[Lua和C++交互总结（很详细）](http://blog.csdn.net/shun_fzll/article/details/39120965)

标签： [lua](http://www.csdn.net/tag/lua)[c++](http://www.csdn.net/tag/c%2b%2b)[交互](http://www.csdn.net/tag/%e4%ba%a4%e4%ba%92)

2014-09-07 18:50 5401人阅读 [评论](http://blog.csdn.net/shun_fzll/article/details/39120965#comments)(10) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/shun_fzll/article/details/39120965#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

C++（14） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg lua语言（1） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[+]](http://blog.csdn.net/shun_fzll/article/details/39120965)

一不小心又标题党了？不好意思哈～

翻看了以前的博客，发下废话太多了自己都看不下去啊，恩，以后尽量写得简洁一些。

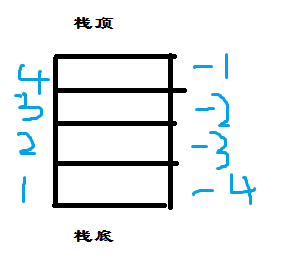
本文不涉及lua语法学习，如果有需要，请移步：<http://book.luaer.cn/>

**一.lua堆栈**

要理解lua和c++交互，首先要理解lua堆栈。

简单来说，Lua和C/c++语言通信的主要方法是一个无处不在的虚拟栈。栈的特点是先进后出。

在lua中，lua堆栈就是一个struct，**堆栈索引的方式可是是正数也可以是负数，区别是：正数索引1永远表示栈底，负数索引-1永远表示栈顶。**如图：



**二.堆栈的操作**

因为lua与c/c++是通过栈来通信，lua提供了C API对栈进行操作。

我们先来看一个最简单的例子：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. #include <iostream>
2. #include <string.h>
3. **using** **namespace** std;
5. **extern** "C"
6. {
7. #include "lua.h"
8. #include "lauxlib.h"
9. #include "lualib.h"
10. }
11. **void** main()
12. {
13. //1.创建一个state
14. lua\_State \*L = luaL\_newstate();
16. //2.入栈操作
17. lua\_pushstring(L, "I am so cool~");
18. lua\_pushnumber(L,20);
20. //3.取值操作
21. **if**( lua\_isstring(L,1)){             //判断是否可以转为string
22. cout<<lua\_tostring(L,1)<<endl;  //转为string并返回
23. }
24. **if**( lua\_isnumber(L,2)){
25. cout<<lua\_tonumber(L,2)<<endl;
26. }
28. //4.关闭state
29. lua\_close(L);
30. **return** ;
31. }

可以简单理解为luaL\_newstate返回一个指向堆栈的指针，其它看注释应该能懂了吧。

如果对extern "C"不熟悉的可以点击这里：<http://blog.csdn.net/shun_fzll/article/details/39078971>。

其他一些栈操作：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. **int**   lua\_gettop (lua\_State \*L);            //返回栈顶索引（即栈长度）
2. **void**  lua\_settop (lua\_State \*L, **int** idx);   //
3. **void**  lua\_pushvalue (lua\_State \*L, **int** idx);//将idx索引上的值的副本压入栈顶
4. **void**  lua\_remove (lua\_State \*L, **int** idx);   //移除idx索引上的值
5. **void**  lua\_insert (lua\_State \*L, **int** idx);   //弹出栈顶元素，并插入索引idx位置
6. **void**  lua\_replace (lua\_State \*L, **int** idx);  //弹出栈顶元素，并替换索引idx位置的值

lua\_settop将栈顶设置为一个指定的位置，即修改栈中元素的数量。如果值比原栈顶高，则高的部分nil补足，如果值比原栈低，则原栈高出的部分舍弃。

**所以可以用lua\_settop(0)来清空栈。**

**三.c++调用lua**

我们经常可以使用lua文件来作配置文件。类似ini，xml等文件配置信息。  
现在我们来使用c++来读取lua文件中的变量，table，函数。

现在有这样一个hello.lua 文件：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. str = "I am so cool"
2. tbl = {name = "shun", id = 20114442}
3. function add(a,b)
4. **return** a + b
5. end

我们写一个test.cpp来读取它：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. #include <iostream>
2. #include <string.h>
3. **using** **namespace** std;
5. **extern** "C"
6. {
7. #include "lua.h"
8. #include "lauxlib.h"
9. #include "lualib.h"
10. }
11. **void** main()
12. {
13. //1.创建Lua状态
14. lua\_State \*L = luaL\_newstate();
15. **if** (L == NULL)
16. {
17. **return** ;
18. }
20. //2.加载lua文件
21. **int** bRet = luaL\_loadfile(L,"hello.lua");
22. **if**(bRet)
23. {
24. cout<<"load file error"<<endl;
25. **return** ;
26. }
28. //3.运行lua文件
29. bRet = lua\_pcall(L,0,0,0);
30. **if**(bRet)
31. {
32. cout<<"pcall error"<<endl;
33. **return** ;
34. }
36. //4.读取变量
37. lua\_getglobal(L,"str");
38. string str = lua\_tostring(L,-1);
39. cout<<"str = "<<str.c\_str()<<endl;        //str = I am so cool~
41. //5.读取table
42. lua\_getglobal(L,"tbl");
43. lua\_getfield(L,-1,"name");
44. str = lua\_tostring(L,-1);
45. cout<<"tbl:name = "<<str.c\_str()<<endl; //tbl:name = shun
47. //6.读取函数
48. lua\_getglobal(L, "add");        // 获取函数，压入栈中
49. lua\_pushnumber(L, 10);          // 压入第一个参数
50. lua\_pushnumber(L, 20);          // 压入第二个参数
51. **int** iRet= lua\_pcall(L, 2, 1, 0);// 调用函数，调用完成以后，会将返回值压入栈中，2表示参数个数，1表示返回结果个数。
52. **if** (iRet)                       // 调用出错
53. {
54. **const** **char** \*pErrorMsg = lua\_tostring(L, -1);
55. cout << pErrorMsg << endl;
56. lua\_close(L);
57. **return** ;
58. }
59. **if** (lua\_isnumber(L, -1))        //取值输出
60. {
61. **double** fValue = lua\_tonumber(L, -1);
62. cout << "Result is " << fValue << endl;
63. }
65. //至此，栈中的情况是：
66. //=================== 栈顶 ===================
67. //  索引  类型      值
68. //   4   int：      30
69. //   3   string：   shun
70. //   2   table:     tbl
71. //   1   string:    I am so cool~
72. //=================== 栈底 ===================
74. //7.关闭state
75. lua\_close(L);
76. **return** ;
77. }

知道怎么读取后，我们来看下如何修改上面代码中table的值：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. // 将需要设置的值设置到栈中
2. lua\_pushstring(L, "我是一个大帅锅～");
3. // 将这个值设置到table中（此时tbl在栈的位置为2）
4. lua\_setfield(L, 2, "name");

我们还可以新建一个table：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. // 创建一个新的table，并压入栈
2. lua\_newtable(L);
3. // 往table中设置值
4. lua\_pushstring(L, "Give me a girl friend !"); //将值压入栈
5. lua\_setfield(L, -2, "str"); //将值设置到table中，并将Give me a girl friend 出栈

**需要注意的是：堆栈操作是基于栈顶的，就是说它只会去操作栈顶的值。**

举个比较简单的例子，函数调用流程是先将函数入栈，参数入栈，然后用lua\_pcall调用函数，此时栈顶为参数，栈底为函数，所以栈过程大致会是：参数出栈->保存参数->参数出栈->保存参数->函数出栈->调用函数->返回结果入栈。

类似的还有lua\_setfield，设置一个表的值，肯定要先将值出栈，保存，再去找表的位置。

再不理解可看如下例子：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. lua\_getglobal(L, "add");        // 获取函数，压入栈中
2. lua\_pushnumber(L, 10);          // 压入第一个参数
3. lua\_pushnumber(L, 20);          // 压入第二个参数
4. **int** iRet= lua\_pcall(L, 2, 1, 0);// 将2个参数出栈，函数出栈，压入函数返回结果
6. lua\_pushstring(L, "我是一个大帅锅～");  //
7. lua\_setfield(L, 2, "name");             // 会将"我是一个大帅锅～"出栈

另外补充一下：

lua\_getglobal(L,"var")会执行两步操作：1.将var放入栈中，2.由lua去寻找变量var的值，并将变量var的值返回栈顶（替换var）。  
lua\_getfield(L,-1,"name") 的作用等价于 lua\_pushstring(L,"name") + lua\_gettable(L,-2)

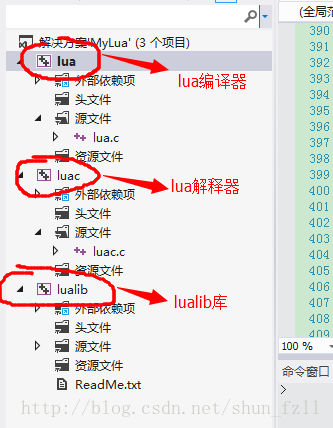
**四.lua调用c++**

我们分三个方法实现它。

**方法一：直接将模块写入lua源码中**

在lua中调用c/c++，我们可以将函数写lua.c中，然后重新编译lua文件。  
也许你用的是lua for windows集成环境，没关系，不会编译lua可以参考这篇：<http://blog.csdn.net/snlscript/article/details/15533373>

编译好后是这样子的：（如图）



然后我们可以在lua.c中加入我们自己的函数。函数要遵循规范（可在lua.h中查看）如下：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. **typedef** **int** (\*lua\_CFunction) (lua\_State \*L);

换句话说，所有的函数必须接收一个lua\_State作为参数，同时返回一个整数值。因为这个函数使用Lua栈作为参数，所以它可以从栈里面读取任意数量和任意类型的参数。而这个函数的返回值则表示函数返回时有多少返回值被压入Lua栈。（因为Lua的函数是可以返回多个值的）  
  
  
然后我们在lua.c中加入如下函数：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. // This is my function
2. **static** **int** getTwoVar(lua\_State \*L)
3. {
4. // 向函数栈中压入2个值
5. lua\_pushnumber(L, 10);
6. lua\_pushstring(L,"hello");
8. **return** 2;
9. }

在pmain函数中，luaL\_openlibs函数后加入以下代码：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. //注册函数
2. lua\_pushcfunction(L, getTwoVar); //将函数放入栈中
3. lua\_setglobal(L, "getTwoVar");   //设置lua全局变量getTwoVar

通过查找lua.h

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

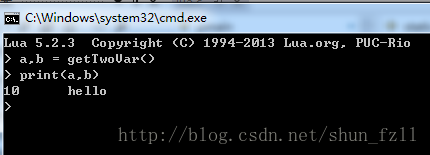
1. /#define lua\_register(L,n,f) (lua\_pushcfunction(L, (f)), lua\_setglobal(L, (n)))

我们发现之前的注册函数可以这样子写：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. lua\_register(L,"getTwoVar",getTwoVar);

运行，结果如图：



当然，一般我们不建议去修改别人的代码，更倾向于自己编写独立的c/c++模块，供lua调用，下面来讲讲如何实现。

**方法二：使用静态依赖的方式**

1.新建一个空的win32控制台工程，记得在vc++目录中，把lua中的头文件和lib文件的目录包含进来，然后->链接器->附加依赖项->将lua51.lib和lua5.1.lib也包含进来。

2.在目录下新建一个avg.lua如下：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

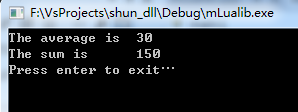
1. avg, sum = average(10, 20, 30, 40, 50)
2. print("The average is ", avg)
3. print("The sum is ", sum)

3.新建test.cpp如下：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. #include <stdio.h>
3. **extern** "C" {
4. #include "lua.h"
5. #include "lualib.h"
6. #include "lauxlib.h"
7. }
9. /\* 指向Lua解释器的指针 \*/
10. lua\_State\* L;
11. **static** **int** average(lua\_State \*L)
12. {
13. /\* 得到参数个数 \*/
14. **int** n = lua\_gettop(L);
15. **double** sum = 0;
16. **int** i;
18. /\* 循环求参数之和 \*/
19. **for** (i = 1; i <= n; i++)
20. {
21. /\* 求和 \*/
22. sum += lua\_tonumber(L, i);
23. }
24. /\* 压入平均值 \*/
25. lua\_pushnumber(L, sum / n);
26. /\* 压入和 \*/
27. lua\_pushnumber(L, sum);
28. /\* 返回返回值的个数 \*/
29. **return** 2;
30. }
32. **int** main ( **int** argc, **char** \*argv[] )
33. {
34. /\* 初始化Lua \*/
35. L = lua\_open();
37. /\* 载入Lua基本库 \*/
38. luaL\_openlibs(L);
39. /\* 注册函数 \*/
40. lua\_register(L, "average", average);
41. /\* 运行脚本 \*/
42. luaL\_dofile(L, "avg.lua");
43. /\* 清除Lua \*/
44. lua\_close(L);
46. /\* 暂停 \*/
47. printf( "Press enter to exit…" );
48. getchar();
49. **return** 0;
50. }

执行一下，我们可以得到结果：



大概顺序就是：我们在c++中写一个模块函数，将函数注册到lua解释器中，然后由c++去执行我们的lua文件，然后在lua中调用刚刚注册的函数。

看上去很别扭啊有木有。。。接下来介绍一下dll调用方式。

**方法三：使用dll动态链接的方式**

我们先新建一个dll工程，工程名为mLualib。（因此最后导出的dll也为mLualib.dll）

然后编写我们的c++模块，以函数为例，我们先新建一个.h文件和.cpp文件。

h文件如下：（如果你不是很能明白头文件的内容，点击这里：<http://blog.csdn.net/shun_fzll/article/details/39078971>。）

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. #pragma once
3. **extern** "C" {
4. #include "lua.h"
5. #include "lualib.h"
6. #include "lauxlib.h"
7. }
9. #ifdef LUA\_EXPORTS
10. #define LUA\_API \_\_declspec(dllexport)
11. #else
12. #define LUA\_API \_\_declspec(dllimport)
13. #endif
15. **extern** "C" LUA\_API **int** luaopen\_mLualib(lua\_State \*L);//定义导出函数

.cpp文件如下：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. #include <stdio.h>
2. #include "mLualib.h"
4. **static** **int** averageFunc(lua\_State \*L)
5. {
6. **int** n = lua\_gettop(L);
7. **double** sum = 0;
8. **int** i;
10. /\* 循环求参数之和 \*/
11. **for** (i = 1; i <= n; i++)
12. sum += lua\_tonumber(L, i);
14. lua\_pushnumber(L, sum / n);     //压入平均值
15. lua\_pushnumber(L, sum);         //压入和
17. **return** 2;                       //返回两个结果
18. }
20. **static** **int** sayHelloFunc(lua\_State\* L)
21. {
22. printf("hello world!");
23. **return** 0;
24. }
26. **static** **const** **struct** luaL\_Reg myLib[] =
27. {
28. {"average", averageFunc},
29. {"sayHello", sayHelloFunc},
30. {NULL, NULL}       //数组中最后一对必须是{NULL, NULL}，用来表示结束
31. };
33. **int** luaopen\_mLualib(lua\_State \*L)
34. {
35. luaL\_register(L, "ss", myLib);
36. **return** 1;       // 把myLib表压入了栈中，所以就需要返回1
37. }

不理解没关系，我们先编译它，然后新建一个lua文件，在lua中我们这样子来调用：**（调用之前记得把dll文件复制到lua文件目录下）**

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. require "mLualib"
2. local ave,sum = ss.average(1,2,3,4,5)//参数对应堆栈中的数据
3. print(ave,sum)  -- 3 15
4. ss.sayHello()   -- hello world!

成功调用了有木有？我们看到了输出信息。**（木有成功你留言，我教你！）**  
  
至此都发生了什么呢？梳理一下：  
  
1.我们编写了averageFunc求平均值和sayHelloFunc函数，  
2.然后把函数封装myLib数组里面，类型必须是luaL\_Reg  
3.由luaopen\_mLualib函数导出并在lua中注册这两个函数。  
  
那么为什么要这样子写呢？实际上当我们在lua中：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. require "mLualib"

这样子写的时候，lua会这么干：

**[cpp]** [view plain](http://blog.csdn.net/shun_fzll/article/details/39120965) [copy](http://blog.csdn.net/shun_fzll/article/details/39120965)

1. local path = "mLualib.dll"
2. local f = package.loadlib(path,"luaopen\_mLualib")   -- 返回luaopen\_mLualib函数
3. f()                                                 -- 执行

**所以当我们在编写一个这样的模块的时候，编写luaopen\_xxx导出函数的时候，xxx最好是和项目名一样（因为项目名和dll一样）**。  
  
需要注意的是：函数参数里的lua\_State是私有的，每一个函数都有自己的栈。当一个c/c++函数把返回值压入Lua栈以后，该栈会自动被清空。

**五.总结**

这篇文章花了好几天才整理的，最后总结一下就是：  
  
lua和c++是通过一个虚拟栈来交互的。

c++调用lua实际上是：由c++先把数据放入栈中，由lua去栈中取数据，然后返回数据对应的值到栈顶，再由栈顶返回c++。

lua调c++也一样：先编写自己的c模块，然后注册函数到lua解释器中，然后由lua去调用这个模块的函数。

转载请注明出处：<http://blog.csdn.net/shun_fzll/article/details/39120965>