lua在游戏开发中非常常见，很好的提升了开发效率，降低逻辑更改成本，甚至实现动态配置逻辑的功能，在与c++的交互中，主要通过一个lua\_state栈进行数据传递，c++操纵lua的内部数据结构也是通过基于lua\_state的api进行，两者进行交互基本用法就包括，lua调用c++提供的接口函数，反过来，c++调用lua函数

举例

c++回调接口

**static int l\_sin (lua\_State \*L)**

**{**

**double d = luaL\_checknumber(L, 1);**

**lua\_pushnumber(L, sin(d));**

**return 1;**

**}**

**上边例子luaL\_checkxxx是lua\_toxxx的加强版本，会自动捕获异常，不过肯定会降低效率，所以用不用就是根据实际情况来定了**

**lua中使用就简单了，假设是该函数定义在global中，那直接就是在lua中添加**

**l\_sin(1.0);**

**反过来，c++调用lua**

**在abc.lua中定义一个函数**

function add(x, y)

return x + y

end

在c++中使用

int main(int argc, char \*argv[])

{

int sum = 0;

L = lua\_open();

luaopen\_base(L);

luaL\_openlibs(L);

luaL\_loadfile(L, "**abc**.lua");

lua\_pcall(L, 0, LUA\_MULTRET, 0);

lua\_getglobal(L, "add");

lua\_pushnumber(L, x);

lua\_pushnumber(L, y);

lua\_call(L, 2, 1);

sum = (int)lua\_tonumber(L, -1);

lua\_pop(L, 1);

printf("The sum is %d\n", sum);

lua\_close(L);

return 0;

}

是不是看着头晕，没错，如果对于一个对lua的api不熟悉的同事来说，在c++中写lua的回调真是要命啊，而且每个函数内部都这么写，实在是累人，这就引出了lua的逻辑封装，封装当然就是为了简单好用，当然最好在提高下效率，好，先从lua的数据封装开始吧

lua的数据内置与c++比较容易交换的数据包括number(double),bool,table,function,string

所谓数据封装，肯定能想到就是type+value(union)的一个数据结构，这个确实如此，然后再加上针对这个数据结构以及各种蛋疼无比的基于lua\_state的栈操作就可以了，这个不深入啰嗦了，用个简单的重载函数封装下参数获取这类api

bool lua\_getparam(lua\_state\* L,int index, const char\* value);

bool lua\_getparam(lua\_state\*L, int index, double& value);

这种就随便写写了

不知道是否有人发出疑问，在c++中如何创建或者保存一个table和function呢？哈，重点终于来了

应该大家最常用的方式就是，没错，string来表示，甚至有 a.b.c.d 这种表示方式以及基于这种路径表示的table获取函数，以前用的公司组件tpf(2009版本)就是这么将c++的ui widget对象与lua table绑定，function同理

这种弱引用方式能很好的解决问题，就是相对来说效率低一些，毕竟反复的查表，尤其是当lua中表的数目多到爆表的时候，也不是个小问题了

解决方法就是，使用Ref，这个的功能官方说法就是，用一个整数来表示一个对象，神奇的是，可以o(1) 得获得这个对象到栈顶，因为那个整数就是一个vector的下标

注意的是：lua的引用机制使用的是全局的注册表，所以在全局注册表中不能使用数字做索引，否则很容易引起冲突！

代码示例：

L是lua\_state\*

int ref = 0;

lua\_newtable(L); //创建的table在栈顶

ref = lua\_ref(L,1);

这样够简单吧，从此以后，ref就代表这个table了，怎么操作这个table呢

lua\_getref(L,ref); //这样刚才创建的table就被放到栈顶

lua\_unref(L,ref); //这样table就被删除了

至于function，貌似在c++中create一个function过于高端了，就不考虑了，但是同样可以利用字符串索引到function中对该值进行引用，这样就不需要在茫茫值海中每次调用都去寻找这个function了

代码示例

L是lua\_State\*

int ref = 0;

//get function 到栈顶

ref = lua\_ref(L,1);

总之呢，ref用法可以引用各种值，一般在c++中为了存取一个table或者调用一个function，用ref能够很好的解决字符串反复查表的问题，而且ref可以保证这个table不会被销毁

相信有了ref这个神器，用过lua的同学封装一个表示lua的所有值的数据结构已经完全不是问题了

附件的ppt演示了下ref的原理，做的不好，感兴趣的同学可以试试