**状态同步与帧同步**

**一、定义**

状态同步：

同步的是游戏中的各种状态。一般的流程是客户端上传操作到服务器，服务器收到后计算游戏行为的结果，然后以广播的方式下发游戏中的各种状态，客户端收到状态后再根据状态显示内容。

是一种不严谨的同步。它的思想中，不同玩家屏幕上的表现的一致性并不是重要指标，只要每次操作的结果相同即可。所以状态同步对网络延迟的要求并不高。

帧同步：

同步的是玩家的操作指令。操作指令包含当前的帧索引。一般的流程是客户端上传操作到服务器，服务器收到后并不计算游戏行为（有些服务器也会计算），而是转发到所有客户端。

帧同步最重要的概念就是相同的输入+相同的时机=相同的输出。

**二、两者的比较**

|  |  |  |
| --- | --- | --- |
|  | 状态同步 | 帧同步 |
| 流量 | 相对高 | 相对低 |
| 安全性 | 服务器实现逻辑，安全性高 | 逻辑在客户端，反外挂压力大。 |
| 服务器压力 | 大 | 小 |
| 断线重连 | 服务器记录状态，从当前状态开始。 | 客户端重跑一遍战斗 |
| 实现方式 | 调优状态同步方式，客户端需要做插值处理. | 客户端按照单机方式开发，保证逻辑层和表现层的分离。 |

总结一下：

1. 对于同步要求比较高的，需要同一时刻不同客户端表现一致的情况，帧同步更容易实现。
2. 对于战斗时大地图MMORPG的，同屏人数很多的情况，状态同步是更好的选择。

**三、AOI(Area of Interest)**

**1.九宫格同步**：

主要思路是将场景地图分为多个格子，每个格子记录其周围的格子信息。

**进入**：角色进入场景，根据其坐标，将其置于一个格子之中。然后向角色所在格子及其周围九个格子中的所有玩家发送“新玩家进入”消息。

**移动：**设玩家移动之前的九宫格集合为oldset，移动之后的集合为newset，则向(oldset - newset)集合中玩家发送“离开”消息。向(newset - oldset)集合中玩家发送“进入”消息。向（oldset & newset）集合中玩家发送“移动”消息。

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 4 | 7 | 10 |
| 2 | 5 | 8 | 11 |
| 3 | 6 | 9 | 12 |

**离开**：角色离开场景，向角色所在格子及其周围格子中的所有玩家发送“玩家离开”消息。

**2.十字链表同步**

首先，根据场景中所有角色的x坐标排序，将其放入一个链表x\_list中。

然后，根据场景中所有角色的y坐标排序，将其放入一个链表y\_list中。

这样，所有的角色同时位于两个链表中。

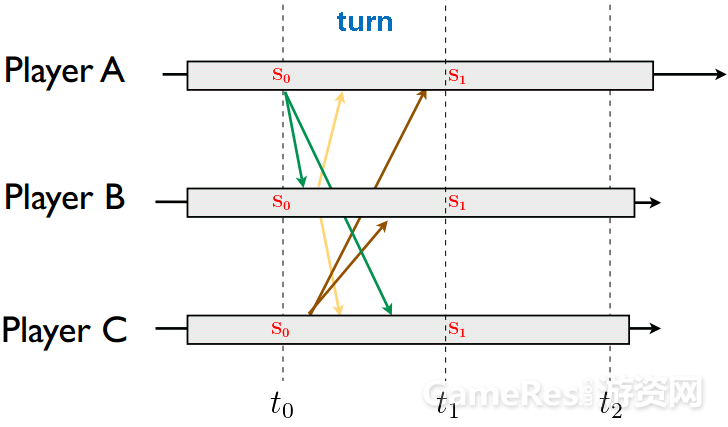
**进入**：玩家进入时，根据x、y坐标排序，分别插入x\_list，y\_list中。同时，根据可视距离，得到x\_list中可视的角色集合x\_set，y\_list中可视的角色集合y\_set，那么（x\_set&y\_set）就是真正可视的角色集合，向其发送“新玩家进入”消息。

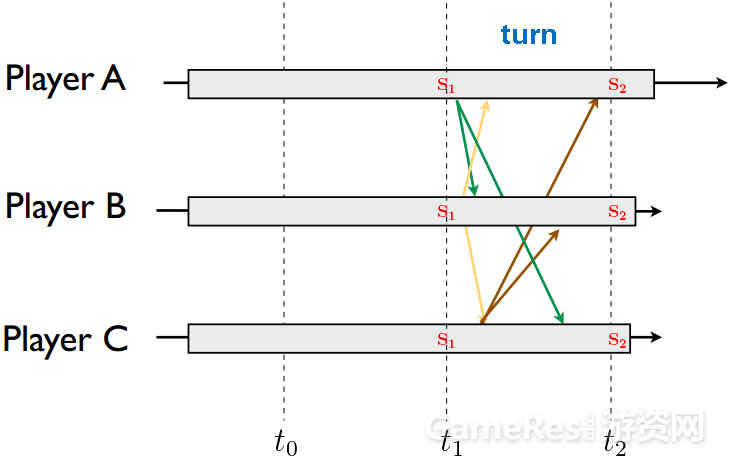
**移动**：根据角色之前的位置可以得到oldset，移动之后，需要根据新的x、y坐标，重新找到角色在x\_list，y\_list中的位置，然后到新的可见角色集合为newset，则向(oldset - newset)集合中玩家发送“离开”消息。向(newset - oldset)集合中玩家发送“进入”消息。向（oldset & newset）集合中玩家发送“移动”消息。

**离开：**向当前真正可视的角色发送“玩家离开”消息，然后从x\_list和y\_list中删除即可。

**四、LockStep同步机制**

帧同步可以说是通过帧率延伸过来的，可以把游戏看成一个巨大的状态机，所有的参与者都采用同一个逻辑帧率来不断的向前推进。





图中是A、B、C三个玩家的时间轴，这个时间轴不是电脑上的本地时间，而是A、B、C联机时定义的一个时间轴。虚线分隔出来的时间片称为turn，可以理解为一帧。箭头表示该玩家将自己的操作指令广播给其他玩家。

所以通过上面我们可以知道：

1、我们把游戏的前进分为一帧帧，这里的帧和游戏的渲染帧率并不是一个，只是借鉴了帧的概念，自定义的帧，我们称为turn。游戏的过程就是每一个turn不断向前推进，每一个玩家的turn推进速度一致。

2、每一帧只有当服务器集齐了所有玩家的操作指令，也就是输入确定了之后，才可以进行计算，进入下一个turn，否则就要等待最慢的玩家。之后再广播给所有的玩家。如此才能保证帧一致。

3、Lockstep的游戏是严格按照turn向前推进的，如果有人延迟比较高，其他玩家必须等待该玩家跟上之后再继续计算，不存在某个玩家领先或落后其他玩家若干个turn的情况。使用Lockstep同步机制的游戏中，每个玩家的延迟都等于延迟最高的那个人。

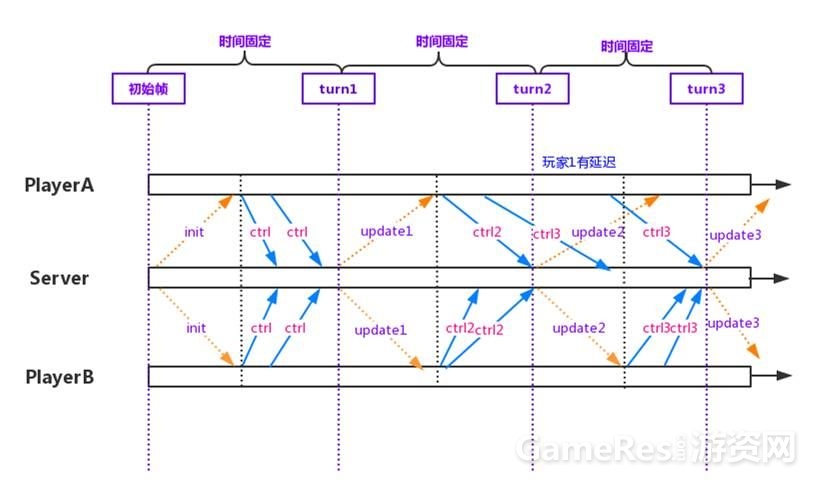
4、由于大家的turn一致，以及输入固定，所以每一步所有客户端的计算结果都一致的。

这种LockStep模式的帧同步，有一个致命的缺陷，若联网的玩家有一个网速慢了，势必会影响其他玩家的体验，因为服务器要等待所有输入达到之后再同步到所有客户端。

**五、乐观锁同步机制**

为了解决上面的问题，采用“定时不等待”的乐观方式在每次Interval时钟发生时固定将操作广播给所有用户，不依赖具体每个玩家是否有操作更新。如此帧率的时钟在由服务器控制，当客户端有操作的时候及时的发送服务器，然后服务器端每秒钟20-50次向所有客户端发送更新消息。

服务器会保存当场游戏的游戏指令以及状态信息，在玩家断线重连的时候，能够恢复到断线前的状态。



我们看到服务器不会再等到搜集所有用户输入再进行下一帧，而是按照固定频率来同步玩家的输入信息到每一个客户端，如果有玩家网络延迟，服务器的帧步进是不会等待的，比如上图中，在第二帧的时候，玩家A的网速慢，那么他这个时候，会被网速快的玩家给秒了，但是网速慢的玩家不会卡到快的玩家， 只会感觉自己操作延迟而已。

**六、需要注意的地方**

**1.定点数**

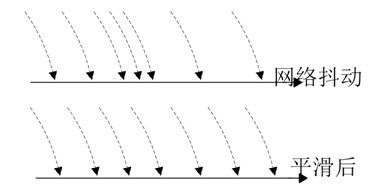
客户端必须保证对网络帧操作的运算过程和结果一致，然而不同系统平台对浮点数的处理有差别，即便差别甚微，也会造成“蝴蝶效应”，导致不同步现象出现。因此需要对游戏对象方位进行定点数改造。

**2.随机种子**

按照帧同步的话，基于相同的输入，每个玩家的客户端都是独立计算伤害的，那么如何保证所有电脑的暴击伤害呢？这个时候就需要用到伪随机了。大部分编程语言内置库里的随机数都是利用线性同余发生器产生的，如果不指定随机种子（Random Seed），默认以当前系统时间戳作为随机种子。一旦指定了随机种子，那么产生的随机数序列就是确定的。就是说两台电脑采用相同的随机种子，第N次随机的结果是一致的。所以在游戏开始前，服务器为每个玩家分配一个随机种子。

**3.网络波动**

帧同步机制下，玩家输入发送到网络，所有响应都必须要等网络逻辑帧才能进行处理。理想环境下，网络帧操作接收到的频率是固定的，能保证客户端表现正常不卡顿。但事实是，绝大多数情况下网络都是不稳定的，时快时慢难以预测。最简单的方案就是建立一个网络逻辑帧的缓冲区，设置一个缓冲区上限，当存入缓存区的帧数满足上限之后，按照固定频率播放。若缓冲区变空，等待其重新填满。通过累积网络逻辑帧延迟，平均分布到固定频率，平滑处理了网络波动造成的卡顿。



**4.丢帧处理**

由于TCP的丢包重传机制会导致较大的延迟，大多数情况下，帧同步都采用UDP协议进行网络通信，这就意味着需要自行解决丢包问题。

预防：关键帧数据包里携带前面两帧的数据，可大大的降低丢包率，但会带来冗余的增加。因此，值得注意的是不能使UDP数据包过大，否则部分路由器会在组合UDP分组时发生错误，建议不超过Internet标准MTU尺寸576byte。

补救：虽然上述方案能起到预防丢帧的作用，仍然无法避免丢帧问题，在出现丢帧问题时，需要客户端根据所需帧序号主动向服务器请求关键帧，包括单帧请求和批量帧请求。为了保证能够获取到所需关键帧。