细谈网络同步在游戏历史中的发展变化（上）

网易游戏雷火事业群
已认证账号
已关注

发布于 2020-04-15 11:44，编辑于 2020-07-24 13:47

本文作者为网易游戏雷火的游戏客户端开发工程师 @Jerish，他参考了大量资料和论文，从网络同步的基本概念讲起，进一步深入到服务器架构与同步算法的实现细节。本文一共分上下两篇。上篇包括网络同步的基本概念与帧同步的实现细节，下篇则将和大家讨论状态同步，通信协议，常见的同步优化手段等，希望对小伙伴们有所助益。

欢迎大家在评论区提问以及和我们互动哦！

另外，由于涉及到比较久远的历史，部分内容无法考证，如果发现文章中有描述错误的地方还请指出。

目录（上篇）：

7.Lockstep与"帧"同步

8.Lockstep小结

一.网络同步概念理解

网络游戏是电子游戏中一个极为重要的品类，自网络诞生以来，游戏行业就没有停止过对多人游戏玩法的探索。但是随着技术的发展，我们发现游戏服务器的架构发展却与通常的WEB服务器架构渐行渐远。在我看来，导致二者差异化的一个重要原因就是**现代游戏服务器**（更准确的说是游戏逻辑服务器）对网络同步有着很高的即时性要求。

那什么是网络同步呢？虽说是游戏中常用的概念，但其实任何与互联网有关的技术都需要网络同步，我这里会尝试用通俗易懂的方式循序渐进的给大家讲解其原理和概念。在其他领域，我们也许可以称他为“数据同步”，比如说用户A用他的手机点击按钮注册了一个QQ账号，那么他的手机号等个人信息就会被存储到服务器上面，这就是一个数据同步的过程。在这个过程中，数据由A的手机流向了QQ的官方服务器。对于游戏来说，其实原理也是一样，不过由于游戏中玩家关注的是游戏的视觉效果，所以我们不仅要同步数据，还要同步表现。可以简单认为，网络同步 = 数据同步+表现同步，数据同步是后端操作，而表现同步就是让前端对后端同步过来的数据进行进一步的处理从而达到表现上的一致。所以这样看来，不仅是游戏，任何拥有前端的产品都是这样的，比如浏览器、各种手机App，当你点击登录的时候，服务器就会要同步的个人信息发给你，并显示在界面上，你用任何一款手机显示的内容都是一样的。

图1–1 一般应用中的数据同步

不过一般Web服务器只是单纯的从服务器向客户端进行数据同步，不会把其他客户端的数据都发给你。而在游戏里面，你需要让N个客户端的显示看起来一模一样（由于网络延迟，同一时刻不可能完全一样），所以需要把其他玩家的一些数据也发给你，不能说A玩家跳了一下，B玩家看到A却趴下了，那样游戏就没法玩了。对于大部分的互联网产品，针对的都是个人用户的，并不需要两个用户之间进行交互，所以也就不需要不同用户的客户端界面保持一致。也行讲到这里，你会拍案而起，“浏览器不久是这样的么？我们所有人看一个网页都是一样的”。没错，最早期的一批网络游戏就是在BBS上进行的，那时候以文字交流的模式进行多人交互，效果本质上和很多页游是非常相似的。假如XXX知乎大佬的页面是一个棋盘的样子，每个人都可以在其首页上面下一枚围棋，那么这不就是一个可以全民参与的围棋游戏么？页游大概就是这么一个原理。

然而，前面我们还忽略了一个游戏中非常重要的需求（尤其是在MMO、FPS这种类型的网游中）——那就是实时性。你可以容忍微信点进去一个文章要花2秒钟，但是你不可能接受你的子弹要2秒后才打到敌人。实际上，在各种电子竞技里面，0.1秒的延迟就足以让整个游戏的局势发生逆转。像浏览器这种页面显示都吞吞吐吐的应用，如何用他流畅的玩FPS和MOBA呢？（关于云游戏这里先不谈）

到这里，我们再次梳理一下网络同步的概念。**可以认为 网络同步 = 实时的多端数据同步+实时的多端表现同步。从计算机的角度来描述，网络同步是一个网络IO与CPU计算同样密集的游戏功能，他对主机CPU的性能要求较高，且单核的性能比并发更为重要。**当然，游戏同步还有一些其他的特点，比如数据一般不要求存储、允许一定程度的丢失等，这里我们就不再进一步讨论。

了解以上网络同步的特点后，我们就可以进一步展开细节来说，比如传输的数据内容有哪些？数据的传递流向是什么样子的？我们平时口中常谈的“帧同步”，“状态同步”如何理解？为了能以一个更全面更权威的视角来讲解网络同步，笔者阅读了大量的论文和资料，对同步技术的发展和细节有了更为清晰的认识（很多内容与网上参差不齐的博客描述并不相同）。由于同步本身与服务器架构无法剥离，文章还会涉及到一些游戏网络架构的发展历史。

二、网络架构与传统同步

1.网络游戏架构的发展

1973年夏天，高中暑期实习生在美国加利福尼亚州NASA的研究中心首次撰写了游戏《迷宫战争》[1]。通过使用串行电缆连接两台算机，增加了两人游戏功能。由于涉及两台对等的计算机，可以使用相同的格式化协议包相互发送信息，因此可以认为这是第一个**P2P架构**的电子游戏。在那个时代，并没有多人在线游戏，互联网也没有诞生，网络同步一词更是无人知晓。不过当两台计算机上的数据进行传递时，最最最简单的同步模型就已经悄无声息的出现了，A把操作信息通过电缆发给B，B收到数据后处理，在把自己的操作数据通过电缆发送给A。比较有意思的一点是，计算机技术的发展或多或少都与电子游戏有着紧密的联系，甚至很多技术的诞生就是源于对游戏交互方式的探索。

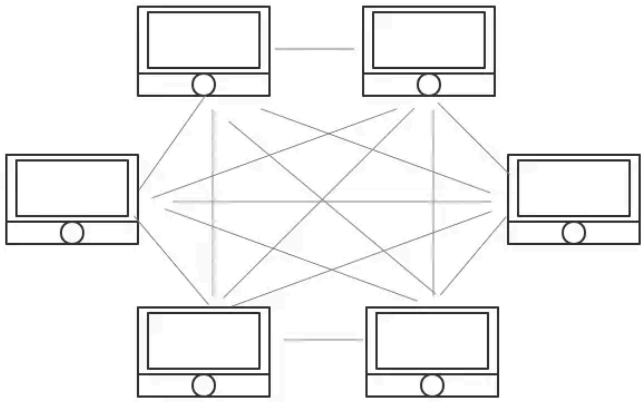

图2–1 P2P架构

1978年，Roy Trubshaw编写了世界上第一个MUD程序《MUD1》，后来又在此基础上诞生了开源的 MudOS（1991），成为众多网游的鼻祖。MUDOS使用单线程无阻塞套接字来服务所有玩家，所有玩家的请求都发到同一个线程去处理，主线程每隔1秒钟更新一次所有对象。这时候所谓的同步，就是把玩家控制台的指令发送到专有的服务器，服务器按顺序处理后再发送给其他所有玩家（几乎没有什么验证逻辑），这是最早的**CS架构**。当时PC图形化还不成熟，MUD早期的系统只有着粗糙的纯文字界面，由于也没有物理引擎等游戏技术，所以对网络延迟、反馈表现要求并不高。

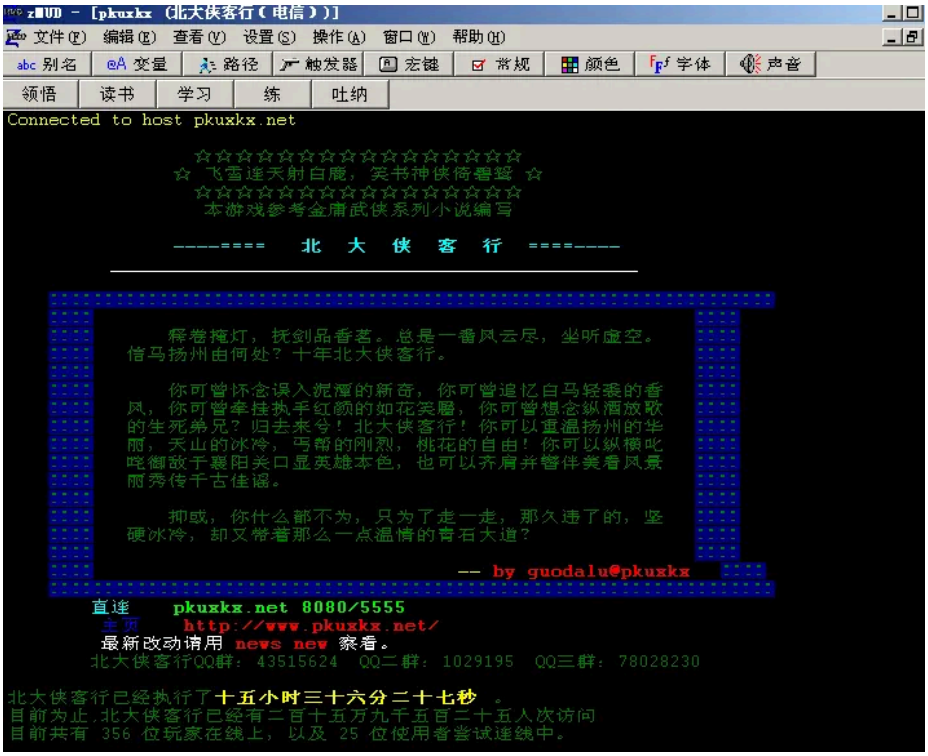

图2-2 国产MUD游戏

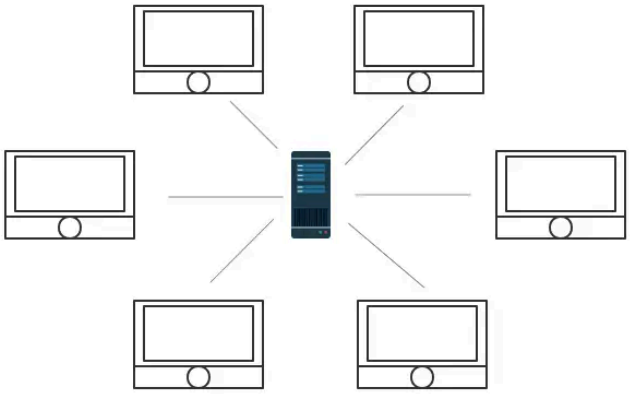

图2-3 早期CS架构

大概在上世纪90年代，在P2P架构的基础上，很自然地诞生了以某个客户端为Host主机（或叫做ListenServer）的CS架构，这样的架构不需要单独都维护一个服务器，任何一个客户端都可以是Sever，能够比较方便的支持局域网内对战，也能节省服务器的运行与开发成本。不过，虽说也是CS架构，如果Host主机不做任何server端的校验逻辑，那么其本质上还是P2P模型，只不过所有的客户端可以把消息统一发送到一个IP，Host再进行转发，这种方式我们称其为Packet Server。

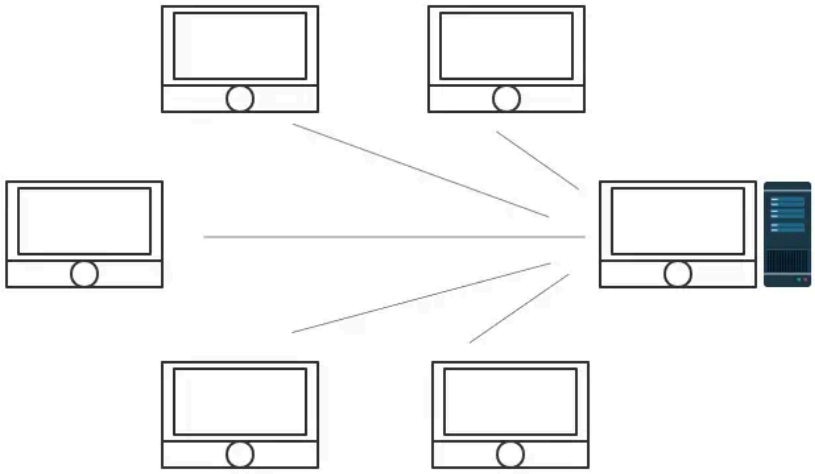

图2-4 带有ListenServer的CS架构

后来一些游戏团队（比如id software）又对CS架构做了进一步调整，先是将大部分的逻辑处理移到服务器上（服务器可能是一个独立的无窗口的后台程序），客户端只负责渲染。随后为了对抗网络延迟提升客户端的流畅性，又把一部分逻辑交回给客户端本地预执行，最终成为很多经典游戏和引擎的架构方式。

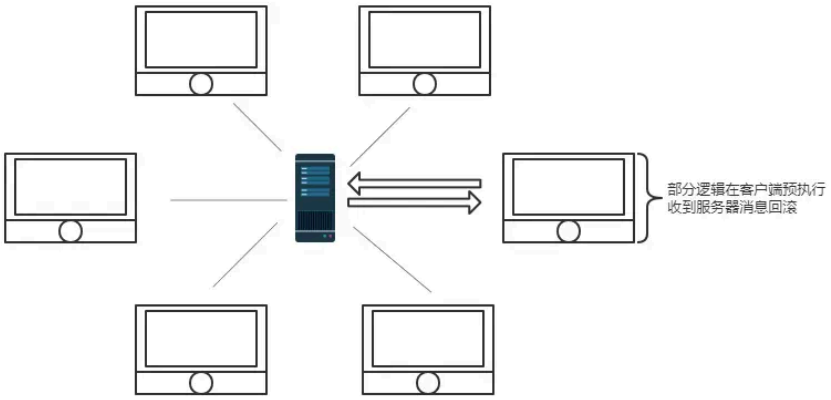

图2-5 客户端与服务器都执行逻辑的CS架构

在2000年后，Eric Cronin的团队在传统多服务器的架构上[2]提出来镜像服务器模型[3]。这种模型提供了多个服务器的拷贝，避免单点崩溃影响到所有玩家的问题。类似CDN，玩家还可以选择就近的服务器进行通信，降低了通信延迟。不过，这种方式增加了服务器的租用和维护成本，在后续的游戏网络架构中并没有被大量使用，倒是WEB服务器广泛采用这种模型并不断将其发扬光大。

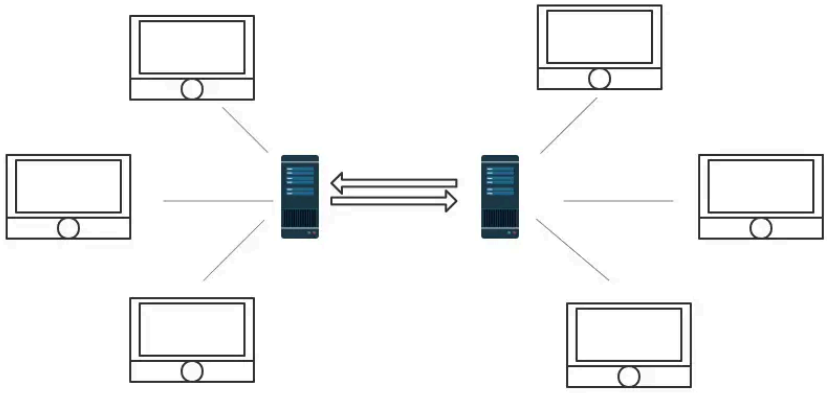

图2-6 镜像服务器架构

再后来，游戏服务器架构不断发展。游戏存储负载和网络连接负载随后从逻辑服上拆分出来，形成独立的服务；玩家数量增多后，又将游戏拆分成多个平行世界，出现了分服和跨服；游戏逻辑进一步复杂后，又开始按照功能去划分成网关服务器、场景服务器、非场景服务器等。我们今天讨论的网络同步几乎都是在逻辑服务器（基本上无法拆分）上进行的，所以后续的这些架构方式与网络同步的关系并不是很大，这里就不再赘述。

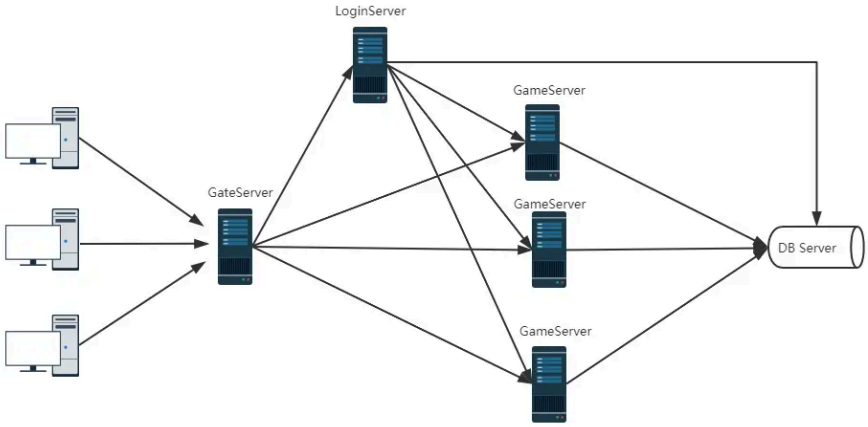

图2-7 分服网络架构

2.传统同步面临的问题

大部分的网络同步技术最早并不是诞生在游戏中，而是出现在各种计算机模拟仿真系统中。在70，80年代就有很多实验室针对Computer Simulation研究各种复杂且有效的同步手段（interactive simulation），我们后面提到的很多游戏同步算法[11]都是由这些早期的研究发展过来的。

网络游戏刚出现的时候，大部分还属于弱交互游戏，可以将其简单理解为一种回合制游戏。在每个回合开始时，所有玩家一同思考并把相关操作指令信息发送给其他玩家，其他玩家收到了别人的消息后就会在本地处理然后结束当前回合，如果没有收到就会进入无限期的等待。由于每个回合有比较长的思考和操作时间，所以网络延迟可以忽略不计，只要保证在回合结束的时候，所有玩家的状态的数据保存一致即可。这种游戏采用的同步方式与计算机网络中的**停等协议**（stop-and-wait-type）非常相似，是一种很自然也很简单的同步模型。不过由于当时网络同步并没有形成体系，所以这种同步方式也没有名字。在局域网盛行以及玩家数量较少的条件下，这种同步方式与架构都是可行的。

- 1.在CS架构下逻辑在客户端执行还是在服务器执行？如果逻辑都在服务器执行，那么客户端的操作都会被发送到服务器运算，服务器计算出结果后通知客户端，客户端拿到结果后再做表现，这样的好处是所有的逻辑由服务器处理和验证，客户端无法作弊，但坏处是会造成客户端的资源被浪费，服务器运算压力过大。如果逻辑在各个客户端执行，那么玩家可以在本地计算后再把本地得到的结果告知服务器，服务器只进行简单的转发，这样的好处是玩家的本地表现很流畅，但坏处是很容易在本地进行作弊。而对于P2P架构，反作弊更是一个严重的问题，我连一个权威服务器都没有，根本无法验证其他客户端消息的真伪，怎么知道其他玩家有没有作弊？
- 2.我们要发送什么数据来进行同步？如果发送每个对象当前的状态，那么如果一个游戏里面有大量的角色，就会大规模的占用网络带宽，造成数据拥塞、丢包等等问题。如果发送玩家指令，那这个指令是要服务器执行还是服务器转发？而且对于大型多人在线游戏又没必要处理所有不相关的玩家信息，同样浪费网络资源。
- 3.面对日益成熟的计算机网络协议，我们选择哪种来进行同步？TCP、UDP还是Http?

这时，游戏开发者们需要面对“发什么数据”，“在哪计算”，“发给谁”等细节问题，他们开始考虑引入更多的其他相关领域的技术（比如计算机模拟仿真）来解决游戏中的同步问题，网络同步概念初见端倪。

三、锁步同步 lockstep（帧同步）

Lockstep就是我们口中常说的“帧同步”。但严格来说，Lockstep并不应该翻译成帧同步，而是——锁步同步算法。（LockStep由军事语境引入，用来表示齐步行军，队伍中的所有人都执行一致的动作步伐）首次引入计算机领域[4]，应该是用于计算机容错系统，即“使用相同的、冗余的硬件组件在同一时间内处理相同的指令，从而保持多个CPU、内存精确的同步”，所以一开始与游戏并没有任何关系。

1.早期的Lockstep

不过，早在1994年，FPS鼻祖Doom就已经采用了类似Lockstep的方式进行网络同步[5]。Doom采用P2P架构，每个客户端本地运行着一个独立的系统，该系统每0.02秒钟对玩家的动作（鼠标操作和键盘操作，包括前后移动、使用道具、开火等）采样一次得到一个 tick command 并发送给其他所有玩家，每个玩家都缓存来自其他所有玩家的 tick commands，当某个玩家收到所有其他玩家的 tick commands 后，他的本地游戏状态会推进到下一帧。在这里， tick command的采集与游戏的推进是相互独立的。

其实当时并没有Lockstep这个说法，doom的整篇论文里面也没有出现过这个词。不过后面概念逐渐清晰后我们会发现，Doom采用的同步方式就是我们常说的原始版本的Lockstep——“确定性锁步同步（Deterministic Lockstep）”。

2.Bucket Synchronization

1999年，Christophe Diot和Laurent Gautier的团队开发了一款基于互联网的页游——MiMaze，基于传统的Time Bucket Synchronization[6]他们在发布的论文里面提出了改进后的Bucket Synchronization同步方法[7]。Bucket Synchronization把时间按固定时长划分为多个Bucket，所有的指令都在Bucket里面执行。考虑到网络延迟的情况，每个玩家在本地的命令不会立刻执行而是会推迟一个时延（该时延的长度约等于网络延迟），用来等待其他玩家的Bucket的到来。如果超过延迟没有到达，既可以选择放弃处理，也可以保存起来用于外插值（Extrapolation）或者使用前面的指令重新播放。在这种方式下，每个玩家不需要按照Lockstep的方式严格等待其他玩家的命令在处理，可以根据网络情况顺延到后面的bucket再执行。Bucket Synchronization可以认为是我们常说的“乐观帧锁定”算法。

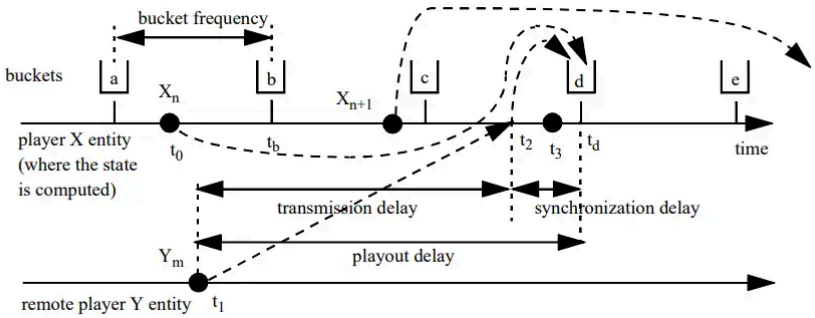

图3–1 Bucket Synchronization同步示意图

3.锁步同步协议 Lockstep protocol

前面提到的Deterministic Lockstep虽然简单，但是问题却很多，包括浮点数跨平台的同步问题、玩家数量增长带来的带宽问题以及显而易见的作弊问题（在P2P架构下几乎没有任何反作弊能力）。说到作弊这里不妨先简单谈一下游戏外挂，外挂这个东西原本是指为增加程序的额外功能而追加的内容，但随着网络游戏的诞生就开始与游戏绑定起来。针对不同的游戏类型有着各式各样的外挂工具，包括游戏加速、透视、自动瞄准、数据修改等。从技术上来讲，有些外挂是无法做到完全避免的。在CS架构下，因为大部分核心逻辑都是在服务器上面计算，很多作弊手段无法生效。但是P2P架构下作弊却变得异常简单，甚至不需要很复杂的修改工具。比如客户端A使用了外挂工具，每次都将自己的操作信息推迟发送，等到看到了别人的决策后再决定执行什么，这种外挂称为lookahead cheats。（或者假装网络信号不好丢弃第K步的操作，第K+1步再发送）

因此，在2001年，Nathaniel Baughman和Brian Neil Levine在IEEE上发表了论文，提出**锁步同步协议 Lockstep protocol** [8]来对抗lookahead cheat类型的外挂。这可能是第一次“Lockstep”一词被正式的用于游戏网络同步的描述中。不过要注意的是，**这里的Lockstep protocol并不是我们前面提到的**Deterministic Lockstep **，**相比之前的在第K步（第K个Tick Command间隔）就直接发送第K+1步的明文操作信息，**Lockstep protocol每一步都分两次发送信息。**大概的流程如下：

- 先针对要发送的明文信息进行加密，生成“预提交单向哈希（secure one–way commitment hash）”并发送给其他客户端。
- 待本地客户端接收到所有其他客户端的第K步预提交哈希值之后，再发送自己第K步的明文信息
- 等到收到所有其他客户端的第K步明文信息后，本地客户端会为所有明文信息逐个生成明文哈希并和预提交的哈希值对比，如果发现XXX客户端的明文哈希值和预提交哈希值不相等，则可以判定该客户端是外挂。反之，游戏正常向前推进。

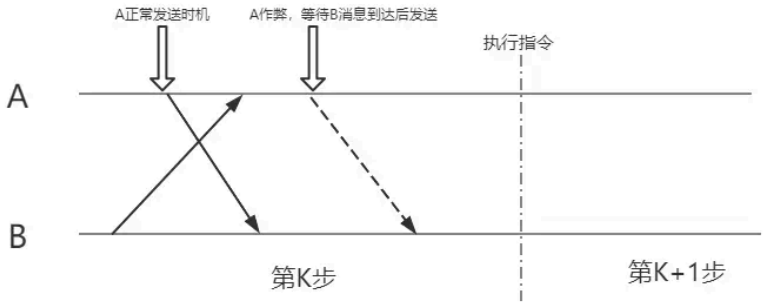

图3–2 Lookahead cheat作弊示意图

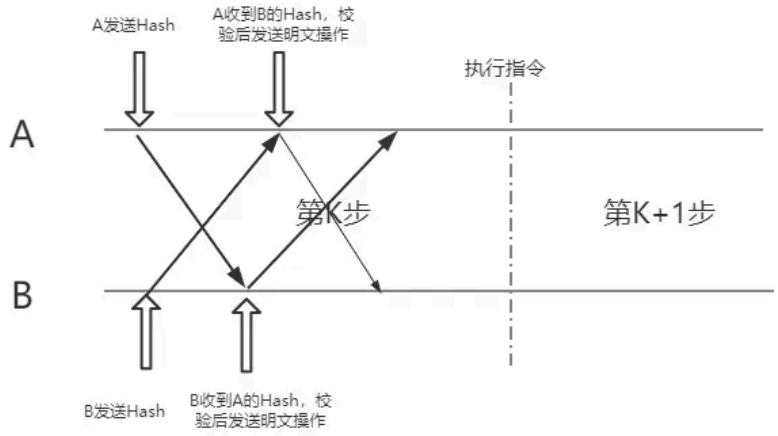

图3-3 Lockstep protocol防作弊流程

这种协议的虽然可以对抗外挂，但是很明显带来了带宽以及性能的浪费，而且网络条件好的客户端会时刻受到网络差的客户端的影响。所以他们又在此基础上提出异步的Lockstep（asynchronous Synchronization lockstep）。大体的思路是利用玩家角色的SOI（Spheres of Influence，和AOI概念差不多），两个玩家如果相距很远互不影响，就采用本地时钟向前推进（非Lockstep方式同步），如果互相靠近并可能影响到对方就变回到严格的LockStep同步，这里并不保证他们的帧序列是完全一致的。

l	Local host
R	The set of all remote hosts
r	A remote host in R
t	The current frame at the local host
S_t^h	State of host h at frame t
$H(S_t^h)$	Hash of state S_t^h for host h for frame t
p_t^h	Potential influence of host h at frame t

Fig. 6. Table of variables.

1. Compute S_t^l
2. Send $H(S_t^l)$
3. Process accepted $H(S_y^r)$ messages that have arrived
4. *foreach* $r \in R$
   Take next S_y^r if any have arrived where $y \leq t$
   Let frame of latest state taken be x
   compute p_t^l, and p_t^r dilated from x
   if $(p_t^l \cap p_t^r = \emptyset)$
   then record l is *not waiting* for r
   else if $H(S_t^r)$ accepted
   then l is *not waiting* for r
   else l is *waiting* for r
5. *if* not *waiting* for any r
   then send S_t^l
   resolve any interactions
   finalize and render turn t
   advance to turn $(t+1)$

图3-3 Asynchronous Synchronization lockstep流程伪代码

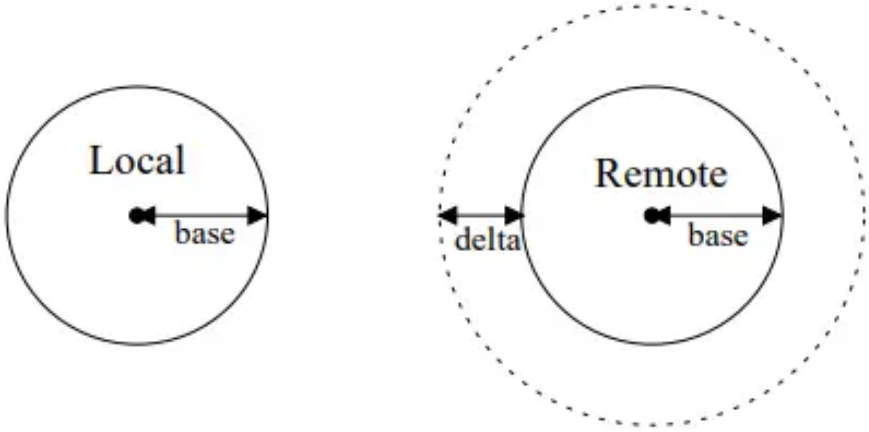

图3–4 SOI示意图

4.RTS中的Lockstep

同一年，2001的GDC大会上，“帝国时代”的开发者Mark Terrano和Paul Bettner针对RTS游戏提出了优化版的锁步协议[9]。早在1996年他们就开始着手于帝国时代1的开发，不过很快就发现在RTS游戏中的网络同步要比其他类型游戏（比如FPS）复杂的多，一是游戏中可能发生位置变化的角色非常多，必须要合理的减少网络同步带宽，二是玩家对同步频率极为敏感，每一秒的疏忽都可能影响局势。所以，他们在传统 Lockstep（当时仍然没有Deterministic Lockstep这个概念）的基础上做了优化，首先保持每一步只同步玩家的操作数据，然后对当前的所有命令延迟两帧执行的方法来对抗延迟。具体来说，就是第K步开始检测到本地命令后会推迟到第K+2步进行发送和执行，K+1步收集到的其他客户端命令会推迟到K+3步去执行，每K步执行前会去判断本地是否有前两步的命令，如果有就继续推进。（关于具体的推进策略，论文里面写的不是很清楚，这里加入了作者自己的判断）

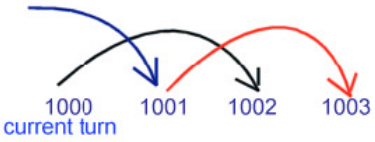

图3–5 延迟执行的Lockstep

此外，为了避免高性能机器受低性能机器的影响而变“卡”，“帝国时代”里面每一步（称为一个turn）的长度是可以调整的，并且完全与渲染分开处理。每个客户端会根据自身的机器性能与网络延迟情况来动态调整步长时间，如果性能优良但是延迟高就会拉长每个turn的时间（多出的时间用于正常进行多个帧的渲染以及Gameplay的处理，虽然可能有误差），如果性能差但是网络正常就会把大部分的时间用于每个turn的渲染，在这种条件下每个客户端相同的turn执行的本地时间虽然不同，但是执行的内容是完全一致的。

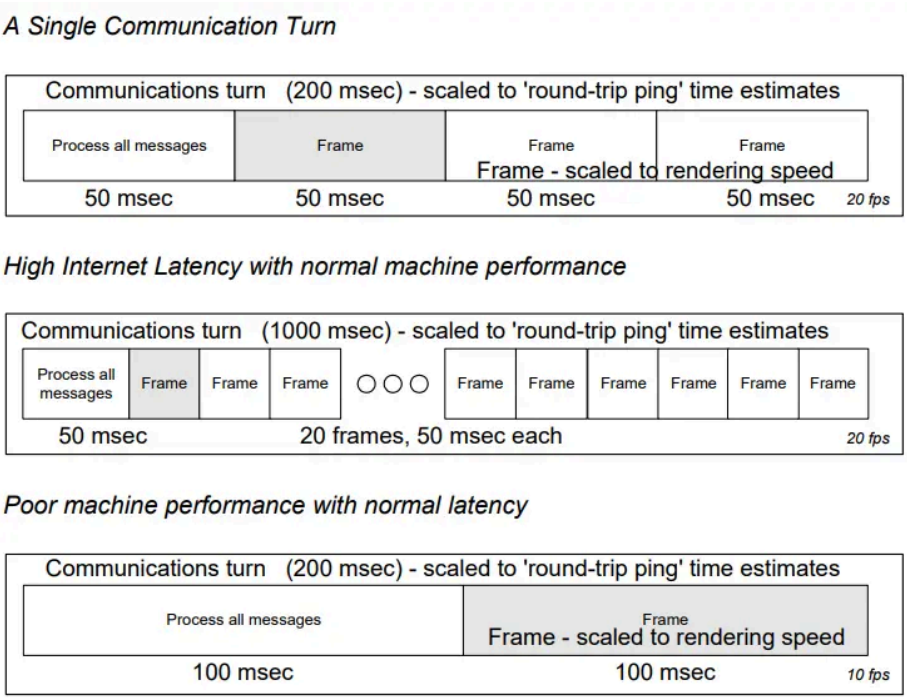

图3–6 根据客户端环境调整的Turn

5.Pipelined Lockstep protocol

流水线操作是一种高效的数据传输模型，在计算机技术里面随处可见（比如计算机存储系统中主存与Cache的交互）。2003年，Ho Lee、Eric Kozlowski等人对Bucket synchronization、Lockstep protocol等协议进一步分析并针对存在的缺点进行优化，提出了Pipelined Lockstep protocol[10]。他们发现只有当前玩家的指令行为不与其他人产生冲突，就可以连续的发送的自己的指令而不需要等待其他人的消息。举个例子，假如一个游戏只有7个格子，玩家A和B分别站在左右两边，每次的指令只能向前移动一格。那么A和B至少可以连续发送三个指令信息而不需要等待对面玩家的数据到来。

Pipelined Lockstep protocol基于Lockstep protocol，为了防止cheatahead外挂同样需要提前发送hash，这种操作同步、不等待超时玩家的确定性锁步的特性逐渐成为“Lockstep”的标准，被广泛应用于网络同步中。

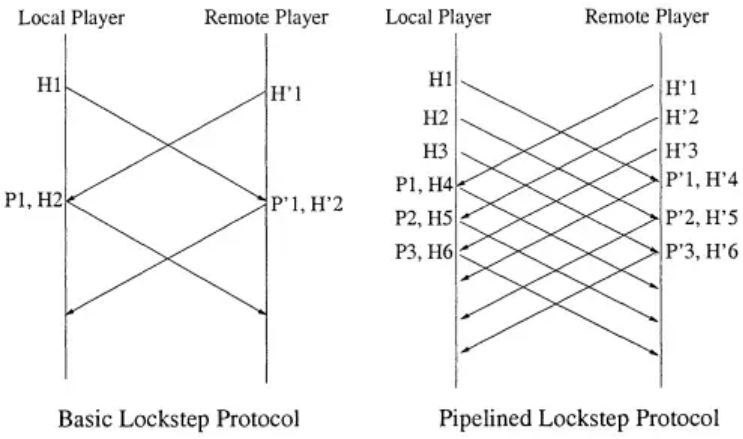

图3–7 流水线Lockstep示意图

6.Time Warp

Time Warp原本是指科幻小说中的时间扭曲，其实早在1982年就被D Jefferson等人引入计算机仿真领域[11]，后续又被Jeff S. Steinrnan进行优化和调整[6][12]。Time Warp算法基本思路是多个

前面提到的Pipelined Lockstep protocol可以流畅的处理玩家互相不影响的情况，但是却没有很好的解决状态冲突与突发的高延迟问题。参考TimeWarp这种思路，我们可以将本地执行过的所有操作指令进行保存同时把对应时刻的游戏世界状态存储成一个快照（Snapshot），本地按照Pipelined Lockstep protocol的规则进行推进，如果后期收到了产生冲突的指令，我们可以根据快照回滚到冲突指令的上一个状态，然后把冲突后续执行过的事件全部取消并重新将执行正确的指令。 这样如果所有玩家之间没有指令冲突，他们就可以持续且互不影响的向前推进，如果发生冲突则可以按照回退到发生冲突前的状态并重新模拟，保持各个端的状态一致。

7.Lockstep与"帧"同步

前面提到了那么多lockstep的算法，但好像没有一个算法使用到“帧”这个概念。其实“帧同步”属于一个翻译上的失误，宽泛一点来讲“帧同步”是指包含各种变形算法的Lockstep，严格来讲就是指最基本的Deterministic Lockstep。我猜测国内在引入这个概念的时候已经是2000年以后（具体时间没有考证），lockstep算法已经有很多变形，时间帧的概念也早已诞生，所以相关译者可能就把“lockstep”翻译成了“帧同步”。当然也可能是引入的时候翻译成了“按帧锁定同步”，后来被大家以简化的方式（帧同步）传递开来。但不管怎么说，“帧”在实际应用中更多的是指画面渲染的频率，lockstep里面的“step”概念要更宽泛一些才是。

了解游戏的朋友，都知道游戏是有帧率（FPS）的，每一帧都会运行相当复杂的运算（包括逻辑和渲染），如果运算规模达到一定程度就会拉长这一帧的时间，帧率也就会随之下降。所有影视作品的画面都是由一张张图构成的，每一个画面就是一帧，一秒能放多少个画面，就是有多少帧。在游戏里面，渲染器会不停的向渲染目标输出画面，并在帧与帧之间游戏处理各种逻辑，逻辑会改变游戏世界对象的行为，这些行为又会影响游戏画面的变化，这就是游戏的核心框架。早期的lockstep里面渲染和逻辑都是放在一个帧里面去处理的，这样一旦命令受到网络延迟的影响，玩家本地就会卡在一个画面直到消息的到来。为了解决这个问题，有一些游戏会将逻辑和渲染分开处理（分为逻辑帧和渲染帧），逻辑帧每隔固定的时间去处理所有逻辑事件。在不是严格锁帧的情况下，你本地即使没有收到网络数据也可以在继续执行其他的逻辑并维持高频率的渲染（正在移动的对象不会由于短暂的延迟而静止不动）。这里面的逻辑帧就是lockstep里面的“Step”，也可以叫做“turn”，“bucket”或者“步”。

8.Lockstep小结

了解了Lockstep的发展历程，最后我们再总结梳理一下。网络同步，其目标就是时刻保证多台机器的游戏表现完全一致。由于网络延迟的存在，时刻保持相同是不可能的，所我们要尽可能在一个回合（turn）内保持一致，如果有人由于某些原因（如网络延迟）没有同步推进，那么我们就需要等他——这就是Lockstep（或者说一种停等协议）。LockStep其实是很朴素的一种思想，很早就被用于计算机仿真模拟、计算机数据同步等领域。后来网络游戏发展起来后，也很自然的被开发者们拿到了系统设计中。

早期lockstep被广泛用于局域网游戏内（延迟基本可以保持在50ms以内），所以这种策略是很有效的。lockstep每个回合的触发，并不是由收到网络包驱动，也不是由渲染帧数驱动（当然渲染帧率稳定的话也可以以帧为单位，每N帧一个回合），而是采用客户端内在的时钟稳定按一定间隔（比如100ms） 的心跳前进。游戏的一开始，玩家在本地进行操作，操作指令会被缓存起来。在回合结束前（网络延迟在50ms以内），我们会收到所有其他客户端的指令数据，然后和前面缓存的指令一同执行并推进到下一个回合。如果玩家在一个回合开始到50ms（网络延迟的一半）都没有任何操作，我们可以认为玩家发出了一个Idle指令，打包发给其他客户端[13][14]。

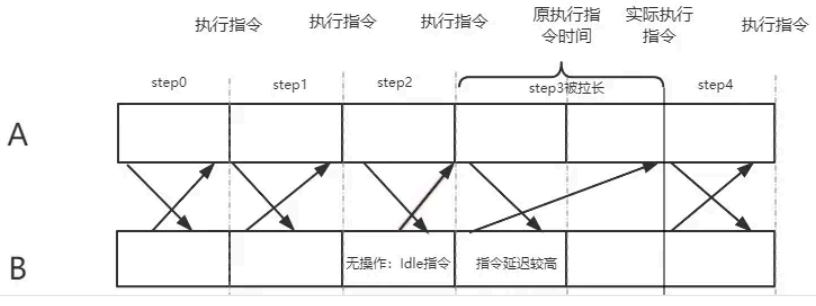

图3-8 传统的Lockstep

换个角度来看，假如一场游戏持续了20分钟，不考虑延迟的情况下整场游戏就是12000个回合（所有客户端都是如此）。现在我们反过去给每个回合添加指令，确保每个回合都收集到所有玩家的指令，那么就可以严格保证所有客户端每个回合的表现都是一样的。假如我们再把这些指令都存储起来，那么就推演出整场比赛，这也是为什么lockstep为什么做回放系统很容易。

至于lockstep为什么要发送指令而不是状态，其实是与当时的网络带宽有关。很多游戏都有明确的人数限制（一般不超过10个人），玩家在每个回合能做的操作也有限甚至是不操作，这样的条件下所有玩家的指令一共也占用不了多少带宽。如果换成同步每个角色的状态数据，那么数据量可能会膨胀10倍不止。从原则上说，锁步数据既可以是游戏角色的状态信息也可以是玩家的操作指令，只不过由于各种原因没有采取状态数据罢了。在下一章，我还会对状态同步做进一步的讲解，他与lockstep的发展是相辅相成的，也不是网上常说的那种对立关系。

```
//lockstep操作指令的结构体
struct Input
{
bool up;
bool down;
bool left;
bool right;
bool space;
bool Z;
};
```

仔细分析一下lockstep，其实就能发现不少缺点。首当其冲的就是网络条件较差的客户端很容易影响其他玩家的游戏体验。为了尽可能保证所有玩家的游戏体验，开发者们不断的对游戏进行更深一步的分析，先后提出了各种优化手段和算法，包括使用乐观帧锁定，把渲染与同步逻辑拆开，客户端预执行，将指令流水线化，操作回滚等。关于回滚还有很多细节[16][18]，我会在下个章节里面进行更详细的阐述。

其次，lockstep的另一个问题就是很难保证命令一致的情况下，所有客户端的计算结果完全一致，只要任何一个回出现了一点点的误差就可能像蝴蝶效应一样导致两个客户端后面的结果截然不同。这个问题听起来容易，实际上执行起来却有很多容易被忽略的细节，比如RPC的时序，浮点数计算的偏差，容器排序的不确定性，随机数值计算不统一等等。浮点数的计算在不同硬件（跨平台更是如此）上很难保持一致的，可以考虑转换为定点数，随机计算保持各个端的随机种子一定也可以解决，但是具体实现起来可能还有很多问题，需要踩坑之后才能真正解决。

（下篇：未完待续）

参考文献：

[1]WIKI "History of video games". Available:en.wikipedia.org/wiki/H...[Accessed: 2020-03-24]

[2]T.A. Funkhouser.”RING: A Client-Server System for Multi-User Virtual Environments“, In Proc. 1995 Available:dl.acm.org/doi/pdf/10.1...[Accessed: 2020-03-24]

[3]Eric Cronin, Burton Filstrup Anthony R. Kurc, Sugih Jamin,"An Efficient Synchronization Mechanism for Mirrored Game Architectures", 2004. Available: citeseerx.ist.psu.edu/v... [Accessed:2020-03-24]

[4]WIKI "Lockstep (computing)". Available: en.wikipedia.org/wiki/L...[Accessed: 2020-03-24]

[5]JMP van Waveren, "The DOOM III Network Architecture",2006. Available:fabiensanglard.net/doom...[Accessed: 2020-03-24]

[6]Jeff S. Steinrnan, "BREATHING TIME WARP" May 1993. Available:dl.acm.org/doi/pdf/10.1...[Accessed:2020-03-24]

[7]Christophe DIOT, Laurent GAUTIER, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet", IEEE, 1999. Available: cs.ubc.ca/~krasic/cpsc5...[Accessed: 2020-03-24]

[8]Nathaniel E. Baughman, Brian Neil Levine, "Cheat-Proof Playout for Centralized and Distributed Online Games", IEEE INFOCOM, 2001. Available: forensics.umass.edu/pub... [Accessed: 2020-03-24]

[9]Mark Terrano,Paul Bettner "Network Programming in Age of Empires and Beyond" GDC 2001. Available:zoo.cs.yale.edu/classes...[Accessed: 2020-03-24]

[10]Ho Lee, Eric Kozlowski, Scott Lenker, Sugih Jamin, "Multiplayer Game Cheating Prevention With Pipelined Lockstep Protocol", 2002. Available: ekozlowski.com/assets/m... [Accessed: 2020-03-24]

[11]Dacid Jefferson,Henry Sowizral "Fast concurrent simulation using the time wrap mechanism " 1982. Available: rand.org/content/dam/ra...[Accessed: 2020-03-24]

[12]J. S. Steinman, J. W. Wallace, D. Davani, and D. Elizandro. "Scalable distributed military simulations using the SPEEDES object-oriented simulation framework". In Proc. of Object-Oriented Simulation Conference (OOS'98), pages 3—23, 1998.

其他参考资料：

[13]云风 "lockstep网络游戏同步方案" Available:blog.codingnow.com/2018...

[14]Skywind "再谈网游同步技术" Available: skywind.me/blog/archive...

[15]DonaldW "网络游戏同步技术概述" Available:zhuanlan.zhihu.com/p/56...

[16]Gordon "帧同步联机战斗(预测，快照，回滚)" Available: zhuanlan.zhihu.com/p/38...

[17]zhepama "帧同步的相关问题"Available:igiven.com/dotnet/lock-...

[18]kisence"关于帧同步和网游游戏开发的一些心得"Available:kisence.com/2017/11/12/...

[18]Glenn Fiedler "Deterministic Lockstep" Available:gafferongames.com/post/...

[19]Qing Wei Lim "How do multiplayer games sync their state" Available:medium.com/@qingweilim/...

[20]treeform "Don't use Lockstep in RTS games"Available: medium.com/@treeform/do...

[21]Maksym Kurylovych "Lockstep protocol" Available: ds.cs.ut.ee/courses/cou...

[22] Glenn Fiedler, "What Every Programmer Needs To Know About Game Networking A short history of game networking techniques", 2010. Available: gafferongames.com/post/...

发布于 2020-04-15 11:44 ，编辑于 2020-07-24 13:47

数据同步 客户端 服务器架构

理性发言，友善互动

27 条评论 默认 最新

Martin
“对当前的所有命令延迟两帧执行的方法来对抗延迟”这个就是关键帧锁定?
刚刚 回复 喜欢

韦易笑 我关注的人
并非翻译错误，我 2000 年初的几篇文章里最先引入“帧锁定同步”这个说法，并不是翻译 lockstep 的资料，我并没有看过任何国外关于 lockstep 的文章，也不认识这个名词，我只是根据早先局域网游戏的表现进行逆向分析，并结合自己实验总结出的这个名词，后简称“帧同步”，只能说和 lockstep 想的差不多吧。
2020-07-28 回复 14

毛毛虫
我也一直不明白帧同步这个名字和其做的事有啥关系。还有人把它和状态同步对立起来，更迷糊。
2020-08-23 回复 2

Jerish
感谢韦老师，那这块没查证的历史原因算是填上了
2020-07-29 回复 2

展开其他 1 条回复 》

fei chen
感谢作者分享，请问， 1.农药这种 帧同步项目， 自身网络延迟并没有影响其他玩家，是乐观锁么？ 2.如果一些不可逆的操作， 不能回滚（死亡 、掉血）一般成熟项目是怎么处理的?
2022-12-12 回复 1

咸蛋
逻辑也是可以拆分的 本质上是房间类服务端 不过这也要看游戏具体逻辑
2020-06-16 回复 喜欢

新生代农民工

2020-05-03 回复 喜欢

张ziqi
对于只玩过游戏没有深入了解内部的我来说很多地方就是天书，但是坚持看完后至少对服务器之类有了个大概的框架
2020-04-20 回复 喜欢

毕纳魏
这种描述技术发展史的方式，复合循序渐进的学习步骤，赞。
2020-04-19 回复 喜欢

睡不着就来找解药
很棒👍👍受益匪浅
2020-04-17 回复 喜欢

Gordon
理论很扎实 👍
2020-04-15 回复 喜欢

网易游戏雷火事业群 作者

罗传月武
前人栽树，后人乘凉，期待下篇~
2020-04-15 回复 喜欢

catcode
很棒，期待下篇~
2020-04-15 回复 喜欢

网易游戏雷火事业群 作者
正在写！
2020-04-15 回复 2

点击查看全部评论 ›

文章被以下专栏收录

SD-WAN技术

推荐阅读

《网络同步在游戏历史中的发展变化》PDF免费放出

Jeris... 发表于游戏开发那...

独立游戏在中国 03：关于手游审批：大限已过，无号上线的...

2017.1月14日更新：因一些朋友私信我版号办理具体流程，写了篇带例子的全攻略放在这儿，供大家参考。 戳我：手游版号办理完全攻略 以下是原文：1.2017年元旦，独立手游之“大限”今天是1月...

糖小渣 发表于独立游戏在...

当获得版号是一款游戏成功的关键时，便是悲哀的开始

佳伦实验室 发表于游戏茶馆归...

游戏用户流失原因及分析建

在游戏公司中，玩家流失一作人、策划、业务运营最为问题之一。本文将围绕用户介绍如何针对流失搭建数据型。我们从玩家流失原因说家为什么会流失？围绕玩家

黑啤

检测到网页存在资源
资源存夸克网盘在线查看或播放
查看资源 收起