**LUA深入浅出**

**目录：LUA深入浅出**

最近做想去了解整个广告市场，因此和老大申请去做一些广告引擎SSP、ADX相关的事情，由于SSP、ADX使用了openresty框架，并且马上要做的要做的移动API也要使用这个框架实现，而这个框架使用了嵌入式语言lua进行业务逻辑的开发，因此学习nginx和LUA变得迫在眉睫了。Lua的语法与javascript很像，但是语法结构有继承了python的有点，使用换行做语句结束符号，没有java那么重的各种集合，使用table完成了所有数据结构的操作，与openresty很好的集成。接下来将由浅入深，从lua的语法开始，到luaVM的原理总结一下所学所感目录：

1. Lua基本语法
2. Lua之Userdata
3. Lua字符串处理
4. Lua之table
5. Lua之模块
6. Lua之IO
7. 协程
8. Lua面向对象
9. Lua异常处理
10. Lua VM原理

接下来的几天将按照目录将坚持高质量的和大家一起分享一下LUA的精彩世界。

**Lua基本语法**

1. **数据类型**

Lua的基本数据类型主要有：nil、boolean、Number、String、function、userdata、threads

**Nil：**空，变量没有赋值，默认值是nil

**Boolean：**Lua中false和nil都看作假的，其它都是真

**Number：**Lua只有一种Number类型-double（双精度类型）

**String:**双引号或单引号，也可以使用[[]] 表示一个字符块，例如：

html = [[

<html>

<head></head>

<body>

<a href="http://blog.csdn.net/code52/article/details/53843804">LUA深入浅出</a>

</body>

</html>

]]

print(html)

结果：

<html>

<head></head>

<body>

<a href="http://blog.csdn.net/code52/article/details/53843804">LUA深入浅出</a>

</body>

</html>

**Function：**作为一个特殊类型，也可以作为参数传递（和javascript一样）

**Userdata：**是用户自定义数据，是用程序或C语言创建的新类型，例如场景，在C语言中创建struct，然后在lua中使用。后面介绍UserData和C语言的结合使用方法。

**Threads：**在lua中使用的是协程，和线程一样有自己独立的栈、局部变量、指令指针，以及可以和其它协程共享全局变量。协程和线程区别：线程可以运行多个，协程序任意时刻只能运行一个，协程只有挂起才会暂停。具体协程的应用在后边专门介绍。

1. **表达式**

**1、算数表达式**

二元运算符：+ - \* / ^  (加减乘除幂)

一元运算符：-  (负值)

算数表达式用于实数，例：x^0.5将计算x的平方根，x^3将计算x的3次方。

**2、逻辑表达式**

and    or     not

逻辑运算符认为false和nil是假（false），其他为真，0也是true

**3、关系表达式**

<      >      <=     >=     ==     ~=

这些操作符返回结果为false或者true；==和~=比较两个值，如果两个值类型不同，Lua认为两者不同；nil只和自己相等。Table、function、userData类型，都是引用比较，也就是说，只有是同一个对象时候，才认为相等

**4、连接表达式**

..         --两个点

字符串连接，如果操作数为数字，Lua将数字转成字符串。由于Lua中字符串是不可改变的，所以连接操作符会创建一个新的字符对象，而不会修改原来的value

**5、表的构造**

在Lua中table既可以作为map使用也可以作为array使用。

构造器是创建和初始化表的表达式。表是Lua特有的功能强大的东西。最简单的构造函数是{}，用来创建一个空表。也可以构造数组，数组下标从1开始。

**1）构造表：**

Local tb1 = {x=10, y=20}

local tb1 = {[“x”] = 10, [“y”] = 20}

tb1 ={};tbl.x=10; tbl.y=20;

这三种构造方式是完全一样的。获取表数据：print(tbx[“x”])和print(tbl.x)也是一样的

**2）构造数组：**

local tbDays ={"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"}

print(tbDays[1]) -->Sunday

如果需要索引结构从0开始，则可以：

local tbDays = {[0]="Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday"}

print(tbDays[1]) -->Sunday

**3）在构造函数的最后的","是可选的，可以方便以后的扩展**。

a = {[1]="red", [2]="green", [3]="blue",}

**4）分割表的元素**

在构造函数中域分隔符逗号（","）可以用分号（";"）替代，通常我们使用分号用来分割不同类型的表元素。

{x=10, y=45**;** "one", "two", "three"}

1. **赋值语句**

Lua的特性：多重赋值

例如：local x,y = 2,4

Print(x) -->2

Print(y) -->4

右边值的个数少于左边变量的个数，多余的变量赋值为nil，如果值的个数多了，则多余的会被忽略掉

1. **控制语句**
2. **代码块与局部变量**

代码块在lua中使用do... End区分，不适用分号”;”断行

Lua中使用local定义局部变量，没有local关键字的都是全局变量

do

local a1 = 10

local a2 = 10

end

1. **控制语句**

所有的编程语言都有控制语句，对lua同样，lua支持的控制语句有if条件，循环的while、for、repeat-util语句，其中if、while、for都是以end结尾，而repeat以util结尾。同时lua不支持switch语句，但是我们可以通过程序实现这种控制结构

**1）If：**

在lua中false和nil是假的，其它都是真的

if conditions then

    then-part

elseif conditions then

    elseif-part

..            --->多个elseif

else

    else-part

end;

1. **while：**

local a = 10

while a > 0 do

a = a - 1

-- Do something else

End

1. **for：**

**数值型：**

for var = exp1, exp2, exp3 do

-- Do something

End

var从exp1变化到exp2，每次变化都以exp3作为步长进行递增。第三个表达式exp3是可选的，若不指定的话，Lua会将步长默认为1

注意：

1. for的三个值exp1、exp2、exp3是在计算一次，并且是在循环开始前
2. Var会被自动声明为局部变量，在for的do...end中有效

**泛型for循环：**

Lua提供了基础的库函数，例如：ipairs：遍历数组；pairs遍历tables元素

**4）repeat-util**

--[ repeat loop execution --]

a = 10

repeat

print("value of a:", a)

a = a - 1

until( a < 0 )

只要a<0则终止循环

**5)return/break**

两个都可以跳出循环代码块，但是return是函数返回结果，lua中的函数都有一个默认的return。

1. **函数**

函数对各种编程语言都是至关重要的，同样LUA中的函数有一些自己的特性，由于各种语言的函数基本相同，这章重点介绍这些特性

1. **lua函数必须以end结尾**

function testFunction(param)

Print(“hello Lua”)

end

**2、函数调用**

当函数只有一个字符串参数或者参数是table时候，可以不写圆括号，例如：

print "Hello World" --> print("Hello World")等价

print [[a multi-line message]] -->

print([[a multi-line message]]) 等价

-- f是一个函数

f{x=10, y=20} -->f({x=10, y=20}) 等价

调用函数的实际参数和定义函数的形式参数个数可以不相同，Lua会自动调整数量，如果实参多与形参，则舍去多余的，如果实参少于形参，则多余的形参赋值为nil

1. **多重返回值**

function foo0() end -- 无返回值

function foo1() return "a" end -- 返回一个结果

function foo2() return "a", "b" end -- 返回两个结果

table构造式可以完整的接收一个函数调用的所有结果，即不会有任何数量，但是这种情况只有在函数调用作为最后一个元素是才会发生，在其它位置时候，只能获取一个结果，例如以下两种情况：

情况一（函数调用是表的最后一个元素）：

local t = {foo0()} -- t = {}(一个空的table)

local t = {foo1()} -- t = {"a"}

local t = {foo2()} -- t = {"a", "b"}

情况一（函数调用不是表的最后一个元素）：

local t = {foo0(), foo2(), 4} -- t[1] = nil, t[2] = "a", t[3] = 4

1. **变长参数**

1）基本用法：

Lua中可以向C语言、java一样接受长度不定的参数列表，使用(...)表示变长参数列表

-- 打印所有的参数

function VarArguments(...)

for i, v in ipairs{...} do

print(v)

end

end

VarArguments(1, 2, 3)

1. 固定参数加上可变参数

例如：

function g (a, b, ...) end

g(3)              a=3, b=nil, arg={n=0}

g(3, 4)           a=3, b=4, arg={n=0}

g(3, 4, 5, 8)     a=3, b=4, arg={5, 8; n=2}

Lua会将前面的实参传递给固定参数，后边的实参传递给arg表中。

3）获取变长参数

变长参数有时候需要故意设置某一个参数为nil，这时候获取参数就需要select了。用法：

for i = 1, select('#', ...) do

local arg = select(i, ...) -- 得到第i个参数

-- Do something else

End

select(‘#’, …)会返回所有变长参数的总数，其中包括nil

参考：

<http://www.jellythink.com/archives/587>

http://www.lua.org/download.html

**Lua之字符串**

在任何编程语言中字符串的处理都是非常重要的，lua中的字符串是不可变的，对字符串处理常用的库函数，下面介绍下：

**string.len(s)**返回s的长度。

**string.rep(s, n)**返回重复s字符串n次的字符串。

**string.lower(s)**返回一份已将大写转成小写的字符串s的拷贝

**string.upper(argument)**:字符串全部转为大写字母

lower,upper都是使用本地字符集的，另外，如果你想对一个string数组进行排序,并不区分大小写，你可能像这么写：

table.sort(a, function(a, b)

return string.lower(a) < string.lower(b)

end)

**string.sub(s, i, j)**将从s提取一段字符串，从i到j(闭区间[i, j])，当然你可以使用负索引值，将从字符串尾部算起，-1是最后一个字符，-2是倒数第二，等等,这么做的好处是当我们要提取直到末尾几个字符时，从后面数起就很方便。例如：

s = "[hello,world]"

print(string.sub(s, 2, -2)) --> hello,world

**string.gsub(mainString,findString,replaceString,num)**在字符串中替换,mainString为要原始字符串， findString 为被替换的字符，replaceString 要替换的字符，num 替换次数（可以忽略，则全部替换）,例如：

string.gsub("aaaa","a","z",3);

zzza 3

**string.find (str, substr, [init, [end]])**

在一个指定的目标字符串中搜索指定的内容(第三个参数为索引),返回其具体位置。不存在则返回 nil。例如

print(string.find("Hello Lua user", "Lua", 1))

7 9

**string.reverse(arg)**

字符串反转

**string.char(arg) 和 string.byte(arg[,int])**

char 将整型数字转成字符并连接， byte 转换字符为整数值(可以指定某个字符，默认第一个字符)

**string.format(...)**

返回一个类似printf的格式化字符串

print(string.format("日期格式化 %02d/%02d/%04d", date, month, year))

**模式匹配**

lua string库里最强大的函数是那些模式匹配函数：find, match, gsub, gmatch。和其他脚本语言不同，lua既没有用POSIX的正则表达式，也没有用perl的正则表达式。原因是实现这些导致lua占用更多内存，而lua的初衷是小巧的，嵌入应用的语言

**模式匹配函数：find, match, gsub, gmatch**

string.match和string.find类似，都是在指定的string中查找相应的模式。不同的是，他返回的是找到的那部分string

例如：

date = "now is 2014/10/6 17:58"

d = string.match(date, "%d+/%d+/%d+")

print(d) --> 2014/10/6

**string.gmatch**函数将返回一个迭代器，用于迭代所有出现在给定字符串中的匹配字符串，例如：

local str = "Hello World"

local iteratorFunc = string.gmatch(str, "%a+") -- %a+表示匹配所有单词

for i in iteratorFunc do

print(i)

end

**#可以获取字符串长度**，例如：

>str='2016-09-01'

> print(#str)

10

**Lua 之table**

Lua中所有数组、符号表、集合、记录、队列和其他数据结构都是通过table实现，也可以通过table来实现模块（module）、包（package）和对象（Object）。table是对象，可以类比Java中的数组b['name']和b.name等价，而对于数字只能使用b['3']的形式，同Javascript一样，当没有table的引用时，Lua的垃圾回收器最终回收table。

Table的初始化等相关操作是非常简单的，这儿就不一一列举，介绍一下table的特性，和使用过程中需要注意的地方：

1.table是一个“关联数组”，数组的索引可以是数字或者是字符串

2.table 的默认初始索引一般以 1 开始

3.table 的变量只是一个地址引用，对 table 的操作不会产生数据影响

4.table 不会固定长度大小，有新数据插入时长度会自动增长

5.table的长度可以通过#获取（#还可以获取字符串的长度），在第一次碰到nil时，就认为是结束标志

**获取table的长度：table.getn(t)等价于 #**

但是它计算的是数组元素。不包括hash键值，而且数组是以第一个nil元素来判断数组结束。

#只计算array的元素个数，它实际上调用了对象的metatable 的\_\_len函数。

对于有\_\_len 方法的函数返回函数返回值。不然就返回数组成员数目。

**Lua中如何判断Table是空的**

1、local a = {}

2、对a进行处理

3、对a是否为空表进行判断

关于对a是否为空表的判断，如果直接这样做：

if a == {} then

这样的结果就是a == {}永远返回false，是一个逻辑错误。因为这里比较的是table a和一个匿名table的内存地址。

也有些代码如此做：

if table.maxn(a) == 0 then

这样做也不保险，除非table的key都是数字，而没有hash部分。

难道真的要遍历table发现有东西就return false跳出才能断定它是否为空吗？这样做是可以的，至少目前项目中前人是这么封装的，不过代码太难看.

lua内置的next函数，即如此用：if next(a) == nil then

next其实就是pairs遍历table时用来取下一个内容的函数.

**对table排序（sort）：**

因为所有的数据结构都是通过table实现的，因此table的排序方法十分常用：

Table的排序函数是table.sort有两个参数，第一个是排序的集合，第二个参数是排序function。

如果没有第二个参数的情况下，默认是升序排列

例如

tb = {20, 10, 2, 3}

table.sort(tb)

for \_, v in ipairs(tb) do

print(v)

End

结果：

2

3

10

20

修改为降序排列：

table.sort(tb,function(a,b) if a>b then return true end end)

for \_, v in ipairs(tb) do

print(v)

End

结果：

20

10

3

2

很多时候我们相对表的key进行排序，这时候简单的对table进行sort处理，没有办法达到期望的排序结果，我们需要先pairs将key装载一个array中，对key进行sort，然后再根据排好顺序的key获取value，这种方式也是非常简单的，就不再写demo了，直接借用果冻（http://www.jellythink.com/archives/539）的例子吧：

local tb = {x = 20, z = 10, y = 30, m = 2, n = 8} -- 这是一个key无序的table

local keyTb = {}

for k, \_ in pairs(tb) do

keyTb[#keyTb + 1] = k

end

table.sort(keyTb)

for \_, v in ipairs(keyTb) do

print(v .. " = " .. tb[v])

end

**Table常用的库函数：**

有很多对table库函数的介绍，在list一遍没有意义，直接参考：

http://www.runoob.com/lua/lua-tables.html

**Lua之元表**

**Lua之UserData**

**Lua之IO**

Lua的文件处理分为简单模式和完全模式

**Lua模块**

Require：

**Lua协同**

**Lua面向对象**

**Pcall、xpcall、debug**

Lua中没有提供try catch 的用法，因此异常处理的时候必须使用Pcall：

Pcall的用法是：第一个参数是执行的函数，或者匿名函数，第二个是执行函数需要的参数

Pcall是在保护模式下执行函数，可以捕获到任何异常、错误，如果执行正常，则返回true和函数结果，但是如果执行异常，则返回nil和错误信息，pcall返回的错误信息不全，返回时候已经销毁了部分栈信息。如果调试还是需要使用xcall查看全部栈信息

Xpcall用法，第一个是需要执行的函数，第二个是错误处理函数，第三个是执行函数需要的参数，在错误处理函数中会打印全部栈信息，这时候需要用到debug库

debug库提供了两个通用的错误处理函数:

debug.debug：提供一个Lua提示符，让用户来价差错误的原因

debug.traceback：根据调用桟来构建一个扩展的错误消息

SOCKET

Lua并没有提供socket相关的操作，socket需要引用Lua socket的包

**Lua基本函数列表**

assert (v [, message])

功能：相当于C的断言，

参数：

v：当表达式v为nil或false将触发错误,

message：发生错误时返回的信息，默认为"assertion failed!"

--------------------------------------------------------------------------------

collectgarbage (opt [, arg])

功能：是垃圾收集器的通用接口，用于操作垃圾收集器

参数：

opt：操作方法标志

"Stop": 停止垃圾收集器

"Restart": 重启垃圾收集器

"Collect": 执行一次全垃圾收集循环

"Count": 返回当前Lua中使用的内存量(以KB为单位)

"Step": 单步执行一个垃圾收集. 步长 "Size" 由参数arg指定　(大型的值需要多步才能完成)，如果要准确指定步长，需要多次实验以达最优效果。如果步长完成一次收集循环，将返回True

"Setpause": 设置 arg/100 的值作为暂定收集的时长

"Setstepmul": 设置 arg/100 的值，作为步长的增幅(即新步长＝旧步长\*arg/100)

--------------------------------------------------------------------------------

dofile (filename)

功能：打开并且执行一个lua块,当忽略参数filename时，将执行标准输入设备(stdin)的内容。返回所有块的返回值。当发生错误时，dofile将错误反射给调用者

注：dofile不能在保护模式下运行

--------------------------------------------------------------------------------

error (message [, level])

功能：终止正在执行的函数，并返回message的内容作为错误信息(error函数永远都不会返回)

通常情况下，error会附加一些错误位置的信息到message头部.

Level参数指示获得错误的位置,

Level=1[默认]：为调用error位置(文件+行号)

Level=2：指出哪个调用error的函数的函数

Level=0:不添加错误位置信息

--------------------------------------------------------------------------------

\_G全局环境表(全局变量)

功能：记录全局环境的变量值的表 \_G.\_G = \_G

--------------------------------------------------------------------------------

getfenv(f)

功能：返回函数f的当前环境表

参数：f可以为函数或调用栈的级别，级别1[默认]为当前的函数,级别0或其它值将返回全局环境\_G

--------------------------------------------------------------------------------

getmetatable(object)

功能：返回指定对象的元表(若object的元表.\_\_metatable项有值，则返回object的元表.\_\_metatable的值)，当object没有元表时将返回nil

--------------------------------------------------------------------------------

ipairs (t)

功能：返回三个值 迭代函数、表、0

多用于穷举表的键名和键值对

如：for i,v in ipairs(t) do

end

每次循环将索引赋级i，键值赋给v

注：本函数只能用于以数字索引访问的表 如：t={"1","cash"}

--------------------------------------------------------------------------------

load (func [, chunkname])

功能：装载一个块中的函数，每次调用func将返回一个连接前一结的字串，在块结尾处将返回nil

当没有发生错误时，将返回一个编译完成的块作为函数,否则返回nil加上错误信息，此函数的环境为全局环境

chunkname用于错误和调试信息

--------------------------------------------------------------------------------

loadfile ([filename])

功能：与load类似，但装载的是文件或当没有指定filename时装载标准输入(stdin)的内容

--------------------------------------------------------------------------------

loadstring (string [, chunkname])

功能：与load类似，但装载的内容是一个字串

如：assert(loadstring(s))()

--------------------------------------------------------------------------------

next (table [, index])

功能：允许程序遍历表中的每一个字段，返回下一索引和该索引的值。

参数：table：要遍历的表

　　　index：要返回的索引的前一索中的号，当index为nil[]时，将返回第一个索引的值，当索引号为最后一个索引或表为空时将返回nil

注：可以用next(t)来检测表是否为空(此函数只能用于以数字索引的表与ipairs相类似)

--------------------------------------------------------------------------------

pcall (f, arg1, ···)

功能：在保护模式下调用函数(即发生的错误将不会反射给调用者)

当调用函数成功能返回true,失败时将返回false加错误信息

--------------------------------------------------------------------------------

print (···)

功能：简单的以tostring方式格式化输出参数的内容

--------------------------------------------------------------------------------

rawequal (v1, v2)

功能：检测v1是否等于v2，此函数不会调用任何元表的方法

--------------------------------------------------------------------------------

rawget (table, index)

功能：获取表中指定索引的值，此函数不会调用任何元表的方法，成功返回相应的值，当索引不存在时返回nil

注：本函数只能用于以数字索引访问的表 如：t={"1","cash"}

--------------------------------------------------------------------------------

rawset (table, index, value)

功能：设置表中指定索引的值，此函数不会调用任何元表的方法，此函数将返回table

--------------------------------------------------------------------------------

select (index, ···)

功能：当index为数字将返回所有index大于index的参数:如：select(2,"a","b") 返回 "b"

当index为"#"，则返回参数的总个数(不包括index)

--------------------------------------------------------------------------------

setfenv (f, table)

功能：设置函数f的环境表为table

参数：f可以为函数或调用栈的级别，级别1[默认]为当前的函数,级别0将设置当前线程的环境表

--------------------------------------------------------------------------------

setmetatable (table, metatable)

功能：为指定的table设置元表metatable，如果metatable为nil则取消table的元表，当metatable有\_\_metatable字段时，将触发错误

注：只能为LUA\_TTABLE 表类型指定元表

--------------------------------------------------------------------------------

tonumber (e [, base])

功能：尝试将参数e转换为数字，当不能转换时返回nil

base(2~36)指出参数e当前使用的进制，默认为10进制，如tonumber(11,2)=3

--------------------------------------------------------------------------------

tostirng(e)

功能：将参数e转换为字符串，此函数将会触发元表的\_\_tostring事件

--------------------------------------------------------------------------------

type(v)

功能：返回参数的类型名("nil"，"number", "string", "boolean", "table", "function", "thread", "userdata")

--------------------------------------------------------------------------------

unpack (list [, i [, j]])

功能：返回指定表的索引的值,i为起始索引，j为结束索引

注：本函数只能用于以数字索引访问的表,否则只会返回nil 如：t={"1","cash"}

--------------------------------------------------------------------------------

\_VERSION

功能：返回当前Lua的版本号"Lua 5.1".

--------------------------------------------------------------------------------

xpcall (f, err)

功能：与pcall类似，在保护模式下调用函数(即发生的错误将不会反射给调用者)

但可指定一个新的错误处理函数句柄

当调用函数成功能返回true,失败时将返回false加err返回的结果

参考文档

<http://www.runoob.com/lua/lua-coroutine.html>

<http://jinnianshilongnian.iteye.com/blog/2235572>

<http://jinnianshilongnian.iteye.com/blog/2187775>

<https://my.oschina.net/wangxuanyihaha/blog/186401>

<http://blog.csdn.net/wzzfeitian/article/details/8832017>

<https://github.com/openresty/>