链滴

Redis Lua 脚本中学教程（下）

作者：Hawkpool

原文链接：https://ld246.com/article/1584512019675

来源网站：链滴

在中学教程的上半部分我们介绍了Redis Lua相关的命令，没有看过或者忘记的同学可以步行前往直使用机票Redis Lua脚本中学教程（上）。今天我们来简单学习一下Lua的语法。

在介绍Lua语法之前，先来介绍一下Lua的身世。Lua是由简称为PUC-Rio的团队设计、开发和维护的Lua在葡萄牙语中是月亮的意思，所以它不是简写，而是一个名词。所以只能写成Lua，而不能写成LU或者其他什么的。接下来我们正式入门Lua。

变量

变量名可以是由字母、数字和下划线组成的字符串，但不能以数字开头。另外需要注意的是，需要尽避免使用下划线加一个或多个大写字母格式的变量名，因为这是Lua的保留字，除了这种格式以外，有一些普通格式的保留字：

and	**break**	**do**	**else**	**lseif**
end	false	for	function	oto
if	in	local	nil	not
or	repeat	return	then	rue
until	while			

Lua是大小写敏感的，and是保留字，但And和AND不是。

全局变量

前面我们提到过Redis不支持Lua的全局变量，但Lua本身是支持全局变量的。

全局变量不需要声明，直接一个未初始化的变量时，它的值是nil。

```
> b --> nil
> b = 10
> b --> 10
```

如果显示的将nil赋值给某个全局变量，Lua会认为我们不再使用这个变量。

局部变量

Lua的变量默认是全局变量，局部变量需要显示声明。局部变量可以避免增加不必要的名称来混淆全环境，同时也能避免两部分代码的命名冲突。另外，访问局部变量要比访问全局变量的速度更快。

局部变量的使用范围是有限制的，只在它声明的块中可用。（块可以是控制结构体或函数体或者是整文件中）

```
x = 10
local i = 1 -- local to the chunk
while i <= x do
local x = i * 2 -- local to the while body
print(x) --> 2, 4, 6, 8, ...
i = i + 1
end
if i > 20 then
local x -- local to the "then" body
x = 20
print(x + 2) -- (would print 22 if test succeeded)
else
print(x) --> 10 (the global one)
end
print(x) --> 10 (the global one)
```

在交互模式下，每次输入都是一块代码，当你输入local i = 1时，就定义了一个局部变量i，而当你在一行使用i时，发现它又成了全局变量。因此上面的栗子就不能用了。为了解决这个问题，我们需要在序中显式的使用**do-end**标记代码块的范围。

```
local x1, x2
do
  local a2 = 2*a
  local d = (b^2 - 4*a*c)^(1/2)
  x1 = (-b + d)/a2
  x2 = (-b - d)/a2
end                     -- scope of 'a2' and 'd' ends here
print(x1, x2)           -- 'x1' and 'x2' still in scope
```

使用这种方式标记代码块范围是一种良好的习惯，而使用局部变量编程也要优于使用全局变量，因此很多人呼吁Lua默认应该定义局部变量，但是这样也会存在问题。最好的解决方案是不要默认，使用有的变量之前都要声明。

Lua有一个常见的习语：

```
local foo = foo
```

这里定义了一个局部变量foo，并把全局变量foo的值赋给局部变量。这一习语主要用来提升变量foo

访问速度，或者对变量进行暂存，防止其他函数改变这个变量的值。

注释

单行注释

Lua的单行注释使用双横线“--”表示，双横线后的内容为注释内容。

多行注释

多行注释的一种表现是以双横线加双左中括号开始，以双右中括号结束。例如：

```
--[[A multi-line
long comment
]]
```

解注释

```
---[[
print(10) --> 10
--]]
```

这里稍微解释一下这种写法的原理，注释时，后一组双横线在注释内容中，因此不起作用，只为了对，效果和普通多行注释一样。而解注释时，第一组双横线前又加了一个横线，就不能认为是多行注释，只能当做单行注释，因此，第一行被注释掉了，这时后一组双横线就会起作用了，注释掉后面的双中括号。

数据类型

Lua是一种动态类型语言，它有8种基本类型：nil，Boolean，number，string，userdata，functio，thread和table。type函数可以返回指定值的类型：

```
> type(nil) --> nil
> type(true) --> boolean
> type(10.4 * 3) --> number
> type("Hello world") --> string
> type(io.stdin) --> userdata
> type(print) --> function
> type(type) --> function
> type({}) --> table
> type(type(X)) --> string
```

Nil

Nil类型的值只有一种，就是nil，它是一种没有值的表现。

Boolean

Boolean类型有两种取值，@false{} and @true{}。但是Boolean类型并不能囊括所有的条件值：在件判断时，Lua会将false和nil判断为假，其他的都判断为真。

画外音：Lua把0和空字符串也判断为真，这点感觉设计的不太好啊

and、or和not是Lua的逻辑运算符。

and的运算方法是，判断第一个操作数是不是false，如果不是，结果就是第二个操作数。

or的运算方法是，
判断第一个操作数是不是真，如果不是，结果就是第二个操作数。

```
> 4 and 5 --> 5
> nil and 13 --> nil
> false and 13 --> false
> 0 or 5 --> 0
> false or "hi" --> "hi"
> nil or false --> false
```

Table

Table是Lua中主要的（也是唯一的）结构化数据表现类型。它可以用来表现很多种数据类型，如数组集合、记录等。

每个表的key可以是不同类型的，对于未定义索引的表元素，它的默认值是nil。和其他大部分语言不的是**Lua中表的下标是从1开始的。**

Table有两种格式：record-style和list-style

record-style可以直接用"."访问，list-style可以用下标来访问。定义时可以一起定义。

```
polyline = {color="blue",
            thickness=2,
            npoints=4,
            {x=0,   y=0},
            {x=-10, y=0},
            {x=-10, y=1},
            {x=0,   y=1}
}
```

当我们访问一个可能为空的Table，往往需要先判断非空

```
if lib and lib.foo then ....
```

使用这种方式访问结构比较深的表示就会非常痛苦：

```
zip = company and company.director and
                    company.director.address and
                      company.director.address.zipcode
```

Lua没有像C#一样提供?.这样的操作，不过我们可以使用or {}的形式来处理。

流程控制

Lua提供了一些基本的流程控制语句：

- if用于条件判断
- while、repeat和for用于便利
- end可以终止if、for和while
- until可以终止repeat
- break用于跳出循环
- return用于跳出函数
- goto会跳转到指定位置

函数

Lua中函数可以接收的参数是list，如果没有参数，也需要写一对空的括号"()"（一句废话）。如果只一个参数，则括号可写可不写。Lua还提供了一种特殊的函数访问方法，有兴趣的话可以参考https://ww.lua.org/pil/16.html

```
o:foo(x)
```

Lua程序中既可以使用定义在Lua中的函数，也可以使用定义在C语言中的函数。

Lua函数有一个非常方便的特性：可以返回多个结果。

```
function maximum (a)
  local mi = 1
  local m = a[mi]
  for i = 1, #a do
    if a[i] > m then
      mi = i; m = a[i]
end end
  return m, mi
end

print(maximum({8,10,23,12,5}))     --> 23   3
```

Lua可以自动调整返回结果的数量，当函数作为语句调用时，会舍弃所有返回值；当函数作为表达式用时，只保留第一个返回值；如果要获得全部返回值，函数调用需要是表达式最后一个。

Lua函数也支持可变参数：

```
function add (...)
  local s = 0
  for _, v in ipairs{...} do
        s=s+ v
  end
return s
end

print(add(3, 4, 10, 25, 12))   --> 54
```

总结

来简单总结一下，本文我们介绍了Lua的基本语法，包括如何定义变量（包括全局变量和局部变量），

种基本数据类型，流程控制语句以及Lua中函数的一些特性。相信看完本文，你就可以写一些简单的Lu脚本了。

对Lua感兴趣的同学可以自行前往Lua官网继续深造。