# UE4源码分析

Uureal Engine是一款非常著名的游戏引擎，UE4已经在Github中开源。

【UE4源码仓库】<https://github.com/EpicGames/UnrealEngine>

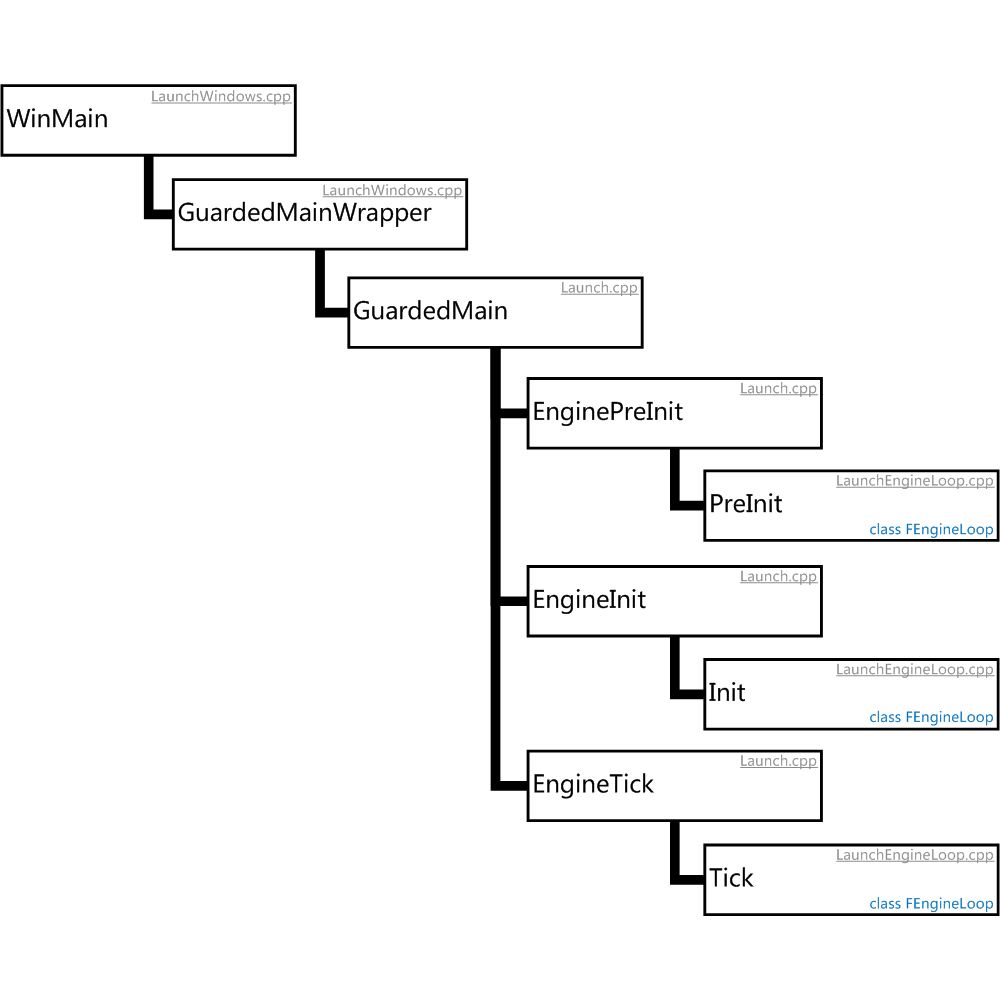
下载到本地后无需配置环境，直接使用VS2013就可以编译运行了。

## 运行调用分析

UE4的代码量很恐怖，想直接一个一个看.H文件与.CPP文件就了解UE4的架构显然是不现实的，为此我决定以UE4 BasicSample的Main函数为切入点，通过分析UE4在运行的过程中涉及到了哪些模块的调用，从而对它的架构建立一个初步印象。

首先在BasicSample的解决方案中搜索WinMain函数，再以它为入口点分析引擎中比较重要的函数调用：

### UE4进入主循环的启动顺序



从上图中可以看到UE4对WinMain函数进行了两层包裹。

**引擎初始化**

UE4将引擎初始化过程分为了两步：预初始化**PreInit**与初始化**Init**。

在预初始化过程中，主要执行的工作是根据应用程序传入的命令行（CommandLine）参数对引擎的各个部分进行配置：

* 设置字符编码格式
* 设置工作目录
* 获取日志控制台（Log Console）
* 初始化文件管理器
* 设置线程关联掩码？
* 设置游戏/渲染线程优先级？
* 配置基准测试（Benchmarking）
* 验证项目文件
* 加载核心组件（Core Modules）
* 初始化渲染参数缓存
* 初始化游戏物理？
* 其他工作？

在初始化函数**Init**中，引擎做了以下工作：

* 定义**ScopeSlowTask**？

FScopedSlowTask SlowTask(100);

SlowTask.EnterProgressFrame(10);

* 确定是否使用引擎的变体版本（如带有Unreal Editor的版本），并创建引擎对象（UEngine类型）
* 将载入界面传递给游戏本体？显示载入界面？

GetMoviePlayer()->PassLoadingScreenWindowBackToGame();

* 分析命令行参数？

GEngine->ParseCommandline

* 初始化计时？

InitTime();

* 初始化引擎对象

GEngine->Init(this);

* 设置内存警告句柄？

FPlatformMisc::SetMemoryWarningHandler(EngineMemoryWarningHandler);

* 初始化子系统？

FURL::StaticInit();

FLinkerLoad::StaticInit(UTexture2D::StaticClass());

* 检查命令行中的默认字符串映射？
* 设置运行时平均刷新时间（每两帧之间的时间间隔）

InitializeRunningAverageDeltaTime();

* 将引擎对象加入到根集合（root set）中，以防被垃圾回收机制删除
* 如果存在的话，初始化头戴可视设备（HMD）？

InitializeHMDDevice();

* 禁止屏幕保存器？

EnableScreenSaver( false );

* 如果应用程序不是一个专用服务器或命令行应用，则获取当前的石板程序（Slate Application，我的理解是可由开发者拓展的模板应用），并初始化程序声音模块。

CurrentSlateApp.InitializeSound(TSharedRef<FSlateSoundDevice>(new FSlateSoundDevice()));

* 指定略缩图压缩器（用于图片压缩/解压缩）

FObjectThumbnail::SetThumbnailCompressor( new FPNGThumbnailCompressor() );

* 这个函数的作用未知？

LoadObject<UClass>(UEngine::StaticClass()->GetOuter(), \*UEngine::StaticClass()->GetName(), NULL, LOAD\_Quiet|LOAD\_NoWarn, NULL );

* 加载引擎配置文件

LoadConfig();

* 根据配置文件设置进程的内存限制

SetConfiguredProcessLimits();

* 设置选中对象的材质与边框颜色为默认值

bIsOverridingSelectedColor = false;

SelectedMaterialColor = DefaultSelectedMaterialColor;

SelectionOutlineColor = DefaultSelectedMaterialColor;

* 初始化对象引用？

InitializeObjectReferences();

* 根据配置文件设置是否在屏幕中显示Debug信息
* 设置脚本的最大循环迭代次数

FBlueprintCoreDelegates::SetScriptMaximumLoopIterations(GEngine->MaximumLoopIterationCount);

* 初始化音频设备管理器

InitializeAudioDeviceManager();

* 如果不是ShippingBuild模式或TestBuild模式：
  + 获取命令行参数中的EXEC命令
  + 获取命令行参数中的ExecCmds命令
  + 获取命令行参数中的vsync命令
  + 获取命令行参数中的novsync命令
* 获取派生数据缓存

GetDerivedDataCache()

GetDerivedDataCacheRef().NotifyBootComplete();

* 删除可能存在的残余崩溃记录，防止不能访问该模块。

FString CrashVideoPath = FPaths::GameLogDir() + TEXT("CrashVideo.avi");

IFileManager::Get().Delete(\*CrashVideoPath);

* 注册接收漫游与网络错误的广播

OnTravelFailure().AddUObject(this, &UEngine::HandleTravelFailure);

OnNetworkFailure().AddUObject(this, &UEngine::HandleNetworkFailure);

* 输出纹理数据流是否可用的信息

UE\_LOG(LogInit, Log, TEXT("Texture streaming: %s"), IStreamingManager::Get().IsTextureStreamingEnabled() ? TEXT("Enabled") : TEXT("Disabled") );

* 配置在线子系统（Online Subsystem，一系列支持与web层服务通信的接口）？
* 初始化可视化系统缓存数据？

GetBufferVisualizationData().Initialize();

* 初始化引擎分析提供器？

FEngineAnalytics::Initialize();

* 注册屏幕捕捉工具？

HandleScreenshotCapturedDelegateHandle = UGameViewportClient::OnScreenshotCaptured().AddUObject(this, &UEngine::HandleScreenshotCaptured);

* 加载流暂停渲染模块（streaming pause rendering module）？

FModuleManager::LoadModulePtr<IModuleInterface>(TEXT("StreamingPauseRendering"));

* 将各个状态加入引擎状态列表？

下面是其中的一个例子：

EngineStats.Add(FEngineStatFuncs(TEXT("STAT\_Version"), TEXT("STATCAT\_Engine"), FText::GetEmpty(), &UEngine::RenderStatVersion, NULL, true));

* 对监听器发送每个状态更新的广播
* 记录对头戴可视设备的分析？

RecordHMDAnalytics();

* 加载晚于引擎初始化的模块（post-engine init modules）

IProjectManager::Get().LoadModulesForProject(ELoadingPhase::PostEngineInit)

IPluginManager::Get().LoadModulesForEnabledPlugins(ELoadingPhase::PostEngineInit)

* 等待载入画面与片头电影播放完毕

GetMoviePlayer()->WaitForMovieToFinish();

* 加载自动化工作模块（automation worker）？

FModuleManager::Get().LoadModule("AutomationWorker");

* 启动游戏渲染？

FViewport::SetGameRenderingEnabled(true, 3);

* 初始化硬件检查？

GEngine->InitHardwareSurvey();

* 这个函数的作用暂时不太清楚？

FCoreDelegates::StarvedGameLoop.BindStatic(&GameLoopIsStarved);

### 引擎主循环

UE4的循环主体是在FEngineLoop类的Tick函数中实现的。它主要执行了以下工作：

* 检查电影状态，如果电影未结束则不执行帧渲染。
* 调用监听控制台变量改变的回调函数

IConsoleManager::Get().CallAllConsoleVariableSinks();

* 刷新debug输出

GLog->FlushThreadedLogs();

* 如果当前帧数超过了基准测试模式中限定的最大值，则退出程序
* 更新程序中记录的当前时间、距离上一帧的时间以及等待以满足最大帧率的时间

GEngine->UpdateTimeAndHandleMaxTickRate();

GEngine->TickFPSChart(FApp::GetDeltaTime());

* 更新平台内存及内存分配器的状态

FPlatformMemory::UpdateStats();

GMalloc->UpdateStats();

* 为当前帧更新状态

FStats::AdvanceFrame( false, FStats::FOnAdvanceRenderingThreadStats::CreateStatic( &AdvanceRenderingThreadStatsGT ) );

* 计算平均FPS

CalculateFPSTimings();

* 设置需要在渲染结束后更新的资源

FDeferredUpdateResource::ResetNeedsUpdate();

FlushPendingDeleteRHIResources\_RenderThread();

* 如果启用了闲置模式，则完全中断当前线程，睡眠0.1秒
* 获取输入设备状态，并处理用户输入操作。

SlateApp.PollGameDeviceState();

SlateApp.FinishedInputThisFrame();

* 调用UEngine对象的Tick函数，对引擎进行更新

GEngine->Tick( FApp::GetDeltaTime(), bIdleMode );

* 等待电影播放完毕

GetMoviePlayer()->WaitForMovieToFinish();

* 处理着色器的异步编译结果

GShaderCompilingManager->ProcessAsyncResults(true, false);

* 将距离区域异步队列（Distance Field Async Queue）中完成的任务应用到场景中

GDistanceFieldAsyncQueue->ProcessAsyncTasks();

* 处理在一个世界时钟周期中积累的石板操作（slate operations）

ProcessLocalPlayerSlateOperations();

* 执行石板应用程序的时钟周期函数

FSlateApplication::Get().Tick();

* 清理正在等待中的属性组（stat group，统计组？）通知

ClearPendingStatGroups();

* 如果当前编译模式包含编辑器，则执行自动化控制（AutomationController）模组的Tick函数

FModuleManager::GetModuleChecked<IAutomationControllerModule>(AutomationController).Tick();

* 根据编译设置执行自动化工作器（AutomationWorker）模组的Tick函数

FModuleManager::GetModuleChecked<IAutomationWorkerModule>(AutomationWorkerModuleName).Tick();

* 啊
* 啊
* 啊
* A
* 啊