依赖于网络异步调用的多人网络游戏

deepwaterooo

February 12, 2023

Contents

1 因为先前的大框架还没有过关，想写一两小依赖异步网络调用的多人网络小游戏打基础. 现搜集一些想法，或是现有网络项目，自己可以再加工的.

- 一个样本思路：狠小，但大致覆盖到了几个必要的方面（通过立方体发射子弹多人游戏）
  - https://msyqgzt.gitee.io/blogs/2020/10/21/f68e6db7d782/#KCP%E5%8D%8F%E8%AE%AE

2 桢同步与状态同步

两种同步的技术特点各不相同各有优劣，但是通常规模不大的项目还是偏向帧同步。

主要原因还是在于开发周期短，好维护，好移植，流量消耗小。而且一般只要做好一套 框架 可以复用于多个游戏。

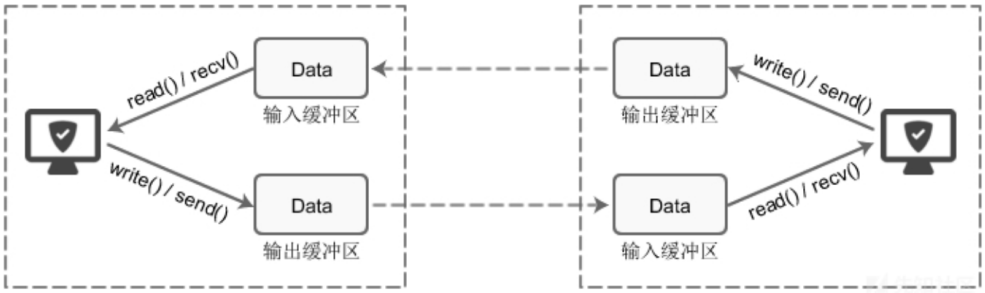
但是在对数据安全性要求很高，同步度要求很高的游戏还是应该使用状态同步。

类型	原理	安全性	维护性
状态同步	逻辑在服务器，服务器运算战斗结果通知客户端，客户端做表现。	数据都在服务器，非常安全	1.实现原理容易。 2.开发的沟通成本高(逻辑和表现分散在服务器和客户端需要调试大量接口)。 3.可移植性差，每款游戏斗得从头来一遍。 4.流量消耗大。
帧同步	逻辑和表现都在客户端，客户端将玩家的操作上发给服务器，服务器接收后广播给所有客户端，客户端在执行。	数据在客户端，玩家可以修改数据 需要用其他办法保证数据安全性	1.实现比较考研架构设计。 2.开发沟通成本极低(服务器只做转发)。 3.可移植性较好(服务器基本上没什么改的，客户端也能保留一套基础框架）。 4.流量消耗极小。

- 当采用桢同步，当客户端的游戏规模大到一定的程度，也就是成为各种各个客户端所看到的场景不一样，因为与服务器交互所造成的各种延迟等，游戏就崩了.....

3 Socket 的缓冲区

- 每个 socket 被创建后，都会分配两个缓冲区，输入缓冲区和输出缓冲区。
- write()/send() 并不立即向网络中传输数据，而是先将数据写入缓冲区中，再由 TCP 协议将数据从缓冲区发送到目标机器。一旦将数据写入到缓冲区，函数就可以成功返回，不管它们有没有到达目标机器，也不管它们何时被发送到网络，这些都是 TCP 协议负责的事情。
- TCP 协议独立于 write()/send() 函数，数据有可能刚被写入缓冲区就发送到网络，也可能在缓冲区中不断积压，多次写入的数据被一次性发送到网络，这取决于当时的网络情况、当前线程是否空闲等诸多因素，不由程序员控制。
- read()/recv() 函数也是如此，也从输入缓冲区中读取数据，而不是直接从网络中读取，如下图所示



- 这些 I/O 缓冲区特性如下：
  - I/O 缓冲区在每个 TCP 套接字中单独存在；
  - I/O 缓冲区在创建套接字时自动生成；
  - 即使关闭套接字也会继续传送输出缓冲区中遗留的数据；
  - 关闭套接字将丢失输入缓冲区中的数据。

4 几种网络调用的原理与区别

4.1 KCP：

★★★★★ KCP - A Fast and Reliable ARQ Protocol

源码： https:// github .com/skywind3000/kcp

c#包装：https://github.com/RainsSoft/kcp-csharp

KCP是一个快速可靠协议，能以比 TCP浪费10%-20%的带宽的代价，换取平均延迟降低 30%-40%，且最大延迟降低三倍的传输效果。纯算法实现，并不负责底层协议（如UDP）的收发，需要使用者自己定义下层数据包的发送方式，以 callback的方式提供给 KCP。 连时钟都需要外部传递进来，内部不会有任何一次系统调用。

整个协议只有 ikcp.h, ikcp.c两个源文件，可以方便的集成到用户自己的协议栈中。也许你实现了一个P2P，或者某个基于 UDP的协议，而缺乏一套完善的ARQ可靠协议实现，那么简单的拷贝这两个文件到现有项目中，稍微编写两行代码，即可使用。

KCP协议比较

如果网络从来不丢包，那么你直接用 TCP就行了，甚至直接裸UDP都没关系，但是网络因为丢包造成卡顿，特别是高峰时期丢包会上到10%的情况，移动设备上这个情况更糟糕。

我自己评测过很多，asio_kcp 的作者做过比较详细的评测，在网络变糟糕的情况下，KCP的延迟比 libenet低三倍以上：

4.2 UDP：

- UDP 的基本原理: Socket 通信是典型的 CS 结构。服务器端为数据接受和处理方，客户端为数据请求方。首先使用 C#，我们建立服务端的启动代码。
- 服务器端：

```
// 0. 定义服务器的接收 IP。这个 IP 应该是公开的，可以被所有访问者看到。
IPEndpoint serverIP = IPEndPoint(IPAddress.Parse("127.0.0.1"), 60010);

// 1. 定义一个 Socket，协议为 UDP
Socket serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

// 2. 绑定 127.0.0.1:60010
serverSocket.Bind(IPEndPoint(IPAddress.Parse("127.0.0.1"), 60010));

// 3. 数据接收
EndPoint clientIP = new IPEndPoint(IPAddress.Any, 0);
byte[] revData = new byte[1024];
int len = serverSocket.ReceiveFrom(revData, ref clientIP);
string receivedData = Encoding.UTF8.GetString(revData, 0, len);
Console.WriteLine("S: Received \"{0}\" from {1}", receivedData, clientIP);

// 4. 发回数据
byte[] returnData = CurCoding.GetBytes(" 你好，我是服务器");
serverSocket.SendTo(returnData, sendData.Length, SocketFlags.None, clientIP);
```

- 说明
  - 第 0 步：定义公开的 IP 地址和端口。地址是本机 IP，而端口根据实际情况进行指定。
  - 第 1 步：对 socket 进行定义。由于要使用 UDP 协议，在初始化时指明 ProtocolType 为 Udp.
  - 第 2 步：对地址进行绑定。
  - 第 3 步：接收数据。这里要注意两点。
    - 线程阻塞：调用 ReceiveFrom() 以后，如果没有数据进入，线程会阻塞在这个地方，以等待数据接入。

        * 接收地址返回：通过对 ref clientIP 的引用，在收到数据后，会返回数据发送方的 IP 地址和端口。这里与 TCP 是不同的，因为在 TCP 协议中，是有连接的，所以 IP 地址是可以通过方法获得的。而 UDP 是无连接，所以在此获得。
    - 第 4 步：发回数据。根据获得的 clientIP，将数据返回。

- 客户端：

```
// 1. 定义一个基于 UDP 协议的 Socket
Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Dgram, ProtocolType.Udp);

// 2. 发送数据
byte[] data = Encoding.UTF8.GetBytes(" 你好服务器，我是客户端！ ");
clientSocket.SendTo(data, data.Length, SocketFlags.None, ServerIP);

// 3. 获得返回数据。
EndPoint serverIP = new IPEndPoint(IPAddress.Any, 0);
byte[] sendData = new byte[1024];
int recv = server.ReceiveFrom(sendData, ref serverIP);
Console.WriteLine("C: Received \"{0}\" from {1}.", CurCoding.GetString(sendData, 0, recv), serverIP.ToString());

// 4. 关闭通信
clientSocket.Close();
```

- 客户说明
    - 第 1 步：对 socket 进行定义为 UDP 协议。
    - 第 2 步：向服务器发送数据。注意：当成功发送数据后，同样也会有自己的端口可以接收数据。这点非常重要，我们可以利用这点，实现两局域网的数据通信。
    - 第 3 步：接收数据，同以上的服务器商。
    - 第 4 步：关闭 socket。
- 注意：上面的源码极其简单，因为它采用的是同步阻塞的方式，在实际项目中用得极少，大家都用异步调用，并封装成流式写法. 只给自己时时提醒一下它的原理，分不清三个的区别

5 ETVoid vs ETTask

5.1 相同点

- ETTask 可以有返回值, 可等待返回结果，他会等待自身内部任务完成再执行下面的语句，通俗点说可以把异步方法变成同步那样的写法
- ETVoid 不能有返回值，不可等待返回结果，不做特殊处理，它的状态是不可获得的，他不会等待自身内部任务完成再执行下面的语句

5.2 不同点

5.2.1 ETTask

- 具体来说就是你异步加载一个超级大的模型，要花 5,6 秒钟，而下面的操作是在模型已经加载完成的基础上进行的，
- 这个时候，**一般情况下是要写回调函数的，然而，使用 ETTask 修饰这个加载模型的函数，调用的时候加上 await 关键字，就会等到模型加载完成再进行下面的操作，**
- 换句话说，await 本身可以看成一个空的回调函数，异步操作处理完了，才会执行 await 下面的语句

5.2.2 ETVoid

- 比如你 Map 服务器从 Gate 服务器收到创建一个游戏物体的请求，就可以使用 ETVoid(因为不需要返回什么东西)
- 再比如你点击一个按钮，所触发的事件函数可以用 ETVoid（emmm，貌似 ETTask 更合理一点？因为你要等他处理完这一次的事件再处理即将到来的事件）
- 因为你不用关心它的状态
- 总来说网络层使用 ETVoid 要比客户端正常的逻辑层频繁的多得多

5.3 具个例子

5.3.1 错误的做法

```
// 这是一个初始化资源的操作，如果这些资源没有添加成功就执行相关操作，就会报空
public async ETVoid Init()
{
^^ITimerComponent timerComponent = ETModel.Game.Scene.GetComponent<TimerComponent>();
^^I// 等待 5 秒
^^Iawait timerComponent.WaitAsync(5000);
^^Iawait ETModel.Game.Scene.GetComponent<FUIPackageComponent>().AddPackageAsync(FUIPackage.FUILogin);
^^Iawait ETModel.Game.Scene.GetComponent<FUIPackageComponent>().AddPackageAsync(FUIPackage.FUILobby);
}

// 下面代码与 Init 在同一调用层级

// 函数调用，运行到第一个 await 之后就会返回，准确的说是运行到第一个返回的 ETTask 就返回，
// 然后时间组件会自己走那个 5s 倒计时
Init().Coroutine();
// 这里会报空，因为资源并没有添加完毕
Game.EventSystem.Run(EventIdType.ShowLoginUI);
```

5.3.2 正确的做法

```
// 这是一个初始化资源的操作，如果这些资源没有添加成功就执行相关操作，就会报空
public async ETTask Init()
{
^^ITimerComponent timerComponent = ETModel.Game.Scene.GetComponent<TimerComponent>();
^^I// 等待 5 秒
^^Iawait timerComponent.WaitAsync(5000);
^^Iawait ETModel.Game.Scene.GetComponent<FUIPackageComponent>().AddPackageAsync(FUIPackage.FUILogin);
^^Iawait ETModel.Game.Scene.GetComponent<FUIPackageComponent>().AddPackageAsync(FUIPackage.FUILobby);
}

// 下面代码与 Init 在同一调用层级

// 函数调用，会等到 Init 中的语句全部执行完毕再执行下面一行代码
await Init(); // <<<<<<<<<<<<<<<<<<<< 这里使用了等待，所以会等到所有的任务执行完了才执行到下一步
// 这里不会报空，因为所有资源已经添加完毕
Game.EventSystem.Run(EventIdType.ShowLoginUI);
```

6 一个带网络请示的：存在问题，自己也可以好好去想它存在的问题：

- https://juejin.cn/post/6844903470865055758
- 项目存在自己的本地. 可以看下网络相关的部分.
- **总结：**这个项目更像是如自己现在这般开发自己的第一第二个网络多人游戏可能会存在的问题，更多的是一个设计上的天生不健全，就是多个客户端之间的同步问题，在设计之初，应该就设计为由服务器端来逻辑同步多个客户端.

- 自己是在用 unity 去尝试做游戏的，中间也遇到了很多很多各种各样的问题，也都在努力去解决。到目前为止也取得了很明显的成果：主流程都是通的，现在允许多个玩家同时在线，在同一个场景中移动，转向，释放技能；服务器也能妥当的同步各个玩家的信息给所有人。不过问题也越来越多，由于现在网络通讯机制的问题，总是会出现莫名其妙的网络断开，而且断开的只是客户端接收响应的通道，客户端依然能够发给服务端消息，服务端也可以收到，当然也正常地返回结果给客户端，然而这里客户端就收不到了。另外一个情况是：现在的同步机制是客户端各自模拟逻辑和运动，服务端只是负责同步各自的位置，某个玩家把自己的位置告知服务端，服务端再广播给所有其他玩家。这样就会有个问题，由于网络延迟的存在，不同玩家看到的场面局势是不一样的，而且现在也没有对技能释放出来的抛出物进行同步，那些火球的攻击判定也是客户端自行判断的，而且不同客户端的判断结果很有可能不一样，这已经很难再进行优化了，要改的话必须改成由服务端计算所有移动判定逻辑，客户端只负责发送指令和显示服务端推送的位置信息，自己移动时客户端不做移动，仅当传送给服务端的指令信息被处理，服务端推送给客户端位置变化的

7 mac 上简单易用的图片修理工具（todo: 去找）

8 应该自己记一个 EME 各个教室 Spring 2023 的课表，自己好比较有准备

- 下面就是字体对不齐的表现，中英文等宽字体会比较好，能够对得齐
- 下面的二楼课表，记录可能有错，要再用一个周左右的时间再校正一下

	星期日	星期一	星期二	星期三	星期四	星期五	星期六
8:30-9:30						前中后都有课	
9:30-10:30		中间有课	有课	中间有课	中间有课	不知道	
10:30-11:30				有课	有课	前中后都有课	
11:30-12:30						后仍有课	
1:30-2:30					有课		
12:00 - 3:00			249 有课				
3:00 - 6:00			249 有课				

- macOS 下，snipaste 的截图质量发白，有点儿关工. 用它是因为一个 F1 键就可以截屏，但是如果质量太差，需要解决，或是换用其它工具 (还是偶尔会出现这样的问题呢？)
- 在自己阅读源码的过程中，无数的标记要手动转换输入法是极其麻烦的事，需要编辑器 emacs 给点儿帮助，在特定加载模式，在特定键的输入后，作必要的输入法自动转换，减少人为操作误差与精力.（大家喜欢 emacs, 也都是因为它的高度可订制化.）爱表哥，爱生活!!! 把那个 sis 包弄懂，用到自己的工具中去. 晚上弄

n- spectacular 分屏工具的使用还是了解一下，可以提高开发效率

- mac 系统下的安卓模拟器：没弄明白是怎么安装的？https://blog.csdn.net/linxinfa/article/details/124844236
  - https://github.com/google/android-emulator-m1-preview/releases
  - 这个还要弄一下，因为必要的时候总是还是需要这些东西的

9 因为对自己系统的开发环境仍然是感觉不熟悉，先试着整个小应用熟悉一下

- 想写一个麻将连连看连连打，可是无从下手，会参考别人的教程，再自己网络搜索一下，试图解决这个问题. 若是实在没有图片，就用别的图片代替.
- 麻将文:https://blog.csdn.net/dlyhy/article/details/121325244?utm_medium=distributepc_relevant.none-task-blog-2~default~baidujs_baidulandingword~default-1-121325244pc_relevant_aa&spm=1001.2101.3001.4242.2&utm_relevant_index=4
- 菜单：最开始学的是 Visual Basic 窗体设计与实现，但是现在全忘光了，连什么方法写在哪里都还需要再科普一下..... 但是会把它写出来
  - https://www.dongchuanmin.com/net/2540.html
  - https://www.dongchuanmin.com/net/2550.html

10 emacs org-mode Skim pdf 双向跳转（以及可能的 latex 与 pdfr 的双向跳转）的实现

- latex, pdf: https://shengdie.github.io/2017/05/29/Mac-Emacs-Skim/
- org-pdfviewer: https://github.com/malb/emacs.d/blob/master/malb.org#pdf-viewer
- pdf-tools: https://github.com/vedang/pdf-tools
- Org-ref integration with Skim.app [Update: 双向跳转: https://emacs-china.org/t/org-ref-integration-with-skim-app-update/4614?page=2
- https://emacs-china.org/t/org-ref-integration-with-skim-app-update/4614/3
- [Emacs PDF 笔记体验优化 #1] org-pdftools 更精准的 PDF links:
  - https://emacs-china.org/t/emacs-pdf-1-org-pdftools-pdf-links/10165
- 早上想好好看看源码，中午下午或是傍晚可以再折腾这些东西

11 熟悉系统开发环境的小项目参考:

- 按照他们这些过来人的经验说法，这个框架不值得一碰，坑太多，有这些精力可能还不如去学习一些别的更有价值的.
- 爱表哥，爱生活!!! 珍爱生命，远离那些坑. 爱表哥，爱生活!!!
- 可能接下来会试着接触更多更好的第三方库，学习那些库来完善这个系统平工程款的开发
- https://www.cnblogs.com/whuanle/p/17060473.html#%E4%B8%BA-maui-blazor-%E8%AE%BE%E7%BD%AE%E8%AF%AD%E8%A8%80
- https://www.dongchuanmin.com/net/2540.html
- https://www.dongchuanmin.com/net/2550.html