网游服务器架构变迁

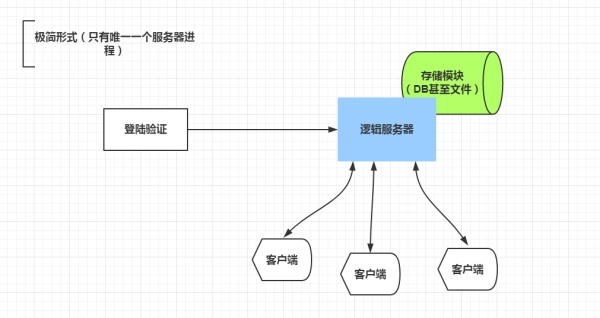
2018年7月27日

15:24

**一、早期网游服务器**

网游刚出现的时候，游戏服务器和小Web服务器没有区别。

从《传奇》的时代开始，游戏服务器内部就出现了游戏逻辑，既能用于同步每个玩家看到的世界，又能让逻辑与客户端分离，避免早期的网络游戏那种毫无防范的逻辑体系（对外挂防御能力为0）。



这种服务器在客户端通过某种形式验证登陆以后，就和服务器通过TCP直接连接了。这种服务器的承载能力不高。

靠多开服务器来弥补单服人数不足的问题 。

**二、早期游戏服务器的改进版本**

扩 厄 彪 式 ， 出 庵 和 连 接 
报 务 
隽 器 
存 冁 
《 司 以 島 文 
进 报 器 

游戏逻辑服务依旧是在一台服务器上，单进程（逻辑处理本身肯定是在一个线程中，可以有子线程负责内网通信）。但是我们自然地想到，储存负载和网络负载可以从逻辑服务器上拆出来。

连接服务器负责把客户端和服务器之间的消息转化为服务器的消息，可以顺便做一下加密的工作。

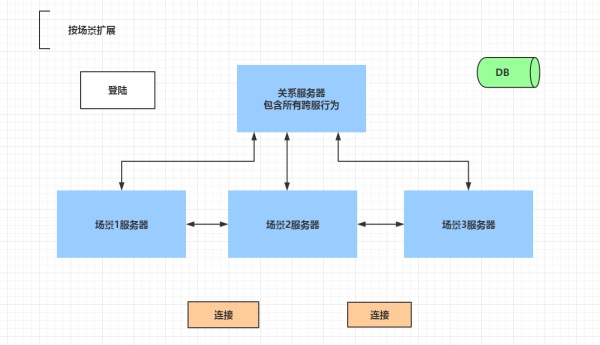
由于连接服务器本身没有时序性，很容易做分布式的（其实大部分游戏还是只有一个连接服），存储服务不要求高实时性，高峰期存盘间隔可以稍长一些，不会对游戏服造成影响。

**三、成熟形态的服务器框架**

逻辑服务器的负载均摊方法一：按照功能划分多个服务器进程

嫣 景 服 身 器 
角 色 在 中 的 战 4 ． 

逻辑服务器的负载均摊方法二：按照场景划分多个服务器进程



这种服务器的难点就在逻辑的设计上，要想做手术一样把本来一体的功能切开，并抽象出若干个API来保持联系（服务器之间是TCP连接）。

在分解时，要找联系相对最薄弱的环节入手比如 场景和场景之间分开、单独抽出聊天服务、组队服务、好友服务。

无论如何分解，最终结果只能是有限个服务。而且分解的越细，开发难度越大。因为跨服务器逻辑时把简单的同步逻辑编程异步Callback逻辑，而且容易出现时序问题等不易测试的问题。

单个场景在服务几乎是无法分解的。分解单个场景难度巨大以至于出现BigWorld引擎来专门解决场景分隔问题。

**附：开房间式的网络游戏**

开房间式的网络游戏是游戏的重要分支，moba和卡牌类游戏都属于这种形式。

这些游戏房间内几乎没有交互，只有大厅内有交互，可以理解为原始形态的游戏服务器的平行拓展。

房间式游戏扩展难度较小，只是需要根据玩家数量动态扩展游戏房间的数量、服务数量。很像网站的架构。

化 的 w 架 
服 迸 隉 1 
服 迸 隉 3 
存 代 了 孩 心 辑 

**小结：游戏服务器框架特点**

1.真正的数据都在内存中，数据库性能不那么重要（注：很多大型游戏采用了共享内存，避免宕机时损失过大）

2.单CPU性能比CUP数量重要得多

3.目前有很多游戏，特别是手游，使用Redis读写代替内存读写，甚至也有用Mongo的。

4.开新服、旧区合服的情况特别适合云平台。

**四、先进服务器框架**

* BigWorld

BigWorld的核心理念时场景分割，利用平面切分的原理，将场景划分为小块，不同的块可以运行在不同的服务器上。而且块的划分是动态的，根据玩家密度程度，数量动态调整动态整块的大小。

具体技术上，使用了Actor模型，要求每个对象都是独立的Entity，对象之间只能通过消息协作，严格限制对象之间的直接交互。

* Skynet

Skynet是新兴的一种通用型服务器框架（完全开源），它游走在传统不易分布服务器和分布式服务器之间。

* 以Go语言为主的其他框架

Go语言的goroutine特性，给游戏开发者带来巨大的想象空间。

**戏服务器特征**

游戏服务器端，是一个会长期运行的程序，并且它还要服务于多个不定时、不定点的网络请求。

所以这类软件的特点是要非常关注 **稳定性**和 **性能**。这类程序如果需要多个协作来提高承载能力，则还要关注部署和扩容的便利性；同时，还需要考虑如何实现某种程度的容灾需求。由于多进程协同工作，也带来了开发的复杂度，这也是需要关注的问题。

**功能约束**，是架构设计的决定性因素。基于游戏领域的功能特征，对服务器端系统来说，有以下几个特殊的需求：

* 对于游戏数据和玩家数据的存储
* 对玩家数据进行数据广播和同步
* 把一部分游戏逻辑在服务器上运算，做好验证，防止外挂。

针对以上的需求特征，在服务器端，我们往往会关注对电脑内存和 CPU 的使用，以求在特定业务代码下，能尽量满足承载量和响应延迟的需求。最基本的做法就是 **空间换时间**，用各种缓存的方式来求得 CPU 和内存空间上的平衡。

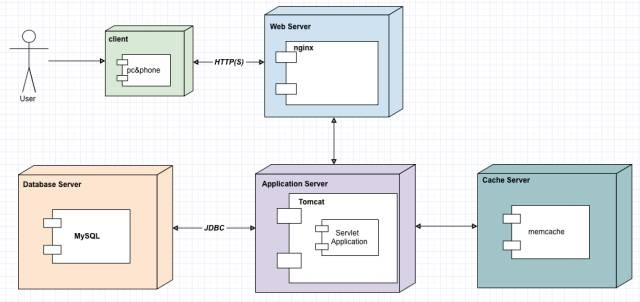
在 CPU 和内存之上，是另外一个约束因素：网卡。网络带宽直接限制了服务器的处理能力，所以游戏服务器架构也必定要考虑这个因素。

**游戏服务器架构要素**

对于游戏服务端架构，最重要的三个部分就是，如何使用 CPU、内存、网卡的设计：

* 内存架构：主要决定服务器如何使用内存，以最大化利用服务器端内存来提高承载量，降低服务延迟。
* 逻辑架构：设计如何使用进程、线程、协程这些对于 CPU 调度的方案。选择同步、异步等不同的编程模型，以提高服务器的稳定性和承载量。可以分区分服，也可以采用世界服的方式，将相同功能模块划分到不同的服务器来处理。
* 通信模式：决定使用何种方式通讯。基于游戏类型不同采用不同的通信模式，比如 **http,tcp,udp** 等。

服务器基于游戏类型不同，所采用的架构也有所不同，我们先讲一下简单的模型，采用 HTTP 通信模式架构的服务器:



这种服务器架构和我们常用的 web 服务器架构差不多，也是采用 Nginx 负载集群支持服务器的水平扩展，memcache 做缓存。唯一的不同点在于通信层需要对协议再加工和加密。一般每个公司都有自己的一套基于 HTTP 的协议层框架，很少采用开源框架。

**长连接游戏服务器**

长连接游戏和弱联网游戏不同的地方在于，长连接中，玩家是有状态的，服务器可以时时和 client 交互。数据的传送，不像弱联网一般每次都需要重新创建一个连接，消息传送的频率以及速度上都快于弱联网游戏。

**服务器的三种类型功能：**

**场景服务器**：它负责完成主要的游戏逻辑，这些逻辑包括角色在游戏场景中的进入与退出、角色的行走与跑动、角色战斗（包括打怪）、任务的认领等。场景服务器设计的好坏是整个游戏世界服务器性能差异的主要体现，它的设计难度不仅仅在于通信模型方面，更主要的是整个服务器的体系架构和同步机制的设计。

**非场景服务器**：它主要负责完成与游戏场景不相关的游戏逻辑，这些逻辑不依靠游戏的地图系统也能正常进行，比如公会聊天或世界聊天，之所以把它从场景服务器中独立出来，是为了节省场景服务器的 CPU 和带宽资源，让场景服务器能够尽可能快地处理那些对游戏流畅性影响较大的游戏逻辑。

**网关服务器**: 在类型一种的架构中，玩家在多个地图跳转或者场景切换的时候采用跳转的模式，以此进行跳转不同的服务器。还有一种方式是把这些服务器的节点都通过网关服务器管理，玩家和网关服务器交互，每个场景或者服务器切换的时候，也有网关服务器统一来交换数据，如此玩家操作会比较流畅。

通过这种类型服务器架构，因为压力分散了，性能会有明显提升，负载也更大了，包括目前一些大型的 MMORPG 游戏就是采用此架构。不过每增加一级服务器，状态机复杂度可能会翻倍，导致研发和找 bug 的成本上升，这个对开发组挑战比较大，没有经验，很容出错。

3、三类型 - 无缝地图

魔兽世界的中无缝地图，想必大家印象深刻，整个世界的移动没有像以往的游戏一样，在切换场景的时候需要 loading 等待，而是直接行走过去，体验流畅。

现在的游戏大地图采用无缝地图多数采用的是 9 宫格的样式来处理，由于地图没有魔兽世界那么大，所以采用单台服务器多进程处理即可，不过类似魔兽世界这种大世界地图，必须考虑 2 个问题：

* 多个地图节点如何无缝拼接，特别是当地图节点比较多的时候，如何保证无缝拼接

* 如何支持动态分布，有些区域人多，有些区域人少，保证服务器资源利用的最大化

为了解决这个问题，需要一组服务器来处理，每台 Node 服务器用来管理一块地图区域，由 NodeMaster（NM）来为他们提供总体管理。更高层次的 World 则提供大陆级别的管理服务。

一个 Node 所负责的区域，地理上没必要连接在一起，可以统一交给一个 Node 去管理，而这些区块在地理上并没有联系在一起的必要性。一个 Node 到底管理哪些区块，可以根据游戏实时运行的负载情况，定时维护的时候进行更改 NodeMaster 上面的配置。

**结语**

游戏服务开发需要应对高并发，低时延的业务场景，必须要懂得正确地分拆各个模块。