，也就是类图中的GameMsg\_Map\_Entry类，每一类消息对应一个GameMsg\_Map\_Entry，GameMsg\_Map\_Entry类中封装了消息的创建函数指针m\_fCreator，以及所有注册过该消息的组件单元与消息处理入口映射表，遍历该表，调用组件注册的GameMsg\_Processor\_Entry信息，快速找到组件，分发网络消息。组件注册GameMsg\_Processor\_Entry信息中包含组件的获取方法m\_pObjGet以及消息的处理入口函数m\_fProcessor。

组件通过继承GameMsg\_Processor类获得通用的消息注册方法，以及通过实现层RoleComponentImpl去调用CGetRoleProcessorObj创建快速查找该组件的IProcessorObjGet模块。

整个底层消息框架通过引入设计模式和查找支持模块，实现速度最优的查找算法。为架构上层组件之间的高效交互，奠定了坚实的基础。

**（3）关键技术实现**

网络Socket与消息系统，我们主要使用的核心技术是分组多线程与环形缓冲区、消息自定位以及组件注册。

1．分组多线程与环形缓冲区技术

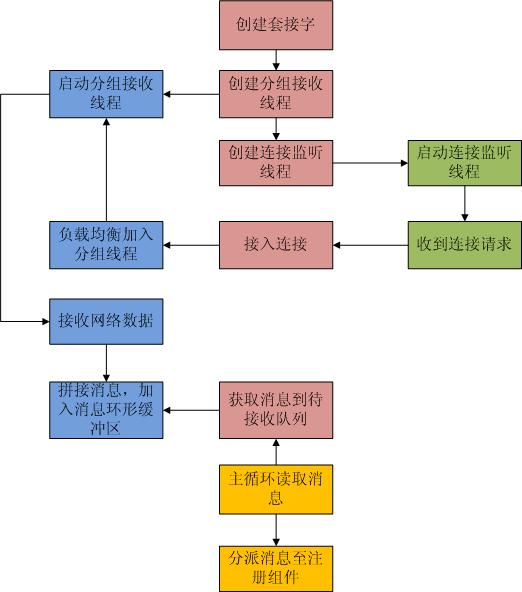


图4-10 分组线程技术流程图

整个Socket的接收网络消息流程如上图，总共有不同颜色的三个线程协同工作。网络线程创建套接字创建，创建分组线程和连接监听线程，当有新连接接入时，连接监听线程汇报网络线程，根据负载均衡算法分配到接收组线程。接下来，接收组线程侦收网络消息，SLOT拼接环形缓冲区消息，消息转入待接收队列，主循环线程读取网络消息，消息系统分派消息至注册组件。发送消息流程与之相反。

环形缓冲区实现原理如下图，缓冲区设头尾两个指针pHead和pTail，pHead端为读操作，pTail端为写操作。环形缓冲区的设计不仅可以有效控制缓冲区大小，避免反复的内存分配开销。而且在多线程环境下，可以方便的进行互斥保护。读写两端的自由控制，同样可以让读与写操作解除彼此耦合，可以独立进行，逻辑清晰，使用方便。其缺点就是缓冲区的大小会依据不同项目特点而不同，需要实践经验去设定合理值[16]。

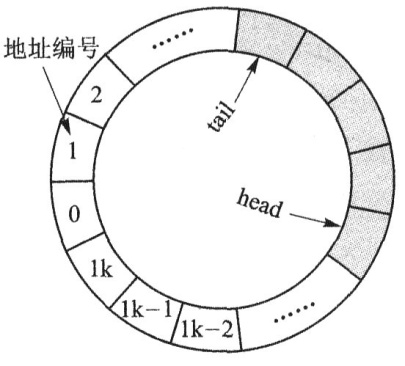


图4-11 环形缓冲区

网络游戏服务端一般会在windows系统环境开发，在linux系统部署运行，因此在实现层，Socket需要同时支持这两种操作系统。两者的主要不同是线程锁与临界区以及系统网络层提供的socket接口。线程锁使用如下宏定义即可以解决。系统网络层接口不同需要在调用处加以区分。

#ifdef WIN32

#define Lock(x) EnterCriticalSection(x)

#define Unlock(x) LeaveCriticalSection(x)

#define WaitThread(x) {WaitForSingleObject(x, INFINITE);CloseHandle(x);}

#define Notify() {SetEvent(m\_hEventNotify);}

#else

#define Lock(x) pthread\_mutex\_lock(x)

#define Unlock(x) pthread\_mutex\_unlock(x)

#define WaitThread(x) pthread\_join(x,NULL)

#define Notify() { Lock (pMutex); pthread\_cond\_signal(m\_pCond); Unlock (pMutex)}

2.消息自定位与组件注册的实现

消息自定位与组件注册技术实现的关键目的只有一个，那就是能够保障网络消息能够以最快的速度达到对应组件。支撑组件间的高交互需求。

消息自定位技术实现的关键：

（1）网关服务器GateServer在完成角色登录后，建立查找速度优先的map或者hash映射，存储角色在游戏服中的索引值nIndex以及角色的当前连接的回话ID，nSessionID。供收到客户端消息时进行填充。

（2）游戏服务器GameServer中，建立通过角色索引值nIndex查找算法复杂度为1的角色表，使得角色管理器，能够通过消息中的nIndex索引值直接找到对应角色，并通过nSessionID值进行校验，确保消息与角色的对应关系无误。

消息自定位技术保障了客户端的网络消息以最快查找算法到达服务端对应角色。接下来的关键就是消息怎样以最快的速度到达关注该消息的组件。也就是组件注册技术提供的保障。

组件注册技术实现的关键：

（1）角色与组件，是一对多的关系，消息系统想要最快的找到组件，需要建立一个查找复杂度为1的对应关系。组件种类繁多，多种多样，在支持组件随机组合的模式中如何建立这样的对应关系呢？最好的办法就是组件自己负责如何让消息系统找到自己。那就是组件需要做两件事，一是组件需要注册自己关注的消息类别，二是注册的同时，需要提供消息系统如何找到自己的办法。

（2）组件注册消息，引入观察者模式，每一类组件通过消息系统提供的注册接口，注册所有自己关注的消息，最后呈现在消息系统数据结构中的将是一个一类消息对应多种组件的map映射。消息系统循环该map映射，即可完成该类消息的全部转发。

（3）如何提供让消息系统快速找到组件自身的办法，其实原理很简单，因为组件想要变为游戏对象（如角色）的一部分，必定存在一个添加的操作，而添加操作背后隐藏的即是一个查找办法。将一个组件添加到游戏对象身上，那该组件在游戏对象身上的所有组件中，一定存在一个先后顺序，这个先后顺序，就是我们快速找到该组件的关键索引。因此只要在某一个组件被添加到游戏对上身上时，记录下该组件索引，在组件向消息系统注册消息的同时，同时注册一个封装了该索引值的查找接口，及设计类图中的IprocessorObject接口。

如下为消息系统消息自定位与组件注册的关键代码：

//GameMsg\_Map

//消息系统提供的组件注册方法

bool AddMsgProcessor(unsigned short nType,

IProcessorObjGet \* pProcssorGet,Processor\_MSG fProcssor){

if( nType != 0 ){

MSGEntryCollection::iterator it = m\_mapEntry.find(nType);

if(it != m\_mapEntry.end()){

GameMsg\_Map\_Entry \* pEntry = it->second;

if( pEntry ){

returnpEntry->AddProcessor(pProcssorGet,fProcssor);

}

}

}

return false;

}

//GameMsg\_Map\_Entry类

//消息系统循环遍历某类消息被注册的map映射，转发消息到所有注册组件

void::ProcessMsg(GameMsg\_Base&rfMsg,CSlotPeer&SlotPee){

MSGProcessorMap::iterator it = m\_mapProcessor.begin();

MSGProcessorMap::iterator endit = m\_mapProcessor.end();

for( ;it != endit;it++ ){

GameMsg\_Processor\_Entry \* pProcessorEntry = it->second;

if( pProcessorEntry ){

pProcessorEntry->OnProcess( rfMsg,SlotPee );

}

}

}

// GameMsg\_Processor\_Entry类

//消息系统通过IprocessorObject接口m\_pObjGet,实现消息的组件定位

void::OnProcess(GameMsg\_Base&rfMsg,CSlotPeer&SlotPeer){

if( m\_pObjGet){

GameMsg\_Processor \*pObj = m\_pObjGet->ProcessorObjectGet(rfMsg);

if(pObj && m\_fProcessor){

(pObj->\*m\_fProcessor)(rfMsg,SlotPeer);

}

}

}

// CgetRoleProcessorObj类

//组件提供的实现IprocessorObject接口查找自身方法的实现类

GameMsg\_Processor \* ProcessorObjectGet(GameMsg\_Base & rfMsg){

GameMsg\_Processor \* pProcessor = NULL;

if(g\_pPlayerManager){

CRoleEntity \* pEntity = g\_pPlayerManager->GetEntity(rfMsg.nIndex);

if(pEntity&&pEntity->GetSessionID()==rfMsg.SessionID){

pProcessor = pEntity->GetComponent(m\_nIndex);

}

}

return pProcessor;

}

4.2.5 组件架构数据查询系统

组件架构DB数据查询框架与消息框架设计非常类似。也采用了查询定位技术与组件注册技术双重保障。

组件注册技术与消息框架完全相同，不再重复介绍。查询定位，与消息不同的是，在查询基类Query中引入查询索引nIndex。

另外DB数据查询框架设计到数据库操作，因此专门创建了查询线程，引入查询管理器QueryManager与数据库接口CDBInterface，负责所有直接面对数据库的操作。如连接、打开、关闭、重连数据库，处理查询的添加查询、处理查询、完成查询一系列流程化操作。执行查询、更新、删除DB数据的sql指令等等。

数据库查询采用异步式操作，流程分为三步，AddQuery，ProcessQuery以及FinishQuery，三步依次阻塞式进行，确保查询栈中的查询先到先出原则。每一个服务器都为自己的每一个数据库连接创建自己的线程，主线程与查询线程独立工作，主线程在心跳函数中处理查询线程的返回。查询线程的独立是为了确保主线程能够顺畅的维持游戏主逻辑，所有在线玩家能够顺畅的体验游戏。

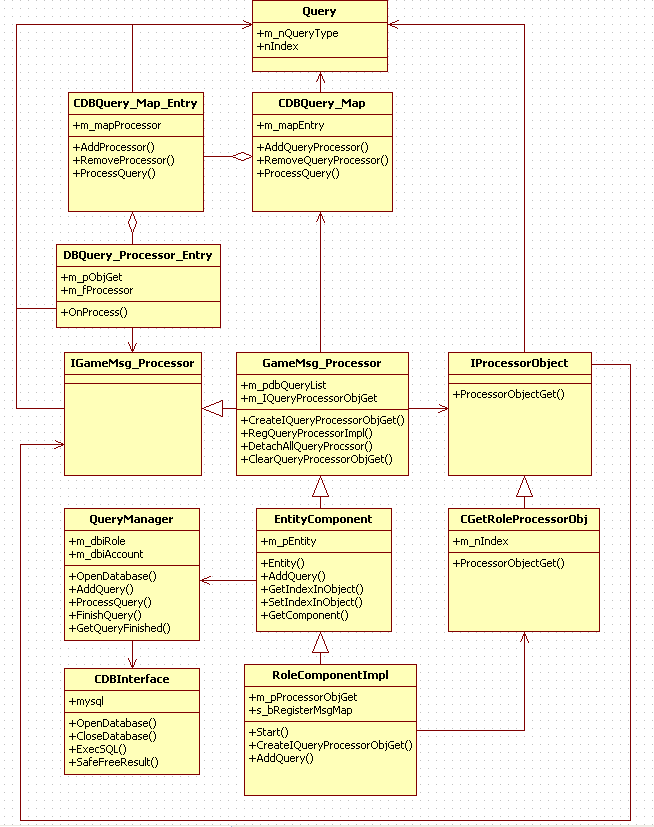
****

图4-12 组建模式DB数据查询架构类图

4.2.6 独立组件

组件模式服务端架构中，可以说所有的系统设计，都是为了能够方便的进行组件的生产、装载、检索以及使用。组件系统中，我们介绍过组件的装载和检索。这一小节，我们重点讨论组件如何方便快速的进行生产和使用。

之所以引入组件模式，原因就在于组件模式在使用对象和对象组合时可以一视同仁，客户程序可以忽略组件复杂的内部实现，应用接口进行装载和使用，所以只要新生产的组件符合组件架构中统一的接口标准。这些组合起来就能协同工作，因此很容易增加新的组件。

至于组件架构中的接口标准，本文讨论的范围仅包含了组件最基本的需求。因此接口标准中暂时只包含这最基本的一部分。而游戏开发是一个庞大的系统工程，其中有一部分需求非常的相似，比如物品系统对物品的管理，比如商城系统对商品的贩卖的管理，比如好友系统对好友增删交互的管理，不管游戏类别有多大的不同，这些系统总体上都没有太大的差别。所以对于这一部分，组件的接口定义相对简单，容易形成统一的标准。当然，也会存在一部分需求差异化较大，不易形成统一的接口标准，面对这种情况，我们有两种解决办法，一是，运用设计模式中的Open-Close开放封闭原则，即“对可变性的封装”，找到一个可变性因素，将它封装起来。实现Open-Close原则，抽象化是关键，因为抽象所以稳定，不变应万变。面对这些不同的需求，我们找到不同，并且找到这些不同处的变化点，然后进行封装设计，抽象出接口。二是，面对一些非常难于抽象与封装变化的需求，我们可以采取定制化组件生产，即面对特定领域，我们进行特定生产，也是组件生产的方式之一[17]。

总之，组件架构模式，系统对所有的组件一视同仁，并不关注组件本身的复杂度和具体实现。只要新生产的组件符合统一的接口标准，组件即能被组装或使用。

下面将以游戏中的物品组件为例，详细的介绍组件的接口定义以及生产和使用。

游戏物品系统，需求包括物品的添加、删除、更新、交换、查找、使用，物品从DB的读取、更新至DB，序列化等。因此，根据需求，我们可以确定物品组件的统一接口如下：

class EntityItemComponent : public EntityComponent{

public:

//添加

int AddItem();

//删除

void DeleteItem();

//更新

void UpdateItem();

//交换

int SwapItem();

//搜索

CItemBase\* SearchItem();

//使用

int UseItem();

//从DB读取

bool CreateFromDB();

//保存至DB

bool PacketToDB();

//序列化

bool Packet();

}

至此，接口统一之后，使用者即无需关注EntityItemComponent组件是如何实现的，只需要关心该组件提供了哪些功能接口，然后根据自身的需求决定是否使用。这样游戏中物品系统的开发，将会从原始的代码开发变成物品组件的整合。如果组件市场上类似的物品组件丰富多彩，整合的过程将还具备充足的选择余地。开发团队将无需自行开发，只需将组件市场中适合自身项目的物品组件引入，并进行游戏中的物品管理。开发速度的提高，将非常显著！

4.2.7 组件间的相互协作

同一组件之间的交互是如此简易高效，那组件之间的协作又会怎样呢？基于组件模式的服务端架构，组件之间的协作，非常重要，只有实现了不同组件之间的协作，简单组件才有可能组装成复杂组件，实现复杂需求。只有实现了不同组件之的协作，组件才能进行不同方向的细分化发展，组件架构才有可能成为一个优秀的架构，通过丰富的组件，开发出游戏的游戏。因此组件间的协作，在组件的架构设计中，规划出了组件相互协作的两种方式，消息传递和组件引用[18]。

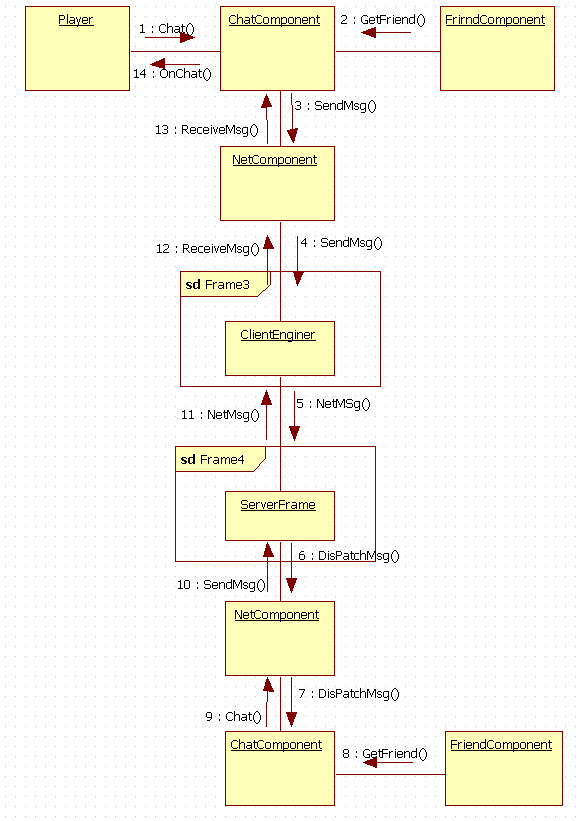


图4-13 组件协作图

消息传递，顾名思义，通过组件之间发送事件消息的方式，达到彼此交互协作。其优点是组件间的依赖少。可以定义不同的事件类型，通过组件基类Component中的OnEvent()方法进行接收处理。

组件引用，在组件Component基类中，我们设计了Start()和GetComponent()两个基类接口，Start()方法会在所有组件加载完毕时统一调用，并实现依赖组件间的相互引用。如此一来，在组件内可以方便的与依赖组件协作，此方法适合需求明确的有相互协作需求的组件之间使用。而另一个方法GetComponent()则提供通过组件查找的方式，直接找到对应组件，并与之交互协作。有这两类方式的保障，组件之间的协作，将会非常的简单。

组件架构中，我们可以方便的进行组件的开发、引进以及组合，可以实现客户端与服务端组件的一一对应，进行高效组件间交互，也可以实现不同组件之间的互通协作，组件模式服务端架构，在Unity3D的组建模式基础上，让整个游戏世界，无论是客户端，还是服务端，都实现了组件化。而组件化了的服务端架构，将为服务端引擎的横空出世打下坚实的架构基础。

# 第五章 组件模式服务端架构的应用实例

《恋舞天使OL》是一款使用Unity3D引擎以及组件模式服务端架构进行开的移动平台休闲类舞蹈网游。其特点就是客户端组件与服务端组件遥相呼应。支持组件之间点对点高性能交互。遵循组件设计的基础性、独立性、通用性以及简化原则，其组件设计与实现过程如下：

第一步：需求分析

舞蹈类游戏与其他类网络游戏相同的基础性需求有：网络通信与管理、消息收发、角色管理、物品管理、商城系统、好友系统、邮件系统等。

舞蹈类游戏特有的需求有：房间系统、模式玩法、服装特效系统。特殊需求，特殊对待，组件按照独立性原则，每一个组件实现某一类需求。运用通用性原则，可积累某一专业领域组件库。

如果企业组件库，或组件市场中，积累了大量通用性或者专业性组件，即可避免相同功能的重复性开发，不仅提高开发团队的效率和速度，也整合了行业资源，避免不必要的资源浪费。

第二步：组件设计

根据需求分析，在组件架构中对各功能系统进行进行组件化设计，其中需要的基础性组件包括：

网络组件EntityNetComponent

角色属性组件EntityAttributeComponent

物品组件EntityItemComponent

商城组件EntityMallComponent

好友组件EntityFriendComponent

聊天组件EntityChatComponent

邮件组件EntityMailComponent。

如物品组件，在各类网络游戏中，其服务端的功能几乎完全相同，包括物品的添加、删除、更新、使用等，因此物品组件适宜设计为基础性的通用组件。下图为《恋舞天使OL》游戏中物品组件的界面呈现。开发过程中除了美术开发外，其余部分完全由物品组件内部实现，客户程序无需关注这些内部实现。因此在服务端的物品系统开发过程中，只需实现一次物品组件，即可达到多游戏项目、跨团队、跨公司的行业性合作。



图5-1 基础性组件一例——物品组件

舞蹈类游戏专业组件：

房间组件RoomComponent

传统模式组件NormalModelComponent

节奏模式组件RhythmModelComponent

控制特效的基因组件EntityGeneComponent。

舞蹈类游戏最大的特点就是不同的玩家分布在不同的房间，每一个房间拥有众多模式的舞蹈玩法，尽管在客户端画面表现和玩家操作上有诸多不同，但抽象过后的服务端逻辑，拥有非常相似的处理过程，比如音乐的选择、播放、关卡的控制、评分的管理、比赛结果的管理、奖励的发放等，都是相同的处理流程，不同的只是这些流程中各步骤的具体实现，因此设计了传统模式组件NormalModelComponent、节奏模式组件RhythmModelComponent等不同组件，供房间组件来处理不同模式的具体实现部分。玩家在房间中选择传统模式，则装载传统模式组件，玩家选择节奏模式，则装载节奏模式组件。下图为房间组件以及房间组件通过控制不同模式组件，实现游戏效果。



图5-2 房间组件



图5-3传统模式组件



图5-4节奏模式组件

各组件具备独立性，因此无论是设计还是实现，无需依赖系统底层以及其他系统，适宜大团队以及不同团队间协作完成任务。

第三步：组件的实现

在基础架构的保障下，《恋舞天使OL》游戏中的组件实现，也非常的简单，接下来将通过商城组件的实现过程来例证组件模式架构下组件实现的方便性。

在组件架构中，角色相关的组件，只需要继承基类EntityComponent，即可拥有组件的所有公共属性。包括处理组件装载、组件查找、网络消息处理、数据查询以及组件间协作等能力。因此商城组件无需专注与自身功能无关的部分，只需完成商城购买、赠送、续费、索要、许愿等商城相关的独立性功能。

class EntityMallComponent : public EntityComponent{

virtual void Start();

void RegComponentNetMsgMap();

void Buy();

void Send();

void RenewItem();

void AskFor();

void BeAskedFor();

void Desire();

}

Start()方法在所有组件被装载后会被立即调用，其作用是完成不同组件间彼此协作关系的预处理，商城系统因为与物品系统密切相关，因此在Start()方法实现中会保存物品组件的引用。供商城组件为角色添加物品。

void EntityMallComponent::Start(){

m\_pRoleItem = GetComponent<CRoleItem>();

}

因为商城组件需要关注与商城功能相关的网络消息，因此商城需要在加载成功时向底层架构注册网络消息。

void CEntityMallComponent::RegComponentNetMsgMap(){

GAMEMSG\_REGISTERCREATOR(GameMsg\_C2S\_MallBuy);

GAMEMSG\_REGISTERCREATOR(GameMsg\_C2S\_MallSend);

RegMsgProcessor(MSG\_C2S\_MALL\_BUY,Buy);

RegMsgProcessor(MSG\_C2S\_MALL\_SEND,Send);

}

其中实现了，购买功能 Buy()，赠送功能Send()，续费功能RenewItem()，索要功能 AskFor()，被索要响应功能BeAskedFor()，许愿功能Desire()等，通过加载商城组件，《恋舞天使OL》项目实现的商城界面效果如下：



图5-5 商城组件

总之，组件模式架构下，组件的设计、生产、实现、装载都非常方便，其中所有与组件独立性功能无关的部分，组件架构已统筹处理，组件只需实现自身需要提供的功能即可。商城组件实现后，角色对象装载商城组件，即可进行商城内商品的浏览、试穿、购买、赠送、索要、许愿等商城功能操作。

组件模式架构，让游戏中的复杂逻辑，进行了自然分离解耦。复杂的网络游戏服务端开发，变成了简单的组件生产与装载过程。已经初步具备了服务端开发引擎化、工具化的基础性条件。这也是本文对网络游戏服务端组件架构研究的意义所在！

# 第六章 结论

游戏研发技术的飞速发展，客户端从手工作坊转变成了成熟的商业化引擎开发。工具化，标准化，成为未来的主流。而相对滞后的网络游戏服务端研发技术，需要从架构上寻求突破。本文借鉴Unity3D引擎中组件模式架构的成功经验，依托现有网络游戏服务端的成熟技术，进行了架构上的探索和创新，形成了类Unity3D组件模式网络游戏服务端架构。

类Unity3D组件模式服务端架构，同样具备工具化、标准化的特点，底层架构的封装，让服务端开发人员，无需再关注复杂的系统底层和繁琐的功能逻辑，继而转变为简单的组件式开发和整合。而这些组件，不仅仅来自身团队的开发，更大的可能是商业化组件的提供。如此，服务端的开发，不仅速度上得到极大的提升，而且质量和稳定性方面，也能得到商业化的足够保障！

类Unity3D组件模式服务端架构比起现有的服务端架构无论是在开发速度、质量、稳定性保障还是面对日趋重要的用户高交互性性能方面，都具备出色的表现。而更重要的是，组件模式架构是服务端走向引擎化的必经之路。

## 6.1 类Unity3D组件模式网络游戏服务端架构的系统特点

与传统的网络游戏服务端架构相比，本架构具有以下几个特点：

（1）建立了一个开放的高性能组件化平台

组件模式服务端架构，为服务端开发提供了一个组件化的新模式，让游戏开发不再单单是一家公司的事情，而是全行业的参与，让原本一家承担的复杂任务，进行行业细分分工。架构对组件生产、装配、使用以及运行的高效支持，也让该架构成为了充分展现各类组件的平台。

（2）组件重用性高，使用方便，开发速度极大提升

组件模式的运营，让架构底层无需关注组件的复杂度和实现，对待所有组件一视同仁，这让组件的重用变得异常方便，因此开发过程变成了对组件的整合，开发速度得到大幅提升。

（3）性能高，支持组件间点对点通信

组件模式服务端架构中，为了支持组件之间高性能的交互，专门设计了针对网络游戏行业特点的网络Socket库和底层消息系统。高效的查找和定位算法，让组件之间的点对点通信变得非常的高效。

（4）可扩充性强，新增组件方便，适合商业化组件运作

这也是组件设计模式的最大优点，组件设计模式以不遵守单一责任原则换取透明性，从而使得客户程序与组合对象（复杂元素）的内部结构解耦。因此新增组件将无需关注底层细节，所以扩充性得到了极大保障。如能做到商业化运作，组件架构将能更大限度发挥它的优势。

（5）系统稳定性高，经过市场检验的组件具备极高的稳定性

由于商业化组件的运作，每一个组件再投入市场后，想要取得成功，往往需要功能和稳定性两方面都兼具，因此再使用这些经过检验的组件进行再生产的时候，质量和稳定性也能最大限度的得到保障。

## 6.2 不足与展望

使用组件模式架构，用继承的方法来增加新的行为很困难，因此在组件的需求分析中，工作需要做得更周全和细致。本文在做组件架构的分析论述中，也难免存在疏忽和不妥之处，有待进一步改善的空间。

另外，类Unity3D组件模式网络游戏服务端架构的实现，只是实现了在服务端引擎化道路上的一个重要环节，还不是全部，要想实现最终的服务端开发引擎，我们还需要在组件接口的标准化上，进行更多的行业研究和讨论。而要想形成组件商业化运作的愿景，还需要许许多多的行业人为之努力！

# 参考文献

[1] BloomGameR.2011中国游戏产业调查报告[2012-03]

[2] 百科网络游戏[EB/OL]: http://baike.baidu.com/view/3543.htm

[3] 李政忠，陈启彰.多人线上游戏伺服器之动态负载平衡机制的设计与制作[D].台湾：义守大学，2006

[4] 胡章优.集群式游戏服务器架构方案设计开发[D].长春：吉林大学，2006

[5] 文睿研究.国内网络游戏开发技术现状和趋势[2011-03]

[6] 杨友斌.浅谈数据结构——树[J].信息与电脑（理论版），2011(2):1-2

[7] [美] Eric Freeman & Elisabeth Freeman with Kathy Sierra & Bert

Bates.Head First设计模式[M].北京：中国电力出版社,2007

[8] Unity圣典用户手册[EB/OL]:http://game.ceeger.com，2012

[9] 依赖注入模式[EB/OL]:

http://cl314413.blog.163.com/blog/static/190507976201191172512571

[10] Unity3D界面插件NGUI核心组件说明[EB/OL]:

http://unity3d.9ria.com/?p=480

[11] C#组件化程序设计浅谈[EB/OL]:

http://developer.51cto.com/art/200909/148991.htm

[12] Lua Documentation[EB/OL]:http://www.lua.org/docs.html

[13] [美]W.Richard Stevens,Stephen A.Rego.UNIX环境高级编程(第二

版)[M].北京：人民邮电出版社,2006

[14] 刘树杰.网络游戏服务器端的设计与实现[D].成都：电子科技大学，2009

[15] [美]Michelle Menard，史晓明、李强译.Unity游戏开发实战.北京：机械工业出版社，2012刘树杰.网络游戏服务器端的设计与实现[D].成都：电子科技大学，2009

[16] 刘树杰.网络游戏服务器端的设计与实现[D].成都：电子科技大学，2009

[17] 苗雨.基于组件的游戏引擎研究与实现[D].北京：北方工业大学，2008

[18] 某游戏公司(笔者曾就职，保密原因，不透露名称).大型多人在线网络游戏服务端架构设计[D]，2010

[19] 孟明.网络游戏服务器端引擎技术研究及应用[D].大连：大连理工大学，2006

[20] 宋斌，李国栋，柳长安.组件化工作流模型的研究与设计[D].北京：华北电力大学，2008

[21] Unity3D之角色控制器组件研究[EB/OL]:

http://www.51xflash.com/article/unity3d/201209/04-17206.html，2012

[22] 叶长青.数字化教学游戏的组件化研究[D].上海：华东师范大学，2010

[23] 刘龙.COM组件化程序设计方法研究[D].哈尔滨：哈尔滨工程大学，2006

# 致谢

首先感谢我的导师。本文的选题、撰写、定稿都是在他的悉心指导下完成的。在论文创作期间，导师李旻老师非常关心论文的进展情况，并且耐心细致地给予我指导。导师广博的学识、严谨的治学态度、踏实的处事作风给我留下了深刻的印象，我为能遇到这样一位优秀的老师感到万分庆幸。本文的顺利完成无不凝聚着导师的心血，在此谨向我的导师献上由衷的感谢！

其次我要感谢软件学院所有的老师两年多以来给予我的帮助，从他们那里获取的知识和学习方法令我受益匪浅，并为我的论文打下了坚实的基础。在此谨向软件学院各位老师表达我真诚的感谢！

再次我要感谢我的同事们，在我论文完成期间，他们给予了我必不可少的帮助。

另外，我要感谢我的妻子和父母，感谢他们对我学业的支持和鼓励！感谢我的宝贝女儿，是她给了我无穷的前进动力！

最后，衷心感谢各位专家、学者为评阅本文所付出的辛勤劳动。谢谢！

**论文独创性声明**

本论文是我个人在导师指导下进行的研究工作及取得的研究成果。论文中除了特别加以标注和致谢的地方外，不包含其他人或其它机构已经发表或撰写过的研究成果。其他同志对本研究的启发和所做的贡献均已在论文中作了明确的声明并表示了谢意。

作者签名： 梅久华 日期： 2012-08-28

**论文使用授权声明**

本人完全了解复旦大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其它复制手段保存论文。保密的论文在解密后遵守此规定。

作者签名： 梅久华 导师签名： 日期： 2012-08-28