2017-7-11

|  |
| --- |
| KAYAKEGAME | liu lian you |



|  |  |
| --- | --- |
| EPIC | UMG |

**前言**

这个是我在学习UNREAL ENGINE 4 的学后笔记，因此有些东西会不全面，也比较杂乱。里面有些内容在官方网站上也提到过，或者具体讲过，但是官方网站里面的内容一来涉及的某一个功能的具体代码框架没有详细的说明，而来网站上的内容实在是有点混乱，有些东西你觉得你在哪里看到过，但是反过来找就是找不到了。雾里看花的感觉真的是非常非常不爽。

因为我是一名程序员，所以我主要写下来的东西是关于代码方面的内容，而且UMG这个玩意儿的编辑器里的操作也就是那么几样，最多也就是锚点的设置会让人感觉云里雾里的感觉。

我没有用过UE3的UISCENE 功能，但是跟UE3 的SCALEFORM 相比， 我觉得最大的优势就是把开发规模缩小了，用SCALEFORM 需要AS 程序员，FLASH 美术，PHOTOSHOP 美术， UNREAL SCRIPT 程序员， C++ CLIENT 程序员，C++ SERVER 程序员。而UMG 只需要 UI 美术， UI 策划， C++ CLIENT 程序员， C++ SERVER 程序员。开发结构变小能够使得工程蠕变的效果变小。代码结构的纯洁性能够保障。

当然如果硬要说那也可以2个人包揽所有活：美术和程序。但是这么做的结果就是程序会非常非常的累，然后非常非常的反感做UI。太多的需要堆时间修改的细节会耗掉一个人的雄心壮志。这样做出来的UI只能说功能有了，但是外观美学只能说呵呵了。毕竟一个程序员还是感觉未知的代码更有雌性的魅力。

# 概述：

UMG 是UE4的制作游戏内的UI的制作工具，它的基础是用的UE4的SLATE 框架。

在UE4 引擎编辑器里所有的UI界面都是用SLATE框架来实现的。而UE4 只是对SLATE 再次做了一次封装，之所以这么做的原因有下面几点：

* UMG通过和 BLUEPRINT相结合能够做到所见即所得的开发模式，而SLATE 只能在代码层面来做UI,因此只能在运行的时候看到效果
* UMG 通过对SLATE 的再次封装隐藏了一些必要的但是繁琐的细节，他们之间的关系就我感觉就像 WIN32 API 和 MFC 的关系一样。
* UMG 能够让美术和策划更加方便的制作UI，这样就能分担程序员的很多任务。让每一个人都能做自己喜欢的事情。

因为 BLUEPRINT 是UMG 的好伙伴，SLATE 是 UMG 的基础，我会在后面分为3个大章节来大致描述一下：

* [BLUEPRINT](#_BLUEPRINT): 游戏里的策划的主要的GAMEPLAY 制作场地
* SLATE： 负责了所有的EDITOR 的窗口。
* UMG： 官方推荐唯一指定游戏内UI制作工具

我没有做过UE3 的UISCENE 所以我没办法拿这两个做比较，我也不知道是不是UMG 只是UMG 的升级，但是我觉得应该不是升级，而是直接重做的，如果有哪位大神对UISCENE 代码熟悉的，希望能够发我邮件让小弟膜拜膜拜。

上面3个章节将要主要讲述各个功能的实现细节。在最后我应该还需要添加两个章节：

* UE4 的内存管理
* UE4 的RENDER PIPELINE

# BLUEPRINT

1. 概述：

BLUEPRINT 是UE4 主要宣传的卖点之一，它的意图是让没有代码经

验的非程序员也能够编写GAMEPLAY 或者说是简单的编程。这种可视化编程我不知道非程序员感觉怎么样，就在我角度来看，这种编程完全取代不了文本代码的功能。至于那些叫嚣尘上的游戏程序员的末日完全不可信。

BLUEPRINT 既可以说是脚本语言，又可以说仅仅是META DATA.

说是脚本语言那因为他的运行机制很大程度上继承了UE3 里的UNREAL SCRIPT， KISMET

又是META DATA是因为在UE4 中C++代码中支持BLUEPRINT 其实就是把那些宏UFUNCTION, UCLASS, UENUM, UPRIORITY, UPARAMT里面的数据翻译成C++代码。所以要坚信一点：BLUEPRINT 不是一门新开发的语言，它和C#里面的类似，但是完全不同。整个UE4 代码编译器用的还是GC。和C++代码一个流程

它的优点有下面几个：

* 提供给非程序员一个操作游戏物件的方法。
* 和UE4 的核心代码完全融合在一起。在游戏运行的时候可以省略很多非必要的资源读取。非常多的资源已经完全编译到map里面去了。
* 操作简单方便，可视化代码结构。
* 因为是和UE4无缝连接，所以在使用BLUEPRINT的时候可以省去非常多的LOAD操作，就比如用UMG来创建UI，官方文档里的方法来创建UI，那么WIDGETBLUEPRINT 在游戏运行的时候是不会去LOAD 的，因为这个已经整合到Map里面去了。
* 在用BLUEPRINT 来做一些场景中的GAMEPLAY 可以让策划非常方便的重用

当然它还有很多其他优点但是缺点也有不少：

* 它里面的所有的代码逻辑最终都会序列化2进制文件当中去，这样就导致在多个版本中，不能分辨有哪些修改，不利于MERGE 或者区分版本区别
* BLUEPRINT是用的图形化编程，这样就导致一旦逻辑复杂的时候整个界面会非常的混乱。
* BLUEPRINT 的IDE使用我觉得不方便，没有VS或者其他的IDE 使用便捷。在做DEBUG的时候不方便。
* BLUEPRINT 使用了UE3 里面的Unreal Script 的一部分流程，这个虚拟器就会导致性能的下降。
* BLUEPRINT 可以说是一门语言，就像FLASH里面的Action Script差不多，因为开发使用的时间不长，所以导致会有很多问题。使用有非常多的不方便。

就凭BLUEPRINT里的缺点，我觉得作为一个程序员就不会非常喜欢BLUEPRINT了，因为我们有更好，更方便，更快捷，更舒心，更美观，更时尚，更有成就感的选择 C++ CODE!

1. 实现细节：

我只知道大体的流程，具体的元数据的翻译没有去细看，因为语言的解析是非常非常繁琐饶人的一件事情

Blueprint 在UE4里就是当作一个普通Asset，这样每次修改，你都会编译并且在编译的时候任何和它相关的都会做更新，在代码里他会调用KISCOMPILE类来做编译，UE3里就是大量用的KIS,所以可以把BLUEPRINT当成UE3里的KISMET的一个重大提升。

UE4 对Blueprint的支持是通过 UNREALHEADTOOL 工具来实现的，可以在代码工程里找到这个工程。

UNREALHEADTOOL 会把所有的元数据翻译成C++ 文件，这些文件是动态生成的所以是不在工程目录下。有的时候你全工程搜索一个函数的实现的时候没有找到，但是编译仍然通过，那你就可以去IMMEDIATE 文件夹去找代码定义。这些额外的C++文件，在UE4 BUILD TOOL 编译的时候会整合到工程里面去。

我现在还不知道到底是Unreal Build Tool 会把各个平台的文件分类，还是说Unreal Head Tool 来做这个事情的，反正我发现有些平台相关的代码是在编译的时候决定的,就像VS工程里的exclude 功能

UE4 的编译流程是先启动UNREALBUILDTOOL 然后再启动UNREALHEADTOOL生成中间零时文件，然后再把同一个工程下面的文件整合到几个大的文件，并且会动态生成与编译头文件，然后再是收集各种编属性，在UNREALBUILDTOOL 的编译时多线程编译，一般比CPU 核心多1/2的线程数，这就会导致在编译的时候你的电脑会卡死。

1. **SLATE**
2. 概述：

SLATE 是UE4 的核心工程之一，因为所有UE4 界面都是通过SLATE 来做的，就连我们下面的UMG 只是在它的基础上加了一个壳子而已。

SLATE 就相当于自己实现了一套界面交互框架，这个框架我觉得非常棒，给我提供一整套用户交互解决方案。

SLATE 这个框架如果你看代码可以看到，它的代码就算脱离UE4，只要改动很小的地放就可以单独运行的。它没有大量运用UOBJECT 来作为类的基类，所有的类都是自己的定义，只是继承了UE4 的智能引用指针，在SLATE里的内存我现在看到他用的最多的就是SNEW 这个宏，最后还是用的C++原生的new来实现的内存的生成，我目前内存看到的还不是特别多，不知道UE4的内存到底是怎么样的，但是就目前看到SLATE 好像没有单独的去做内存管理。

SLATE 的代码我看到几天发现代码的机构规范很清楚，脉络不复杂。所有的功能都是分开的，至于重复的代码在不同地方出现，有但是不多。在看SLATE 代码的时候能够学到很多的C++11 之后的特性，和一些小小的TRICK。总之我感觉到看代码非常舒服的。

SLATE 支持部分序列化的，尤其是在MainFrameModuel 里可以看到主框架的LAYOUT 是读取EngineLayOut.ini文件来实现的，它的序列化石把信息按照JSON的格式来存储的。

SLATE 里的所有的Widget 在内存中都是单独一份的，在Widget之间是不纯在直接的拷贝，或者赋值，Widget之间的通信都是透过指针，或者引用（UE4 里是通过TSharePtr,TShareRef这些引用计数指针做的。）这样做的显然好处太多了。

正如上面说的优点，SLATE 还是会有很多缺点的：

（这里我希望我以后能增加的更过，因为这个缺点发现的越多，bug发现的越多，我的能力就越强，我的理解就越深入，我的未来就越明亮,想想有点小激动）

* SLATE 好像天生就支持鼠标的操作，这个如果来做主机平台的，或者手机平台的话我觉得就是对性能非常浪费。内存和CPU的双重浪费，内存至少要增加1MB，CPU就更不用说了，那么多检索实在是太耗了的
* 用SLATE 来做UI 会非常的 非常的麻烦，所有的UI 逻辑，资源配置都是写在代码里的，所以这就导致代码耦合性强，数据和代码不能有效的分开。同一个UI只能一个人做。

1. 实现细节：
2. 对象的生成：

SLATE 的所有对象生成都是通过**SNEW** 或者**SASIGNNEW**来做的，通过这两个宏可以看到使用者两个宏会让代码简洁方便易于管理，可以直接通过连的....来实现对象的初始化,非常的酷。

这两个宏里面包含了内存的分配的，在里面有两种内存的分配：

* 直接调用没有重载过的new,
* 还有就是通过TSharePtr

就我目前看到的情况大部分都是通过TSharePtr 来分配内存的。之所以有这两个我认为是因为SLATE本来就是一个独立的功能，如果没有UE4，它也可以放到任何一个项目中去。它和UE4 的关系是标准的聚合的关系

1. SLATE 框架概述：

在UE4 里 SLATE 分为3个主要组成部分和2个应用部分。

组成部分分为：

* SLATE

这个部分聚焦的是一个个可以实现显示的最终组件，它是对SLATE CORE的最初级的整合应用，当然也包括很组件之间的组织形式，比如SButton, SText 等等一系列可以直接拿来用的组件就在这里定义

* SLATE CORE

这个部分聚焦的是最基本的原始功能。它支撑着SLATE.比如SLATE 里的SText里面的FONT 就在这里管理。获得鼠标指定的widget功能等等那些基本的原始的都在这里实现

* SLATERHIRENDERER

这个看名字就知道，这里定义了SLATE里所有的纹理，定点，着色， 材质的行为。

应用部分分为：

* MAINFRAMEMODUEL

这个就是UE4 Editor 的主要窗口框架，它是对SLATE 的一个完全应用。它通过组合调用SLATE里的功能实现了UE4 里所有的界面。

* UMG

这个是我们游戏里UI的实现与生产工具，它基于SLATE 添加了可供用户可视化操作的接口。

1. SLATE CORE

一般上来说我们把UI 拆分为5个互不相干的单独部分：

* + Input：这里就包含了所有UI对鼠标的响应的方式和结果，比如说鼠标在屏幕上移动我们该怎么去判定哪一个UI元件被选中了。被选中UI应该怎么样去触发事件，等等这些都包含在Input里面
  + Layout：这个包含了每一个UI元素在屏幕上的位置，它可以是各个元件间的相对位置（包括规则），也可以是屏幕的绝对位置
  + Sound: UI交互肯定是有声音反馈的，不然你不知道当前鼠标点击或者其他的事件有没有触发。
  + Material:这里的材质是一个泛指，所有的用来表现一个元件的属性都可以算是材质，比如说贴图，法线，拉伸变形规则，颜色等等用来表现一个元件的属性，在这个里面也可以有坐标，这里的坐标和Layout里的坐标是两个概览，其实就跟本地坐标系和世界坐标系的关系，Material里的坐标是可以当成本地坐标而Layout 里的坐标可以当成全局坐标。
  + Animation：严格说来动画算是Layout 和 Material 的统合，因为在UI里所谓的动画归根结底也就是位置，色彩（包括贴图），形状上在时间轴上的变化。我在这里单独列出来是因为做一个漂亮动画的时间就和其他模块一样或者更多。
  + Font：之所以把这个单独列出来，因为现在很多的文字的字元都是用的第三方资源，比如TTF，OTF，这些第三方资源专业性高，可靠性，可扩展性，复杂度都非常高。很少会有自己去做一张贴图把所有的字元罗列在一起。

用第三方的资源优点很多，缺点就是版权费会很多，如果单独做出来一个用第三方工具做的字体成本非常大，TTF的制作非常专业，一个单纯的美术是做不来的。

自己做一个文字贴图，虽然简单但是在多分辨率下会出现字体的拉伸从而模糊，有毛边。这个是很大的问题。而且像汉子，棒子语常用字特别多的。这样就导致贴图死命的大。

在UE4 里的SLATE CORE 里，作者对UI的分类有些不同但是大体上是相同的。

LayOut：就是工程里的Layout文件夹，在这个文件夹里包含了

1. **UMG**
2. 概述：
3. 实现细节:
4. **UE4 MEMORY MANAGE**
5. **UE4 RENDER PIPELINE**
6. **番外**
7. DELEGATE
8. 概述:

在SLATE 里的所有的事件函数都是通过DELEGATE来实现的，不同于UE3的DELEGATE UE4里的DELEGATE 实现是通过模板实现。在看实现代码可以加深对模板的理解。随着UE4 版本的不断升级DELEGATE 的有些实现细节会一直改变，但是基本的设计思路还是一直沿用的。

其实DELLEGATE 就是用的函数指针来实现的，每当你通过宏自定义一个DELEGATE 类型，通过这个类型定义一个DELEGATE 对象，DELETEGATE 的内存是用UE4 的内存管理的。然后在使用的地方只需要通过特定的函数，参数为函数作用域，函数地址就可以生成一个有功能的托管函数。

1. 实现细节：
2. 文件布局：

UE4 里的DELEGATE 分为两个部分：单通道和多通道

UE4 的Unicast DELEGATE 函数实现的代码主要集中在：

* DelegateSignatureImpl.inl

这个里面就包含了所有的Delegate的实际对外接口比如CreateRaw, StaticCreate 等等。并且在这里会调用DelegateInstanceInterface 来实现Create和Execute 功能

* DelegateInstanceInterface.h

这个文件是用来支持DelegateSignatureImpl.inl里面的Execute功能

* DelegateCombinations.h

这个文件用来定义一些组合宏来让其他模块能够更好的调用 DELEGATE 函数。

* DelegateInstanceImp.h

实际上的一个Delegate的数据结构，以及一些相关的行为。

* Delegate.h

这个文件算是DelegateCombinations.h 和 DelegateSignatureImpl.inl的中间文件，它会将所有在DelegateCombinations.h的宏解释成DelegateSignatureImpl.ini里面的具体类或者实现

* DelegateBase.h

实际进行内存分配的地方。

1. 内存分配：

DELEGATE 的内存是用UE4统一处理的，通过FDefaultAllocator 来申请内存，在做内存分配的时候我看到一个小花招：通过先申请一个栈内存，他会临时生成一个TBaseDelegate 对象，这个对象里面的所有值都是没有初始化的，除了DelegateSize,它只有一个作用用来通过placement new 来调用UE4 的内存管理工具分配内存。

在通过DelegateBase.h文件的阅读可以清楚的看到Delegate 是16位对齐，并且默认的一个Delegate 对象大小也就是32位。