# 摘 要

Plain framework（以下简称PF）重量级服务器模型构架文档主要从服务器整个服务器的构架和服务器之间的关系来分析其优缺点，并用简明易懂的语言来描述服务器用到的技术以及实现的相关原理。

这里所阐述的指网络游戏的服务器模型，存在一定的针对性，所以需要各位同仁不要一位生搬硬套。游戏服务器（game server）在网络游戏中起着举足轻重的地位，有的人将之称之为游戏的大脑，更有人戏谑的称之为“皇上”，不管怎样乖僻的称呼，总之是说明其在整个游戏中至关重要。

重量级服务器主要针对大型游戏，其设计之初参考了部分优秀的大型角色扮演在线游戏（MMORPG），但是最终的目的并不限制于某种类型的游戏，所以在代码的结构层会将这些有异的功能加以区分出来。

我们在这篇文章中将会仔细的剖析各个游戏的功能，以及其存在的关键技术应用，从实现的复杂度，与其的优点进行一定说明和分析。

# Abstract

Plain framework (hereinafter referred to as PF) heavyweight server model architecture document, mainly from the relationship between the server and the server for the entire server architecture to analyze their strengths and weaknesses, and use straightforward language used to describe the server technology and related principles to achieve .

Set forth herein refers to the network game server model, there are some specific, so you need not a rote colleagues. Game server (game server) in the network plays a pivotal role in the game, some people will call the game's brain, but it was jokingly called "emperor", however perverse call, in short, explain its is critical throughout the game.

Heavyweight server mainly for the big game, which made reference to the beginning of the design of a large part of the excellent online role-playing game (MMORPG), but the ultimate goal is not limited to a certain type of game, so the code structure layer translates these have to distinguish different functions.

We will carefully analyze the function of each game, and the key technology of its existence in this article, certain descriptions and analyzes the complexity of implementation, and its advantages from.

# 目录

[摘 要 1](#_Toc408050145)

[Abstract 2](#_Toc408050146)

[目录 3](#_Toc408050147)

[前 言 4](#_Toc408050148)

[第一篇 构架 5](#_Toc408050149)

[1.1 构架思想 5](#_Toc408050150)

[1.2 构架模型 5](#_Toc408050151)

[1.3 性能分析 5](#_Toc408050152)

[第二章 关系与作用 6](#_Toc408050153)

[2.1 关系 6](#_Toc408050154)

[2.2 作用 7](#_Toc408050155)

[第三章 技术分析 8](#_Toc408050156)

[3.1 关键技术 8](#_Toc408050157)

[3.2 技术剖析 9](#_Toc408050158)

[总结 9](#_Toc408050159)

[致 谢 10](#_Toc408050160)

[参考文献 11](#_Toc408050161)

# 前 言

在前面已经大致说明了本文的内容，但其只限于PF的商业版开发。每个人的思想没有百分百的完美，所以文章中可能还有些不太完善的地方，希望每一位PF的核心人员能够踊跃的指正错误。

我们需要大概的想象，哪怕那件事自己认为不能实现，但是你可以分享出来。不能实现的东西，在人类的进化史中逐渐都实现了，说不定你一说出来的时候有人就能实现了。不过在紧张的开发中，我们可以将这些奇特的想法暂时记录下来，在空余的时间再来讨论。

本文档适用于PF商业版开发过程中核心的参考资料，请勿任意使用于其他商业用途或传播该资料，否则PF成员有权利追求其所带来的法律责任。

# 第一篇 构架

## 1.1 构架思想

重量级的大方向就是将各个功能尽量的细化，特别是针对服务器压力这块采取了分布式系统的设计概念，多服务器分流式解决玩家涌入的并发问题。

## 1.2 构架模型

我们的游戏服务器分为：消费服务器（billing）、登陆服务器（login）、中心服务器（center）、游戏服务器（server）、共享内存服务器（sharememory）、数据中心（dbcenter）。

## 1.3 性能分析

客户端的登陆请求首先是请求登陆服务器，如果没有得到登陆服务器的认可，将其认证成功的消息发送到中心服务器，再由中心服务器转送到游戏服务器的话，那么任何无效的与游戏服务器之间的连接都将被阻断。其一将游戏服务器隐藏在登录服务器之下，其二更是屏蔽了客户端直接连接游戏服务器占用其连接的资源。

其次消费服务器作为游戏中重要的部分，没有直接与游戏服务器集成的原因是我们可以更多的选择充值的方式，而且账号的数据也由此服务器进行负责到底，对接游戏平台也有了更多的选择，我们可以用先入库的方式记录充值信息，不用担心游戏服务器因为过多的压力而不能正常的处理充值的问题。

再者数据的读写（sharememory）、逻辑的处理（server）、数据库存储（dbcenter）的分离让服务器的数据有了更多保障同时，也缓解了频繁操作数据带来的累积压力。

最后这里的游戏服务器还有一层设计，那就是场景的分离，一组服务器维护起一个完整的游戏世界，分散了服务器的压力，不用担心因为场景过多的负载造成服务器资源不足的问题。这种设计方式，几乎很多大型游戏都使用到了（当然你可以提出你更好的想法）。

任何设计都有缺点，如这里将这些模块进行详细的分开后，势必会增加更多的处理，特别是网络消息方面，各个服务器的进程之间的通讯问题成为此种设计最大的缺憾。

# 

# 第二章 关系与作用

## 2.1 关系

在一个完整的游戏服务器中，其实这些服务器的功能几乎都是需要用的，只不过设计不同而已。那么这些服务器之间是否有关联？就像我们在日常生活里的人际关系一样？

针对上述的两个问题，答案是肯定的，它们之间肯定是存在一些紧密的关联。正如日常生活中，人们之间的关系一样，每个人都有自己的生存技能，我们之间很多时候都是彼此互利的。

它们的关系大致是这样的：客户端首先发送到登录服务器请求登陆，登录服务器收到客户端的请求后再请求消费服务器验证账号，消费服务器将验证的账号结果返回给登录服务器，如果账号验证成功，则首先将账号的信息同步到中心服务器，并从数据库读出玩家的基础数据，将结果返回给客户端，此时客户端能够进入选人场景。

进入场景的流程：客户端申请进入场景后，登陆服务器将请求发送给中心服务器，中心服务器将通过玩家的信息保存并尝试找出玩家所在场景的服务器，并请求将该玩家加入到游戏服务器，游戏服务器返回结果给中心服务器，中心服务器再将此结果返回给登陆服务器。如果账号的验证和游戏服务器的加入场景都成功，则登陆服务器会返回一个成功的消息，并将对应游戏服务器的IP和端口发送给客户端，此时客户端才能正式的进入场景，此时客户端也断开了和登录服务器的连接。

切换场景的流程：客户端首先向自己切换场景的消息发送给当前的游戏服务器，游戏服务器收到后会根据场景表判断是否是自己服务器上的场景，如是则直接切换场景，否则将发送切换服务器的信息到中心服务器，中心服务器此刻将接受到服务器传来的切换场景的消息，并搜索玩家需要切换的场景所在的游戏服务器信息，并将结果返回给当前的游戏服务器。当前的游戏服务器收到结果后将信息返回给客户端，客户端收到此信息后将断开与当前服务器的连接，并尝试进入目标游戏服务器。

充值流程：消费服务器收到平台或者WEB服务器的消息，成功充值后将此信息发送给登陆服务器，登陆服务器并将此消息发送到中心服务器，中心将此消息分派给对应的游戏服务器，前提是玩家在线（这种模式考虑舍弃）。目前充值模式，还有一种玩家查询的模式，玩家可以充值点数，不用充值到对应的游戏服务器，每次在游戏中兑换后与其他游戏的平台充值一样的效果即可。

## 2.2 作用

消费服务器：一切和金钱有关的数据处理。

登陆服务器：起到网关的作用，处理玩家与游戏服务器之间的连接，处理消费服务器与游戏服务器之间的连接。同时登陆服务器还有简单的数据读写的功能，如创建角色、删除角色、角色查询等功能。

中心服务器：一切公用数据与功能的处理，如全局的工会信息维护，游戏中的活动维护等等。具体的有：处理登陆服务器通讯，并实现排队功能、处理与游戏服务器通讯，实现进入游戏场景功能、处理公用数据保存，通过共享内存、处理定时活动、处理跨服逻辑，即从游戏服务器中获取相应的角色数据、处理游戏服务器的一些重要数据，如货币的交换等、维护在线玩家列表。

共享内存：提供共享内存服务，并能定时与数据中心服务器读写数据。

数据中心服务器：与数据库服务器交互，进行数据的读写。

游戏服务器：游戏主要逻辑的处理，一切复杂的游戏逻辑都放在这里处理，主要有模块系统的处理、场景的管理等。

# 第三章 技术分析

## 3.1 关键技术

在这里我们所用到的关键技术主要有：网络通讯的处理、数据存储技术、事件触发机制、脚本技术、线程技术。

网络通讯：网络通讯（net）在网络游戏中是必不可少的部分，在这里分为两个部分处理，就如构架图里的关系一样，服务器与客户端、服务器与服务器都有一根线紧密连接在一起。

这根线的建立以及维护，我们在这里就使用到了通讯的技术，不过有的不是net通讯，如共享内存和服务器之间，使用的通讯技术是内存共享。而网络通讯中，我们主要采取tcp通讯技术，使用系统给出的套接字（socket）接口，前期战士不使用udp技术，后期可能会加入日志服务器概念。

数据存储技术：除了内存的存储（memory）以外，我们需要将一些重要的数据存档到数据库，所以我们不得不用到数据库相关的技术。PF框架中，提供了一个名为ODBC的接口，可以方便的连接到各种数据库。我们针对数据库的操作，也利用了ODBC作为桥接的方式进行。

事件触发：完整的架构中有着许多的事件（event），就好比接收到网络包，我们对其进行逻辑处理一样，它也可以看做一个事件，只是这个事件比较单一。而在游戏世界中，分为全局的事件、模块的事件、玩家的事件等等。

有的时候我们有必要对复杂的事件进行统一的管理，这就需要用到PF框架里的事件模块。

脚本技术：利用脚本（script）处理一些简单的逻辑，让开发和维护成本大大降低。我们在这里优先选用PF框架提供的lua脚本接口技术。

线程技术：线程（thread）可以让代码分块的运行，提高多任务时处理的效率。PF框架中集成了线程模块，我们可以找到相关接口来实现这项任务。

## 3.2 技术剖析

消费服务器：数据库存储（DB）、网络通讯（NET）

登陆服务器：数据库存储（DB）、网络通讯（NET）、线程（THREAD）

中心服务器：网络通讯（NET）、共享内存（MEMORY）、脚本处理（SCRIPT）、事件处理（EVENT）、线程（THREAD）

共享内存服务器：共享内存（MEMORY）、网络通讯（NET）

数据服务器：数据库技术（DB）、网络通讯（NET）、线程（THREAD）

游戏服务器：共享内存（MEMORY）、网络通讯（NET）、脚本处理（SCRIPT）、事件处理（EVENT）、线程（THREAD）

# 总结

暂时没想到

# 致 谢

暂时没想到。

# 参考文献