# CryEngine3的LUA框架

## Cry3的entity系统

游戏中的实体，是逻辑的基本单位，例如角色，怪物（cry3里没怪物，只有actor，怪物猎人加的），场景物件（可破坏物），物品（手雷，武器）等都是一个entity，这些entity大多是用LUA脚本定义的，entity系统结构图如下



一个entity的结构如图的组件式模型，第一层的方框是proxy，代表组件基本功能，gameobject是用户扩展逻辑，一个entity如果对应游戏中的实际逻辑，取决于gameobject下的extension是什么，例如extension1是一个CMonster类，则该entity是一个怪物，如果是CPlayer类，则是一个角色

## 脚本模块功能

集成LUA功能的便是ScriptProxy，大多数entity类型是通过LUA脚本定义，LUA的主要功能是

1.类型定义

2.属性方法定义

3.C++组件的组合者

4.简单状态机

5.基于状态机事件响应(c++大量调用LUA函数)

## 具体示例

Player类型的定义，定义了player的属性和方法等，self关键字可以理解为C++的this指针，

”:”就是C++内的”::”这样就好理解了

Script.ReloadScript( "SCRIPTS/Entities/actor/BasicActor.lua");

Player = {

AnimationGraph = "HumanMaleFullBody.xml",

UpperBodyGraph = "HumanMaleUpperBody.xml",

type = "Player",

foreignCollisionDamageMult = 0.1,

vehicleCollisionDamageMult = 7.5,

//属性定义

Properties =

{

-- AI-related properties

soclasses\_SmartObjectClass = "Player",

groupid = 0,

esFaction = "Players",

}

};

//在C++对象中会调用OnUpdate，用Ref索引

function Player:OnUpdate(frameTime)

if (self.stopEPATime and self.stopEPATime < 0) then

//调用C++模块回调函数

self.actor:SetParams({followCharacterHead = 0,});

self.actor:SetMovementTarget(g\_Vectors.v000,g\_Vectors.v000,g\_Vectors.v000,1);

//调用C++基础entity的回调函数

self:HolsterItem(false);

end;

end

简单状态机不做介绍了，跟LUA特性关系不大

## Entity创建过程

为什么说LUA是组件组织者？

entity结构图列出了entity的重要组件，其中最先初始化的就是脚本组件，entity创建过程用player举例，每个脚本会被加载并且抽象成一个C++类叫EntityClass，这里包含的有，文件名字，class名字（Player），即定义的表名字，以及Player这个LUA table，所就像C++给出class可以创建对象一样，给出一个EntityClass的时候，创建Entity的过程如下

1. 创建了Entity的C++对象，初始化脚本模块，在脚本模块深度拷贝名称为Player的 table并且保存该table，如何保存table？请看链接xxxxx

Player = {

AnimationGraph = "HumanMaleFullBody.xml",

UpperBodyGraph = "HumanMaleUpperBody.xml",

type = "Player",

foreignCollisionDamageMult = 0.1,

vehicleCollisionDamageMult = 7.5,

//属性定义

Properties =

{

-- AI-related properties

soclasses\_SmartObjectClass = "Player",

groupid = 0,

esFaction = "Players",

}

};

1. table复制了一份，但是有个问题，除了table的内容，Player.lua还定义了基于名字为Player的table的函数，如何让复制的table也能访问这些函数？

function Player:OnUpdate(frameTime)

end

函数复制？作者不知道是否有这个功能，但是程序猿都知道，一个对象数据是独有的，函数都是公有的，这时候就引入了LUA的一个重要特性，也是LUA实现继承和多态的关键，metatable，中文叫元表，

举例 table A table B

当A是B的元表，并且A中定义了\_\_index（注意是两个下划线），当在B中无法索引到的元素，则会去A中查找，那么B中没有的函数，则会去调用A的

这样就简单了，只需要把在Player.lua中的名称为Player的table设置为复制出的table的元表，则新创建的Entity独有的一个table就可以访问Player.lua中的函数

1. 复制出的table都是跟Player.lua中定义的名称为Player的table相同的属性，cry3的编辑器可以编辑entity的table并且序列化保存为archetype的文件，所以创建一个entity需要entityclass加上属性特化文件，然后用属性特化文件重新把复制出的table修改下，则一个新的entity诞生了
2. 有了新的table，在C++中调用LUA内的Init回调，Init函数则可以根据table的属性，来决定创建哪些组件，比如需要渲染一个静态mesh，则会创建Render组件，需要进行刚体模拟，会继续创建Physics模块，然后根据属性来初始化这些模块，所以说脚本是组件的整合者，初始化完毕，一个新的entity彻底诞生

## Cry3的C++回调管理

红色字和绿色字部分

//在C++对象中会调用OnUpdate，用Ref索引

function Player:OnUpdate(frameTime)

if (self.stopEPATime and self.stopEPATime < 0) then

//调用C++模块回调函数

self.actor:SetParams({followCharacterHead = 0,});

self.actor:SetMovementTarget(g\_Vectors.v000,g\_Vectors.v000,g\_Vectors.v000,1);

//调用C++基础entity的回调函数

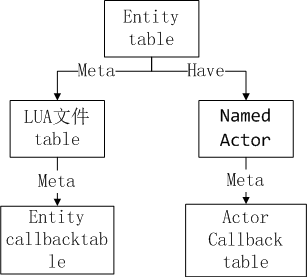
self:HolsterItem(false);

end;

end

这里有个不同，self:HolsterItem(false);与self.actor:SetParams({followCharacterHead = 0,});

的不同之处在？



LUA的C++的回调的组织方式，所有entity公有的C++回调函数，都放在entity callback table内，上图内对照Player，entityTable可以理解为每个Player对象都单独拥有的LUA table，LUA文件table就是在Player.lua文件内定义的名称为Player的table

而self.actor:SetParams({followCharacterHead = 0,});

中的actor是什么呢，往往这里是跟一个extension相关的，流程大概如下

当一个entity有一个新的extension创建的时候，比如Player这个extension，那么在entity的单独table内会创建一个叫做actor的table，并且在这个table内存入entityID的信息，然后把存有所有关于Player相关的C++回调的actor callback table设置为名称为actor的元表，所以当没有Player这个extension的时候，也不会有actor这个表格，实现LUA功能和C++功能扩展的统一，所以两个函数的不同就是，一个是在entity通用table内，一个是在对应extension创建才会生效的table内

## 供C++对象调用的LUA回调函数的管理

这个没什么好纠结的，比如调用

function Player:OnHit(hitParam)

end;

首先，该函数已经通过名字进行了引用，又提到了Ref功能，神器必用

再次广告，km：xxxxxx

调用这类函数没啥好说，就是push进函数后，不能直接push hitParam这个参数，还得多push个table，是哪个table呢？当思考题吧。。。