Recommended reading order:

lmathlib.c, lstrlib.c: get familiar with the external C API. Don't bother with the pattern matcher though. Just the easy functions.

lapi.c: Check how the API is implemented internally. Only skim this to get a feeling for the code. Cross-reference to lua.h and luaconf.h as needed.

lobject.h: tagged values and object representation. skim through this first. you'll want to keep a window with this file open all the time.

lstate.h: state objects. ditto.

lopcodes.h: bytecode instruction format and opcode definitions. easy.

lvm.c: scroll down to luaV_execute, the main interpreter loop. see how all of the instructions are implemented. skip the details for now. reread later.

ldo.c: calls, stacks, exceptions, coroutines. tough read.

lstring.c: string interning. cute, huh?

ltable.c: hash tables and arrays. tricky code.

ltm.c: metamethod handling, reread all of lvm.c now.

You may want to reread lapi.c now.

ldebug.c: surprise waiting for you. abstract interpretation is used to find object names for tracebacks. does bytecode verification, too.

lparser.c, lcode.c: recursive descent parser, targetting a register-based VM. start from chunk() and work your way through. read the expression parser and the code generator parts last.

lgc.c: incremental garbage collector. take your time.

Read all the other files as you see references to them. Don't let your stack

英文比较简单，我就不翻译了。

今天已经读了 lmathlib.c, lstrlib.c, lapi.c 这三个文件。

对于 lmathlib.c 和 lstrllib.c 这个两个文件主要是熟悉如何扩展 Lua，一般使用过 Lua 的人都对这个会有所了解。只是简单的读一读、熟悉代码风格，不用花太多时间（我只看了 20 分钟）

lapi.c 这里就开始有点难度了， 主要是你这里有一些平时没有接触过的东西。阅读这个文件可以结合手册中第三章 C API 来进行。

由于 lapi.c 涉及到很多的宏和结构，建议装一个 vim + ctags 或 Visual studio 2010 做 cross references。

以下是一些关于 lapi.c 笔记:

TValue:由 Value 和 tt 组成， tt 为类型标识，等于 LUA_TNIL 的呢个 Value 是一个联合体，根据 tt 的值不同使用不同的值，lua 中所有的值、对象，都是由这个结构储存。以下是 Value 联合体的定义:

```
typedef union {
  GCObject *gc;
  void *p;
  lua_Number n;
  int b;
} Value;
```

GCObject：里面的内容比较复杂，从字面上理解是指就是 gc 会收集的对象，

可以理解为 Lua 中的基本对象。包括 Thread,Table,UserData,String 都是存储在这个对象。

void* p: 就是指针 对应的内容就是 api 中的 lightuserdata, 这个可以从 lua_pushlightuserdata

和 lua_touserdata 中得到证明。由此，可以知道 lightuserdata 和 number, integer 有相同的行为

index2adr 函数：将栈索引转换成 StkId，伪索引（GLOBAL Index, REGISTRY Index 等）

也是在这里处理。几乎所有 API 都需要调用这个。

StkId：字面意思为栈 ID，实际为 TValue*，使用地址作为一标识。可以想象，

lua_State 中的栈是一个 TValue 数组。

阅读 lapi.c 的时间也不需要太长，不需要太深入，遇到与虚拟机有关的函数，直接 step over 就可以了。

Lua 源代码阅读(2)

作者：future0906 发布时间：May 16, 2011 分类：

这次的分析主要是 lobject.h 文件和 lstate.h 文件。

lobject.h 充满了关于 lua 对象如何在 C 中实现的线索。如果你对这十分好奇，可以详细读一下。

lstate.h 中重要结构体有三个 lua_State,global_State 和 GCObject。

Lua 中用到大量 C 的联合体和结构体组合来巧妙的组织代码。

GCheader 结构体 是所有 GC 可以收集的对象的基础属性。这里类似于 C++中的多态（所有的 Lua 对象在 C 里面都是用这个结构体存储）。

```
GCheader {
      CommonHeader;
}
```

GCObject 结构体,TString, Udata, Closure, Table, Proto, UpVal, lua_State, 都包含一个 CommonHeader

```
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 */
};
```

这是一个非常重要的技巧，可以用 C 来实现多态。

另外，可以留意到的是，上述 GCObject 包含的类型在加上基本类型(NIL, BOOLEAN, LIGHTUSERDATA, NUMBER)Lua 中的所有类型。

TString:从字面上就是字符串

Udata: 用户数据

Closure: 闭包数据(如何实现？)

Table: 表格

UpVal: 函数外局部变量

lua_State: lua 状态对象

所有的对象值都保存为 TValue 结构体

```
typedef struct lua_TValue {
    TValuefields;
} TValue;
#define TValuefields    Value value; int tt
```

根据 tt 的值不同 value 会有不同的含义，这个算多态的表现

这就是 LUA 基本类型的编号,与 CommonHeader 中的 tt 对应

```
#define LUA_TNONE          (-1)
#define LUA_TNIL           0
#define LUA_TBOOLEAN       1
#define LUA_TLIGHTUSERDATA2
#define LUA_TNUMBER        3
```

#define LUA_TSTRING 4

#define LUA_TTABLE 5

#define LUA_TFUNCTION 6

#define LUA_TUSERDATA 7

#define LUA_TTHREAD 8

每个 Lua 虚拟机实例都包含一个 global_state 和一个 lua_State

global_state 对于 C API 来说是不可见，每个 lua_State 都有一个指向 global_state 的指针

global_state 包含 gc 信息等。和 global_state 一起创建的 lua_State 称为 MainThread。

lua_State 中的 stack 由 stack_init 初始化(申请内存等),可以由 luaE_newthread 和 f_luaopen 调用，初始栈的大小为 20*2 +5

标签: none