<http://blog.csdn.net/maximuszhou/article/details/44755517>

Lua将表示成带标识的联合结构，代码如下（lobject.h）

typedef union {

GCObject \*gc;

void \*p;

lua\_Number n;

int b;

} Value;

#define TValuefields Value value; int tt

typedef struct lua\_TValue {

TValuefields;

} TValue;

可以看出TValue有两个成员：

一个是整数tt用来表示数据类型，lua中所有的数据类型定义如下（lua.h）：

#define LUA\_TNONE (-1)

#define LUA\_TNIL 0

#define LUA\_TBOOLEAN 1

#define LUA\_TLIGHTUSERDATA 2

#define LUA\_TNUMBER 3

#define LUA\_TSTRING 4

#define LUA\_TTABLE 5

#define LUA\_TFUNCTION 6

#define LUA\_TUSERDATA 7

#define LUA\_TTHREAD 8

可以看到实现了9种数据类型，其中把语法中userdata分为LUA\_TLIGHTUSERDATA和LUA\_TUSERDATA，其中前一种类型即为轻量级userdata，轻量级userdata是一种表示C指针的值，对Lua虚拟机来说，这种数据类型不需要GC，其指向的内存由用户分配和释放；后一种userdata类型为完全userdata，内存是由Lua虚拟机分配，并有GC机制负责处理。

在lua中，一个变量使用TValue这个类型来存储，int tt就是类型（占4个字节），而value则是一个union（占8个字节），因此通常来说一个TValue就占用12个字节，考虑到字节对齐，它占用了16个字节

lua\_Number n 实际上被lua定义为double

int b boolean值

void \*p light userdata轻量级用户数据

GCObject \*gc 用来指向那些需要垃圾回收的对象，包括string、table、function、完全userdate和thread类型，GCObject用来表示可以垃圾回收的对象，它也是一个联合体（lstate.h）

union GCObject {

GCheader gch;

union TString ts;

union Udata u;

union Closure cl;

struct Table h;

struct Proto p;

struct UpVal uv;

struct lua\_State th; /\* thread \*/

};

其中成员GCHeader主要用于GC垃圾回收，其他成员比如TString等等才是真正存储值的结构，而这些数据结构也会有GCHeader，用于GC管理

typedef struct GCheader {

CommonHeader;

} GCheader;

#define CommonHeader GCObject \*next; lu\_byte tt; lu\_byte marked

最后，关于lua的数据类型实现，值得指出使用带标志的结构体来表示lua的数值类型，使得lua中任何一种类型至少占用的空间是16个字节（结构体还需要对齐），因此，拷贝lua值是比较耗时的

关于nil：

首先我们可以看看table中的luaH\_getnum函数：

const TValue \*luaH\_getnum (Table \*t, int key) {

/\* (1 <= key && key <= t->sizearray) \*/

if (cast(unsigned int, key-1) < cast(unsigned int, t->sizearray))

return &t->array[key-1];

else {

lua\_Number nk = cast\_num(key);

Node \*n = hashnum(t, nk);

do { /\* check whether `key' is somewhere in the chain \*/

if (ttisnumber(gkey(n)) && luai\_numeq(nvalue(gkey(n)), nk))

return gval(n); /\* that's it \*/

else n = gnext(n);

} while (n);

return luaO\_nilobject;

}

}

可以看到，如果没有找到则返回一个luaO\_nilobject对象（lobject.h与lobject.c）

#define luaO\_nilobject (&luaO\_nilobject\_)

LUAI\_DATA const TValue luaO\_nilobject\_;

const TValue luaO\_nilobject\_ = {{NULL}, LUA\_TNIL};

也就是说所有的nil均指向同一个对象luaO\_nilobject\_

cout << sizeof(luaO\_nilobject\_) << endl;

cout << sizeof(luaO\_nilobject) << endl;

cout << luaO\_nilobject << endl;



**个人总结：**

GCObject \*gc; -string/table/userdata/thread/function

void \*p; -lightuserdata

lua\_Number n; -number

int b; -boolean