向 lua 虚拟机传递信息

当程序逻辑交给脚本跑了以后，C/C++ 层就只需要把必要的输入信息传入虚拟机就够了。当然，我们也需要一个高效的传递方法。

以向 lua 虚拟机传递鼠标坐标信息为例，我们容易想到的方法是，定义一个 C 函数 get\_mouse\_pos 。当 lua 脚本中需要取得鼠标坐标的时候，就可以调用这个函数。

但这并不是一个好方法，因为每次获取鼠标坐标，都需要在虚拟机和 native code 间做一次切换。我们应该寻求更高效的方案。

编写脚本的人可以只获取一次鼠标坐标，然后把数据放进一组全局变量。在一个运行片内，不再调用 get\_mouse\_pos 函数，而是通过访问全局变量来得到鼠标的位置。

从这个方案，我们可以引申开，其实这个全局变量可以由 C 程序主动设置，在 native code 的运行片中，Windows 消息处理完后，直接讲鼠标信息设入 lua 虚拟机。代码类似这样的。

lua\_pushstring(L,"MOUSE\_X");  
lua\_pushnumber(L,mouse\_x);  
lua\_settable(L,LUA\_GLOBALSINDEX);

lua\_pushstring(L,"MOUSE\_Y");  
lua\_pushnumber(L,mouse\_y);  
lua\_settable(L,LUA\_GLOBALSINDEX);

但是这里，依旧存在一个效率问题，那就是 lua\_pushstring 。我们知道，lua 虚拟机中，每次产生一个 string ，都需要查对 string 在虚拟机中是否存在相同的拷贝，如果存在，就直接引用已有的；如果不存在，则产生一份新的拷贝。

这里，MOUSE\_X 和 MOUSE\_Y 两个 string 除了第一次运行，以后都是存在于 lua 虚拟机中的，虽然不会产生新的 string，但查找和比较字符串依然会消耗一定的时间。下面，我们来优化这个 lua\_pushstring 操作。

我们可以在程序开始阶段，创建出这两个 string ，并且在 C 中保留引用。  
lua\_pushstring(L,"MOUSE\_X");  
\_mouse\_x\_ref=lua\_ref(L,-1);  
lua\_pushstring(L,"MOUSE\_Y");  
\_mouse\_y\_ref=lua\_ref(L,-1);

那么，以后运行时就不需要再做 lua\_pushstring 操作了，而改成相对较快的 lua\_getref 操作。

lua\_getref(L,\_mouse\_x\_ref);  
lua\_pushnumber(L,mouse\_x);  
lua\_settable(L,LUA\_GLOBALSINDEX);

lua\_getref(L,\_mouse\_y\_ref);  
lua\_pushnumber(L,mouse\_y);  
lua\_settable(L,LUA\_GLOBALSINDEX);

lua\_getref 之所以相对快一些，是因为 lua 对数字做 key 的 table 操作有优化处理，直接变成一次指针操作。而 ref 就是记在一张全局表中的。而且 lua\_getref 不需要 lua\_pushstring 做过的 strcmp 操作。

那么这个方法还没有优化余地呢？答案还是有。

我们其实可以写一个 lua 程序，放在一个单独的文件(mouse.lua)中，程序很短：

return funtion(mx,my) MOUSE\_X,MOUSE\_Y=mx,my end

我们在程序启动的时候运行  
lua\_dofile(L,"mouse.lua");  
\_mouse\_set\_ref=lua\_ref(L,-1);

那么，在设置鼠标坐标的时候就可以简单的做如下操作：  
lua\_getref(L,\_mouse\_set\_ref);  
lua\_pushnumber(L,mouse\_x);  
lua\_pushnumber(L,mouse\_y);  
lua\_call(L,2,0);

这个方案只需要保留一个函数的 ref ，并且把设置的工作交给了虚拟机中的伪指令。单从这个例子(仅仅 MOUSE\_X,MOUSE\_Y两个需要传递的信息)来看，不能说明后者的效率更高一些，毕竟 lua\_call 也有额外的消耗。但是，最后一个方案更加灵活，对于native code 向虚拟机更多数据的交换采用这种方案更加简洁。

ps. lua\_ref 的东西，最后要记得调用 lua\_unref 解引用。