现在脚本可谓在游戏的开发中占有举足轻重的地位啊，从python到lua，甚至还有自己设计脚本语言的，其实说到脚本，我倒是很喜欢javascript的，可以应用有限，很难运用于游戏中来，好了废话不多说，我们回到正题来。

游戏中为什么要加入脚本呢，其实是为了更好的扩展性，比如人物的初始化设置，对话内容，攻击模式甚至剧情走向都可以用脚本来编写，因为如果把这些直接嵌在游戏主程序里面不要生成的文件庞大冗长，而且扩展性极差，如果我们要改变一个很小的设置，就必须要重新编译整个游戏工程，很显然，是不划算的，所以，便有了脚本的出现，再说说脚本的执行效率问题，我们知道，脚本的执行速度普遍比编译语言要低，那么我们就要在众多脚本中选择一个高效轻量的脚本来作为我们的首选，于是lua便映入了我们的视线。

lua这个语言学起来还是蛮简单的，可能就是上手会有点变扭，因为他都是以栈作为操作基础的，我们调用它的一个内部函数基本上都是改变lua栈的运行状态，不过，在理解这一点后，上手还是很快的。

[cpp]

```cpp
class CDXLua
{
public:
    CDXLua();
    ~CDXLua();
    bool Execute(LPCSTR szPath);
    float GetNum(LPCSTR szName);
    LPCSTR GetString(LPCSTR szName);
    float GetTableNum(LPCSTR szTable,LPCSTR szKey);
    LPCSTR GetTableString(LPCSTR szTable,LPCSTR szKey);
    void CallFunc(LPCSTR szFunc);
    lua_State *GetLua();
private:
    lua_State *m_pLua;
};
```

以上便是我封装的一个简单的lua类，Execute便是载入一个lua脚本，而GetNum等函数便是获得脚本里相应的变量值，现有如下脚本：

[plain]

```plain
Eye={x=2,y=30,z=2};
At={x=2,y=10,z=30};
obj=RPG();
obj:AddTree(200,200,"t1");
obj:AddTree(200,100,"t2");
obj:AddTree(100,200,"t3");
obj:AddTree(300,200,"t4");
obj:AddTree(200,300,"t5");
obj:AddEnemy(500,100,"e1");
obj:AddEnemy(500,300,"e2");
obj:AddEnemy(50,30,"e3");
obj:AddHuman(150,150,"h");
obj:AddText("Me:你好");
obj:AddText("NPC:现在局势很乱啊");
obj:AddText("Me:是啊，我一定要好好努力啊");
obj:AddText("NPC:恩，加油啊！");
```

如何应用到游戏里面中去呢，对于Eye和At变量，我们需啊哟载入脚本后读出相应的值就可以了。

```
m_Lua.Execute("lua\\init.lua");
D3DXVECTOR3
vEye(m_Lua.GetTableNum("Eye","x"),m_Lua.GetTableNum("Eye","y"),m_Lua.GetTableNum("Eye","z"));
 //摄像机的位置
D3DXVECTOR3
```

loa

vAt(m_Lua.GetTableNum("At","x"),m_Lua.GetTableNum("At","y"),m_Lua.GetTableNum("At","z")); //观察点的位置

这样的话配置属性便可以得到，那么对于obj=RPG();这样的一个创建对象的语句，我们又是如何做到的呢，这里不得不引入另外一个文件了：#include "lua_tinker.h"，就是这个韩国人编写的短小精悍的Lua辅助库可以完成我们关于lua创建c++类的操作，我们看一下如下代码：

```cpp
class CLuaCall
{
public:
    CLuaCall()
    {

    }
    void AddTree(float x,float z,LPCSTR id);
    void AddEnemy(float x,float z,LPCSTR id);
    void AddHuman(float x,float z,LPCSTR id);
    void AddText(LPCSTR Text);
};
```

我们在lua脚本里创建的便是这个类，然后通过它的一系列方法来操作我们的游戏主对象，给出一个AddTree函数的实现：

```cpp
void CLuaCall::AddTree( float x,float z,LPCSTR id )
{
    USES_CONVERSION;
    CDXRPG::RPGObj->Lua_AddTree(x,z,A2W(id));
}
```

```cpp
void CDXRPG::Lua_AddTree( float x,float z,LPCTSTR id )
{
    D3DXMATRIX mat,mat1;
    D3DXMatrixScaling(&mat1,0.05,0.05,0.05);
    float fHeight=m_Land.GetHeight(x,z);
    D3DXMatrixTranslation(&mat,x,fHeight,z);
    mat=mat1*mat;
    CDXEntity *p1=new CDXEntity(&m_pMeshTree,&mat);
    D3DXVECTOR3 min=p1->GetBoundMin();
    D3DXVECTOR3 max=p1->GetBoundMax();
    min.x+=30;
    min.z+=30;
    max.x-=30;
    max.z-=30;
    p1->SetBoundMin(min);
    p1->SetBoundMax(max);
    m_OctNode.AddEntity(p1,id);
}
```

loa

这样大家就可以把脉络理清楚了吧，首先我们通过lua创建一个CLuaCall的c++对象，然后通过这个对象的AddTree方法调用CDXRPG对象里的Lua_AddTree方法，真正添加一个树的操作就是在Lua_AddTree这个函数里完成的。接下来的问题是，我们怎么让c++和lua之间通信呢，也就是说怎么让lua识别这个CLuaCall对象并且创建它呢，看以下代码：

```cpp
lua_State *pL=m_Lua.GetLua();
    lua_tinker::class_add<CLuaCall>(pL,"RPG");
```

```
    lua_tinker::class_con<CLuaCall>(pL,lua_tinker::constructor<CLuaCall>);
    lua_tinker::class_def<CLuaCall>(pL,"AddTree",&CLuaCall::AddTree);
    lua_tinker::class_def<CLuaCall>(pL,"AddEnemy",&CLuaCall::AddEnemy);
    lua_tinker::class_def<CLuaCall>(pL,"AddHuman",&CLuaCall::AddHuman);
    lua_tinker::class_def<CLuaCall>(pL,"AddText",&CLuaCall::AddText);
```

就是这么几句类的声明和类函数声明的代码，很简单也很清晰的流程，当然，这些代码在游戏初始化的时候，也就是载入脚本前就应该调用，否则找不到类或者函数的声明会出错哦。

本文有不足之处，还望大家多多指正。

loa