Lua Reference Book

Tạ Quang Tùng

18 tháng 12 năm 2016

Mục lục

1 Lua C API

1.1 Các hàm luaL

1.1.1 luaL_newstate

lua_State *luaL_newstate();
Tạo một state mới, độc lập. Trả về NULL nếu thiếu bộ nhớ.

1.1.2 lua_close

void lua_close(lua_State *L);
Destroy toàn bộ mọi thứ liên quan đến State

1.1.3 luaL_openlibs

void luaL_openlibs(lua_State *L);
Mở các thư viện chuẩn của Lua

1.1.4 luaL_loadfile

int luaL_loadfile(lua_State *L, const char *path);
Load file từ path lên và compile nó. Nếu có lỗi thì trả về true và push string lỗi lên stack. Thường gọi hàm lua_pcall(L, 0, 0, 0) ngay sau nó để thực hiện phần thân chương trình.
Các thao tác trên tương đương với luaL_dofile(L, path):
(luaL_loadfile(L, path) || lua_pcall(L, 0, 0, 0))

1.1.5 luaL_loadstring

int luaL_loadstring(lua_State *L, const char *s);
Giống như hàm trên loadfile nhưng load một xâu thay vì từ file

1.1.6 luaL_newlib

void luaL_newlib(lua_State *L, const luaL_Reg *list);
Tạo một mảng các hàm từ *list* - thành một module, push nó lên stack (pop vào biến nào đó cần thiết để sử dụng)

1.1.7 luaopen_*

int luaopen_* (lua_State *L);
đó là tên của module. Hàm thường trả về 1 để chỉ số tham số trả về là 1. Bắt nguồn từ từ khóa require "*", Lua sẽ cố gắng load file .dll hoặc .so rồi gọi hàm luaopen_* để lấy một table các hàm (thường được tạo bằng luaL_newlib) từ top stack.
Ví dụ một hàm:

```
int luaopen_test(lua_State *L) {
    luaL_newlib(L, testlib);
```

```
    return 1; // Chỉ định chỉ có 1 tham số trả về
}
```
Với testlib là mảng các luaL_Reg.

1.1.8 luaL_requiref

void luaL_requiref(lua_State *L,
const char *modname,
lua_CFunction openf, int global);

Load một module vào lua_State (trước khi dùng lua_loadfile) như một thư viện (giống như thư viện chuẩn). Set giá trị của **package.loaded[modname]**. Sẽ sử dụng hàm *openf* để lấy module cần thiết (openf giống như luaopen_* ở trên), và vẫn để nguyên giá trị của module trên top stack (pop đi nếu cần thiết).
Thường được sử dụng khi không muốn tạo .dll hay .so
global chỉ định xem có muốn module ở chế độ global hay không.
Ví dụ:
luaL_requiref(L, "test", luaopen_test, 1);

1.2 Kiểu dữ liệu

1.2.1 lua_Number

Tương ứng với double

1.2.2 lua_Integer

Tương ứng với long long

1.2.3 lua_Unsigned

Tương ứng với unsigned long long

1.2.4 lua_CFunction

typedef int (*lua_CFunction)(lua_State *L);
Prototype các hàm dùng để gọi từ Lua, các tham số đầu vào được lấy từ stack, push lần lượt từ trái sang phải, bắt đầu từ index = 1 (stack hoàn toàn mới mỗi lần gọi). Trả về cũng bằng stack, return n; với n là số tham số trả về

1.2.5 luaL_Reg

```
typedef struct luaL_Reg {
    const char *name;
    lua_CFunction func;
} luaL_Reg;
```
Kiểu dữ liệu cho mảng các hàm để được register bởi luaL_register. *name* là tên của hàm, *func* là con trỏ hàm. Bất kì mảng của các luaL_Reg đều phải kết thúc bằng {NULL, NULL}

1.3 Thao tác với stack

Để tham chiếu tới một phần tử trên stack, ta dùng những index. Phần tử đầu tiên trên stack có index = 1, phần tử thứ hai là 2, phần tử cuối cùng có index = -1

1.3.1 lua_pushnil

void lua_pushnil(lua_State *L);

1.3.2 lua_pushboolean

void lua_pushboolean(lua_State *L, int bool);

1.3.3 lua_pushnumber

void lua_pushnumber(lua_State *L, lua_Number n);

1.3.4 lua_pushinteger

void lua_pushinteger(lua_State *L, lua_Integer n);

1.3.5 lua_pushunsigned

void lua_pushunsigned(lua_State *L, lua_Unsigned n);

1.3.6 lua_pushstring

void lua_pushstring(lua_State *L, const char *s);
Lua tạo một bản copy của string, nên không ảnh hưởng đến string đưa vào tham số
Xác định kết thúc bởi null

1.3.7 lua_pushlstring

void lua_pushlstring(lua_State *L, const char *s, size_t len);
Xác định kết thúc xâu bởi len

1.3.8 lua_checkstack

int lua_checkstack(lua_State *L, int size);
Kiểm tra xem stack đã đầy hay chưa, trả về false nếu không thể grow stack tới size đó

1.3.9 lua_is*

int lua_is* (lua_State *L, int index);
Để kiểm tra xem phần tử nào đó trên stack có đúng type mong muốn hay không (Kiểm tra xem nó có thể convert thành kiểu mong muốn không). * có thể là number, string, ...

1.3.10 lua_to*

<type> lua_to* (lua_State *L, int index);
Chuyển giá trị trên stack có index thành giá trị mong muốn. Không làm thay đổi kích thước stack.
* có thể là string, number, integer, boolean, ...

1.3.11 lua_gettop

int lua_gettop(lua_State *L);
Trả về index của top stack, hay kích thước của stack đó.

1.3.12 lua_settop

void lua_settop(lua_State *L, int index);
Thay đổi top index của stack

1.3.13 lua_pushvalue

void lua_pushvalue(lua_State *L, int index);
Sao chép phần tử tại index rồi đẩy nó lên top stack

1.3.14 lua_remove

void lua_remove(lua_State *L, int index);
Remove một phần tử tại index, shift lại stack

1.3.15 lua_insert

void lua_insert(lua_State *L, int index);
Di chuyển phần tử top và chèn nó vào index bằng cách shift các phần tử phía trên index để tạo khoảng trống

1.3.16 lua_replace

void lua_replace(lua_State *L, int index);
Di chuyển phần tử từ top đến vị trí đã cho bằng cách thay thế vị trí đó

1.3.17 lua_pop

#define lua_pop(L, n) lua_settop(L, -1 - (n))

1.4 Thao tác với các biến global và table

1.4.1 lua_getglobal

void lua_getglobal(lua_State *L, const char *name);
Push lên stack phần tử global có tên name.

1.4.2 lua_setglobal

void lua_setglobal(lua_State *L, const char *name);
Pop phần tử của stack và gán nó và biến global tên name

1.4.3 lua_gettable

void lua_getttable(lua_State *L, int index);
Tham số index chỉ vị trí của biến table đã được push lên. Top của stack là key (dưới dạng string). Hàm sẽ pop key đó ra rồi push table[key] lên stack (Hay giá trị đang muốn nhận)

1.4.4 lua_getfield

void lua_getfield(lua_State *L, int index, const char *key);
Push lên stack giá trị table[key], với table nằm trên stack có vị trí index

1.4.5 lua_settable

void lua_settable(lua_State *L, int index);
index chỉ vị trí của biến table trên stack. Top của stack là value, ngay dưới top là key. Nó sẽ thực hiện gán table[key] = value

1.4.6 lua_setfield

void lua_setfield(lua_State *L, int index, const char *key);
index chỉ vị trí của biến table trên stack. Top của stack là value. Nó sẽ thực hiện phép gán table[key] = value

1.4.7 lua_newtable

void lua_newtable(lua_State *L);
Tạo một empty table và push nó lên stack

1.5 Thao tác gọi hàm

1.5.1 lua_call

void lua_call(lua_State *L, int nargs, int nresults);
Gọi một hàm:

- nargs: số tham số đầu vào
- nresults: số tham số trả về

- Trước khi gọi hàm phải push hàm đó lên stack bằng gọi lua_getglobal. Sau đó push các tham số đầu vào, theo thứ tự từ trái sang phải
- Hàm sẽ pop lấy các tham số đầu vãn, đồng thời pop cả biến hàm đã push trước đó (giá trị để xác định gọi hàm nào). Gọi hàm đó và push các giá trị trả về lên stack theo thứ tự từ trái sang phải.
- Bất kì một lỗi nào xảy ra trong lua_call sẽ được nhảy ra nhờ longjmp

1.5.2 lua_pcall

int lua_pcall(lua_State *L, int nargs, nresults, int errfunc);
Gọi một hàm trong chế độ được bảo vệ.

- nargs: số tham số đầu vào
- nresults: số tham số trả về

- Hàm sẽ pop lấy các tham số đầu vào, đồng thời pop luôn cả giá trị biến hàm đã push trước đó. Thực hiện gọi hàm và push các giá trị trả về lên stack.
- Nếu có bất kì lỗi nào, hàm sẽ catch lỗi đó, push error message lên stack và trả về một error code (khác 0).
- errfunc là vị trí của hàm sẽ handle error trên stack, 0 nếu không chỉ định hàm nào.

Các mã lỗi:

- LUA_ERRRUN: lỗi runtime.
- LUA_ERRMEM: lỗi cấp phát bộ nhớ. (Sẽ không gọi errfunc).
- LUA_ERRERR: lỗi trong khi gọi hàm errfunc.

1.5.3 lua_pushcfunction

void lua_pushcfunction(lua_State *L, lua_CFunction func);
Đẩy func lên stack (pop nó ra một biến global nào đó để có thể sử dụng).

1.6 Gọi một hàm Lua từ C

Các thao tác để gọi một hàm từ C:

1. Push hàm đó lên trên stack bằng lua_getglobal.
2. Push các tham số đầu vào lên stack theo thứ tự từ trái sang phải, dùng các hàm lua_push*.
3. Gọi một trong các hàm lua_pcall, lua_call với các tham số cần thiết.
4. Lấy ra các giá trị trả về nhờ các hàm lua_to*.
5. Pop n giá trị trả về.

1.7 Gọi một hàm C từ Lua

Các bước để gọi một hàm C từ Lua:

1. Định nghĩa các hàm C theo chuẩn của lua_CFunction:
   (a) Lấy các tham số nhờ các hàm lua_to*
   (b) Pop n tham số
   (c) Thực hiện hàm

(d) Push các giá trị trả về

(e) Return số giá trị trả về

2. Đăng kí các hàm đó nhờ một trong các cách:

   - Sử dụng lua_pushfunction để gán nó vào một biến global nào đó.
   - Sử dụng luaL_newlib để push một table các hàm lên rồi gán vào một biến global nào đó.
   - Sử dụng luaL_requiref để đăng kí một module .
   - Sử dụng .dll hoặc .so nhờ định nghĩa hàm luaopen_*.

3. Gọi hàm đó từ trong mã Lua