Lua 脚本语言入门

在这篇文章中，我想向大家介绍如何进行 Lua 程序设计。我假设大家都学过至少一门编程语言，比如 Basic 或 C，特别是 C。因为 Lua 的最大用途是在宿主程序中作为脚本使用的。

Lua 的语法比较简单，学习起来也比较省力，但功能却并不弱。

在 Lua 中，一切都是变量，除了关键字。请记住这句话。

I. 首先是注释

写一个程序，总是少不了注释的。

在 Lua 中，你可以使用单行注释和多行注释。

单行注释中，连续两个减号"--"表示注释的开始，一直延续到行末为止。相当于 C++语言中的"//"。

多行注释中，由"--[["表示注释开始，并且一直延续到"]]"为止。这种注释相当于 C 语言中的"/*...*/"。在注释当中，"[["和"]]"是可以嵌套的。

II. Lua 编程

经典的"Hello world"的程序总是被用来开始介绍一种语言。在 Lua 中，写一个这样的程序很简单：

```
print("Hello world")
```

在 Lua 中，语句之间可以用分号"；"隔开，也可以用空白隔开。一般来说，如果多个语句写在同一行的话，建议总是用分号隔开。

Lua 有好几种程序控制语句，如：

条件控制：if 条件 then ... elseif 条件 then ... else ... end

While 循环：while 条件 do ... end

Repeat 循环：repeat ... until 条件

For 循环：for 变量 = 初值，终点值，步进 do ... end

For 循环：for 变量 1，变量 2，... ，变量 N in 表或枚举函数 do ... end

注意一下，for 的循环变量总是只作用于 for 的局部变量，你也可以省略步进值，这时候，for 循环会使用 1 作为步进值。

你可以用 break 来中止一个循环。

如果你有程序设计的基础，比如你学过 Basic，C 之类的，你会觉得 Lua 也不难。但 Lua 有几个地方是明显不同于这些程序设计语言的，所以请特别注意。

. 语句块

语句块在 C++中是用"{"和"}"括起来的，在 Lua 中，它是用 do 和 end 括起来的。比如：

```
do print("Hello") end
```

你可以在 函数 中和 语句块 中定局部变量。

. 赋值语句

赋值语句在 Lua 被强化了。它可以同时给多个变量赋值。

例如：

```
a,b,c,d=1,2,3,4
```

甚至是：

```
a,b=b,a -- 多么方便的交换变量功能啊。
```

在默认情况下，变量总是认为是全局的。假如你要定义局部变量，则在第一次赋值的时候，需要用 local 说明。比如：

```
local a,b,c = 1,2,3 -- a,b,c 都是局部变量
```

. 数值运算

和C语言一样，支持 +, -, *, /。但Lua还多了一个"^"。这表示指数乘方运算。比如2^3 结果为8，2^4结果为16。

连接两个字符串，可以用".."运处符。如：

```
"This a " .. "string." -- 等于 "this a string"
```

. 比较运算

```
< > <= >= == ~=
```

分别表示 小于，大于，不大于，不小于，相等，不相等

所有这些操作符总是返回true或false。

对于Table，Function和Userdata类型的数据，只有 == 和 ~=可以用。相等表示两个变量引用的是同一个数据。比如：

```
a={1,2}
b=a
print(a==b, a~=b) -- true, false
a={1,2}
b={1,2}
print(a==b, a~=b) -- false, true
```

. 逻辑运算

```
and, or, not
```

其中，and 和 or 与C语言区别特别大。

在这里，请先记住，在Lua中，只有false和nil才计算为false，其它任何数据都计算为true，0也是true！

and 和 or的运算结果不是true和false，而是和它的两个操作数相关。

a and b：如果a为false，则返回a；否则返回b

a or b：如果 a 为true，则返回a；否则返回b

举几个例子：

```
print(4 and 5) --> 5
print(nil and 13) --> nil
print(false and 13) --> false
print(4 or 5) --> 4
print(false or 5) --> 5
```

在Lua中这是很有用的特性，也是比较令人混淆的特性。

我们可以模拟C语言中的语句：x = a? b：c，在Lua中，可以写成：x = a and b or c。

最有用的语句是： x = x or v，它相当于：if not x then x = v end 。

. 运算符优先级，从高到低顺序如下：

```
^
not - （一元运算）
* /
+ -
..（字符串连接）
< > <= >= ~= ==
and
or
```

III. 关键字

关键字是不能做为变量的。Lua 的关键字不多，就以下几个：

and break do else elseif

end false for function if

in local nil not or

repeat return then true until while

IV. 变量类型

怎么确定一个变量是什么类型的呢？大家可以用 type()函数来检查。Lua 支持的类型有以下几种：

Nil 空值，所有没有使用过的变量，都是 nil。nil 既是值，又是类型。

Boolean 布尔值

Number 数值，在 Lua 里，数值相当于 C 语言的 double

String 字符串，如果你愿意的话，字符串是可以包含'\0'字符的

Table 关系表类型，这个类型功能比较强大，我们在后面慢慢说。

Function 函数类型，不要怀疑，函数也是一种类型，也就是说，所有的函数，它本身就是一个变量。

Userdata 嗯，这个类型专门用来和 Lua 的宿主打交道的。宿主通常是用 C 和 C++来编写的，在这种情况下，Userdata 可以是宿主的任意数据类型，常用的有 Struct 和指针。

Thread　　线程类型，在 Lua 中没有真正的线程。Lua 中可以将一个函数分成几部份运行。如果感兴趣的话，可以去看看 Lua 的文档。

V. 变量的定义

所有的语言，都要用到变量。在 Lua 中，不管你在什么地方使用变量，都不需要声明，并且所有的这些变量总是全局变量，除非，你在前面加上"local"。

这一点要特别注意，因为你可能想在函数里使用局部变量，却忘了用 local 来说明。

至于变量名字，它是大小写相关的。也就是说，A 和 a 是两个不同的变量。

定义一个变量的方法就是赋值。"＝"操作就是用来赋值的

我们一起来定义几种常用类型的变量吧。

A. Nil

正如前面所说的，没有使用过的变量的值，都是 Nil。有时候我们也需要将一个变量清除，这时候，我们可以直接给变量赋以 nil 值。如：

```
var1=nil -- 请注意 nil 一定要小写
```

B. Boolean

布尔值通常是用在进行条件判断的时候。布尔值有两种：true 和 false。在 Lua 中，只有 false 和 nil 才被计算为 false，而所有任何其它类型的值，都是 true。比如 0，空串等等，都是 true。不要被 C 语言的习惯所误导，0 在 Lua 中的的确确是 true。你也可以直接给一个变量赋以 Boolean 类型的值，如：

```
varboolean = true
```

C. Number

在 Lua 中，是没有整数类型的，也不需要。一般情况下，只要数值不是很大（比如不超过 100,000,000,000,000），是不会产生舍入误差的。在很多 CPU 上，实数的运算并不比整数慢。

实数的表示方法，同 C 语言类似，如：

```
4 0.4 4.57e-3 0.3e12 5e+20
```

D. String

字符串，总是一种非常常用的高级类型。在Lua中，你可以非常方便的定义很长很长的字符串。

字符串在Lua中有几种方法来表示，最通用的方法，是用双引号或单引号来括起一个字符串的，如：

```
"This is a string."
```

和C语言相同的，它支持一些转义字符，列表如下：

```
\a bell
\b back space
\f form feed
\n newline
\r carriage return
\t horizontal tab
\v vertical tab
\\ backslash
\" double quote
\' single quote
\[ left square bracket
\] right square bracket
```

由于这种字符串只能写在一行中，因此，不可避免的要用到转义字符。加入了转义字符的串，看起来实在是不敢恭维，比如：

```
"one line\nnext line\n\"in quotes\", 'in quotes'"
```

一大堆的"\"符号让人看起来很倒胃口。如果你与我有同感，那么，我们在Lua中，可以用另一种表示方法：用"[["和"]]"将多行的字符串括起来，如：

```
page = [[
<HTML>
    <HEAD>
        <TITLE>An HTML Page</TITLE>
    </HEAD>
    <BODY>
        <A HREF="http://www.lua.org">Lua</A>
        [[a text between double brackets]]
    </BODY>
</HTML>
]]
```

值得注意的是，在这种字符串中，如果含有单独使用的"[["或"]]"就仍然得用"\["或"\]"来避免歧义。当然，这种情况是极少会发生的。

E. Table

关系表类型，这是一个很强大的类型。我们可以把这个类型看作是一个数组。只是C语言的数组，只能用正整数来作索引；在Lua中，你可以用任意类型来作数组的索引，除了nil。同样，在C语言中，数组的内容只允许一种类型；在Lua中，你也可以用任意类型的值来作数组的内容，除了nil。

Table的定义很简单，它的主要特征是用"{"和"}"来括起一系列数据元素的。比如：

```
T1 = {} -- 定义一个空表
T1[1]=10 -- 然后我们就可以象C语言一样来使用它了。
T1["John"]={Age=27, Gender="Male"}
```

这一句相当于：

```
T1["John"]={} -- 必须先定义成一个表，还记得未定义的变量是nil类型吗
T1["John"]["Age"]=27
T1["John"]["Gender"]="Male"
```

当表的索引是字符串的时候，我们可以简写成：

```
T1.John={}
T1.John.Age=27
T1.John.Gender="Male"
```

或

```
T1.John{Age=27, Gender="Male"}
```

这是一个很强的特性。

在定义表的时候，我们可以把所有的数据内容一起写在"{"和"}"之间，这样子是非常方便，而且很好看。比如，前面的T1的定义，我们可以这么写：

```
T1=
{
    10, -- 相当于 [1] = 10
    [100] = 40,
    John= -- 如果你原意，你还可以写成：["John"] =
    {
        Age=27, -- 如果你原意，你还可以写成：["Age"] =27
        Gender=Male -- 如果你原意，你还可以写成：["Gender"] =Male
    },
    20 -- 相当于 [2] = 20
}
```

看起来很漂亮，不是吗？我们在写的时候，需要注意三点：

第一，所有元素之间，总是用逗号"，"隔开；

第二，所有索引值都需要用"["和"]"括起来；如果是字符串，还可以去掉引号和中括号；

第三，如果不写索引，则索引就会被认为是数字，并按顺序自动从1往后编；

表类型的构造是如此的方便，以致于常常被人用来代替配置文件。是的，不用怀疑，它比ini文件要漂亮，并且强大的多。

F. Function

函数，在Lua中，函数的定义也很简单。典型的定义如下：

```
function add(a,b) -- add 是函数名字，a和b是参数名字
 return a+b -- return 用来返回函数的运行结果
end
```

请注意，return语言一定要写在end之前。假如你非要在中间放上一句return，那么请写成：do return end。

还记得前面说过，函数也是变量类型吗？上面的函数定义，其实相当于：

```
add = function (a,b) return a+b end
```

当你重新给add赋值时，它就不再表示这个函数了。你甚至可以赋给add任意数据，包括nil（这样，你就清除了add变量）。Function是不是很象C语言的函数指针呢？

和C语言一样，Lua的函数可以接受可变参数个数，它同样是用"..."来定义的，比如：

```
function sum (a,b,...)
```

如果想取得...所代表的参数，可以在函数中访问arg局部变量（表类型）得到。

如 sum(1,2,3,4)

则，在函数中，a = 1, b = 2, arg = {3, 4}

更可贵的是，它可以同时返回多个结果，比如：

```
function s()
    return 1,2,3,4
end
a,b,c,d = s() -- 此时，a = 1, b = 2, c = 3, d = 4
```

前面说过，表类型可以拥有任意类型的值，包括函数！因此，有一个很强大的特性是，拥有函数的表，哦，我想更恰当的应该说是对象吧。Lua可以使用面向对象编程了。不信？那我举例如下：

```
t =
{
 Age = 27
 add = function(self, n) self.Age = self.Age+n end
}
print(t.Age) -- 27
t.add(t, 10)
print(t.Age) -- 37
```

不过，t.add(t,10) 这一句实在是有点土对吧？没关系，在Lua中，你可以简写成：

```
t:add(10) -- 相当于 t.add(t,10)
```

G. Userdata 和 Thread

这两个类型的话题，超出了本文的内容，就不打算细说了。

VI. 结束语

就这么结束了吗？当然不是，接下来，需要用Lua解释器，来帮助你理解和实践了。这篇小文只是帮助你大体了解Lua的语法。如果你有编程基础，相信会很快对Lua上手了。

就象C语言一样，Lua提供了相当多的标准函数来增强语言的功能。使用这些标准函数，你可以很方便的操作各种数据类型，并处理输入输出。有关这方面的信息，你可以参考《Programming in Lua》一书，你可以在网络上直接观看电子版，网址为：http://www.lua.org/pil/index.html

当然，Lua的最强大的功能是能与宿主程序亲蜜无间的合作，因此，下一篇文章，我会告诉大家，如何在你的程序中使用Lua语言作为脚本，使你的程序和Lua脚本进行交互。

使用流程

1. 函数的使用

以下程序演示了如何在Lua中使用函数, 及局部变量

例 e02.lua

```
-- functions
function pythagorean(a, b)
local c2 = a^2 + b^2
return sqrt(c2)
end
```

```
print(pythagorean(3,4))
```

运行结果
5

程序说明
在Lua中函数的定义格式为:
function 函数名(参数)
...
end
与Pascal语言不同, end不需要与begin配对, 只需要在函数结束后打个end就可以了.
本例函数的作用是已知直角三角形直角边, 求斜边长度. 参数a,b分别表示直角边长,
在函数内定义了local形变量用于存储斜边的平方. 与C语言相同, 定义在函数内的代
码不会被直接执行, 只有主程序调用时才会被执行.
local表示定义一个局部变量, 如果不加local刚表示c2为一个全局变量, local的作用域
是在最里层的end和其配对的关键字之间, 如if ... end, while ... end等。全局变量的
作用域是整个程序。

2. 循环语句
例e03.lua
```
-- Loops
for i=1,5 do
print("i is now " .. i)
end
```

运行结果
```
i is now 1
i is now 2
i is now 3
i is now 4
i is now 5
```

程序说明
这里偶们用到了for语句
for 变量 = 参数1, 参数2, 参数3 do
循环体
end
变量将以参数3为步长, 由参数1变化到参数2
例如:
```
for i=1,f(x) do print(i) end
for i=10,1,-1 do print(i) end
```

这里print("i is now " .. i)中，偶们用到了..，这是用来连接两个字符串的，
偶在(1)的试试看中提到的，不知道你们答对了没有。
虽然这里i是一个整型量，Lua在处理的时候会自动转成字符串型，不需偶们费心。

3. 条件分支语句
例e04.lua

```
-- Loops and conditionals
for i=1,5 do
print(“i is now “ .. i)
if i < 2 then
print(“small”)
elseif i < 4 then
print(“medium”)
else
print(“big”)
end
end
```

运行结果

```
i is now 1
small
i is now 2
medium
i is now 3
medium
i is now 4
big
i is now 5
big
```

程序说明

if else用法比较简单，类似于C语言，不过此处需要注意的是整个if只需要一个end，
哪怕用了多个elseif，也是一个end.

例如

```
if op == "+" then
r = a + b
elseif op == "-" then
r = a - b
elseif op == "*" then
r = a*b
elseif op == "/" then
r = a/b
else
error("invalid operation")
end
```

4.试试看

Lua中除了for循环以外，还支持多种循环，请用while...do和repeat...until改写本文中的for程序

数组的使用

1.简介

Lua语言只有一种基本数据结构，那就是table，所有其他数据结构如数组啦，
类啦，都可以由table实现.

2.table 的下标
例 e05.lua
```
-- Arrays
myData = {}
myData[0] = “foo”
myData[1] = 42

-- Hash tables
myData[“bar”] = “baz”

-- Iterate through the
-- structure
for key, value in myData do
print(key .. “=“ .. value)
end
```

输出结果
```
0=foo
1=42
bar=baz
```

程序说明
首先定义了一个 table myData={}, 然后用数字作为下标赋了两个值给它. 这种定义方法类似于 C 中的数组, 但与数组不同的是, 每个数组元素不需要为相同类型, 就像本例中一个为整型, 一个为字符串.

程序第二部分, 以字符串做为下标, 又向 table 内增加了一个元素. 这种 table 非常像 STL 里面的 map. table 下标可以为 Lua 所支持的任意基本类型, 除了 nil 值以外.

Lua 对 Table 占用内存的处理是自动的, 如下面这段代码
```
a = {}
a["x"] = 10
b = a -- `b' refers to the same table as `a'
print(b["x"]) --> 10
b["x"] = 20
print(a["x"]) --> 20
a = nil -- now only `b' still refers to the table
b = nil -- now there are no references left to the table
```
b 和 a 都指向相同的 table, 只占用一块内存, 当执行到 a = nil 时, b 仍然指向 table, 而当执行到 b=nil 时, 因为没有指向 table 的变量了, 所以 Lua 会自动释放 table 所占内存

3.Table 的嵌套
Table 的使用还可以嵌套，如下例
例 e06.lua
```
-- Table ‘constructor’
myPolygon = {
color=“blue”,
```

```
thickness=2,
npoints=4;
{x=0, y=0},
{x=-10, y=0},
{x=-5, y=4},
{x=0, y=4}
}

-- Print the color
print(myPolygon[“color”])

-- Print it again using dot
-- notation
print(myPolygon.color)

-- The points are accessible
-- in myPolygon[1] to myPolygon[4]

-- Print the second point’s x
-- coordinate
print(myPolygon[2].x)
```

程序说明
首先建立一个table，与上一例不同的是，在table的constructor里面有{x=0,y=0}，这是什么意思呢？ 这其实就是一个小table，定义在了大table之内，小table的table名省略了.
最后一行myPolygon[2].x，就是大table里面小table的访问方式.