






虚幻4高级程序-第三阶段01

UNREAL 4 HIGH INTEGRATION PROGRAM DEVELOPMENT ENGINEER



01 工程目录结构



 .vs	2018/12/6 下午 ...	文件夹	
 Binaries	2018/10/9 下午 ...	文件夹	
 Config	2018/10/22 下午...	文件夹	
 Content	2018/12/14 下午...	文件夹	
 Intermediate	2019/3/18 下午 ...	文件夹	
 Saved	2018/12/14 下午...	文件夹	
 Source	2018/10/9 下午 ...	文件夹	
 UECPP.sln	2018/12/6 下午 ...	Microsoft Visual...	9 KB
 UECPP.uproject	2018/12/6 下午 ...	Unreal Engine Pr...	1 KB
 UECPP.VC.db	2018/12/14 下午...	Data Base File	1,038,836...

1. Binaries:存放编译生成的结果二进制文件。该目录可以gitignore,反正每次都会生成。
2. Config:配置文件。
3. Content:平常最常用到,所有的资源和蓝图等都放在该目录里。
4. DerivedDataCache: “DDC”, 存储着引擎针对平台特化后的资源版本。比如同一个图片,针对不同的平台有不同的适合格式,这个时候就可以在不动原始的uasset的基础上,比较轻易的再生成不同格式资源版本。gitignore。
5. Intermediate: 中间文件 (gitignore), 存放着一些临时生成的文件。有:
 1. Build的中间文件, .obj和预编译头等
 2. UHT预处理生成的.generated.h/.cpp文件
 3. VS.vcxproj项目文件, 可通过.uproject文件生成编译生成的Shader文件。
 4. AssetRegistryCache: Asset Registry系统的缓存文件, Asset Registry可以简单理解为一个索引了所有uasset资源头信息的注册表。CachedAssetRegistry.bin文件也是如此。
6. Saved: 存储自动保存文件, 其他配置文件, 日志文件, 引擎崩溃日志, 硬件信息, 烘焙信息数据等。Gitignore
7. Source: 代码文件。

UnrealBuildTool (UBT) 是一个自定义工具，用于管理跨各种构建配置构建虚幻引擎4 (UE4) 源代码的过程。

UE4分为许多模块。每个模块都有一个build.cs控制其构建方式的文件，包括用于定义模块依赖关系，附加库，包含路径等的选项。默认情况下，这些模块被编译为DLL并由单个可执行文件加载。可以选择在文件中构建单个可执行BuildConfiguration.cs文件。

重要的是要了解构建过程独立于开发环境的任何项目文件执行，例如 .sln 或 .vcproj 文件（对于Visual Studio）。这些文件对于编辑目的很有用，因此提供了一个工具来动态生成它们（基于项目目录树的内容）。

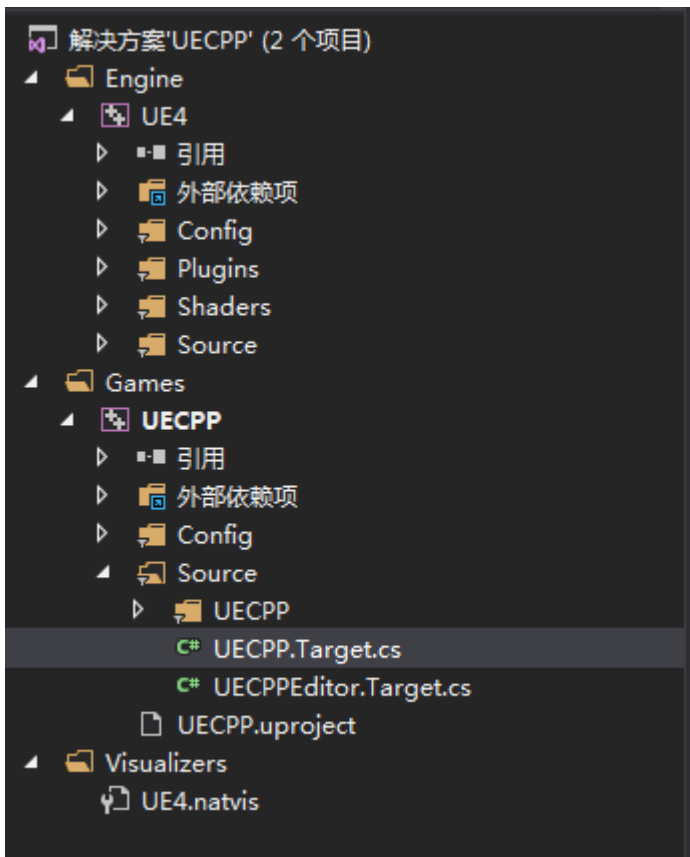
UnrealHeaderTool (UHT) 是一个支持UObject系统的自定义解析和代码生成工具。代码编译分两个阶段进行：

- 1.调用UHT，它解析与虚幻相关的类元数据的C++头，并生成自定义代码以实现各种与UObject相关的功能。
- 2.调用普通的C++编译器来编译结果。

编译时，任一工具都可能发出错误，需要仔细查看。

02 项目目录结构

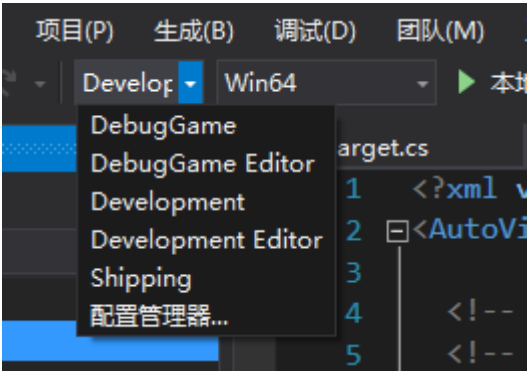




1. Engine 引擎源码文件（只读），虚幻是开源项目我们可以直接在工程中看到引擎源码但是无法修改。如需修改请下载Git源码工程
2. Games 项目工程文件（主要编写逻辑文件），我们的代码需要在此工程中编写。虚幻中采用了编译模块方式进行引擎构建，所以对于引擎来说，我们编写的内容只是一个模块，模块会被动态编译为库文件，加载到引擎中使用。Target.cs文件就是模块配置文件
3. Visualizers虚幻4.21加入的文件夹，VS编辑器配置文件

03 编译类型





状态	Engine	Game	其他
Debug (调试)	Debug	Debug	必须在编辑器上加-debug参数才能反射查看代码更改
DebugGame (调试游戏)	Release	Debug	适合只调试游戏代码
Development (开发)	Release	Release	允许编辑器发射查看代码更改
Shipping (发行)	Release	Release	无控制台命令, 统计数据 and 性能分析
Test (测试)	Release	Release	启用了一些控制台命令, 统计数据 and 性能分析

目标	
空白	不带编辑器的一个独立可执行版本, 需要提前打包烘焙内容资源
Editor (编辑器)	直接在编辑器里打开游戏项目
Client (客户端)	多人联机项目, 生成客户端版本, 需要提供<Game>Client.Target.cs文件
Server (服务器)	多人联机项目, 生成服务器版本, 需要提供<Game>Server.Target.cs文件

04 命名规则



1. 模版类以T作为前缀，比如TArray,TMap,TSet
2. UObject派生类都以U前缀
3. AActor派生类都以A前缀
4. SWidget派生类都以S前缀
5. 全局对象使用G开头，如GEngine
6. 抽象接口以I前缀
7. 枚举以E开头
8. bool变量以b前缀，如bPendingDestruction
9. 其他的大部分以F开头，如FString,FName
10. typedef的以原型名前缀为准，如typedef TArray FArrayOfMyTypes;
11. 在编辑器里和C#里，类型名是去掉前缀过的
12. UHT在工作的时候需要你提供正确的前缀，所以虽然说是约定，但你也得必须遵守。（编译系统怎么用到那些前缀，后续再讨论）

UE遵循帕斯卡命名法则

类型	前缀	说明
Level/Map	L_	关卡
Blueprint	BP_	常规蓝图
Material	M_	材质
StaticMesh	S_	静态网格
Skeletal Mesh	SK_	骨架网格
Texture	T_	纹理
Particle System	PS_	粒子系统
Widget Blueprint	WBP_	组件蓝图

```
|-- 项目名称  
|   |-- Maps  
|   |-- Textures  
|   |-- Materials  
|   |-- Blueprints  
|   |-- Effects  
|   |-- Animations  
|   |-- Sounds  
|   |-- Sources
```

我们建议，在引擎根目录进行细致的命名分化。已达到高效管理资产的目的，命名应遵循清晰，准确，简短的描述分类内容。尽量不要使用模糊描述。项目名称作为最顶层文件夹名称

<https://github.com/uejoy/ue4-style-guide>

05 Actor相关












UE游戏世界中存在元素的根源，用来表示世界中的任何物体的高级抽象类，由Actor进行构建，Component进行行为组件，完成整个游戏世界的元素展示

Acotor的创建方式分为两种，最简单的方式直接在场景中编辑拖拽（静态创建），创建有引擎构建场景时进行创建，优势无需编码，更加直观简单，缺点可能会影响游戏启动速度，增加场景构建负担。第二种可以通过编码动态进行生成（动态创建），相对于前者复杂度上升，但是可控性更强，动态生成的Actor会持有有效的操作指针，我们可以根据实际情况进行生成，更加灵活。

在蓝图中我们已经学习了如何动态生成Actor，我们借助了SpawnActorOfClass。在C++中，我们需要通过UWorld指针进行创建。在UWorld类中，我们可以找到多个生成Actor的函数。对于重载函数，我们根据特定的情况进行选择即可！SpawnActor函数是个工厂函数。下面地址中可以查到UWorld的相关API

<http://api.unrealengine.com/INT/API/Runtime/Engine/Engine/UWorld/index.html>

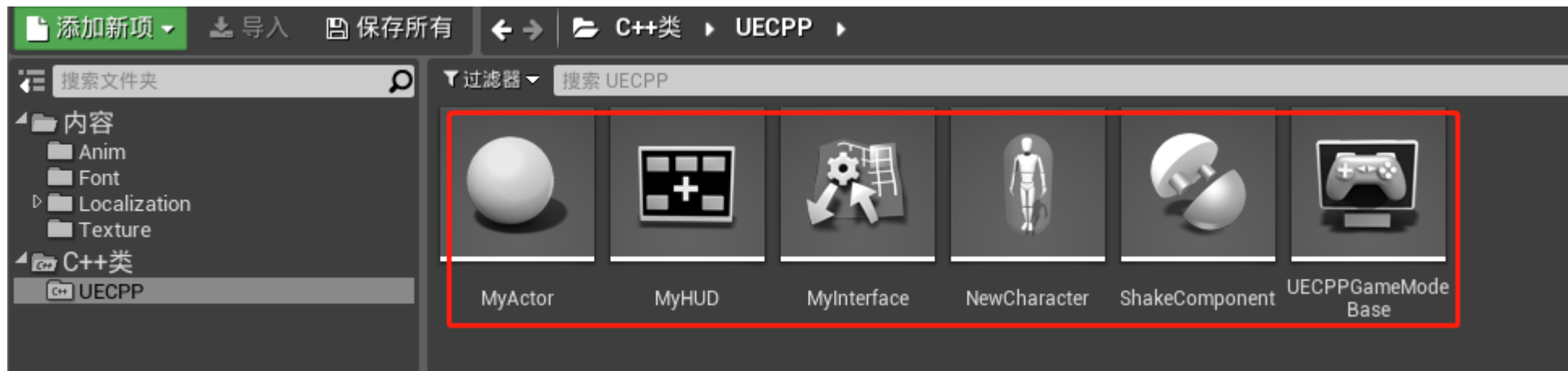
	<code>AActor *</code>	<code>SpawnActor</code> (UClass * Class, FTransform const* Transform, const FActorSpawnParameters & Spawn...)	Spawn Actors with given transform and SpawnParameters Actor that just spawned
	<code>AActor *</code>	<code>SpawnActor</code> (UClass * InClass, FVector const* Location, FRotator const* Rotation, const FActorSpawnParameters & Spawn...)	Spawn Actors with given transform and SpawnParameters Actor that just spawned
	<code>T *</code>	<code>SpawnActor</code> (UClass * Class, FTransform const& Transform, const FActorSpawnParameters & Spawn...)	Templated version of SpawnActor that allows you to specify whole Transform class type via parameter while the return type is a parent class of that type
	<code>T *</code>	<code>SpawnActor</code> (FVector const& Location, FRotator const& Rotation, const FActorSpawnParameters & Spawn...)	Templated version of SpawnActor that allows you to specify location and rotation in addition to class type via the template type
	<code>T *</code>	<code>SpawnActor</code> (UClass * Class, const FActorSpawnParameters & Spawn...)	Templated version of SpawnActor that allows you to specify the class type via parameter while the return type is a parent class of that type
	<code>T *</code>	<code>SpawnActor</code> (const FActorSpawnParameters & Spawn...)	Templated version of SpawnActor that allows you to specify a class type via the template type
	<code>T *</code>	<code>SpawnActor</code> (UClass * Class, FVector const& Location, FRotator const& Rotation, const FActorSpawnParameters & Spawn...)	Templated version of SpawnActor that allows you to specify the rotation and location in addition class type via parameter while the return type is a parent class of that type

首先我们需要获取到UWorld世界对象指针！对于每一个在场景中存在的对象，本身都具备获取UWorld指针的能力！我们只需要调用GetWorld函数即可获得UWorld对象指针。记得要引入头文件（UWorld本身头文件已经在父类中被包含，所以无需额外引入）

```
void AUECPPGameModeBase::BeginPlay()
{
    UWorld* world = GetWorld();
    world->SpawnActor
}
```

合成函数，获取一个UClass指针。旨在将操作类作为一个参数进行传递（传递模版）。虚幻中除了使用模版构建对象外，还增加了映射关系，可以将类作为对象构建依据，这可以方便的将类提供给蓝图使用。

注意：C++本身是编译型语言，我们在编辑器中看到的文件类并不是真正的C++类，只是编译后生成的映射文件。



Actor的创建分为三种方式，三种方式的创建生命均有细微差异，但是总体不大，基本分为一下流程

1. 构造函数调用
2. 初始化成员变量
3. 如有蓝图，则初始化蓝图数据
4. 构建组件
5. BeginPlay（标志着Actor被创建到世界当中）
6. Tick

Destroy函数

调用自身Destroy函数进行强制消亡操作

参数说明

bNetForce 是否强制网络同步删除

bShouldModifyLevel 主要是用来控制先删除actor再修改关卡，还是先修改关卡再删除actor，默认是true，即为先修改关卡，再删除actor（修改关卡即为把actor先移除出场景）

Actor被标记为等待销毁并从关卡的Actor阵列中移除。

SetLifeSpan函数

设置延时删除（单位秒）

Destroyed函数

当对象被删除时（非内存删除）进行回调操作

EndPlay函数

对象被彻底清除时回调，回调会进行删除类型通知

```
void AMyActor::EndPlay(const EEndPlayReason::Type EndPlayReason)
```

删除类型介绍

Destroyed 当actor或是component彻底被删除时（内存中）

LevelTransition 关卡切换时删除回调（非关卡流）

EndPlayInEditor 编辑器关闭时，回调通知

RemovedFromWorld 关卡流切换被释放时调用

Quit 游戏退出时被删除回调

06 屏幕日志输出



借助全局变量GEngine指针调用函数AddOnScreenDebugMessage即可完成屏幕输出

```
void AddOnScreenDebugMessage  
(  
    int32 Key,  
    float TimeToDisplay,  
    FColor DisplayColor,  
    const FString & DebugMessage,  
    bool bNewerOnTop,  
    const FVector2D & TextScale  
)
```

Key如果是-1则添加新的消息，不会覆盖旧有消息

如果不是-1则更新现有消息，效率更高

bNewerOnTop 当key为-1时有效，直接添加到队列最上层

虚幻中用来构建非对象型字符串的关键宏，构建结果是与平台无关的宽字符串（类似C语言字符串），借助对象FString带参构造函数TCHAR类型指针来构建FString对象字符串

```
// If we don't have a platform-specific define for the TEXT macro, define it now.
#if !defined(TEXT) && !UE_BUILD_DOCS
    #if PLATFORM_TCHAR_IS_CHAR16
        #define TEXT_PASTE(x) u ## x
    #else
        #define TEXT_PASTE(x) L ## x
    #endif
    #define TEXT(x) TEXT_PASTE(x)
#endif
```

07 控制台日志输出



通过使用宏UE_LOG进行控制台日志输出（日志会写入本地缓存）。构建日志需要传入三个参数。

1. 日志分类（可以自定义）
 1. 决定了内容输入到控制台时的分类项
2. 日志类型冗余度，分为
 1. Fatal（会终止进程）致命问题
 2. Error 错误问题（红色警告）
 3. Warning、Display、Log（较常用的日志分类项）
 4. Verbose（将日志详细信息记录到日志文档，但不向控制台输出）
 5. VeryVerbose（将日志详细信息记录到日志文档，但不向控制台输出）
3. 日志内容

08 自定义日志分类



分两步：声明和定义

1.声明自己日志分类

```
DECLARE_LOG_CATEGORY_EXTERN(CategoryName, DefaultVerbosity, CompileTimeVerbosity);
```

CategoryName 自定义日志分类名称 Log开头

DefaultVerbosity 日志默认级别，一般使用Log

CompileTimeVerbosity 日志编译级别 高于此级别的不会被编译 一般用All

注意：此操作需要在头文件中完成，并且只需要完成一次即可

此宏用来生成日志分类结构体对象，只需要在头文件中进行一次操作即可

```
DECLARE_LOG_CATEGORY_EXTERN(ZLog, Log, All);
```

2.定义日志分类

```
DEFINE_LOG_CATEGORY(CategoryName);
```

注意：此操作必须在CPP文件中进行，只需要进行一次定义即可

```
DEFINE_LOG_CATEGORY(ZLog);
```

以上操作必须均具备方可

借助宏构建宏可以编写更加简单的日志输入方式

```
#define 快速日志宏 (参数) UE_LOG(日志分类, 日志级别, 输出内容);
```


09 格式化日志输出



类似C语言中的printf输出，在虚幻中UE_LOG支持可变参数进行构建复杂语句格式，通过占位符，进行输入导入，用来编写更加清晰的日志语句。

占位符

%d 整数输出

%f 浮点数输出

%s 输出UE类型字符串（宽字符串指针）

编写带有可变参数的自定义日志

__VA_ARGS__ 可变参数宏，在宏中用来接收可变参数，将可变参数进行传递。

可以借助此宏编写更加简单的日志输出宏

```
Log_Display(msg, ...) UE_LOG(FLLLog, Display, TEXT(msg), ##__VA_ARGS__);
```



感谢聆听

THANK YOU

帮助更多的人实现梦想