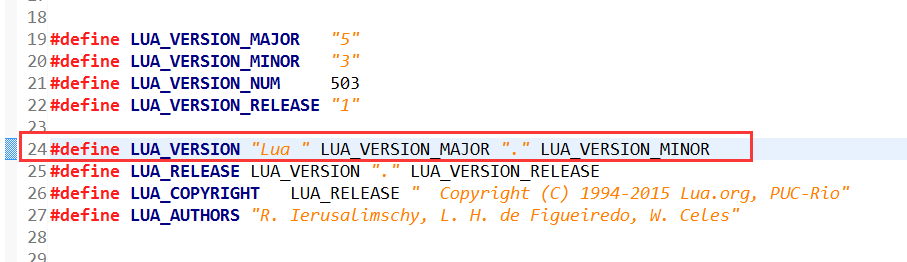
**Lua虚拟机学习笔记**

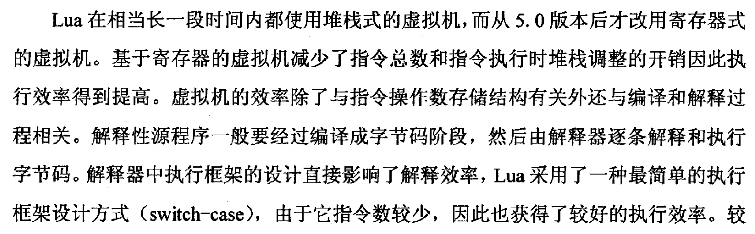
Lua是一种比较轻巧而且快捷的脚本语句，语法简单，但应用很广，很容易扩展。虚拟币后台系统通过官方发布的开源Lua解释器（lua 5.3），解释执行app.lua脚本文件实现了开发者的合约交易应用。

1、我们用的LUA解释器版本：lua 5.3



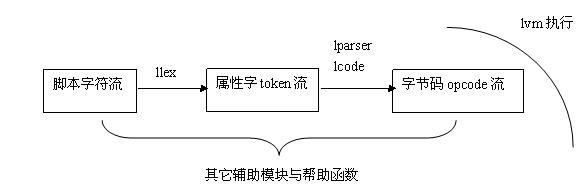
参考：http://www.doc88.com/p-6681238341344.html

2、lua在相当长一段时间内都使用堆栈式虚拟机，而从5.0版本后才改用寄存器式的虚拟机。



3、大概了解一下代码结构：

可以看到lua的一些头文件.h和源代码文件.c，很多文件都是相对应的，比如lcode.c就有一个lcode.h与它同名。所以我们这里先解释每个h文件的作用，至于它的实现可以参照相应同名的c文件。lapi是Lua API的辅助函数。lcode是Lua的代码生成函数，Lua在用语法分析器分析脚本高级语言的时候就会调用lcode里面的函数来生成lua的字节码（opcode）。ldebug是调试模块的辅助函数。ldo是Lua的堆栈和函数调用等处理函数。lfunc有一些处理脚本函数和闭包的辅助函数。lgc是垃圾收集函数。llex是词法分析器，从脚本源代码的字符流分解出属性字（token）流，llimit是一些类型定义和系统依赖定义，相当于MFC的stdafx文件。lmem是内存管理模块，封装了一些内存的分配、释放等函数。lobject是Lua基本对象定义，包括运行时lua对象和脚本函数、哈希表等都在这里定义。lopcodes是Lua的字节码（指令）定义，脚本的源代码都会汇编成字节码给虚拟机执行。lparser是语法分析器，把属性字（token）流分析成一个个Lua的语句单位并调用代码生成函数生成字节码。lstate是全局状态类型定义；lstring是字符串表定义；ltable是lua表定义；ltm是一些标签函数；而lundump是预编译lua块。lvm是本文最重要的模块——lua虚拟机。lzio处理一些字符串流。主要流程可以看下图：



字节码介绍：

 Lua4的字节码是一个32位的无符号整数，它用了低6位来保存操作码，高位来放一些简单的操作数，根据操作数的个数和类型，基本上有四种格式，一种是没有操作数的，还有三种是有不同类型操作数的：

我们可以看一下字节码的定义，可以看到它大概有以下几种类型：

1、函数调用返回等，比如：OP\_CALL。

2、堆栈处理，比如：OP\_PUSHINT。

3、变量和表的获取与设置，比如：OP\_GETLOCAL等。

4、算术运算，比如：OP\_ADD等。

5、跳转判断和循环，比如：OP\_JMP等。

在不同的指令里，操作数有不同的作用，比如在OP\_GETLOCAL中，有一个无符号参数U，它表示局部变量在函数局部变量内存空间的索引。这样你就可以通过这个操作数来得到这个局部变量确切的地址了。我们可以通过GETARG\_U和SETARG\_U等宏来获取和设置这些指令的操作数。

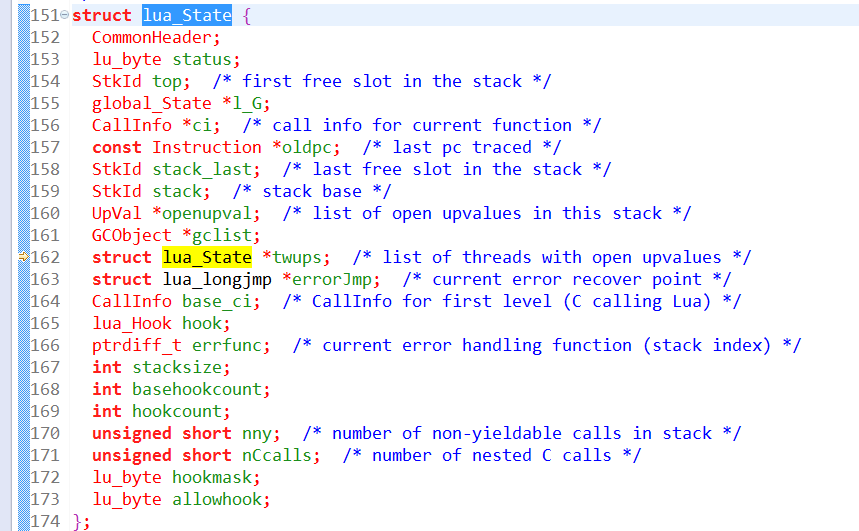
### 4、Lua\_State结构

### 一个很重要的结构是Lua\_State，它是Lua的全局状态结构，当你要打开一个Lua脚本本执行之前，你必须调用lua\_open函数打开并初始化一个Lua全局状态，此后的所有工作，包括注册给脚本使用的API函数，打开运行Lua脚本，调用Lua函数等，都要使用这个Lua的全局状态指针为参数，当你执行完Lua后还必须调用lua\_close关闭Lua\_State即可，从这些可以看出这个结构体的重要性。

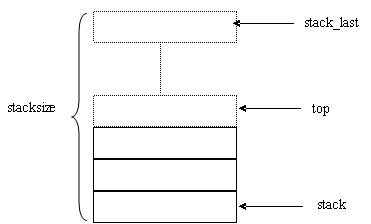
### 

### 

下面我们打开来看一下这个结构的成员。



|  |
| --- |
| 堆栈相关数据 |
| 全局缓冲与其它数据 |
| 全局脚本函数、闭包等 |
| 全局表、字符串表和全局变量等 |
| 垃圾收集与调用跟踪相关数据 |

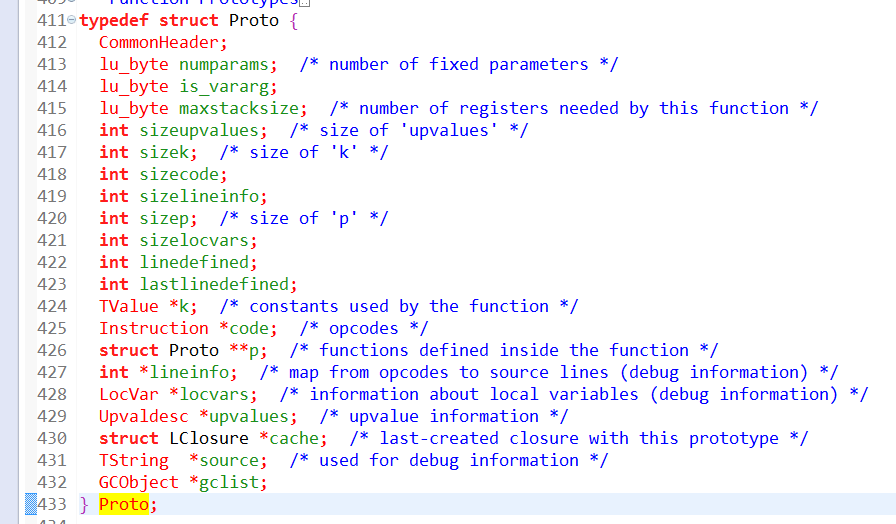
其中堆栈这块数据跟虚拟机关系比较重要，先来看这块，它包括top、stack、stack\_last、stacksize等成员变量，它们的关系如下：  
  


如上图，stack指向栈底，即堆栈最底下的那个元素，top指向栈顶，即即将入栈的元素存放的位置，statcksize表示堆栈最大可能的元素个数，stack\_last表示最后一个允许入栈的元素。

 全局缓冲主要是作为一个临时的缓冲区用，在语法分析等地方会经常用到。全局闭包函数把全局的代码段作为一个主函数，保存，然后脚本里自定义的其它函数作为它的子函数。全局字符串表等保存了脚本里用到的所有字符串，另外还有全局变量等全局数据。

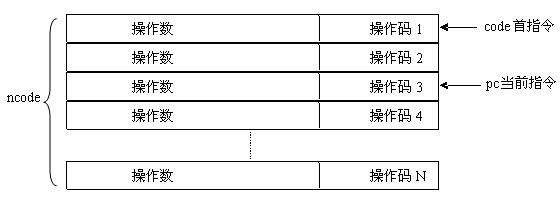
### 5、Proto结构

还有个比较重要的结构是脚本函数Proto结构。它保存了一个脚本函数要用到的所有数据，那些全局作用域的脚本其实也是作为一个脚本函数来处理，其它定义的脚本函数都是作为它的子函数。下面我们来看看这个结构的数据：



|  |
| --- |
| 实数表 knum |
| 字符串表 kstr |
| 子函数表 kproto |
| 指令流 code |
| 参数信息与其它数据 |
| 其它调试相关信息 |

  实数表保存了脚本用到的一些实数，不过整数的话一般不放这个表而是作为一个字节码的立即操作数用；字符串表存放脚本里用的所有字符串，还包括全局变量名和函数名等。这样在遇到存取全局变量和调用函数的时候都用这个字符串当关键字来进行操作。子函数表里保存了这个函数的所有子函数，即函数里定义的函数。指令流就是这个函数的字节码列表了，运行这个函数就是从指令的第一句一直执行到最后一句，当前中间会有一些跳转和循环。下面我们来看看这个结构：



  这样虚拟机执行这个函数的指令就很简单了，找到函数的code变量，然后从第一句开始执行，一直执行到最后一句结束。