性能优化

1，处理这部分应该找清楚性能瓶颈在什么地方，即为什么卡？

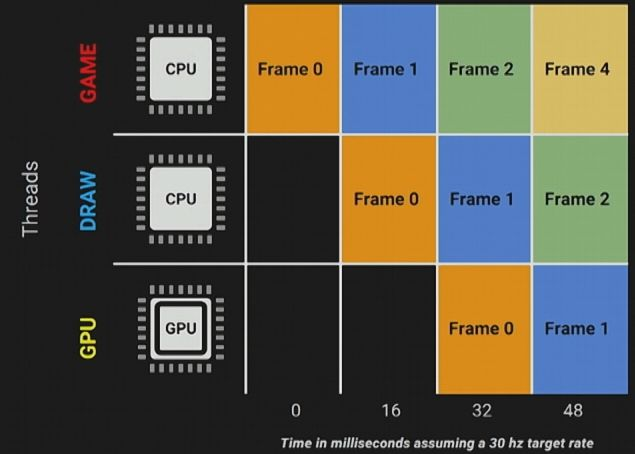
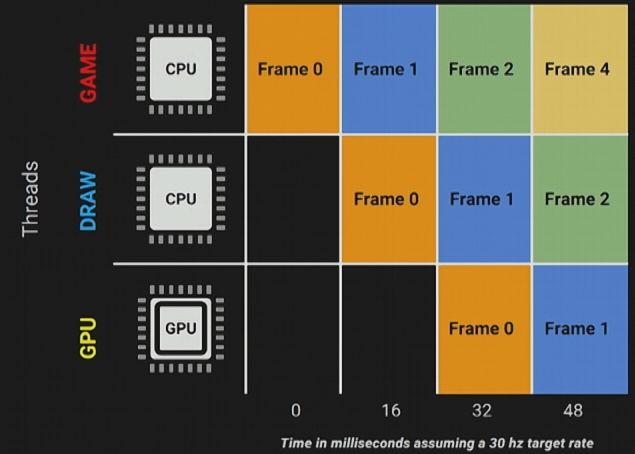
stat fps：帧率

stat unit：

Game: 处理游戏逻辑耗费的时间

Draw:  准备好所有必要的渲染所需的信息，并把它从 *CPU* 发送给 *GPU* 所耗费的时间

GPU：接收到渲染所需信息之后，将像素最终的表现画在屏幕上的耗时



**Game Thread** 首先会对整个游戏世界进行逻辑层面的计算与模拟(*e.g.Spawn* 多少个新的 *actor*、每个 *actor* 在这一帧位于何处、角色移动、动画状态等等)，所有这些信息会被输送到 *Draw Thread*

**Draw Thread**(也叫 *Rendering Thread*) 会根据这些信息，剔除*(Culling)*掉不需要显示的部分*(e.g. 处于屏幕外的物体)*，接着创建一个列表，其中包含了渲染每个物体必备的关键信息(*e.g.* 如何被着色、映射哪些纹理，着色器Shader，材质Material等等)，再将这个列表输送给 *GPU Thread*

**GPU Thread** 在获取了这个列表之后，会计算出每个像素最终需要如何被渲染在屏幕上，形成这一帧的画面

综上，对于每一帧来说，这三者的执行顺序依次为：*Game Thread → Draw Thread → GPU Thread*

一帧的总耗时，很大程度上取决于三者中开销最严重的线程，因为某一个线程卡顿就会造成这一帧出现卡顿

Game Thread 和 Draw Thread 在 CPU 上运行，GPU Thread 在 GPU 上运行

GameThread

主要是C++和蓝图的逻辑处理，瓶颈常见于Tick和代价昂贵的逻辑实现

**stat game**：显示 Tick 的耗时情况

**dumpticks**：可将所有正在 *tick* 的 *actor* 打印到 *log* 中

DrawThread

主要是Visibility Culling和Draw Call

Visibility Culling:基于深度缓存信息，剔除位于相机的视锥体之外和被遮挡住的物体

Draw Call: CPU准备好一系列渲染所需的信息，通知GPU进行一次渲染的过程，很多情况下，不同多边形需要的是同一种颜色，给笔刷蘸好颜色，可以一次性给这些多边形上色。

这个过程就叫做合批，只要合批机制完善，DrawCall的数量也可以非常少。

GPUThread

顶点处理导致的瓶颈

DynamicShadow： ShadowMapping，一种在光栅化计算阴影的技术，当顶点数过多，DynamicShadow 将成为GPU在光栅化阶段的一大性能瓶颈

ShowFlag.DynamicShadow 0:可关闭动态阴影

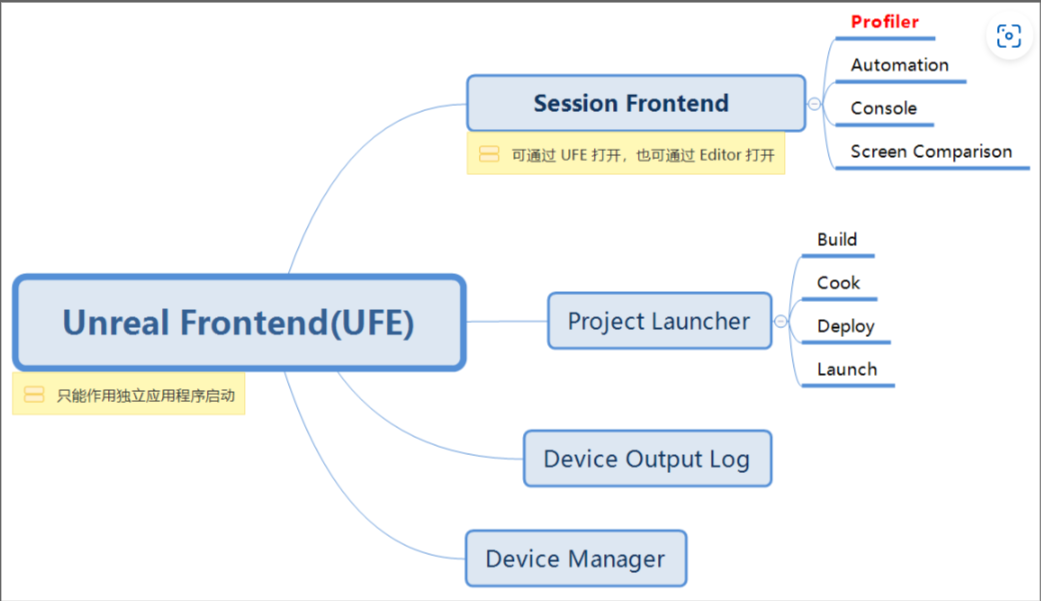
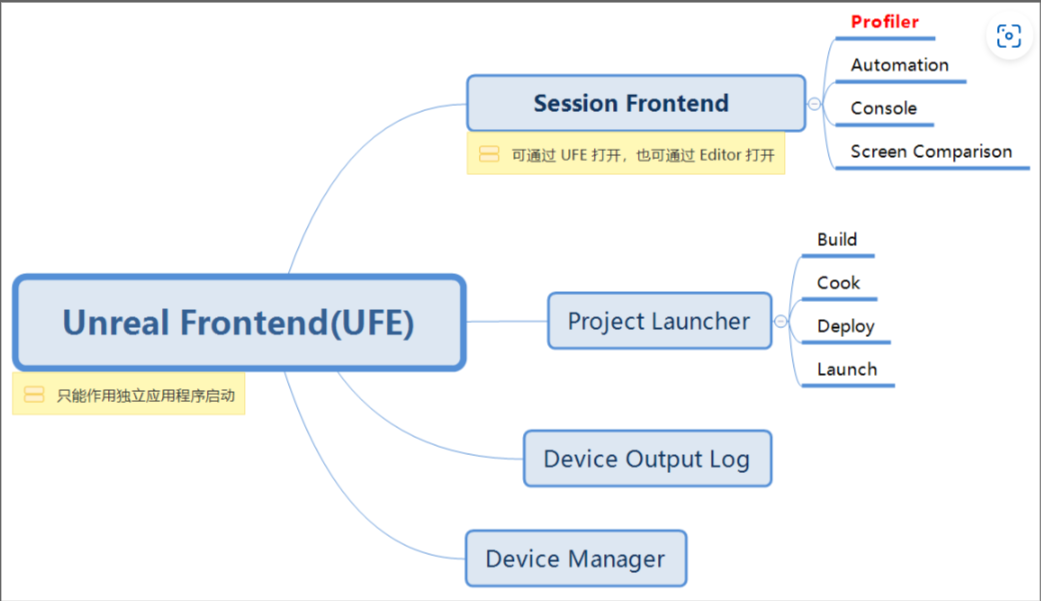
着色导致的瓶颈

r.ScreenPercentage X 可以将渲染的像素数量到X%

场景中存在过多的半透明物体，会显著增加Pixel Shader的计算压力

2，如何通过工具找到卡的原因

①Session Frontend



Unreal Frontend Profiler

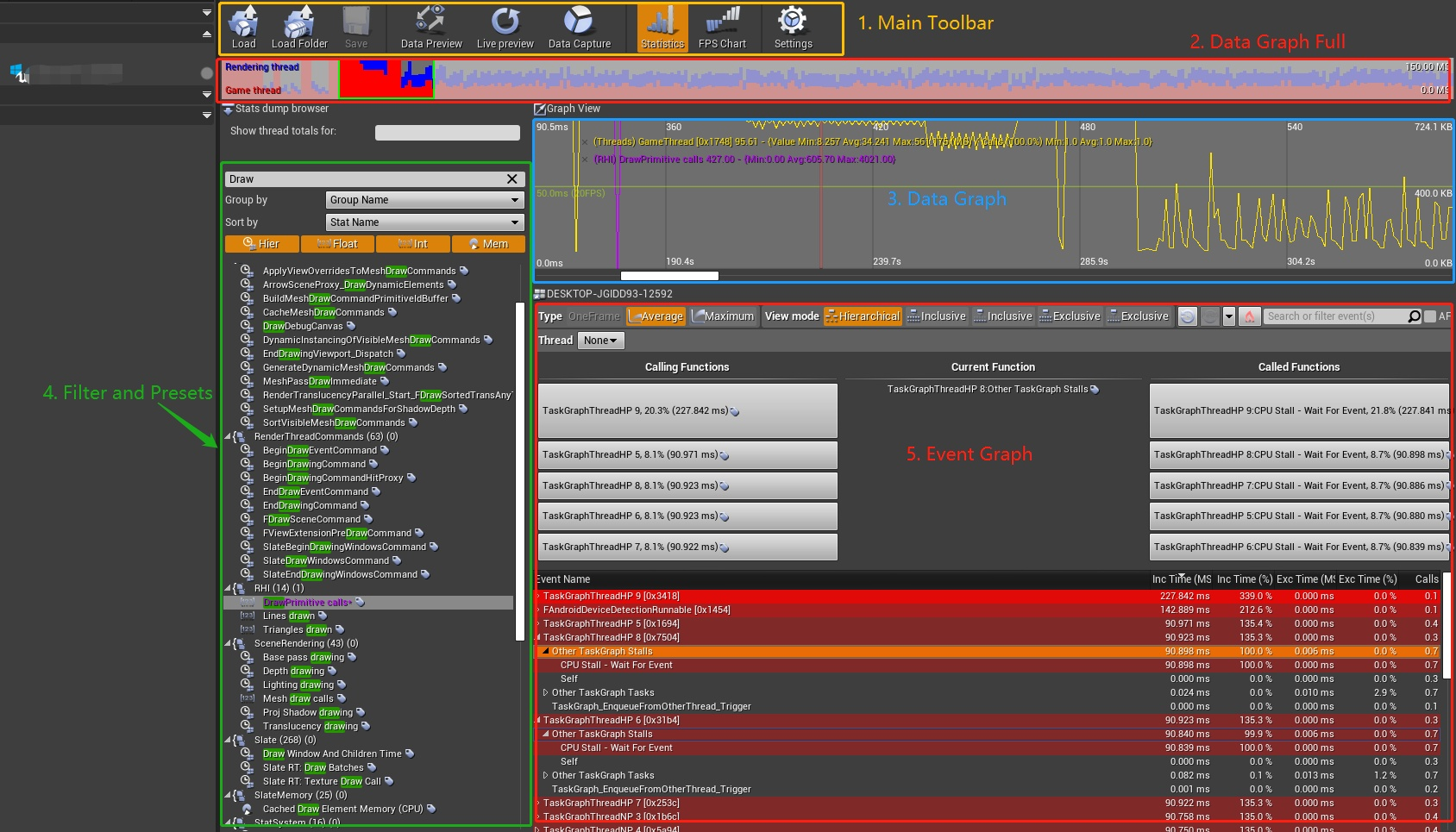
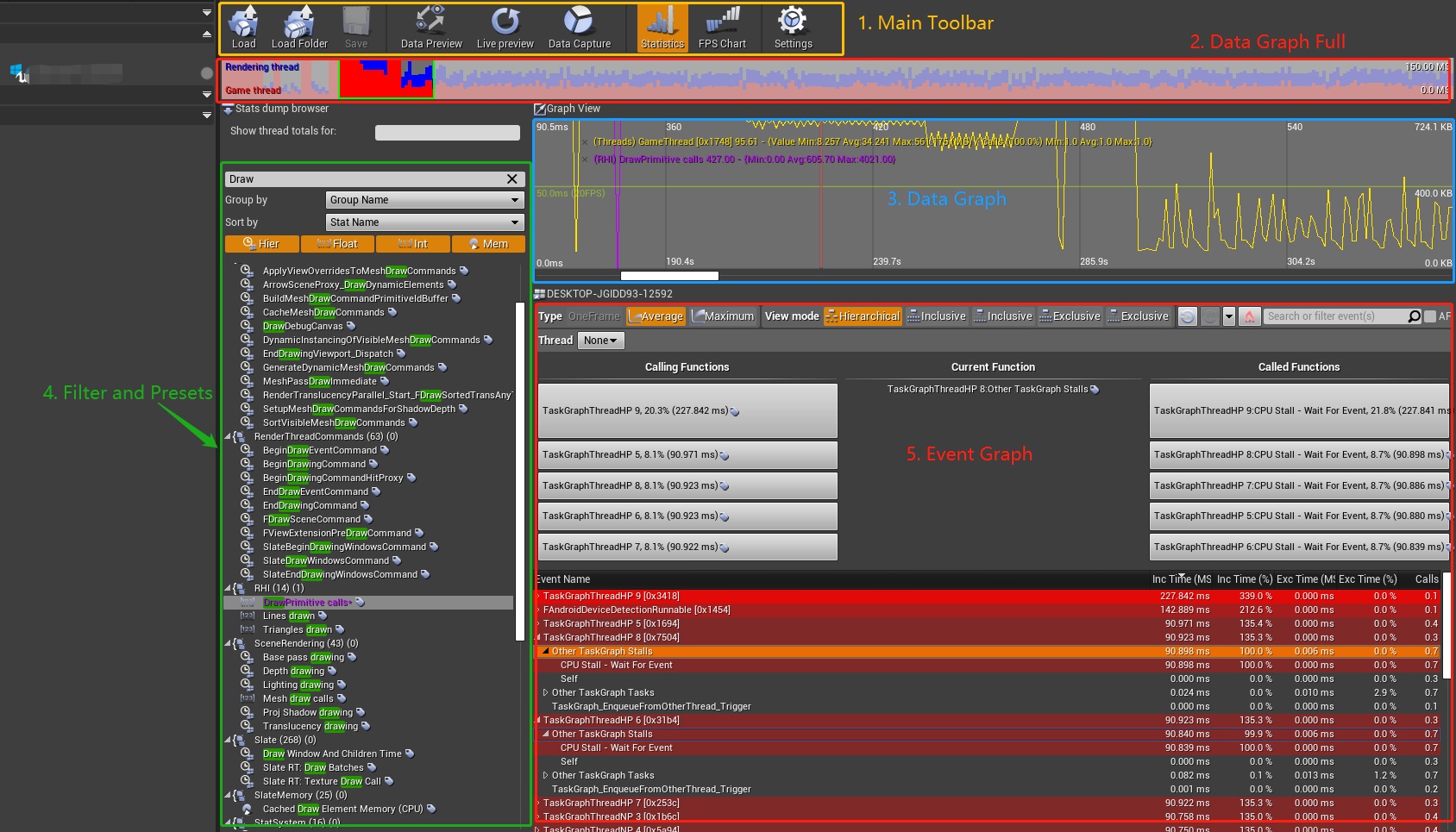
Unreal Insights

GPU Visualizer & RenderDoc

**Unreal Frontend**只能检测到 **CPU** 侧的开销信息，无法用于分析 **GPU** 方面的性能

LiveData:贮存在内存中的实时数据，无法保存以反复查看

CapturedData将实时信息输出成本地文件后的数据，可加载到Profiler中反复查看



②UnrealInsights（这个感觉更好用些）

**原理：**

UnrealInsights主要通过FTraceAuxiliaryImpl类进行驱动，负责解析命令行， 控制连接服务器和打开insight文件，以及Unreal Insights的开启/关闭。

class FTraceAuxiliaryImpl：

Member:

TMap<uint32,FChannel> Channels:当前要进行profiler的channel

TraceDest:连接的服务器地址或写入的文件名

**开启Profiler流程：**

1，游戏 FEngineLoop::PreInitPreStartupScreen函数调用FTraceAuxiliary::Initialize函数，传入CmdLine，开启UnrealInsight功能。

2，写入程序简要信息，包括平台，AppName等

3，解析-trace=channel1,channel2... 加入到channelmaps里，并设置Active状态。如果用户没有提供Channel，UE会去Engine.ini获取Trace.ChannelPresets指定的预置Channel

4，解析-tracehost，查看所需要发送到的insights服务器ip地址（ip:port可指定，默认使用1980端口），并建立一条TCP连接

**收集Profiler数据：**

1，创建profiler标记

常用的是 TRACE\_CPUPROFILER\_EVENT\_SCOPE(Name)，在函数第一行添加就可以记录函数的总执行时间，

（需要注意的是很多地方都使用了封装了这个宏的宏，如：SCOPED\_NAMED\_EVENT(Name,Color)等）

宏最终展开为

**#define TRACE\_CPUPROFILER\_EVENT\_SCOPE\_ON\_CHANNEL\_STR(NameStr, Channel) \**

**static uint32 PREPROCESSOR\_JOIN(\_\_CpuProfilerEventSpecId, \_\_LINE\_\_); \**

**if (bool(Channel|CpuChannel) && PREPROCESSOR\_JOIN(\_\_CpuProfilerEventSpecId, \_\_LINE\_\_) == 0) { \**

**PREPROCESSOR\_JOIN(\_\_CpuProfilerEventSpecId, \_\_LINE\_\_) = FCpuProfilerTrace::OutputEventType(NameStr); \**

**} \**

**FCpuProfilerTrace::FEventScope PREPROCESSOR\_JOIN(\_\_CpuProfilerEventScope, \_\_LINE\_\_)(PREPROCESSOR\_JOIN(\_\_CpuProfilerEventSpecId, \_\_LINE\_\_), Channel);**

上述流程为： PREPROCESSOR\_JOIN宏把两个参数拼接到一起，因此创建了名为\_\_CpuProfilerEventSpecIdXXX的uint32变量，注意这是一个static变量，每个标签都对应一个，可以称为SpecId。之后判断该变量是否为0，即我们是否第一次进到这个标签，如果是第一次，要使用FCpuProfilerTrace::OutputEventType()函数获取新的Id，Id是自增的且可多线程访问。

最后创建FCuProfilerTrace::FEventScope对象，该对象的生命周期就是我们的profile周期，构造函数开始profile，析构函数结束profile。

**FEventScope：**

**struct** **FEventScope**{

FEventScope(uint32 InSpecId, **const** Trace**::**FChannel**&** Channel)

**:** bEnabled(Channel **|** CpuChannel)

{

**if** (bEnabled)

{

OutputBeginEvent(InSpecId);

}

}

**~**FEventScope()

{

**if** (bEnabled)

{

OutputEndEvent();

}

}

**bool** bEnabled;

};

构造函数和析构函数都会判断bEnabled，而bEnabled由Channel决定。所以可以中途通过Trace.start和Trace.stop来动态改变Channel值，从而达到按需开启/关闭Profiler数据收集的目的，但此时TCP连接还在。

。

。

。

。

（解析过程未完）

**Channel定义：**

使用UE\_TRACE\_CHANNEL\_DEFINE定义

创建名为CpuChannelObject的FChannel实例，作为静态变量，然后创建名为CpuChannel的引用对象。之后创建名为FCpuChannelRegister的结构体和实例，主要作用为调用CpuChannelObject.Setup函数进行注册

**如何自己加Insights数据：**

**程序启用方式：**

1，UnrealInsights作为单独程序打开，位置在{UE Engine Folder}/Engine/Binaries/Win64/UnrealInsights.exe

2，程序以独立窗口方式运行，即可进行监听，如果想以PIE方式运行，需要在Editor的启动参数加上-tracehost=127.0.0.1，不加则不会正常进行监听

**客户端的部分启动参数（在AdditionalLaunchParameter进行添加）**

通道列表（逗号间隔，不区分大小写）                                 -trace=Log,bookmark,frame,cpu....

数据传输的DesIp(UnrealInsights启动默认监听端口1980)    -tracehost=局域网ip

指定trace后文件输出目录（相对路径）                                -tracefile=../../../WatchMan/tracefolder/tracetestfile1 可以直接丢到工程目录下

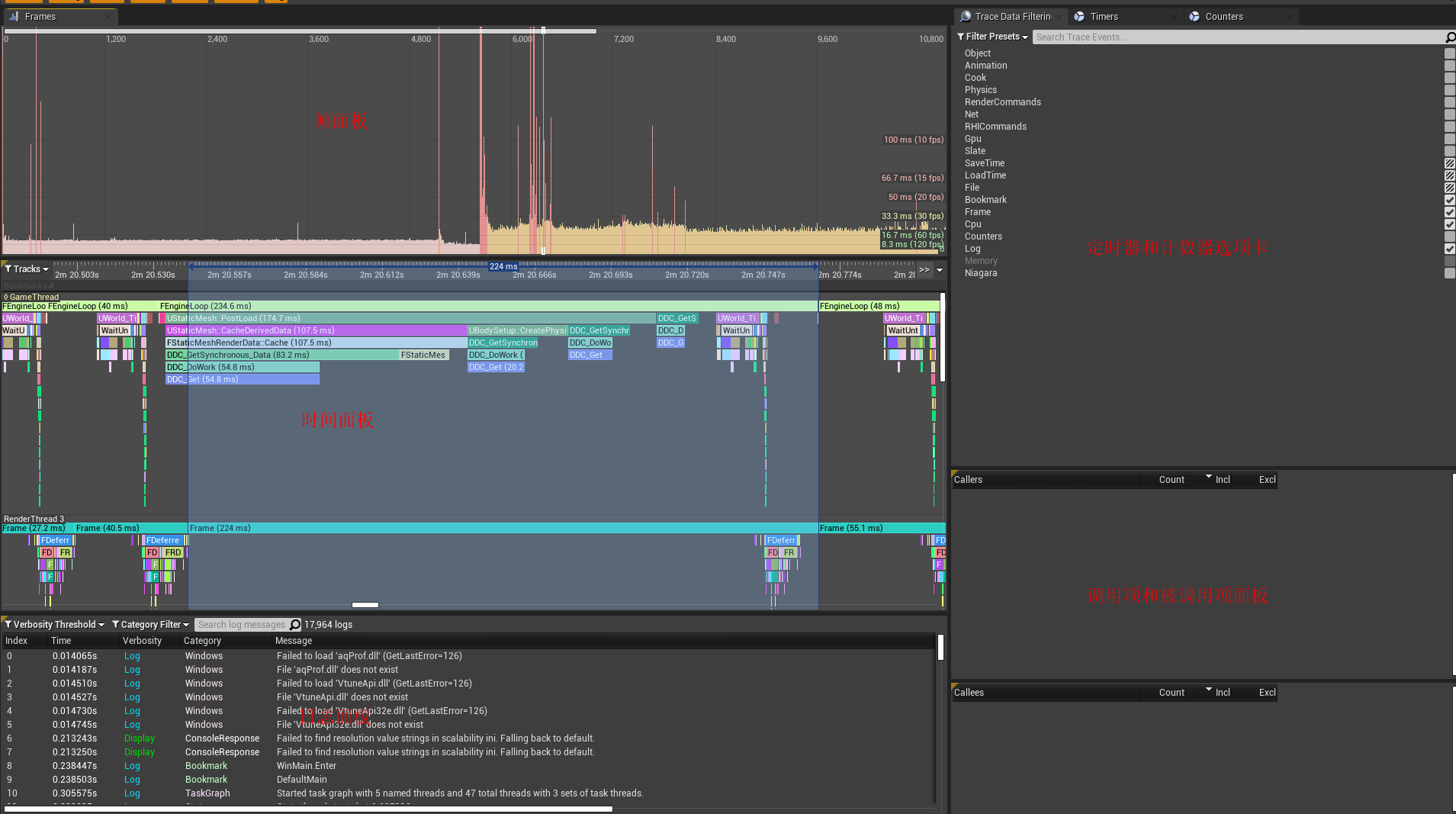
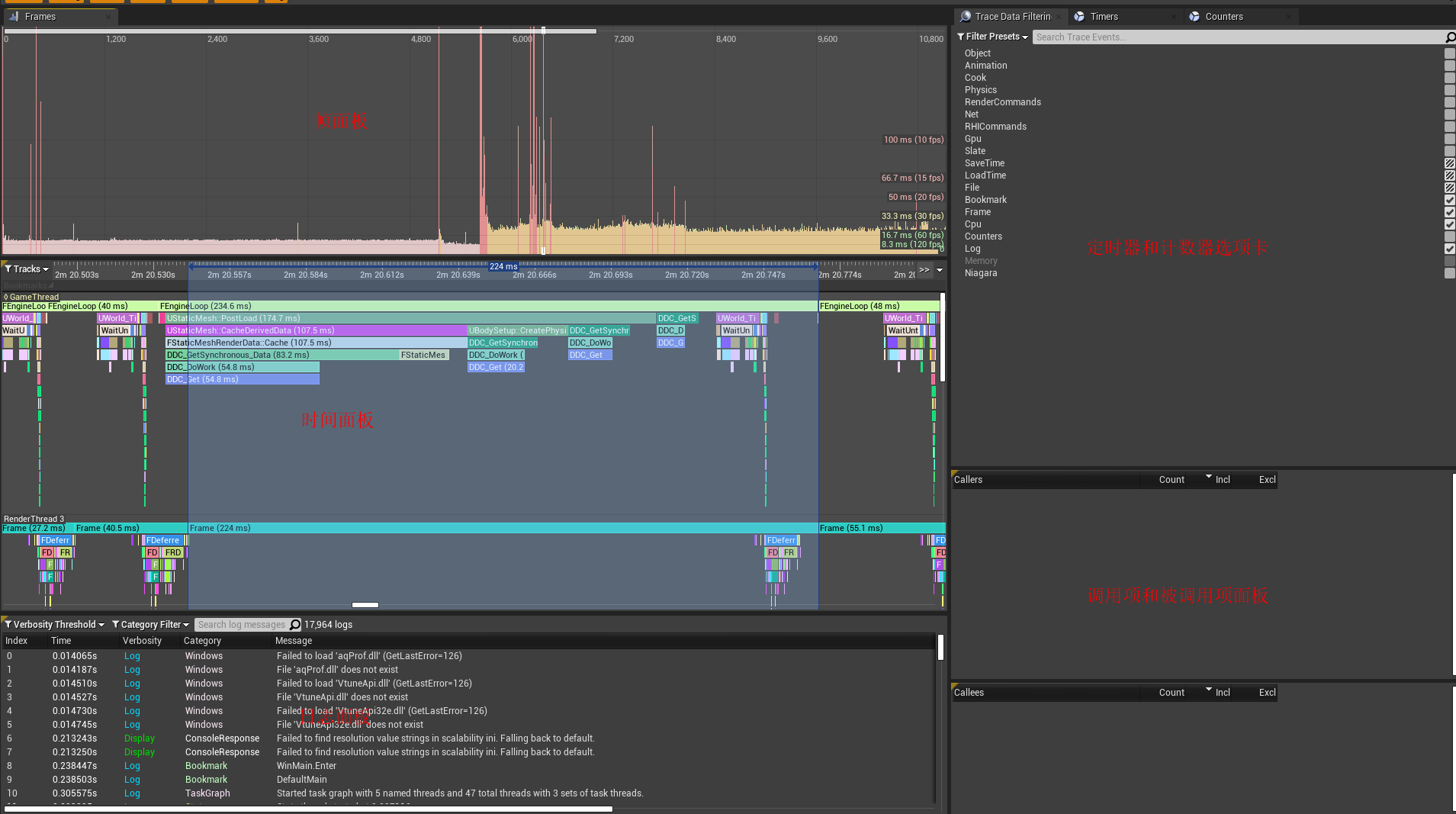
显示各个trace event的名字和开销情况                                -statnamedevents

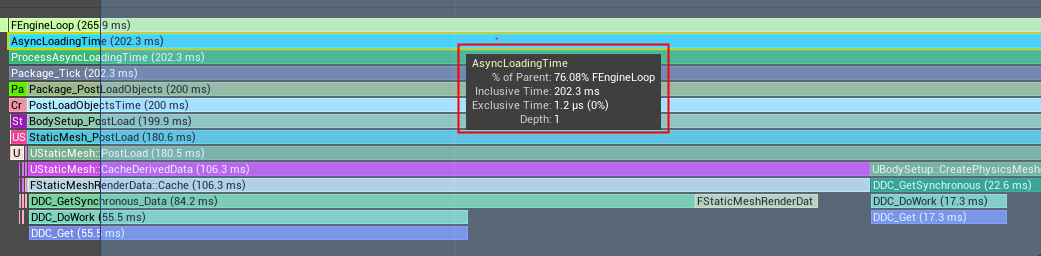
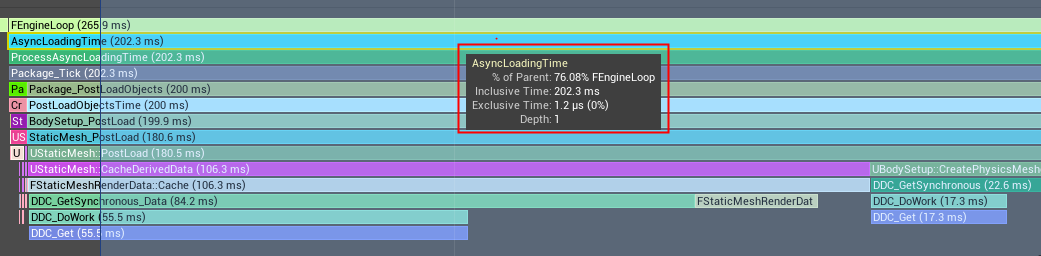
**DS启动**

也只需要加-trace参数即可

如："..\Engine\Binaries\Win64\UE4Editor-Win64-Debug.exe" D:\WatchMan\Client\WatchMan\WatchMan.uproject /Game/WatchMan/92\_Test/Maps/Training\_Course?port=5437 -server -log -trace=Object,Animation,Cook,Physics,RenderCommands,Net,GPU,LoadTime,SaveTime,File,Bookmark,Frame,CPU,Counters,Log

**面板结构**





黄色标：事件名

% of Parent：消耗时间占Caller的百分比

Inclusive Time:包含所有Calles的时间

Exclusive Time:去除掉所有Calles的时间

**AdditionalFunction**

线程置顶操作：右键某个线程标签，置顶

FrameStack：开启后可以让GameFrame和RenderFrame边界更加清晰

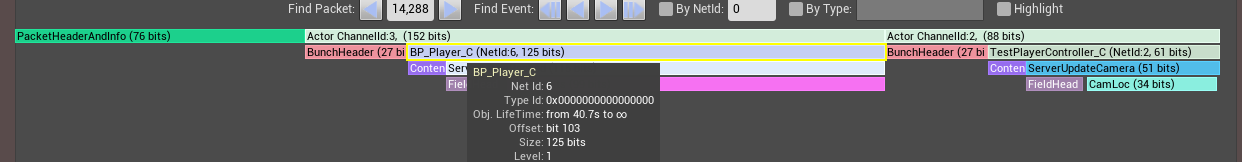
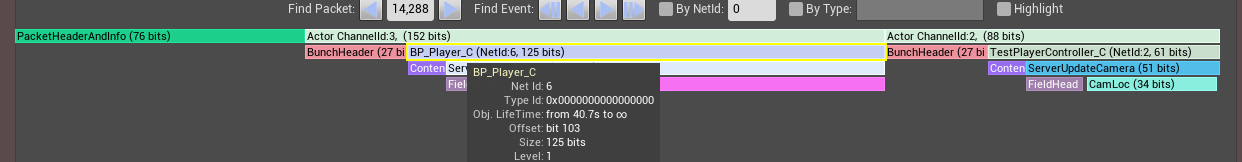
**卡顿分析**

1，LiveTime下观察帧面板，如果有一条线特别长，可以进行分析

NetWorkInsights

为啥会启动两个窗口？

满足并行



主要是这个窗口，代表一个包体信息，可以看到多个Bunch以及每一个Bunch的大小

NetId:GUID

可以用网络延迟等cmd命令做测试

MemoryInsights

内存变化监测

可结合其他Track使用，从而可以看到由事件调用引起的内存改变

AnimationInsights

可以查看动画的混合状态