Lua 编程
第三版

Roberto erusalimschy

2018 年 5 月 29 日

目录

前言

第一部分

语言

第一章　入门

依循惯例，我们的第一个 lua 程序也仅仅只打印一句“Hello World”：

```
print("Hello World")
```

如果使用独立的 Lua 解释器，需要调用解释器（通常叫做 lua 或者 lua5.2）运行第一个程序，在解释器后面带上包含程序的文本文件名。如果上面的程序保存在 hello.lua 文件中，使用如下命令运行：

```
% lua hello.lau
```

再来一个复杂点的例子，下面的程序定义了一个计算给定数字的阶乘的函数，请求用户输入数字，并打印它的阶乘：

```
-- 定义一个阶乘函数
function fact (n)
  if n == 0 then
    return 1
  else
    return n * fact(n-1)
  end
end

print("enter a number:)
a = io.read("*n")               -- 读取一个数字
print(fact(a))
```

1.1　程序块（chunk）

Lua 执行的代码片段，例如一个文件或者交互模式下的一行代码，都被称为“程序块”。程序块就是命令序列（或者语句）。

Lua 的语句间不需要分隔符，如果你愿意，可以使用分号来分隔语句。个人建议仅在两条或多条语句写在同一行时，使用分号。在 Lua 语法中，换行无意义；如下四个程序块都成立且意义相同：

```
a = 1
b = a*2

a = 1;
b = a*2;

a = 1; b = a*2

a = 1 b = a*2                   -- 我很丑，可是我好用
```

程序块可以简单到只有一条语句，就象“Hello World”那个例子。也可以由句子和函数定义（后面我们会看到，它实际上就是赋值操作）混合组成，就象阶乘那个例子。如果你想，程序块可以很大。因为 Lua 也用作数据描述语言，带有几兆字节的程序块并不少见。Lua 解释器在处理大程序块方面完全没有问题。

除了把程序写进文件，还可以在解释器中以交互模式运行。如果不带参数运行 lua，将显示交互提示：

```
% lua
Lua 5.2 Copyright (C) 1994-2012 lua.org, PUC-Rio
>
```

此后输入的每个命令（比如 print"Hello World"）都将立即执行。要退出交互模式和解释器，只需输入文件结束控制符（在 UNIX 中是 ctrl-D，在 Windows 中则是 ctrl-Z），或者调用操作系统库的 exit 函数，即输入 os.exit()。

在交互模式下，Lua 通常把输入的每一行视为一个完整的程序块。但是，如果发现某一行不足以构成完整的程序块，它会等待更多的输入，直到形成一个完整的程序块。通过这种方式，可以输入类似前面的阶乘函数那样的多行定义。不过，把定义写入文件并调用 Lua 运行这些文件，通常更实用方便。

可以使用 -i 参数通知 Lua 在运行给定的程序块后进入交互会话：

```
% lua -i prog
```

这条命令将运行 prog 文件中的程序块，然后显示交互提示。这种方式对于调试和手工测试尤其有用。在本章的最后，将看到独立解释器的其它参数。

运行程序块的另一种方法是使用实时执行文件的 dofile 函数。举个例子，假设有包含如下代码的 lib1.lua 文件：

```
function norm (x, y)
  return (x^2 + y^2)^0.5
end

function twice (x)
  return 2*x
end
```

在交互模式下，输入：

```
> dofile("lib1.lua")            -- 载入库文件
> n = norm(3.4, 1.0)
> print(twice(n))               --> 7.0880180586677
```

dofile 函数在测试代码片段时也很有用。可以在两个窗口中工作：一个窗口是代码编辑器（比如正编辑 prog.lua），另一个则是以交互模式运行 Lua 的控制台。在保存了对代码的修改后，可以在 Lua console 运行 dofile("prog.lua") 来加载新代码；然后试验新代码，调用函数并打印结果。

1.2 词法规约

Lua 中的标识符（或名字）可以是任何由字母、数字和下划线构成，但不以数字开头的字符串；例如：

```
i       j       i10     _ij
aSomewhatLongName       _INPUT
```

要避免使用以一个下划线开头、带有一个或多个大写字母的标识符（比如 _VERSION）；他们在 Lua 中被保留作特殊用途。通常，我保留带有一个下划线的标识符用于虚变量。

在旧版本的 Lua 中，字符的概念依赖于本地化环境。但是程序中的一些字符不能在不支持这些本地化的系统中运行。因此，Lua 5.2 仅允许使用 A-Z 和 a-z 作为标识符可用的字母。

下面这些关键词是保留的；不能把它们用作标识符：

```
and     break   do      else    elseif
end     false   goto    for     function
if      in      local   nil     not
or      repeat  return  then    true
until   while
```

Lua 区分大小写：and 是一个保留字，但是 And 和 AND 是另两个不同的标识符。

注解可以在任意位置以两个连字符开始直到行末。Lua 也支持块注释，以 --[[开始，以]] 结束[1]。一个注释掉代码片段的小窍门就是把代码封在 [[和]] 之间，就像这样：

```
--[[
print(10)               -- 不起作用（注释掉了）
--]]
```

要激活这段代码，只需要在第一行前面添加一个连字符：

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

前一个例子，第一行的 --[[是块注释的起始，而最后一行的两个连字符是注释的一部分。后一个例子，---[[序列开始了一个普通的单行注释，于是第一行和最后一行变量了独立的注释。在这种情况下，print 位于注释外面。

1.3 全局变量

全局变量不需要声明，可以直接使用。访问未初始化的变量不会出错，只会得到一个 nil 值：

```
print(b)                --> nil
b = 10
print(b)                --> 10
```

如果给全局变量赋予 nil 值，Lua 会假定变量不再被使用：

```
b = nil
print(b)                --> nil
```

在这条赋值语句之后，Lua 将回收该变量的内存。

1.4 独立解释器

独立解释器（源文件是 lua.c，可执行文件是 lua）可直接使用 Lua 的小型程序。这一节介绍它的主要参数。

如果文件第一行以脚本符（'#'）开头，解释器在载入时将忽略这一行。这个特性支持在类 UNIX 系统中以脚本方式运行 Lua 文件。使用下面内容开始脚本（前提是独立解释器程序放在 /usr/local/bin 目录中）：

```
#!/usr/local/bin/lua
```

或者使用：

```
#!/usr/bin/env lua
```

就可以跳过 Lua 解释器直接调用这个脚本文件。

lua 命令的用法如下：

[1] 2.4 节将看到，块注释可以比这复杂的多

```
lua [options] [script [args]]
```

每个部分都是可选的。前面我们已经见过，不带任何参数调用 lua 命令将进入交互模式。
-e 参数允许我们直接在命令行输入代码，就像这样：

```
% lua -e "pring(math.sin(12))"          --> -0.53657291800043
```

在 UNIX 中需要使用双引号防止 shell 解析圆括号。
-l 参数加载库。像前面看到的那样，-i 在运行其他参数之后进入交互模式。下面的代码先加载 lib 库，执行赋值 x = 10，最后显示交互提示符：

```
% lua -i -llib -e "x = 10"
```

在交互模式下，使用等号开头，后面跟着一个表达式，可以打印任何表达式的值：

```
> = math.sin(3)                          --> 0.14112000805987
> a = 30
> = a                                    --> 30
```

此特性可用于将 Lua 当成计算器。
在运行参数前，解释器先寻找名为 LUA_INIT_5_2 的系统变量（如果找不到，会寻找 LUA_INIT）。如果这些变量包含 *@filename*，解释器就运行给定的文件。如果 LUA_INIT_5_2（或者 LUA_INIT）变量定义不是以 @ 开头，解释器将其视为 lua 代码运行。此特性为我们提供了通过这些配置项自定义 Lua 解释器的强大能力。我们可以预载包、改变路径、自定义函数、给函数改名、删除函数，等等。脚本可以通过预定义的全局变量 arg 接收到命令行参数。在调用 % lua script a b c 时，运行脚本之前，解释器建立一个包含全部命令行参数的表 arg。脚本名赋给索引 0，第一个参数（本例中的 'a'）赋给索引 1，其余参数依次赋给对应的索引。显示在脚本前的参数赋给负数索引。例如，思考如下调用：

```
% lua -e "sin=math.sin" script a b
```

解释器这样整理参数：

```
arg[-3] = "lua"
arg[-2] = "-e"
arg[-1] = "sin=math.sin"
arg[0] = "script"
arg[1] = "a"
arg[2] = "b"
```

多数情况，脚本只使用正数索引（本例中的 arg[1] 和 arg[2]）。
从 Lua 5.1 开始，脚本也可以通过可变长度参数（vararg）表达式接受接受这些参数。在函数体中，表达式 ...（三个圆点）代表脚本的参数（第 5.2 节将详细说明可变长度参数表达式。）

练习

练习 1.1： 运行阶乘的例子。如果给一个负数会发生什么？修改例子修正这个问题。
练习 1.2： 分别用载入文件和 -l 参数并调用 dofile 运行第二个例子。你喜欢哪种？
练习 1.3： 你能说出使用 – 作为注解标识符的其它语言吗？
练习 1.4： 下面字符串中，哪些是有效标识符？

```
___     _end    End     end     until?  nil     null
```

练习 1.5： 写一个简单脚本，假定事先不知道的情况下打印脚本名称。

第二章　类型和值

Lua 是动态类型语言。其中没有类型定义，每个值持有自己的类型。

Lua 中有 8 种基本类型：nil，boolean，number，string，userdata，function，thread，table。type 函数用于获取给定的值的类型名：

```
print(type("Hello world"))      --> string
print(type(10.4*3))             --> number
print(type(print))              --> function
print(type(type))               --> type
print(type(true))               --> boolean
print(type(nil))                --> nil
print(type(type(X)))            --> string
```

最后一行无论 X 的值是什么，结果都是“string”，因为 type 的值总是字符串。

变量没有预定义类型，每个变量可以包含任意类型的值：

```
print(type(a))                  --> nil (`a' 没有初始化)
a = 10
print(type(a))                  --> number
a = "a string!"
print(type(a))                  --> string
a = print                       --> 你没看错，这是正确的!
print(type(a))                  --> function
```

注意最后两行：在 Lua 中，函数是一等值；所以我们能够像操作其它值一样操作它。（第六章将看到关于这一特性的更多详情。）

通常，将同一变量用于不同类型，会造成混乱。但是，有时候，审慎地使用这一特性很有用，比如使用 nil 来区别正常的返回值和非正常情况。

2.1　Nil

Nil 是一个只带有 nil 一个值的类型（单值类型），它的主要作用是区别其他类型。Lua 使用 nil 作为无效值，表示没有有用的有效值。就像我们看到的那样，全局变量被赋值前的默认值就是 nil，还可以用给全局变量赋 nil 值的方式来删除之。

2.2　Booleans

布尔类型有两个值：false 和 true，用来表示布尔值。但是条件值并未由布尔值独霸：在 Lua 中任何值都能用于条件。条件判断（比如控制结构中的条件）将 false 和 nil 视为假，其余值均被视为真。尤其要注意的是，在条件判断中 Lua 将 0 和空字符串也视为真。

在本书中，我用“假”来表示任务假值，包括布尔值 false 和 nil。当特指布尔值时，我用“false”。同样的规则可以推广到“真”和“true”。

2.3 Numbers

数字类型表示实数（双精度浮点数）。Lua 没有整型。

有些人担心使用浮点数，即使简单的处境和比较也可能会产生怪异的结果。事实并非如此。实际上如今所有平台都遵循 IEEE 754 标准中的表示法。遵循这一标准，仅当数字不能精确表示时，才会出现误差。结果不能精确表示时，才会出现舍入。能够精确表示的结果必须给出精确值。

2^{53}（约等于 10^{16}）以内的整数都能用双精度浮点数精确表示。当使用双精度数表示整数时，不会出现舍入错误，除非其绝对值超出了 2^{53}。Lua 数字能够表示任何 32 位整数而不出现舍入问题。

当然，分数会遇到表示错误。这种情况和使用纸笔没什么区别。如果我们用十进制计算 $1/7$，我们将不得不在某个位置停下。如果我们使用十位数字表示，$1/7$ 将被舍入为 0.142857142。如果我们使用十位数字计算 $1/7*7$，结果将是 0.999999994，并不是 1。此外，在十进制中能够用有穷数字表示的数字，在二进制中将是无穷数。例如，$12.7-20+7.3$ 用双精度数计算，结果不等于 0，因为 12.7 和 7.3 在二进制中都没有精确的有穷表示（详见练习 2.3）。

在继续前进之前，切记：整数有精确表示并且没有舍入错误。

多数现代 CPU 浮点计算和整数计算一样快。不过，使用其它的数字类型（比如长整数或单精度数）编译 Lua 也很容易。对像类似嵌入式这样没有浮点数硬件支持的系统非常有用。详情参见发行版中的 `luaconf.h` 文件。

可以书写带有可选小数部分及可选十进制指数部分的数字常量。有效数字示例如下：

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

还可以加上 $0x$ 前缀书写十六进制常量。从 Lua 5.2 开始，十六进制常量也可以有小数和指数（通过 'p' 或 'P' 前缀），示例如下：

```
0xff (255)      0x1A3 (419)     0x0.2 (0.125)   0x1p-1 (0.5)
0xa.bp2 (42.75)
```

（每个常量后面的括号里是它的十进制表示）

第二部分

标准库

第三部分

C API

第四部分

表和对象