<https://www.jianshu.com/p/f415697fd952?utm_campaign=maleskine&utm_content=note&utm_medium=pc_all_hots&utm_source=recommendation>

<http://www.cnblogs.com/renyuan/p/6812206.html>

<https://blog.csdn.net/chenjiayi_yun/article/details/8877235>

<https://blog.csdn.net/maximuszhou/article/details/21331819>

struct lua\_State {

CommonHeader;

lu\_byte status;

StkId top; /\* first free slot in the stack \*/

StkId base; /\* base of current function \*/

global\_State \*l\_G;

CallInfo \*ci; /\* call info for current function \*/

const Instruction \*savedpc; /\* `savedpc' of current function \*/

StkId stack\_last; /\* last free slot in the stack \*/

StkId stack; /\* stack base \*/

CallInfo \*end\_ci; /\* points after end of ci array\*/

CallInfo \*base\_ci; /\* array of CallInfo's \*/

int stacksize;

int size\_ci; /\* size of array `base\_ci' \*/

unsigned short nCcalls; /\* number of nested C calls \*/

unsigned short baseCcalls; /\* nested C calls when resuming coroutine \*/

lu\_byte hookmask;

lu\_byte allowhook;

int basehookcount;

int hookcount;

lua\_Hook hook;

TValue l\_gt; /\* table of globals \*/

TValue env; /\* temporary place for environments \*/

GCObject \*openupval; /\* list of open upvalues in this stack \*/

GCObject \*gclist;

struct lua\_longjmp \*errorJmp; /\* current error recover point \*/

ptrdiff\_t errfunc; /\* current error handling function (stack index) \*/

};

typedef struct global\_State {

stringtable strt; /\* hash table for strings \*/

lua\_Alloc frealloc; /\* function to reallocate memory \*/

void \*ud; /\* auxiliary data to `frealloc' \*/

lu\_byte currentwhite;

lu\_byte gcstate; /\* state of garbage collector \*/

int sweepstrgc; /\* position of sweep in `strt' \*/

GCObject \*rootgc; /\* list of all collectable objects \*/

GCObject \*\*sweepgc; /\* position of sweep in `rootgc' \*/

GCObject \*gray; /\* list of gray objects \*/

GCObject \*grayagain; /\* list of objects to be traversed atomically \*/

GCObject \*weak; /\* list of weak tables (to be cleared) \*/

GCObject \*tmudata; /\* last element of list of userdata to be GC \*/

Mbuffer buff; /\* temporary buffer for string concatentation \*/

lu\_mem GCthreshold;

lu\_mem totalbytes; /\* number of bytes currently allocated \*/

lu\_mem estimate; /\* an estimate of number of bytes actually in use \*/

lu\_mem gcdept; /\* how much GC is `behind schedule' \*/

int gcpause; /\* size of pause between successive GCs \*/

int gcstepmul; /\* GC `granularity' \*/

lua\_CFunction panic; /\* to be called in unprotected errors \*/

TValue l\_registry;

struct lua\_State \*mainthread;

UpVal uvhead; /\* head of double-linked list of all open upvalues \*/

struct Table \*mt[NUM\_TAGS]; /\* metatables for basic types \*/

TString \*tmname[TM\_N]; /\* array with tag-method names \*/

} global\_State;

在lua\_newstate函数中

LUA\_API lua\_State \*lua\_newstate (lua\_Alloc f, void \*ud) {

int i;

lua\_State \*L;

global\_State \*g;

**void \*l = (\*f)(ud, NULL, 0, state\_size(LG));**

if (l == NULL) return NULL;

L = tostate(l);

g = &((LG \*)L)->g;

L->next = NULL;

L->tt = LUA\_TTHREAD;

g->currentwhite = bit2mask(WHITE0BIT, FIXEDBIT);

L->marked = luaC\_white(g);

set2bits(L->marked, FIXEDBIT, SFIXEDBIT);

**preinit\_state(L, g);**

g->frealloc = f;

g->ud = ud;

g->mainthread = L;

g->uvhead.u.l.prev = &g->uvhead;

g->uvhead.u.l.next = &g->uvhead;

g->GCthreshold = 0; /\* mark it as unfinished state \*/

g->strt.size = 0;

g->strt.nuse = 0;

g->strt.hash = NULL;

setnilvalue(registry(L));

luaZ\_initbuffer(L, &g->buff);

g->panic = NULL;

g->gcstate = GCSpause;

g->rootgc = obj2gco(L);

g->sweepstrgc = 0;

g->sweepgc = &g->rootgc;

g->gray = NULL;

g->grayagain = NULL;

g->weak = NULL;

g->tmudata = NULL;

g->totalbytes = sizeof(LG);

g->gcpause = LUAI\_GCPAUSE;

g->gcstepmul = LUAI\_GCMUL;

g->gcdept = 0;

for (i=0; i<NUM\_TAGS; i++) g->mt[i] = NULL;

if (luaD\_rawrunprotected(L, **f\_luaopen**, NULL) != 0) {

/\* memory allocation error: free partial state \*/

close\_state(L);

L = NULL;

}

else

luai\_userstateopen(L);

return L;

}

通过lua\_newState创建一个新的lua虚拟机时，第一块申请的内存用来保存主线程和这个全局状态机。Lua的实现尽可能的避免内存碎片，同时也减少内存分配和释放的次数，它采用了一个小技巧，利用一个LG结构，把主线程lua\_state和global\_state分配在一起。

static void preinit\_state (lua\_State \*L, global\_State \*g) {

G(L) = g; **// 给lua\_State中的global\_State赋值**

L->stack = NULL;

L->stacksize = 0;

L->errorJmp = NULL;

L->hook = NULL;

L->hookmask = 0;

L->basehookcount = 0;

L->allowhook = 1;

resethookcount(L);

L->openupval = NULL;

L->size\_ci = 0;

L->nCcalls = L->baseCcalls = 0;

L->status = 0;

L->base\_ci = L->ci = NULL;

L->savedpc = NULL;

L->errfunc = 0;

setnilvalue(gt(L));

}

static void f\_luaopen (lua\_State \*L, void \*ud) {

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

UNUSED(ud);

**stack\_init(L, L)**; /\* init stack \*/

**sethvalue(L, gt(L), luaH\_new(L, 0, 2))**; /\* table of globals，实际上是lua\_state中的l\_gt \*/

**sethvalue(L, registry(L), luaH\_new(L, 0, 2))**; /\* registry，实际上是global\_state中的l\_registry \*/

**luaS\_resize(L, MINSTRTABSIZE)**; /\* initial size of string table，**MINSTRTABSIZE = 32** \*/

**luaT\_init(L); // 元表TString**

**luaX\_init(L); // 保留字符串TString**

**luaS\_fix(luaS\_newliteral(L, MEMERRMSG));// 设置no enough memory提示字符串TString不可回收**

g->GCthreshold = 4\*g->totalbytes;

}

static void stack\_init (lua\_State \*L1, lua\_State \*L) {

**/\* initialize CallInfo array \*/**

L1->base\_ci = luaM\_newvector(L, BASIC\_CI\_SIZE, CallInfo);

L1->ci = L1->base\_ci;

L1->size\_ci = BASIC\_CI\_SIZE; **// BASIC\_CI\_SIZE = 8**

L1->end\_ci = L1->base\_ci + L1->size\_ci - 1;

**/\* initialize stack array \*/**

L1->stack = luaM\_newvector(L, BASIC\_STACK\_SIZE + EXTRA\_STACK, TValue); **// 实际stack是指向TValue的指针，BASIC\_STACK\_SIZE = 40**

L1->stacksize = BASIC\_STACK\_SIZE + EXTRA\_STACK;

L1->top = L1->stack;

L1->stack\_last = L1->stack+(L1->stacksize - EXTRA\_STACK)-1;

**/\* initialize first ci \*/**

L1->ci->func = L1->top;

setnilvalue(L1->top++); /\* `function' entry for this `ci' \*/

L1->base = L1->ci->base = L1->top;

L1->ci->top = L1->top + LUA\_MINSTACK;

}