# 做一个多线程游戏框架可以多简单

众所周知现在各种游戏终端的发展十分迅猛。其中一个共同的特征是“多核化”，由此带来了游戏开发的“多线程化”，但大量的切入点主要集中在引擎层、WorkerThread层。那游戏逻辑层呢？这里笔者想通过一个demo来说说游戏逻辑的多线程化，然后在推销一下它下面的地基GLogic。

这里我们先定一个小目标：

**“一套代码，支持各种线程模式，开发还很简单”**

然后我们来看看demo:

GitHub: <https://github.com/darktemplar216/GLogicPublicDemo>

请用Unity2018.2.0f1打开这个工程。实际上真正有意义的代码并不挑Unity版本，只是笔者拖了个UI，所以有兼容性问题。

## 命名规范

* L 代表 Logic
* M 代表 Main
* N 代表 Net
* 线程模式里的LMN、LM\_N、L\_M\_N的含义
  + LMN：LogicMainNet都在一个线程里；
  + LM\_N： LogicMain在一个线程里，Net在单独的线程里
  + L\_M\_N：Logic Main Net各自在自己单独的线程里

## Demo1

打开Demo1；

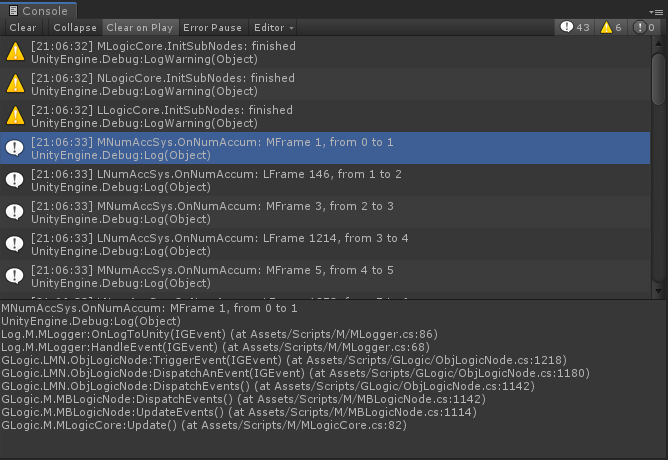
选中Entrance节点；

在inspector上选择线程模式；

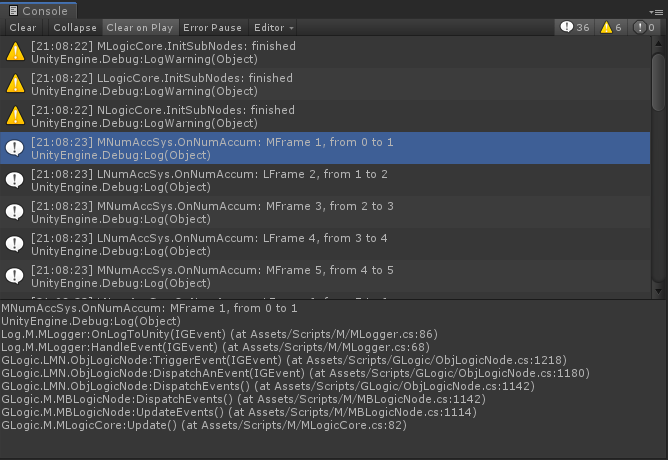
点击播放按钮；

在Console窗口下观察不同；

1. L\_M\_N模式下你应该看到这样的输出：



1. LM\_N和LMN模式下你应该看到这样的输出：



然后，然后这个无聊的demo就完了。那这个无聊的demo到底做了啥？

这个无聊的demo通过两个类的协同工作来累加一个值，一个是MNumAccSys,另一个是LNumAccSys，他们的前缀L或者M代表了我们对它的抽象，L代表了逻辑线程，M代表了主线程。敏感的同学应该可以意识到笔者这里想扯的是表现与逻辑分离，但我们这里还不想展开说。

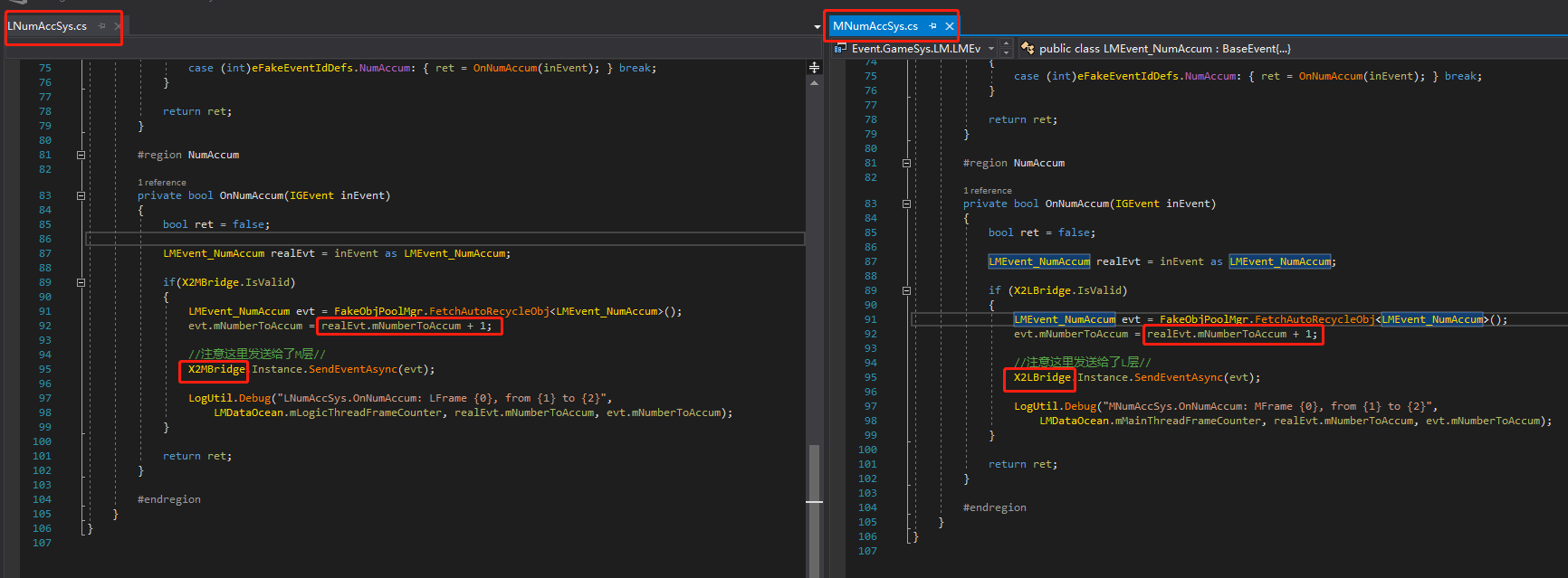
回来观察Log都打些什么：

* 可以看到在L\_M\_N模式下，MFrame(主线程帧号)第一帧的时候把数值从0加到了1，然后由LFrame(逻辑线程帧号)在146帧的时候把这个值从1加到了2。继续看下去的话会发现这两个帧号就如同你和你的前男/女友一样-没有半毛钱关系；
* 接着我们看LM\_N和LMN模式，会发现这两个帧号变得如胶似漆-你现男/女友的感觉；

接着你可以继续跑下去或者编个手机包测试一段时间，观察有没有多线程崩溃或错误。如果没有，那欢迎感兴趣者继续阅读。

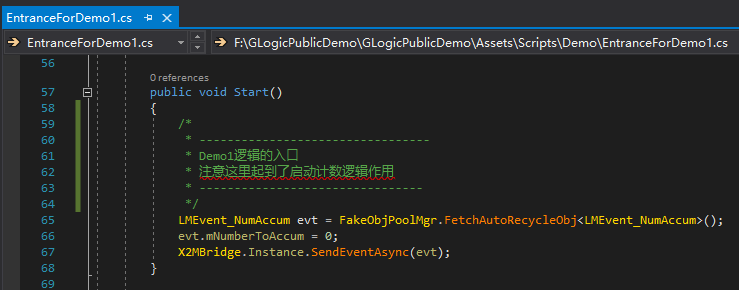
我们意识到这个demo有这样一个特点：在不同线程模式中维持了相同的时序，所以无论哪个线程跑得快慢、多少，运行结果相同。这里大家敏感的话可以意识到笔者想扯帧同步，没错，但是我们这里也不想马上展开说。

那我们这个累加逻辑是怎么实现的呢？



可见我们实际上是在利用消息机制，在多线程的时候是线程间异步通讯，在单线程的时候是线程内异步通讯，所以我们的时序可以保证。

同时提一下，整个逻辑的起始，放在了EntranceForDemo1中：

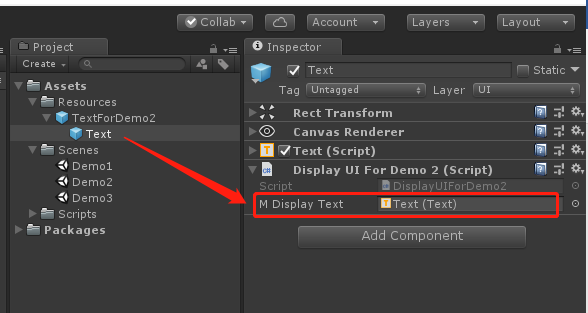


恭喜你现在你做了个任意线程模式下都能跑的游戏

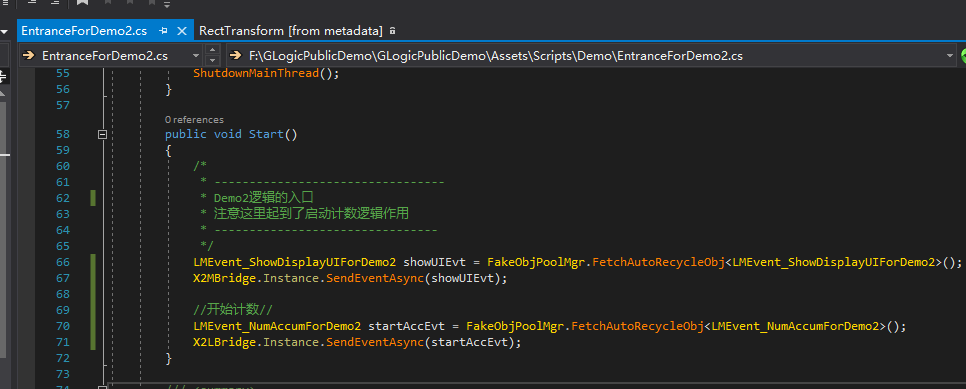
## Demo2

如果我们改下代码，让计数只在L层的LNumAccSys计算，而在M层为一个UI.Text控件赋值呢(DisplayUIForDemo2)？那恭喜你，你已经做到了任意线程模式下的逻辑和表现分离，懒得自己写的同学请运行Demo2。如果出现Unity版本导致的UI不兼容，就请你自己怎么搞下，笔者就不管了。

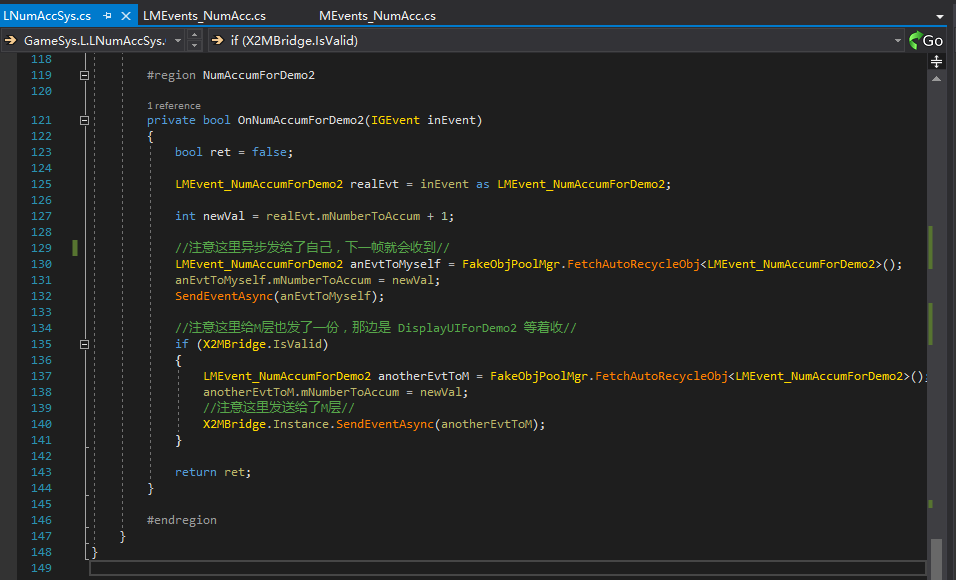
Prefab长这样：



入口在这：



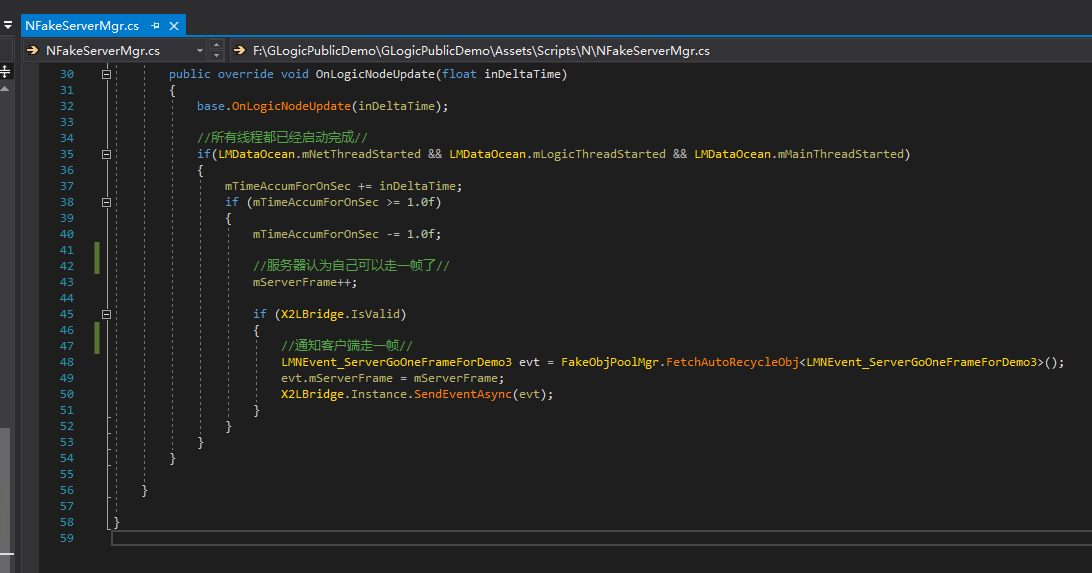
实际逻辑在这：



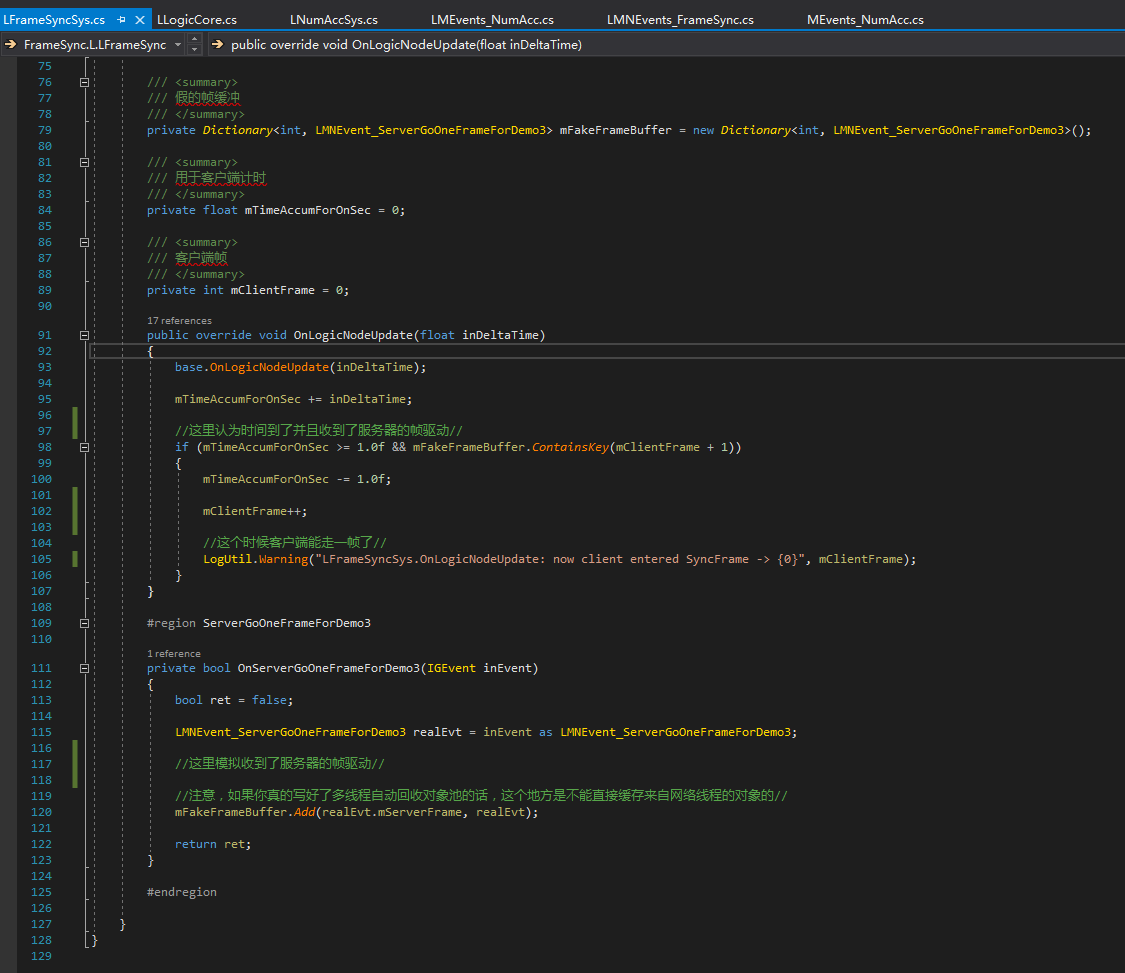
## Demo3

那如果我们再改下呢？我们加入一个NFakeServerMgr，用于提供每秒一次的逻辑帧驱动。那么一旦这个NFakeServerMgr变成真Server，频率变成66毫秒一次，那我们就变成了一个任意线程模式、表现逻辑分离的帧同步游戏了。仍然懒得自己写的同学就请运行Demo3。

模拟帧同步服务器：



示例帧同步客户端：

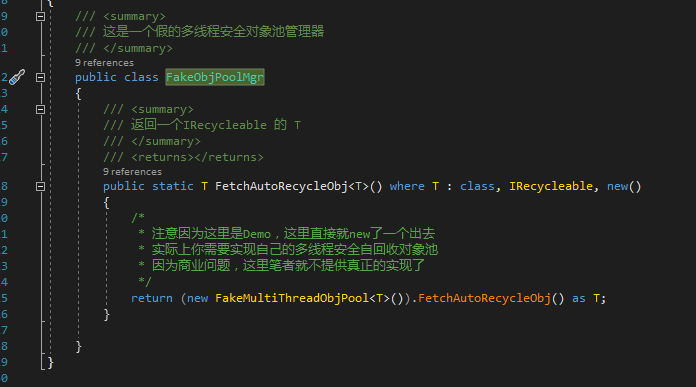


## Demo4

那如果我们再继续猛烈的改一下呢？数据分离，加入实体，XXXSys…呢？恭喜你，你已经有了一个任意线程模式、表现逻辑分离、ECS的帧同步游戏了…。懒得写的同学也自己去写，这么多代码已经超过了Demo的容量。

## 本文Demo中没说的东西

* 你可能注意到了FakeObjPoolMgr是假的



这里只描述一下该写什么东西：你需要有一个多线程自释放对象池

* 其实LogUtil也值得你看下
* 其实笔者本身在这个GLogic上做的东西远不止文中所提及的内容，大家请发挥想象力

## 你说了这么多，这GLogic到底是啥

限于篇幅我们这里简单说说里面最基本的一些概念，其它更深层的用法请大家自己阅读代码吧，别担心，里面有充足的注释。

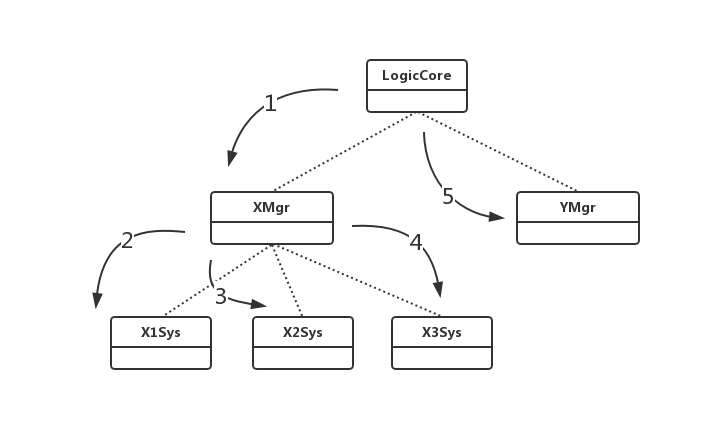
基本上GLogic作为一个逻辑框架，提供了两个东西：

1. **时序控制**
2. **消息机制**

它通过实现一个叫做“逻辑树”的概念来达成以上两点。

1. **逻辑树**
   1. 逻辑树由逻辑节点互相挂接组成，逻辑节点是一个实现了IGLogicNode接口的类；逻辑树的挂接形态是时序的基础；
   2. IGEvent和IGEventListener分别作为消息和消息监听器的接口，提供了在逻辑树中监听消息的能力；
   3. 逻辑树的根节点一般称为LogicCore，本身仍然是一个逻辑节点，但额外担负了循环入口的重担；
   4. 逻辑树之间的相互挂接实现了上面各Demo中的任意线程模式切换
2. **时序控制**

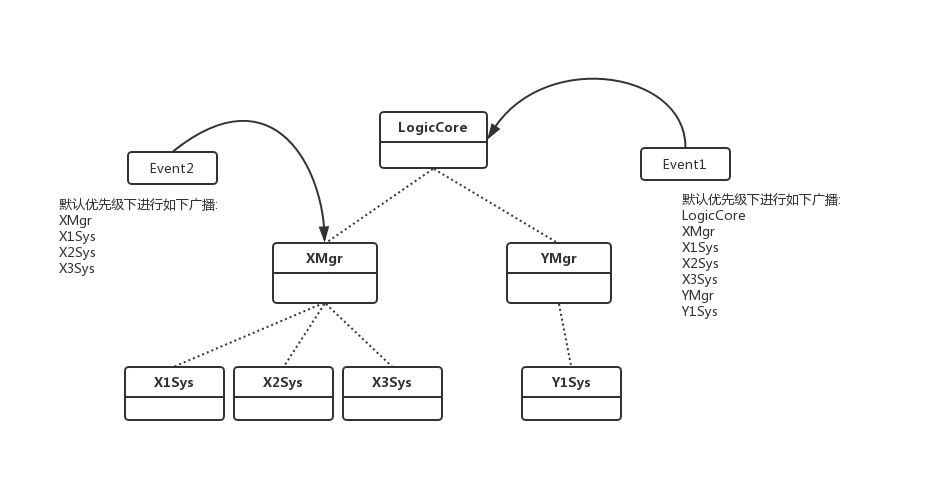
**树状结构，深度优先遍历**



**图中弧线表示默认执行顺序，**

**本执行顺序可通过“节点优先级”进行深度定制**

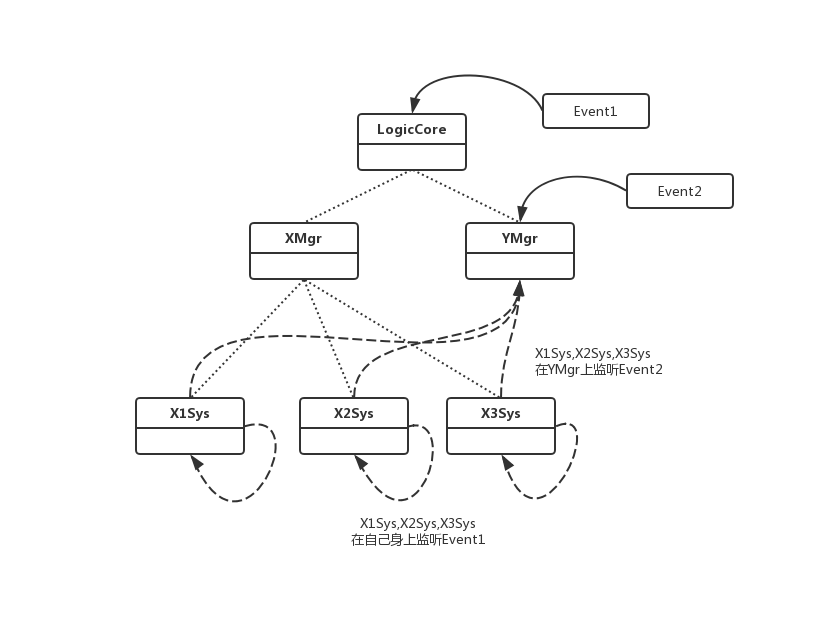
1. **消息机制**
   1. **层级消息广播**



**可见在不同层级上抛出的消息，它的辐射面是不同的**

**本顺序可以通过“消息优先级”进行深度定制**

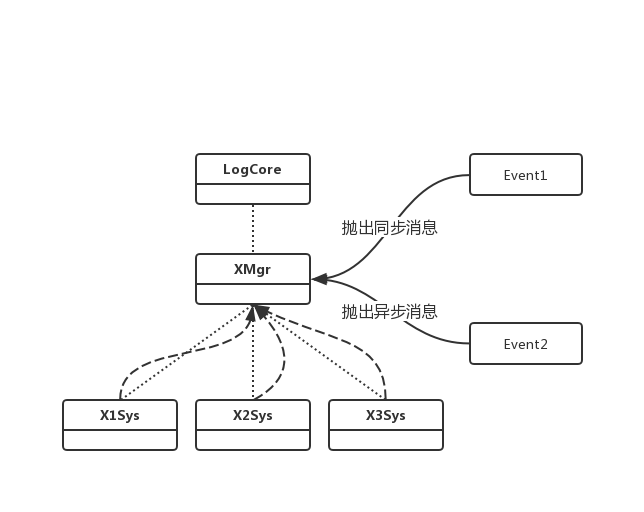
* 1. **消息监听**



**根据我们的广播原理，各Sys都将监听到Event1和Event2**

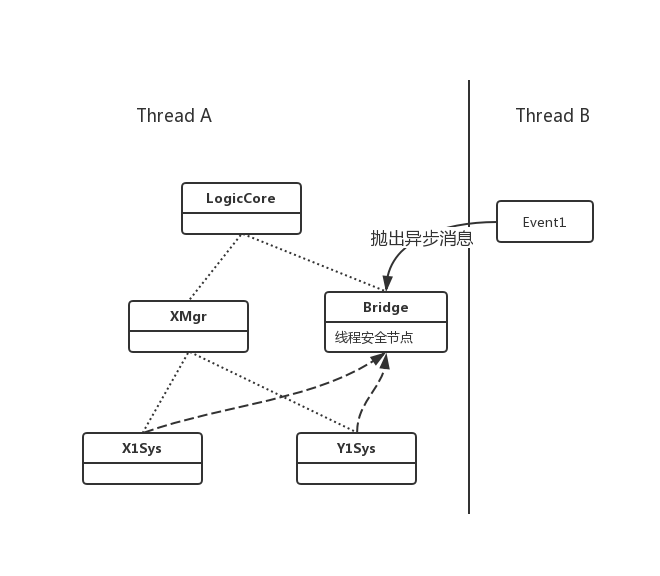
**但Event1的广播量明显有浪费**

* 1. **同步及异步消息**



**各Sys会立即收到Event1，而Event2则会在下一帧收到**

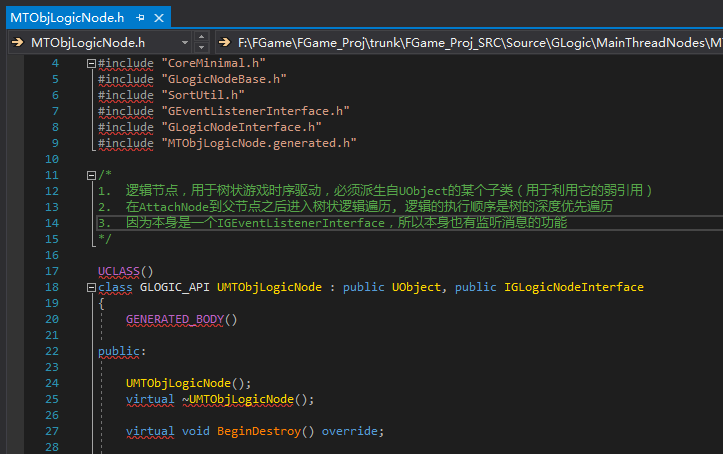
* 1. **线程安全**



**Bridge是一个线程安全节点，这样另一个线程就能安全的在它上面抛出Event1**

**各Sys将在ThreadA的时序内收到Event1，无需担心线程安全问题**

1. **其它：**
   1. 消息拦截，在HandleEvent中返回true来阻止消息继续广播。这个特性在异步Job分配、负载控制、loading拦截输入等场景尤其便利。
   2. 节点优先级、消息监听优先级，决定了节点的执行顺序及收到消息的顺序，这个机制在动态启动高优先级逻辑、数据池优先于所有逻辑感知数据变化等场景尤其便利。
   3. C++版本？GLogic有适用于Unreal4的C++版本，逻辑思想类似，大家可以自行改造。



# 综述

说这么多，笔者无非想表达一个意思：希望GLogic能帮助你在当下多核设备环境中搭建高性能、高灵活度的游戏框架。

使用代码的话请保留作者声明。