游戏同步的基本概念

多人网络游戏是电子游戏中一个极为重要的游戏品类，随着技术的发展，游戏服务器的架构发展与常规的 Web 服务器机构有了很大区别。在应用场景上，多人网络游戏，更加重视数据的即时性，我们需要玩家的输入，快速的表现在游戏中，例如玩家进行的移动输入，游戏画面就应当快速进行反馈表现。而常规的 Web 应用就不一样，及时性要差很多。

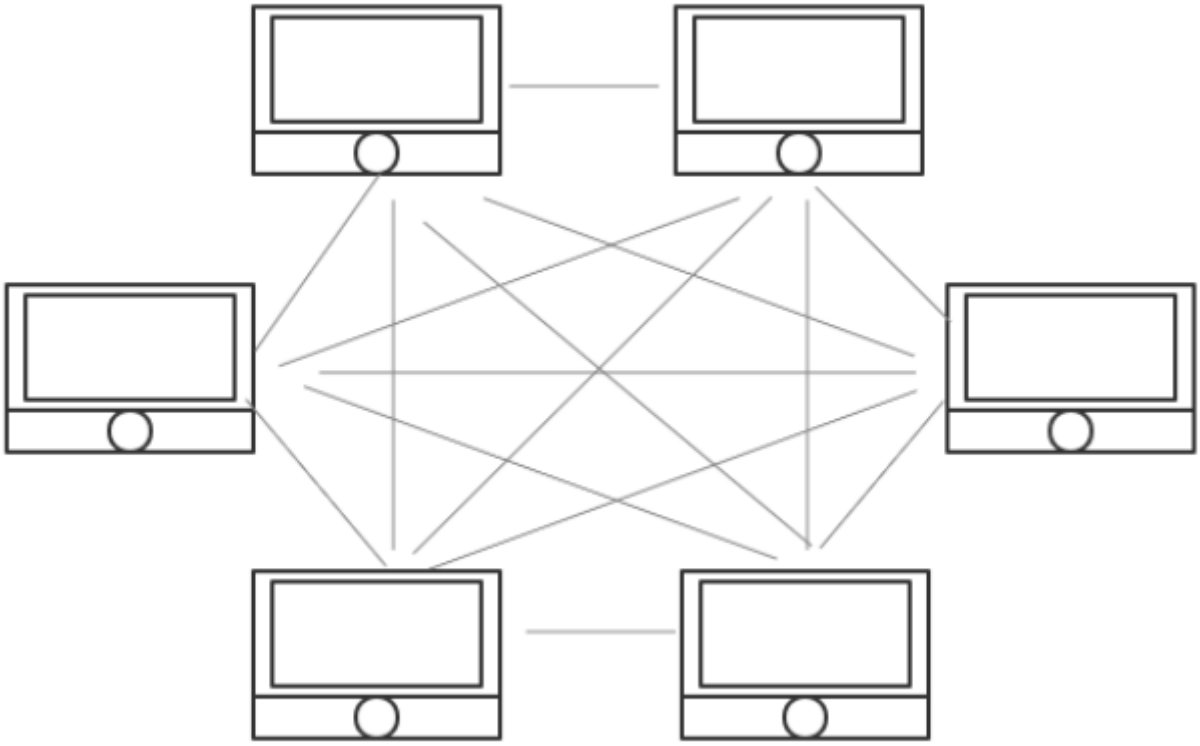
通常情况下，游戏同步强调的是数据的同步，当数据层的数据同步了，那么显示层就可以通过数据做到同步。数据层同步，再驱使表现层同步。

网络游戏架构的发展

P2P 架构

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。

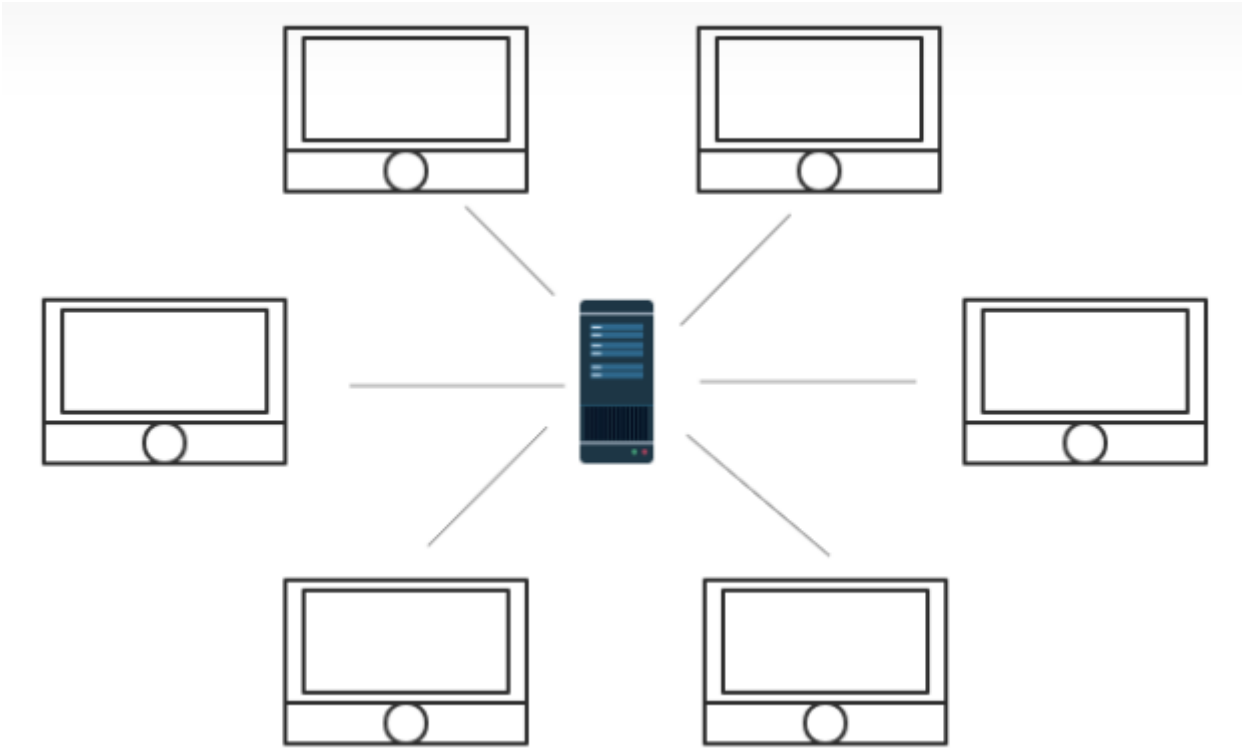
比较有意思的一点是，计算机技术的发展或多或少都与电子游戏有着紧密的联系，甚至很多技术的诞生就是源于对游戏交互方式的探索。



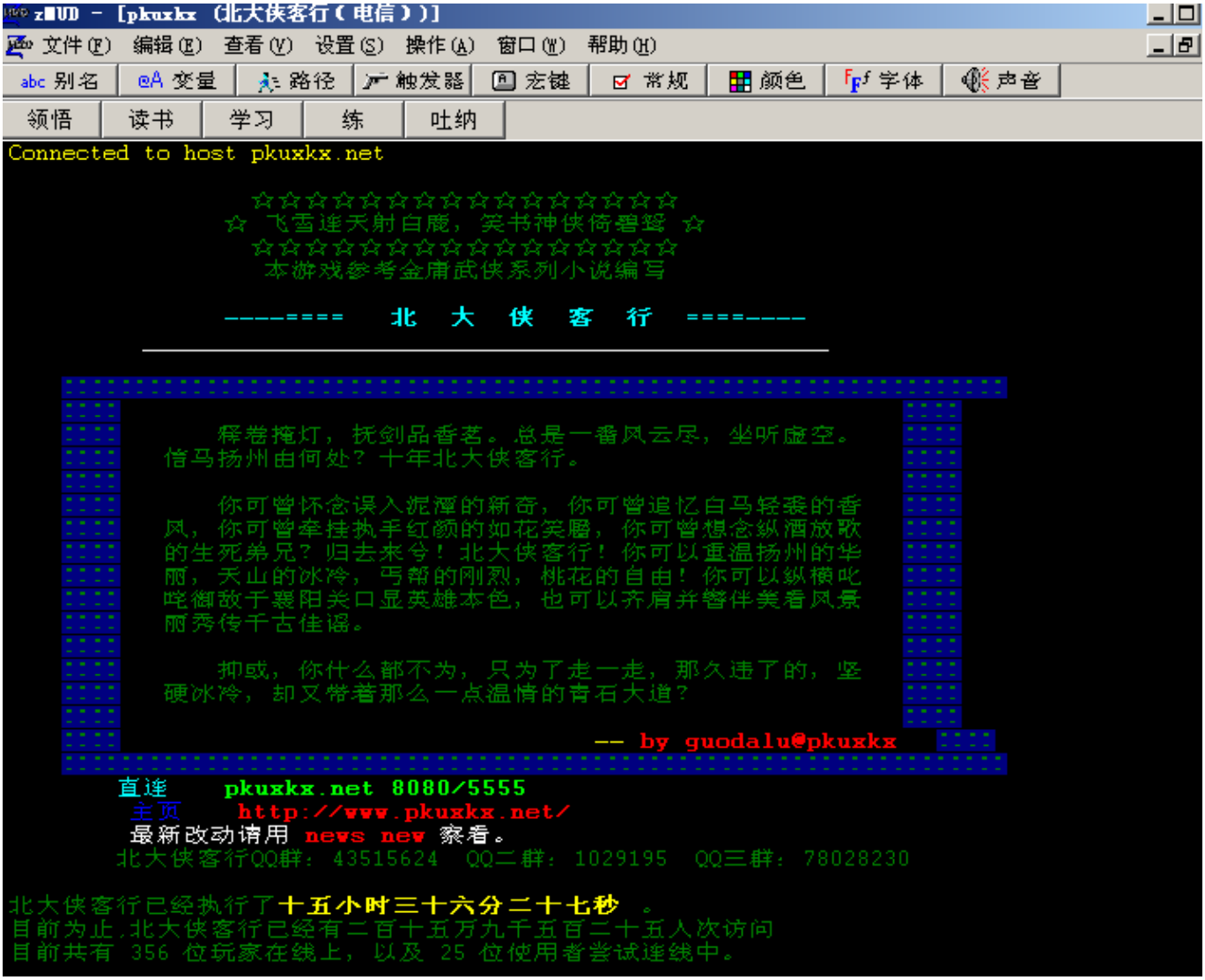
https://baijiahao.baidu.com/s?id=1689217666386425537

CS 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。

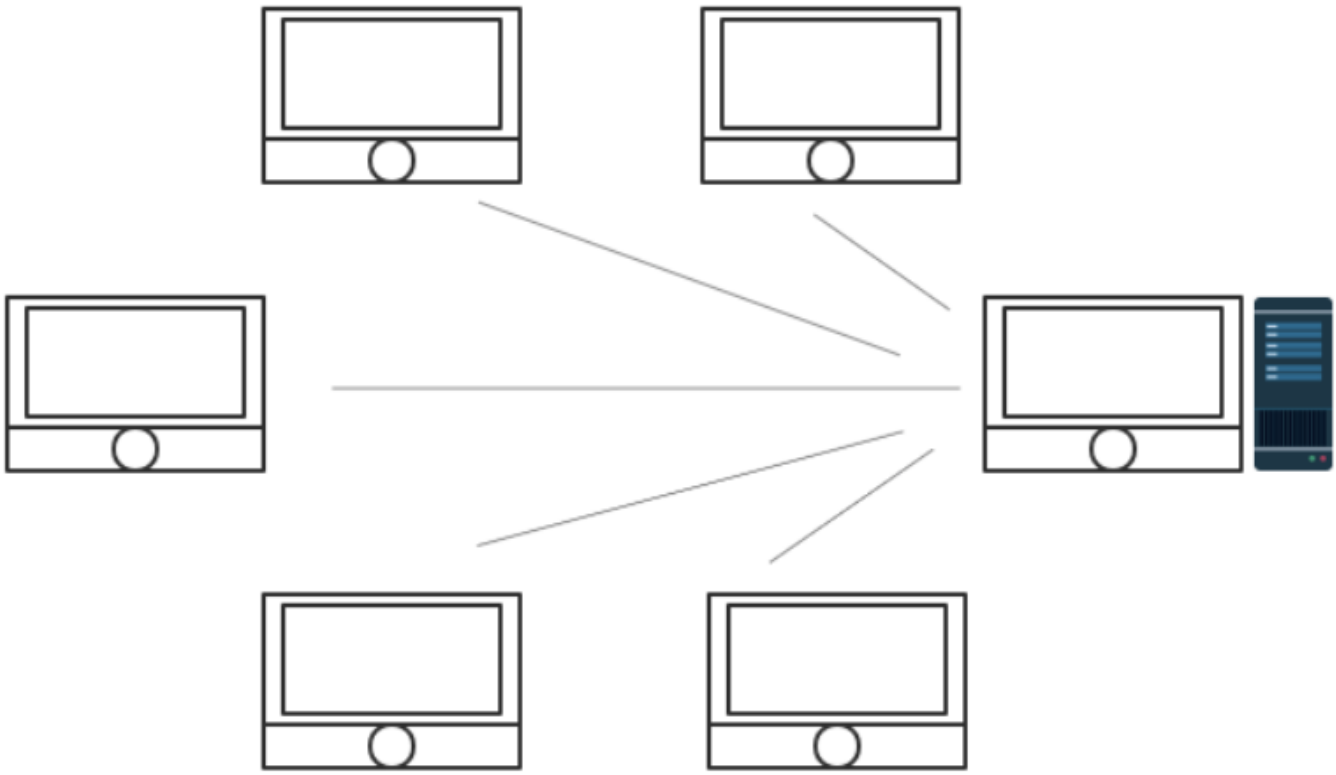


国产MUD游戏，《北大侠客行》



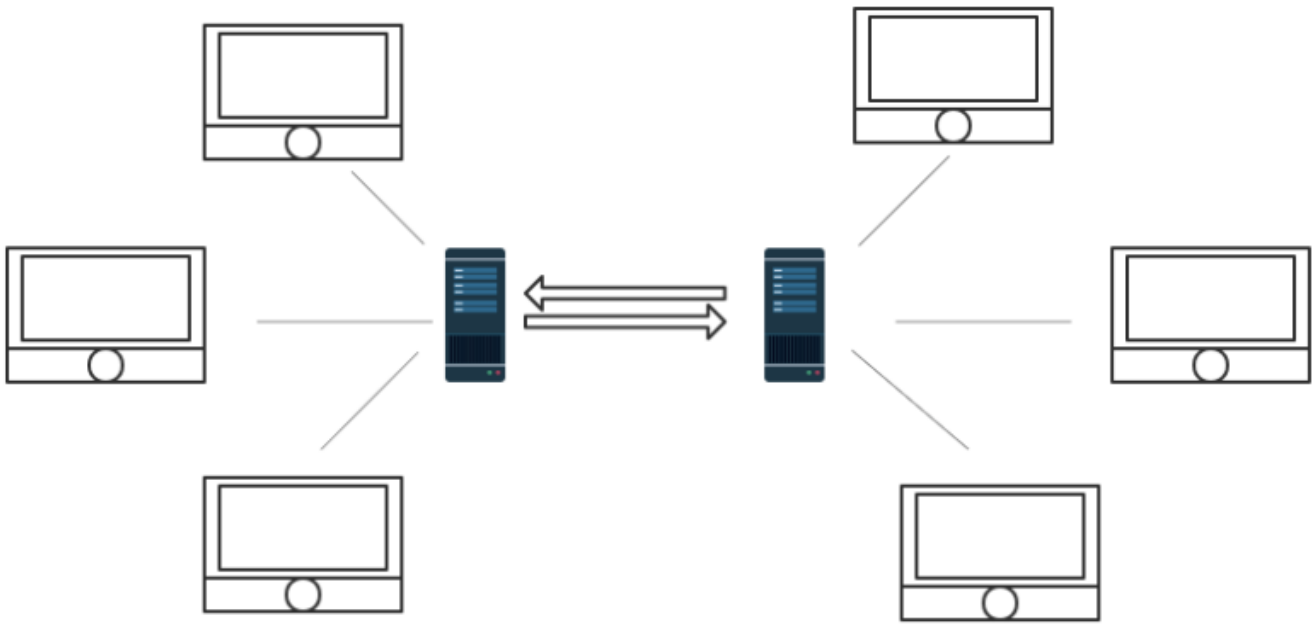
Host主机式 CS 架构

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。



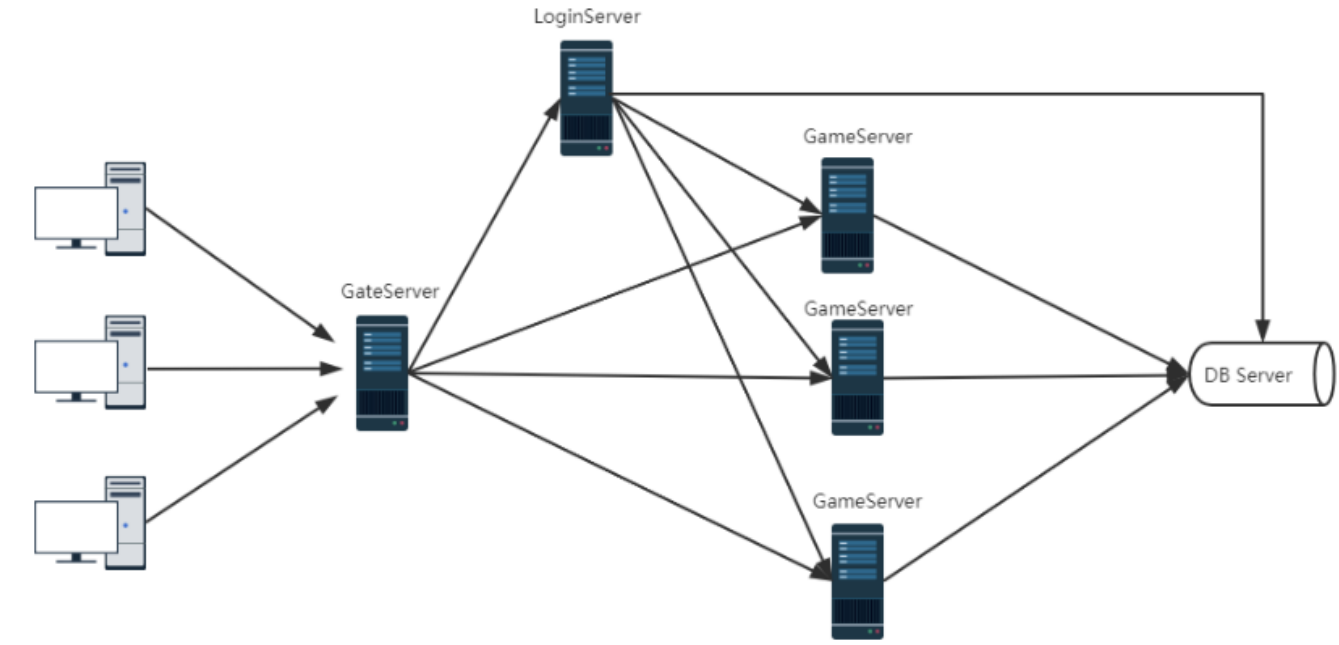
镜像服务器 CS 架构

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。



分布式 CS 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。



关于帧同步

LockStep，即我们所说的帧同步.

帧同步的防外挂策略

关于状态同步

状态同步，顾名思义，强调的是状态数据，多数情况下，由服务器下发状态数据，客户端更加状态信息进行效果的表现。

状态同步需要注意以下情况：

- 状态数据的同步压力；
- 客户端的预表现与预测；

状态同步的数据压力

在状态同步下，如果不做特殊处理，那么同步下发的状态数据就是全量包，对于很多MMO游戏来说，这个数据量太大了，所以往往状态同步都是做增量同步以及AOI。

增量同步是指只同步有被修改的数据。

AOI（area of interest），可以理解为只关心小范围数据，例如在一个超大的MMO游戏世界内，我们可以通过目标客户端角色所处的游戏世界位置，定位到一个小范围区域内，也就是说只需要下发小范围区域内的数据。另外再通过白名单和黑名单的方式，将状态数据再筛选一遍，那么同步的数据包又可以减小一部分。

特殊的物理同步

在游戏中，如果希望有更加真实的物理效果，那么就需要用上物理引擎去表现，重力、弹力、摩擦力等等都可以让游戏对象表现得更加真实。 理论上来说，重力、弹力、摩擦力等等都是对物理对象的坐标、朝向做计算，这让人觉得物理同步似乎和其他常规的数据同步没有区别，这样的观点是错误的。

物理引擎本身就有不确定性，以多年的游戏经验来说，哪怕是3A单机游戏，里面的物理效果，总能找到一些物理表现BUG,

物理世界必须遵循能量守恒定律，浮点数如果变成定点数计算的话，那么就是顶点数忽略的就是小数点能量，它在物理世界是有很大意义的，向上取整意味着无端端增加能量，向下取整就是丢失能量。

https://www.bilibili.com/video/BV1ss411o7Dx?spm_id_from=333.337.search-card.all.click&vd_source=4ef13c12c37e96927eed265ec739144b

同步的优化技巧

同步问题排查记录

在7月底，开始了一波对历史遗留的PVP问题进行逐个修复 其中主要包括：

- 常规报错，导致运行代码中断;
- 不同设备的浮点数计算误差;
- 回滚过程中，数据的重置清理;

- 行为树的数据使用规范，行为树之间的传值规则;
- 显示层数据不应当作为判定条件;
- 逻辑不严谨，对引用类型的数据做了错误修改;

浮点数计算精度误差

判定来源不可靠

游戏流程问题

不严谨的逻辑代码

同步问题 Log 排查技巧

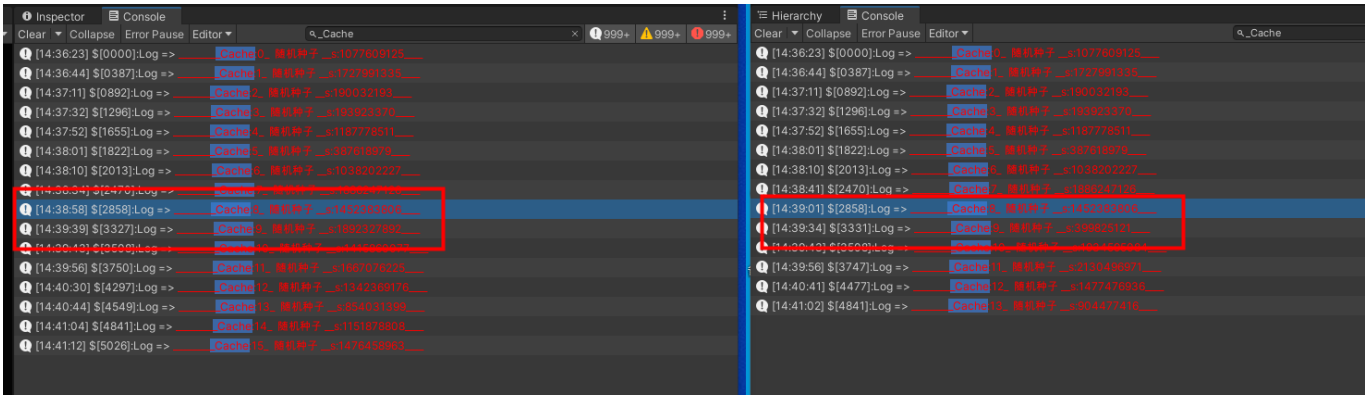
首先我们需要遵守一些规则，如下：

- 只输出关键Log，防止过多无用Log，增加排查难度；
- Log 数据必须带有帧号，帧号是游戏内真正的时间线，是时序判定的唯一途径；
- 必要时 Log 也需要带有角色信息，表明是某个游戏对象输出的Log；

有了以上规则，Log 输出将规范不少。然后我们可以对于两个客户端的日志做搜索。

搜索缓存帧、随机数

搜索 "_Cache"，如下图所示：

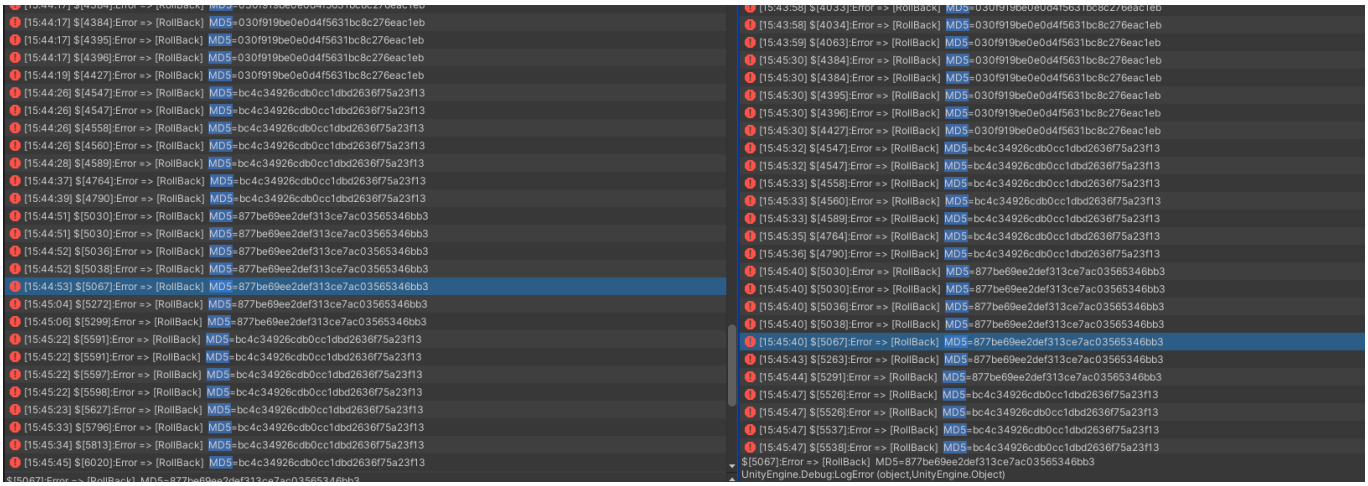


_Cache 是对缓存帧的唯一标识，而缓存帧是回滚数据依据的时间点，通常情况下，每一次回滚都是以最新一次缓存帧作为回滚点，开始回滚时，将游戏内的对象状态数据重置到缓存帧的数据，然后从缓存帧开始，加速追帧到最新的一帧。

如果某一次的缓存帧下的随机数对应不上，说明在这一次的缓存帧之前，出现了一次数据不一致的问题，导致了缓存帧的帧号错位，或者随机数错位。

搜索球员Buff信息

搜索 "MD5="，如下图所示：



MD5 是根据两边队伍的数据，生成的MD5，所以 MD5 必须是帧号与数据的严格一致，才能保证到最终的同步结果。

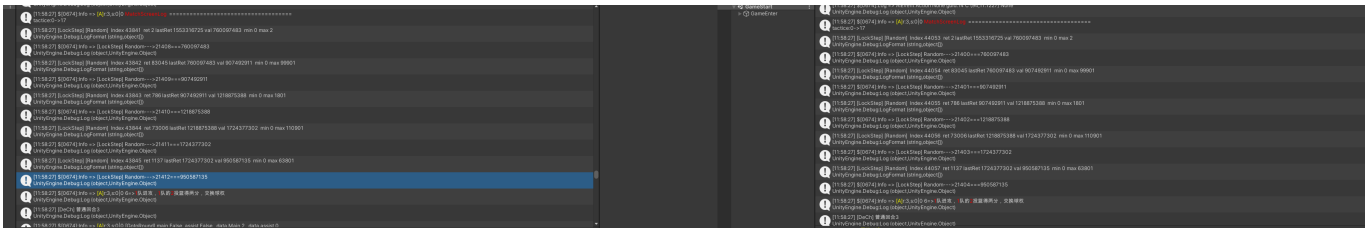
搜索指令信息

搜索 "添加指令" 或 "执行指令"，如下图所示：

指令数据也需要对角色的帧号、所处格子、角色身份验证一致，才能保证最终的数据同步。

随机数时序对齐

搜索 "Random"，如下图所示：



随机数是对一些筛选或假随机的一种特殊实现方式，也必须保证一个帧号与随机数的一致性。

自动化测试工具

有了上文提到的排查Log的方式，那么我们也可以把这些工作交给计算机处理。

大概的设计如下：

- 两个客户端实现自动对局的测试代码，这样可以连续测试多局，无需手动操作；
- 两个客户端各自存储对局Log，带上唯一标识参数；
- 多次对局后，两个客户端把输出的Log进行一对一对比；
- 对比的方式，把帧号当作Key，并且同帧号的数据Log维护成一个队列，保持Log的时序，同步的前提是Log对比一致。如果一致，则本局游戏同步，如果不一致，打出第一行不一致的Log，提供给程序排查，减少人工排查的工作量。