Lua 脚本语法说明

Lua 脚本语法说明（增加 lua5.1 部份特性）

Lua 的语法比较简单，学习起来也比较省力，但功能却并不弱。

所以，我只简单的归纳一下 Lua 的一些语法规则，使用起来方便好查就可以了。估计看完了，就懂得怎么写 Lua 程序了。

在 Lua 中，一切都是变量，除了关键字。

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（在 lua5.1 中，中括号中间是可以加若干个"="号的，如 [==[...]==]），见下面的字符串表示说明。

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：

控制语句	格式	示例
If	if 条件 then ... elseif 条件 then ... else ... end	if 1+1=2 then print("true") elseif 1+2~=3 then print("true") else print("false") end

While	while 条件 do ... end	while 1+1~=2 do print("true") end
Repeat	repeat ... until 条件	repeat print("**Hello**") until 1+1~=
For	for 变量=初值, 终点值, 步进 do ... end	for i = 1, 10, 2 do print(i) end
For	for 变量 1, 变量 2, ... 变量 n in 表或枚举函数 do ... end	for a,b in mylist do print(a, b) end

注意一下，for 的循环变量总是只作用于 for 的局部变量；当省略步进值时，for 循环会使用 1 作为步进值。

使用 break 可以用来中止一个循环。

相对 C 语言来说，Lua 有几个地方是明显不同的，所以面要特别注意一下：

. **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

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。假如需要定义局部变量，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

比较符号	<	>	<=	>=	==	~=
含义	小于	大于	小于或等于	大于或等于	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)  --输出 true, false
a={1,2}
b={1,2}
print(a==b, a~=b)  --输出 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) --输出 5
print(nil and 13) --输出 nil
print(false and 13) --输出 false
print(4 or 5) --输出 4
print(false or 5) --输出 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，从低到高顺序如下：

```
or
and
<     >     <=    >=    ~=    ==
.. (字符串连接)
+     -
*     /     %
not   #(lua5.1 取长度运算)     - (一元运算)
^
```

和 C 语言一样，括号可以改变优先级。

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，只有两个有效值：true 和 false
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（这和 C 语言总是以"\0"结尾是不一样的）
Table	关系表类型，这个类型功能比较强大，请参考后面的内容。
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，Us 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
	的文档。
	现在回过头来看看，倒觉得不是线程类型。反而象是用来做遍历的，象是 Iterator 函数。
	如：
	`function range(n)`
	`  local i = 0`
	`  while(i < n) do`
	`  coroutine.yield( i )`
	`  i = i + 1`
	`  end`
	`end`
	可惜的是要继续运行，需要 coroutine.resume 函数，有点鸡肋。请指教。

V. 变量的定义

所有的语言，都要用到变量。在 Lua 中，不管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，除非我们在前面加上"local"。这一点要特别注意，因为我们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：

```
theBoolean = 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 WindowsXP 能跑的当今主流 PC 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，我们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：

"That's go!"

或

'Hello world!'

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。（lua5.1：中括号中间可以加入若干个"="号，如 [==[...]==]，详见下面示例）

示例：下面的语句所表示的是完全相同的字符串：

```
a = 'alo\n123"'
a = "alo\n123\""
a = '\97lo\10\04923"'
a = [[alo
123"]]
a = [==[
alo
123"]==]
```

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。假如我们非要在中间放上一句 return，那么就应该要写成：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
```

当重新给 add 赋值时，它就不再表示这个函数了。我们甚至可以赋给 add 任意数据，包括 nil（这样，赋值为 nil，将会把该变量清除）。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（lua5.1：取消 arg，并直接用"..."来代表可变参数了，本质还是 arg）。

如 sum(1,2,3,4)

则，在函数中，a = 1, b = 2, arg = {3, 4} （lua5.1： a = 1, b = 2, ... = {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？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，我们可以简写成：

```
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，我们需要用 Lua 解释器，来帮助理解和实践了。相信这样会更快的对 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，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，我们可以参考《Programming in Lua 》一书，也可以在网络上直接观看电子版，网址为：

http://www.lua.org/pil/index.html