UE4类型系统

Object基类
优点：
1. 便于追踪
2. 通用属性和接口
3. 便于GC
4. 统一序列化
5. 反射

缺点：
1. 接口和属性并不一定所有对象都用得上（冗余
2. 有可能菱形继承

UE4实现反射的方案：和QT中的类似，用宏做标记，然后用UHT分析生成generated.h/.cpp文件后再一起编译。

模拟构建的流程：生成，收集和注册，链接

```cpp
#include "UObject/NoExportTypes.h"
#include "MyClass.generated.h"


UCLASS()
class HELLO_API UMyClass : public UObject
{
    GENERATED_BODY()
};
```

UHT代码生成阶段

代码里面GENERATED_BODY()的宏定义：

```
#define BODY_MACRO_COMBINE_INNER(A,B,C,D) A##B##C##D
#define BODY_MACRO_COMBINE(A,B,C,D) BODY_MACRO_COMBINE_INNER(A,B,C,D)
#define GENERATED_BODY(...) BODY_MACRO_COMBINE(CURRENT_FILE_ID,_,__LINE__,_GENERATED_BODY)
```

GENERATED_BODY最终只是生成另一个宏的名称，此处的__LINE__是标准宏，CURRENT_FILE_ID定义在生成的generated.h里面。

如果MyClass类需要UMyClass(const FObjectInitializer& ObjectInitializer)的构造函数自定义实现，则需要用GENERATED_UCLASS_BODY宏来让最终生成的宏指向Hello_Source_Hello_MyClass_h_11_GENERATED_BODY_LEGACY,这样最终展开会多一个构造函数的显示。

```
PRAGMA_DISABLE_DEPRECATION_WARNINGS
#ifdef HELLO_MyClass_generated_h
#error "MyClass.generated.h already included, missing '#pragma once' in MyClass.h"
#endif
#define HELLO_MyClass_generated_h

#define Hello_Source_Hello_MyClass_h_11_RPC_WRAPPERS    //先忽略
#define Hello_Source_Hello_MyClass_h_11_RPC_WRAPPERS_NO_PURE_DECLS  //先忽略
#define Hello_Source_Hello_MyClass_h_11_INCLASS_NO_PURE_DECLS \
    private: \
    static void StaticRegisterNativesUMyClass(); \
    friend HELLO_API class UClass* Z_Construct_UClass_UMyClass(); \
    public: \
    DECLARE_CLASS(UMyClass, UObject, COMPILED_IN_FLAGS(0), 0, TEXT("/Script/Hello"), NO_API) \
    DECLARE_SERIALIZER(UMyClass) \
    /** Indicates whether the class is compiled into the engine */ \
    enum {IsIntrinsic=COMPILED_IN_INTRINSIC};


#define Hello_Source_Hello_MyClass_h_11_INCLASS \
    private: \
    static void StaticRegisterNativesUMyClass(); \
    friend HELLO_API class UClass* Z_Construct_UClass_UMyClass(); \
    public: \
    DECLARE_CLASS(UMyClass, UObject, COMPILED_IN_FLAGS(0), 0, TEXT("/Script/Hello"), NO_API) \
    DECLARE_SERIALIZER(UMyClass) \
    /** Indicates whether the class is compiled into the engine */ \
    enum {IsIntrinsic=COMPILED_IN_INTRINSIC};


#define Hello_Source_Hello_MyClass_h_11_STANDARD_CONSTRUCTORS \
    /** Standard constructor, called after all reflected properties have been initialized */ \
    NO_API UMyClass(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get()); \
    DEFINE_DEFAULT_OBJECT_INITIALIZER_CONSTRUCTOR_CALL(UMyClass) \
    DECLARE_VTABLE_PTR_HELPER_CTOR(NO_API, UMyClass); \
DEFINE_VTABLE_PTR_HELPER_CTOR_CALLER(UMyClass); \
private: \
    /** Private move- and copy-constructors, should never be used */ \
    NO_API UMyClass(UMyClass&&); \
    NO_API UMyClass(const UMyClass&); \
public:
```

```
#define Hello_Source_Hello_MyClass_h_11_ENHANCED_CONSTRUCTORS \
    /** Standard constructor, called after all reflected properties have been initialized */ \
    NO_API UMyClass(const FObjectInitializer& ObjectInitializer = FObjectInitializer::Get()) : Super(ObjectInitializer) { }; \
private: \
    /** Private move- and copy-constructors, should never be used */ \
    NO_API UMyClass(UMyClass&&); \
    NO_API UMyClass(const UMyClass&); \
public: \
    DECLARE_VTABLE_PTR_HELPER_CTOR(NO_API, UMyClass); \
DEFINE_VTABLE_PTR_HELPER_CTOR_CALLER(UMyClass); \
    DEFINE_DEFAULT_OBJECT_INITIALIZER_CONSTRUCTOR_CALL(UMyClass)


#define Hello_Source_Hello_MyClass_h_11_PRIVATE_PROPERTY_OFFSET     //先忽略
#define Hello_Source_Hello_MyClass_h_8_PROLOG   //先忽略
#define Hello_Source_Hello_MyClass_h_11_GENERATED_BODY_LEGACY \ //两个重要的定义
PRAGMA_DISABLE_DEPRECATION_WARNINGS \
public: \
    Hello_Source_Hello_MyClass_h_11_PRIVATE_PROPERTY_OFFSET \
    Hello_Source_Hello_MyClass_h_11_RPC_WRAPPERS \
    Hello_Source_Hello_MyClass_h_11_INCLASS \
    Hello_Source_Hello_MyClass_h_11_STANDARD_CONSTRUCTORS \
public: \
PRAGMA_ENABLE_DEPRECATION_WARNINGS


#define Hello_Source_Hello_MyClass_h_11_GENERATED_BODY \    //两个重要的定义
PRAGMA_DISABLE_DEPRECATION_WARNINGS \
public: \
    Hello_Source_Hello_MyClass_h_11_PRIVATE_PROPERTY_OFFSET \
    Hello_Source_Hello_MyClass_h_11_RPC_WRAPPERS_NO_PURE_DECLS \
    Hello_Source_Hello_MyClass_h_11_INCLASS_NO_PURE_DECLS \
    Hello_Source_Hello_MyClass_h_11_ENHANCED_CONSTRUCTORS \
private: \
PRAGMA_ENABLE_DEPRECATION_WARNINGS

#undef CURRENT_FILE_ID
#define CURRENT_FILE_ID Hello_Source_Hello_MyClass_h    //前文说过的定义
PRAGMA_ENABLE_DEPRECATION_WARNINGS
```

倒数第一个HELLO_SOURCE宏中的一个

```
#define DEFINE_DEFAULT_OBJECT_INITIALIZER_CONSTRUCTOR_CALL(TClass) \
    static void __DefaultConstructor(const FObjectInitializer& X) { new((EInternal*)X.GetObj())TClass(X); }
```

因为到时候可能需要用名字构造对象，需要保存函数指针。

```
class COREUOBJECT_API UClass : public UStruct
{
    typedef void (*ClassConstructorType) (const FObjectInitializer&);
    ClassConstructorType ClassConstructor;
}
```

而DECLARE_CLASS内主要定义了类的一些常用函数。

```
#define DECLARE_CLASS( TClass, TSuperClass, TStaticFlags, TStaticCastFlags, TPackage, TRequiredAPI  ) \
private: \
    TClass& operator=(TClass&&);   \
    TClass& operator=(const TClass&);   \
    TRequiredAPI static UClass* GetPrivateStaticClass(const TCHAR* Package); \
public: \
    /** Bitwise union of #EClassFlags pertaining to this class.*/ \
    enum {StaticClassFlags=TStaticFlags}; \
    /** Typedef for the base class ({{ typedef-type }}) */ \
    typedef TSuperClass Super;\
    /** Typedef for {{ typedef-type }}. */ \
    typedef TClass ThisClass;\
    /** Returns a UClass object representing this class at runtime */ \
    inline static UClass* StaticClass() \
    { \
        return GetPrivateStaticClass(TPackage); \
    } \
    /** Returns the StaticClassFlags for this class */ \
    inline static EClassCastFlags StaticClassCastFlags() \
    { \
        return TStaticCastFlags; \
    } \
    DEPRECATED(4.7, "operator new has been deprecated for UObjects - please use NewObject or NewNamedObject instead") \
    inline void* operator new( const size_t InSize, UObject* InOuter=(UObject*)GetTransientPackage(), FName InName=NAME_None, EObjectFlags InSetFlags=RF_NoFlags ) \
    { \
        return StaticAllocateObject( StaticClass(), InOuter, InName, InSetFlags ); \
    } \
    /** For internal use only; use StaticConstructObject() to create new objects. */ \
    inline void* operator new(const size_t InSize, EInternal InInternalOnly, UObject* InOuter = (UObject*)GetTransientPackage(), FName InName = NAME_None, EObjectFl
    { \
        return StaticAllocateObject(StaticClass(), InOuter, InName, InSetFlags); \
} \
    /** For internal use only; use StaticConstructObject() to create new objects. */ \
    inline void* operator new( const size_t InSize, EInternal* InMem ) \
    { \
        return (void*)InMem; \
    }
```

关于StaticClass的代码调用流程如下：

```
inline static UClass* StaticClass() \
{ \
    return GetPrivateStaticClass(TPackage); \
} \
```

对应cpp文件中：
上方的宏定义其实就是一个static自动注册过程，进行类的信息的收集。

```
#define IMPLEMENT_CLASS(TClass, TClassCrc) \
static TClassCompiledInDefer<TClass> AutoInitialize##TClass(TEXT(#TClass), sizeof(TClass), TClassCrc); \   //延迟注册
UClass* TClass::GetPrivateStaticClass(const TCHAR* Package) \   //.h里声明的实现，StaticClas()内部就是调用该函数
{ \
    static UClass* PrivateStaticClass = NULL; \ //又一次static lazy
    if (!PrivateStaticClass) \
    { \
        /* this could be handled with templates, but we want it external to avoid code bloat */ \
        GetPrivateStaticClassBody( \    //该函数就是真正创建UClass*,以后
            Package, \  //Package名字
            (TCHAR*)TEXT(#TClass) + 1 + ((StaticClassFlags & CLASS_Deprecated) ? 11 : 0), \//类名，+1去掉U、A、F前缀，+11去掉_Deprecated前缀
            PrivateStaticClass, \   //输出引用
            StaticRegisterNatives##TClass, \
            sizeof(TClass), \
            TClass::StaticClassFlags, \
            TClass::StaticClassCastFlags(), \
            TClass::StaticConfigName(), \
            (UClass::ClassConstructorType)InternalConstructor<TClass>, \
            (UClass::ClassVTableHelperCtorCallerType)InternalVTableHelperCtorCaller<TClass>, \
            &TClass::AddReferencedObjects, \
            &TClass::Super::StaticClass, \
            &TClass::WithinClass::StaticClass \
        ); \
    } \
    return PrivateStaticClass; \
}
```

UE4中用的收集方法:

```
127  template <typename TClass>
128  struct TClassCompiledInDefer : public FFieldCompiledInInfo
129  {
130      TClassCompiledInDefer(const TCHAR* InName, SIZE_T InClassSize, uint32 InCrc)
131      : FFieldCompiledInInfo(InClassSize, InCrc)
132      {
133          UClassCompiledInDefer(this, InName, InClassSize, InCrc);
134      }
135      virtual UClass* Register() const override
136      {
137          return TClass::StaticClass();
138      }
139  };
140
141  static TClassCompiledInDefer<TClass> AutoInitialize##TClass(TEXT(#TClass), sizeof(TClass), TClassCrc);
```

```
void GetPrivateStaticClassBody(
const TCHAR* PackageName,
const TCHAR* Name,
UClass*& ReturnClass,
void(*RegisterNativeFunc)(),
uint32 InSize,
EClassFlags InClassFlags,
EClassCastFlags InClassCastFlags,
const TCHAR* InConfigName,
UClass::ClassConstructorType InClassConstructor,
UClass::ClassVTableHelperCtorCallerType InClassVTableHelperCtorCaller,
UClass::ClassAddReferencedObjectsType InClassAddReferencedObjects,
UClass::StaticClassFunctionType InSuperClassFn,
UClass::StaticClassFunctionType InWithinClassFn,
bool bIsDynamic /*= false*/
)
{
ReturnClass = (UClass*)GUObjectAllocator.AllocateUObject(sizeof(UClass), alignof(UClass), true);//分配内存
ReturnClass = ::new (ReturnClass)UClass //用placement new在内存上手动调用构造函数
(
EC_StaticConstructor,Name,InSize,InClassFlags,InClassCastFlags,InConfigName,
EObjectFlags(RF_Public | RF_Standalone | RF_Transient | RF_MarkAsNative | RF_MarkAsRootSet),
InClassConstructor,InClassVTableHelperCtorCaller,InClassAddReferencedObjects
);
InitializePrivateStaticClass(InSuperClassFn(),ReturnClass,InWithinClassFn(),PackageName,Name);//初始化UClass*对象
RegisterNativeFunc();//注册Native函数到UClass中去
}
```

```
COREUOBJECT_API void InitializePrivateStaticClass(
    class UClass* TClass_Super_StaticClass,
    class UClass* TClass_PrivateStaticClass,
    class UClass* TClass_WithinClass_StaticClass,
    const TCHAR* PackageName,
    const TCHAR* Name
    )
{
    //...
    if (TClass_Super_StaticClass != TClass_PrivateStaticClass)
    {
        TClass_PrivateStaticClass->SetSuperStruct(TClass_Super_StaticClass);    //设定类之间的SuperStruct
    }
    else
    {
        TClass_PrivateStaticClass->SetSuperStruct(NULL);    //UObject无基类
    }
    TClass_PrivateStaticClass->ClassWithin = TClass_WithinClass_StaticClass;    //设定Outer类类型
    //...
    TClass_PrivateStaticClass->Register(PackageName, Name); //转到UObjectBase::Register()
    //...
}
```

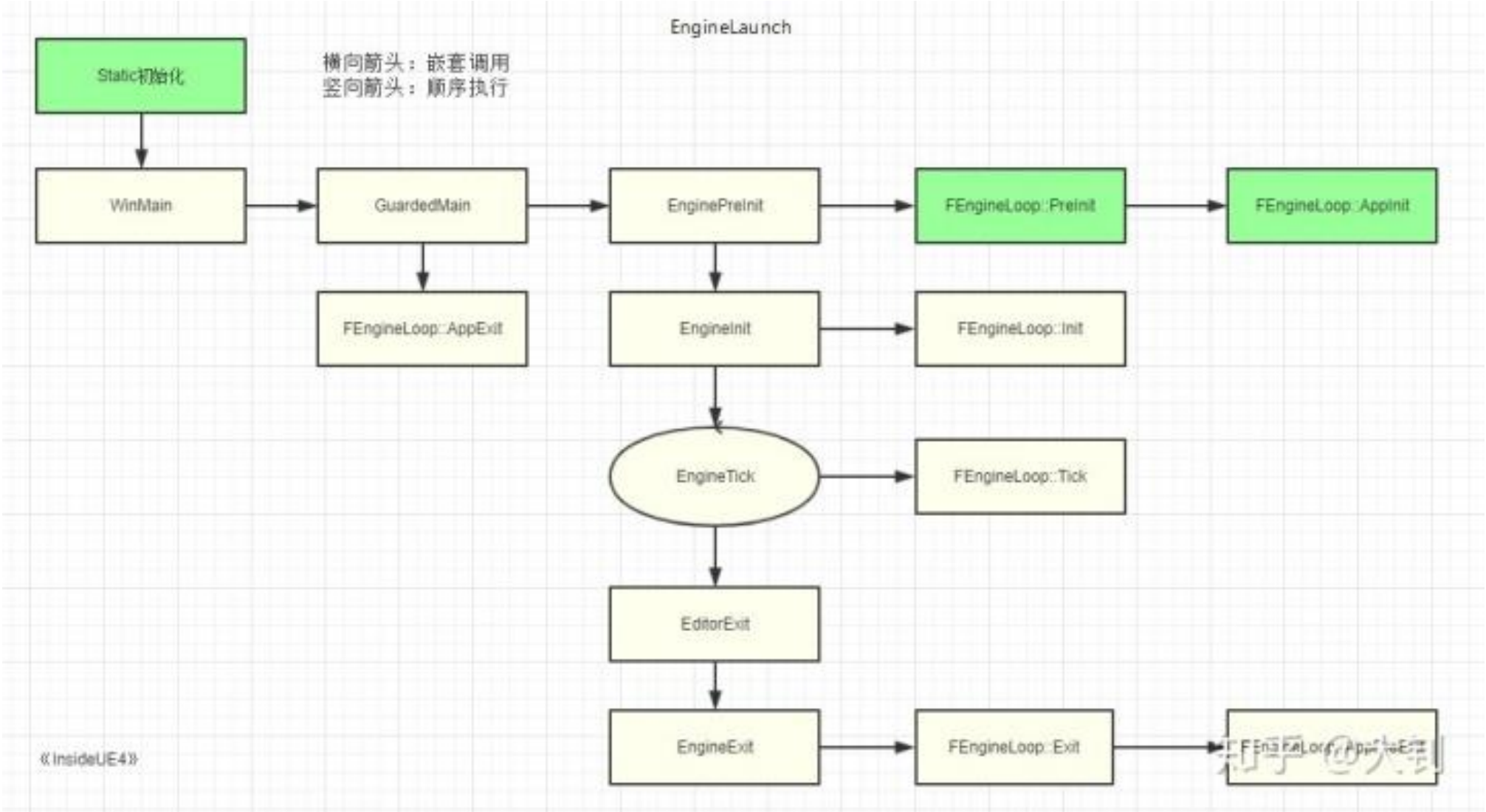
```cpp
struct FPendingRegistrantInfo
{
    const TCHAR*    Name;   //对象名字
    const TCHAR*    PackageName;    //所属包的名字
    static TMap<UObjectBase*, FPendingRegistrantInfo>& GetMap()
    {   //用对象指针做Key，这样才可以通过对象地址获得其名字信息，这个时候UClass对象本身其实还没有名字，要等之后的注册才能设置进去
        static TMap<UObjectBase*, FPendingRegistrantInfo> PendingRegistrantInfo;
        return PendingRegistrantInfo;
    }
};
//...
struct FPendingRegistrant
{
    UObjectBase*    Object; //对象指针，用该值去PendingRegistrants里查找名字。
    FPendingRegistrant* NextAutoRegister;   //链表下一个节点
};
static FPendingRegistrant* GFirstPendingRegistrant = NULL;  //全局链表头
static FPendingRegistrant* GLastPendingRegistrant = NULL;   //全局链表尾
//...
void UObjectBase::Register(const TCHAR* PackageName,const TCHAR* InName)
{
    //添加到全局单件Map里，用对象指针做Key，Value是对象的名字和所属包的名字。
    TMap<UObjectBase*, FPendingRegistrantInfo>& PendingRegistrants = FPendingRegistrantInfo::GetMap();
    PendingRegistrants.Add(this, FPendingRegistrantInfo(InName, PackageName));
    //添加到全局链表里，每个链表节点带着一个本对象指针，简单的链表添加操作。
    FPendingRegistrant* PendingRegistration = new FPendingRegistrant(this);
    if(GLastPendingRegistrant)
    {
        GLastPendingRegistrant->NextAutoRegister = PendingRegistration;
    }
    else
    {
        check(!GFirstPendingRegistrant);
        GFirstPendingRegistrant = PendingRegistration;
    }
    GLastPendingRegistrant = PendingRegistration;
}
```

记录的数据结构为一个Map加上一个链表，Map保证快速查找，链表是顺序结构用来辅助，因为有些情况需要遵循添加的顺序。这里只是简单记录一下信息，并没有做实际的操作。因为此时还在static阶段，UObject对象分配索引什么的还没初始化好，无法进行实际操作。

Uobject这个基类就是在最开始通过IMPLEMENT_VM_FUNCTION(EX_CallMath, execCallMathFunction)调用触发自己的StaticClass调用产生的。

```
#define IMPLEMENT_FUNCTION(func) \
static FNativeFunctionRegistrar UObject##func##Registar(UObject::StaticClass(),#func,&UObject::func);

//...
IMPLEMENT_VM_FUNCTION(EX_CallMath, execCallMathFunction);//ScriptCore.cpp里的定义
```


EngineLaunch
Static初始化
横向箭头：嵌套调用
竖向箭头：顺序执行
WinMain
GuardedMain
EnginePreInit
FEngineLoop::PreInit
FEngineLoop::AppInit
FEngineLoop::AppExit
EngineInit
FEngineLoop::Init
EngineTick
FEngineLoop::Tick
EditorExit
EngineExit
FEngineLoop::Exit
《InsideUE4》