DEC 9TH, 2012

Accessing C++ Objects From Lua

Continuing the post about lua integration with C++. Now to more serious stuff. Let's try to write a wrapper for a `std::list<int>`. Imagine that you have a `std::list<int>` in your C++ that you want to share with the Lua enviroment. So both C++ and Lua can access the list.

Keep in mind, that I'm going to use a `std::list<int>` as an example but you could apply the same idea to any other type: user-defined or builtin.

Basics

First, let's get the `Makefile` in place

```
LUAHOME=$(HOME)/tmp/lua-5.2.1/src

all: sampleluahost

sampleluahost: sampleluahost.cpp
  g++ -g sampleluahost.cpp -llua -L$(LUAHOME) -I$(LUAHOME) -o sampleluahost
```

Lua script

```
function entries(arg) -- iterator
  return function()
        return arg:pop();
        end
end

for i in entries(the_list) do
  io.write("From LUA:  ", i, "\n")
end

for i=1,10 do
  the_list:push(50+i*100);
end
```

This script when executed will empty `the_list` printing its contents and will fill it again with new content. That will illuestrate that Lua can access the underlyint `std::list<int>` backing up `the_list`.

Note that we use the colon notation to call methods on `the_list`. So when I write `arg:pop()` it's tranlated to `arg.pop(arg)`. The first argument to the function will be the object itself (Think on that argument like the implicit `*this` in C++ methods or `self` argument in python).

Note: that I wrote an iterator for the list in lua. That is a function that returns a closure. the `for` will call this returned function over and over until it returns `nil`.

The `arg:pop()` will pop a element from the front of the `std::list<int>` and will return `nil` when the list is empty.

The C++ host application

++

```
#include <lua.hpp>
#include <iostream>
#include <list>
#include <assert.h>

extern "C" {
  static int l_list_push(lua_State *L) { // Push elements from LUA
    assert(lua_gettop(L) == 2); // check that the number of args is exactly 2
    std::list<int> **ud = static_cast<std::list<int> **>(luaL_checkudata(L,1, "ListMT")); // first arg is the list
    int v =luaL_checkint(L,2); // seconds argument is the integer to be pushed to the std::list<int>
    (*ud)->push_back(v); // perform the push on C++ object through the pointer stored in user data
    return 0; // we return 0 values in the lua stack
  }
  static int l_list_pop(lua_State *L) {
    assert(lua_gettop(L) == 1); // check that the number of args is exactly 1
    std::list<int> **ud = static_cast<std::list<int> **>(luaL_checkudata(L, 1, "ListMT")); // first arg is the userdata
    if ((*ud)->empty()) {
      lua_pushnil(L);
      return 1; // if list is empty the function will return nil
    }
    lua_pushnumber(L,(*ud)->front()); // push the value to pop in the lua stack
                                      // it will be the return value of the function in lua
    (*ud)->pop_front(); // remove the value from the list
    return 1; //we return 1 value in the stack
  }
}
class Main
{
public:
  Main();
  ~Main();
  void run();

  /* data */
private:
  lua_State *L;
  std::list<int> theList;
  void registerListType();
  void runScript();
};

Main::Main() {
  L = luaL_newstate();
  luaL_openlibs(L);
}

Main::~Main() {
  lua_close(L);
}

void Main::runScript() {
  lua_settop(L,0); //empty the lua stack
  if(luaL_dofile(L, "./samplescript.lua")) {
    fprintf(stderr, "error: %s\n", lua_tostring(L,-1));
    lua_pop(L,1);
    exit(1);
  }
  assert(lua_gettop(L) = 0); //empty the lua stack
}

void Main::registerListType() {
  std::cout << "Set the list object in lua" << std::endl;
  luaL_newmetatable(L, "ListMT");
  lua_pushvalue(L,-1);
  lua_setfield(L,-2, "__index"); // ListMT .__index = ListMT
  lua_pushcfunction(L, l_list_push);
  lua_setfield(L,-2, "push"); // push in lua will call l_list_push in C++
  lua_pushcfunction(L, l_list_pop);
  lua_setfield(L,-2, "pop"); // pop in lua will call l_list_pop in C++
}
```

```
71
72  void Main::run() {
73    for(unsigned int i = 0; i<10; i++) // add some input data to the list
74      theList.push_back(i*100);
75    registerListType();
76    std::cout << "creating an instance of std::list in lua" << std::endl;
77    std::list<int> **ud = static_cast<std::list<int> **>(lua_newuserdata(L, sizeof(std::list<int> *)));
78    *(ud) = &theList;
79    luaL_setmetatable(L, "ListMT"); // set userdata metatable
80    lua_setglobal(L, "the_list"); // the_list in lua points to the new userdata
81
82    runScript();
83
84    while(!theList.empty()) { // read the data that lua left in the list
85      std::cout << "from C++: pop value " << theList.front() << std::endl;
86      theList.pop_front();
87    }
88
89
90  }
91
92  int main(int argc, char const *argv[])
93  {
94    Main m;
95    m.run();
96    return 0;
97  }
```

The idea is to set up the basic lua enviroment thought the luaL_newstate and luaL_openlibs.

Then we create a metatable with luaL_newmetatable. A metatable is just a regular table that can be associated with lua values such as userdata. The metatable is where Lua goes to search for metamethods. You can see the list of available metamethods in Lua Reference. In this case we define the metamethod `__index`, which is the metamethod used by Lua when in cannot find a given index in a table or userdata. So imagine that `a` is a userdata and we type `a.elem`. `a` has no `elem` in it so it invokes `__index` on a's metatable to see what to do. It's kind of `method_missing` in Ruby if you are familiar with Ruby. Now the `__index` metamethod it's a little bit special in the sense that I doesn't need to be a method/function at all. If Lua finds out that the `__index` field of the metatable is actually a table and not a function it will just use that table to find the key. So going back to `a.elem` example, that will be translated to `getmetatable(a)["__index"].elem`.

We will use the `"ListMT"` metatable to hold the methods for lists. We associate the metatable entries for `push` and `pop` with two static C functions `l_list_push` and `l_list_pop`. This functions must be of type `lua_CFunction`. that is they should take a `lua_State *` as paramter and return an integer. That's how the Lua communicates with C++, via the `lua_State` and its stack.

The functions themselves are quite straighforward. They must be defined as `extern "C"` because Lua is compiled as a C library and it will call all the `lua_CFunction` with a C linkage (that determines the order in which the function parameters will be pushed into the machine's stack, etc.) so we need to make sure that the function that we are generating here can be called from C.

The function are designed so that the first parameters is always the "object" in this case a userdata of type "ListMT". The lua function luaL_checkudata will check that the metatable of the userdata matches and it will provide the pointer to the userdata. When Lua call the funciton the arguments to the functions are always pushed into the stack so that the first parameter lands on the stack position 1. So arguments are easy to address.

Finally the Lua resources are freed with lua_close.

Output

```
1  Set the list object in lua
2  creating an instance of std::list in lua
3  From LUA:  0
4  From LUA:  100
5  From LUA:  200
6  From LUA:  300
7  From LUA:  400
```

```
From LUA:  500
From LUA:  600
From LUA:  700
From LUA:  800
From LUA:  900
from C++: pop value 150
from C++: pop value 250
from C++: pop value 350
from C++: pop value 450
from C++: pop value 550
from C++: pop value 650
from C++: pop value 750
from C++: pop value 850
from C++: pop value 950
from C++: pop value 1050
```

Things to remember

- Understand userdata, metatables and metamethods
- Lua and C++ communicate though the Lua stack
- check the number of arguments with `assert(lua_gettop(L) == x);
- empty the lua stack or assert that it's empty where do you know that the stack should be empty

Posted by Ruben Laguna • Dec 9th, 2012

« Sublime Text 2 integration with RVM and Rspec: Take number 2

Comments

Recent Posts

Accessing C++ objects from Lua

Sublime Text 2 integration with RVM and Rspec: Take number 2

GDB posix_spawn failed on Mac OS X Mountain Lion

First steps in LUA-C++ integration

Sublime Text 2 RVM and RSpec

GitHub Repos

dotfiles

jletty
LDAP server implemented in java

en4j
Java Desktop Client to Evernote

nevernote
Nevernote is an incomplete clone of Evernote designed to run on Linux. This repository is a development fork from original. You can see original distribution from http://nevernote.sourceforge.net/

rosert-petal-parser
An ANTLR based parser for RoseRT petal files

@ecerulm on GitHub

Latest Tweets

Status updating...

On Delicious

ICEpdf - Open Source Java PDF, Java PDF Viewer, Java PDF Rendering, Java PDF Extraction

Länna Sport, Stockholm sportbutik sportaffär golfbutik cykelbutik skor uteliv konfektion alpint

Windsurf sweden

My Delicious Bookmarks »

My Pinboard

Aho/Ullman Foundations of Computer Science

algorithms, theory, ullman, book, automata

GNU Octave

octave, manual, documentation

bubble bobble mcu decapping

bubble, bobble, rom, decapping

My Pinboard Bookmarks »