luaL\_openlibs加载lua基本库（linit.c）

typedef struct luaL\_Reg {

const char \*name;

lua\_CFunction func;

} luaL\_Reg;

static const luaL\_Reg **lualibs**[] = {

{"", luaopen\_base},

{LUA\_LOADLIBNAME, luaopen\_package},

{LUA\_TABLIBNAME, luaopen\_table},

{LUA\_IOLIBNAME, luaopen\_io},

{LUA\_OSLIBNAME, luaopen\_os},

{LUA\_STRLIBNAME, luaopen\_string},

{LUA\_MATHLIBNAME, luaopen\_math},

{LUA\_DBLIBNAME, luaopen\_debug},

{NULL, NULL}

};

#define LUA\_FILEHANDLE "FILE\*"

#define LUA\_COLIBNAME "coroutine"

LUALIB\_API int (luaopen\_base) (lua\_State \*L);

#define LUA\_TABLIBNAME "table"

LUALIB\_API int (luaopen\_table) (lua\_State \*L);

#define LUA\_IOLIBNAME "io"

LUALIB\_API int (luaopen\_io) (lua\_State \*L);

#define LUA\_OSLIBNAME "os"

LUALIB\_API int (luaopen\_os) (lua\_State \*L);

#define LUA\_STRLIBNAME "string"

LUALIB\_API int (luaopen\_string) (lua\_State \*L);

#define LUA\_MATHLIBNAME "math"

LUALIB\_API int (luaopen\_math) (lua\_State \*L);

#define LUA\_DBLIBNAME "debug"

LUALIB\_API int (luaopen\_debug) (lua\_State \*L);

#define LUA\_LOADLIBNAME "package"

LUALIB\_API int (luaopen\_package) (lua\_State \*L);

关键函数

LUALIB\_API void luaL\_openlibs (lua\_State \*L) {

const luaL\_Reg \*lib = **lualibs**;

for (; lib->func; lib++) {

lua\_pushcfunction(L, lib->func); **// 实际在数据栈中添加了一个闭包**

lua\_pushstring(L, lib->name);

lua\_call(L, 1, 0);

}

}

**1）压入C函数**

#define lua\_pushcfunction(L,f) lua\_pushcclosure(L, (f), 0)

LUA\_API void lua\_pushcclosure (lua\_State \*L, lua\_CFunction fn, int n) {

Closure \*cl;

lua\_lock(L);

luaC\_checkGC(L);

api\_checknelems(L, n);

cl = **luaF\_newCclosure**(L, n, **getcurrenv**(L));

cl->c.f = fn; **// 函数指针赋值**

L->top -= n;

while (n--) **// 将n个参数放到upvalue中**

setobj2n(L, &cl->c.upvalue[n], L->top+n); **// top实际上是一个TValue的指针，upvalue是TValue**

**setclvalue**(L, L->top, cl); **// 此时cl已经包含了一个函数指针以及多个参数，压入到栈顶**

lua\_assert(iswhite(obj2gco(cl)));

api\_incr\_top(L);

lua\_unlock(L);

}

static Table \*getcurrenv (lua\_State \*L) {

if (L->ci == L->base\_ci) /\* no enclosing function? \*/

return hvalue(gt(L)); /\* use global table as environment \*/

else {

Closure \*func = curr\_func(L);

return func->c.env;

}

}

Closure \*luaF\_newCclosure (lua\_State \*L, int nelems, Table \*e) {

Closure \*c = cast(Closure \*, luaM\_malloc(L, sizeCclosure(nelems)));

**luaC\_link(L, obj2gco(c), LUA\_TFUNCTION); // 将当前的Closure添加到rootgc链表（实际上是global\_state的变量）中**

c->c.isC = 1;

c->c.env = e;

c->c.nupvalues = cast\_byte(nelems);

return c;

}

void luaC\_link (lua\_State \*L, GCObject \*o, lu\_byte tt) {

global\_State \*g = G(L);

**o->gch.next = g->rootgc;**

g->rootgc = o;

o->gch.marked = luaC\_white(g);

o->gch.tt = tt;

}

#define setclvalue(L,obj,x) \

{ TValue \*i\_o=(obj); \

i\_o->value.gc=cast(GCObject \*, (x)); i\_o->tt=LUA\_TFUNCTION; \

checkliveness(G(L),i\_o); }

再看看闭包的定义：

#define ClosureHeader \

CommonHeader; lu\_byte isC; lu\_byte nupvalues; GCObject \*gclist; \

struct Table \*env // 环境

typedef struct CClosure {

ClosureHeader;

lua\_CFunction f;

TValue upvalue[1];

} CClosure;

typedef struct LClosure {

ClosureHeader;

struct Proto \*p;

UpVal \*upvals[1];

} LClosure;

typedef union Closure {

CClosure c;

LClosure l;

} Closure;

**2）压入字符串**

LUA\_API void lua\_pushstring (lua\_State \*L, const char \*s) {

if (s == NULL)

lua\_pushnil(L);

else

lua\_pushlstring(L, s, strlen(s));

}

LUA\_API void lua\_pushnil (lua\_State \*L) {

lua\_lock(L);

setnilvalue(L->top);

api\_incr\_top(L);

lua\_unlock(L);

}

#define setnilvalue(obj) ((obj)->tt=LUA\_TNIL)

LUA\_API void lua\_pushlstring (lua\_State \*L, const char \*s, size\_t len) {

lua\_lock(L);

luaC\_checkGC(L);

setsvalue2s(L, L->top, luaS\_newlstr(L, s, len));

api\_incr\_top(L);

lua\_unlock(L);

}

#define setsvalue(L,obj,x) \

{ TValue \*i\_o=(obj); \

i\_o->value.gc=cast(GCObject \*, (x)); i\_o->tt=LUA\_TSTRING; \

checkliveness(G(L),i\_o); }

**3）lua\_call**

LUA\_API void lua\_call (lua\_State \*L, int nargs, int nresults) {

StkId func;

lua\_lock(L);

api\_checknelems(L, nargs+1);

checkresults(L, nargs, nresults);

func = L->top - (nargs+1);

**luaD\_call(L, func, nresults);**

adjustresults(L, nresults);

lua\_unlock(L);

}

/\*

\*\* Call a function (C or Lua). The function to be called is at \*func.

\*\* The arguments are on the stack, right after the function.

\*\* When returns, all the results are on the stack, starting at the original

\*\* function position.

\*/

void luaD\_call (lua\_State \*L, StkId func, int nResults) {

if (++L->nCcalls >= LUAI\_MAXCCALLS) {

if (L->nCcalls == LUAI\_MAXCCALLS)

luaG\_runerror(L, "C stack overflow");

else if (L->nCcalls >= (LUAI\_MAXCCALLS + (LUAI\_MAXCCALLS>>3)))

luaD\_throw(L, LUA\_ERRERR); /\* error while handing stack error \*/

}

if (luaD\_precall(L, func, nResults) == PCRLUA) /\* is a Lua function? \*/

luaV\_execute(L, 1); /\* call it \*/

L->nCcalls--;

luaC\_checkGC(L);

}

添加自定义的lib

#define luaI\_openlib luaL\_openlib

static luaL\_Reg mylib[] = {

{ "my\_sin", my\_sin },

{ NULL, NULL }

};

luaL\_openlib(L, "mylib", mylib, 0);

LUALIB\_API void luaI\_openlib (lua\_State \*L, const char \*libname,

const luaL\_Reg \*l, int nup) {

if (libname) {

int size = libsize(l); **// 获取lib中有多少个函数组**

/\* check whether lib already exists \*/

luaL\_findtable(L, LUA\_REGISTRYINDEX, "\_LOADED", 1); **// 在注册表中查找\_LOADED表，如果没有则创建一个\_LOADED表，并放在注册表中，最后\_LOADED表会放在栈顶**

lua\_getfield(L, -1, libname); /\* get \_LOADED[libname] \*/ **// 在\_LOADED表中查找是否有libname表**

if (!lua\_istable(L, -1)) { /\* not found? \*/

lua\_pop(L, 1); /\* remove previous result \*/

/\* try global variable (and create one if it does not exist) \*/

if (luaL\_findtable(L, LUA\_GLOBALSINDEX, libname, size) != NULL)

luaL\_error(L, "name conflict for module " LUA\_QS, libname);

lua\_pushvalue(L, -1);

lua\_setfield(L, -3, libname); /\* \_LOADED[libname] = new table \*/

}

lua\_remove(L, -2); /\* remove \_LOADED table \*/ **// 移除\_LOADED表，此时就剩余libname表**

lua\_insert(L, -(nup+1)); /\* move library table to below upvalues \*/ **// 将libname表放到nup最底下**

}

for (; l->name; l++) {

int i;

for (i=0; i<nup; i++) /\* copy upvalues to the top \*/

lua\_pushvalue(L, -nup);

lua\_pushcclosure(L, l->func, nup);

lua\_setfield(L, -(nup+2), l->name);

}

lua\_pop(L, nup); /\* remove upvalues \*/

}

**部分api对栈的影响：**

lua\_settable -2 lua\_gettable 0

lua\_rawset -2 lua\_rawget 0

lua\_setfield -1 lua\_getfield +1

lua\_rawseti -1 lua\_rawgeti +1