## Lua

Lua是一种轻量小巧的脚本语言，用标准C语言编写并以源代码形式开放，其设计目的是为了嵌入应用程序中，从而为应用程序提供灵活的扩展和定制功能。

## Lua特性

轻量级:它用标准C语言编写并以源代码形式开放，编译后仅仅一百余K，可以很方便的嵌入别的程序里。

可扩展:Lua提供了非常易于使用的扩展接口和机制：由宿主语言(通常是C或C++)提供这些功能，Lua可以使用它们，就像是本来就内置的功能一样。

支持面向过程(procedure-oriented)编程和函数式编程(functionalprogramming)；

自动内存管理；只提供了一种通用类型的表（table），用它可以实现数组，哈希表，集合，对象；

语言内置模式匹配；闭包(closure)；函数也可以看做一个值；提供多线程（协同进程，并非操作系统所支持的线程）支持；

通过闭包和table可以很方便地支持面向对象编程所需要的一些关键机制，比如数据抽象，虚函数，继承和重载等。

## Linux系统安装

Linux上安装Lua安装非常简单，只需下载源码包并在终端解压编译即可：

curl-R-Ohttp://www.lua.org/ftp/lua-5.3.0.tar.gz

tarzxflua-5.3.0.tar.gz

cdlua-5.3.0

makelinuxtest

makeinstall

## 第一个Lua程序

使用Lua来输出"HelloWorld!"

vimhello.lua

print("HelloWorld!")

运行luahello.lua，会在屏幕上显示Hello，world!。

## Lua基本语法

1. 程序块

连续的语句之间并不需要分隔符，如果愿意也可以用分号进行分割。

1. 标识符

Lua标示符用于定义一个变量，函数获取其他用户定义的项。标示符以一个字母A到Z或a到z或下划线\_开头后加上0个或多个字母，下划线，数字（0到9）。

最好不要使用下划线加大写字母的标示符，因为Lua的保留字也是这样的。Lua不允许使用特殊字符如@，$和%来定义标示符。Lua是一个区分大小写的编程语言。

1. 关键字

以下列出了 Lua 的保留关键字

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

一般约定，以下划线开头连接一串大写字母的名字（比如 \_VERSION）被保留用于 Lua 内部全局变量。

1. 注释

单行注释：两个减号是单行注释:

--

多行注释：

--[[

多行注释

多行注释

--]]

1. 全局变量

变量一般总是认为是全局的，如果一个变量的生命周期较短，可以将其声明为局部变量local。全局变量不需要声明，给一个变量赋值后即创建了这个全局变量，访问一个没有初始化的全局变量也不会出错，只不过得到的结果是：nil。如果你想删除一个全局变量，只需要将变量负值为nil。

## 数据类型

变量：

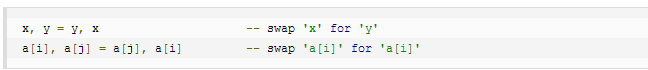
Lua是动态类型语言，变量不要类型定义，只需要为变量赋值。值可以存储在变量中，作为参数传递或结果返回。变量在使用前，必须在代码中进行声明，即创建该变量。编译程序执行代码之前编译器需要知道如何给语句变量开辟存储区，用于存储变量的值。

Lua 变量有三种类型：全局变量、局部变量、表中的域。函数外的变量默认为全局变量，除非用 local 显示声明。函数内变量与函数的参数默认为局部变量。局部变量的作用域为从声明位置开始到所在语句块结束（或者是直到下一个同名局部变量的声明）。变量的默认值均为 nil。应该尽可能的使用局部变量，有两个好处：1. 避免命名冲突。2. 访问局部变量的速度比全局变量更快。

Lua可以对多个变量同时赋值，变量列表和值列表的各个元素用逗号分开，赋值语句右边的值会依次赋给左边的变量。



遇到赋值语句Lua会先计算右边所有的值然后再执行赋值操作，所以我们可以这样进行交换变量的值.



当变量个数和值的个数不一致时，Lua会一直以变量个数为基础采取以下策略：

a. 变量个数 > 值的个数 按变量个数补足nil

b. 变量个数 < 值的个数 多余的值会被忽略

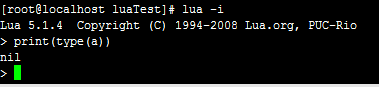
数据类型：

Lua中有8个基本类型分别为：nil、boolean、number、string、userdata、function、thread和table。

|  |  |
| --- | --- |
| 数据类型 | 描述 |
| nil | 这个最简单，只有值nil属于该类，表示一个无效值（在条件表达式中相当于false）。 |
| boolean | 包含两个值：false和true。 |
| number | 表示双精度类型的实浮点数 |
| string | 字符串由一对双引号或单引号来表示 |
| function | 由 C 或 Lua 编写的函数 |
| userdata | 表示任意存储在变量中的C数据结构 |
| thread | 表示执行的独立线路，用于执行协同程序 |
| table | Lua 中的表（table）其实是一个"关联数组"（associative arrays），数组的索引可以是数字或者是字符串。在 Lua 里，table 的创建是通过"构造表达式"来完成，最简单构造表达式是{}，用来创建一个空表。 |

1. nil:

nil 类型表示一种没有任何有效值，它只有一个值 nil，例如打印一个没有赋值的变量，便会输出一个 nil 值：



对于全局变量和 table，nil还有删除变量功能。

1. boolen

boolean 类型只有两个可选值：true（真） 和 false（假），Lua 把 false 和 nil 看作是"假"，其他的都为"真"。

1. number

Lua 默认只有一种 number 类型 -- double（双精度）类型（默认类型可以修改 luaconf.h 里的定义）

1. string

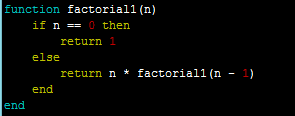
字符串由一对双引号或单引号来表示，也可以用 2 个方括号 "[[]]" 来表示"一块"字符串。字符串拼接使用的是”..”。

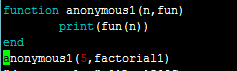
在对一个数字字符串上进行算术操作时，Lua 会尝试将这个数字字符串转成一个数字。 使用 # 来计算字符串的长度，放在字符串前面。

1. function

在 Lua 中，函数是被看作是"第一类值（First-Class Value）"，函数可以存在变量里。

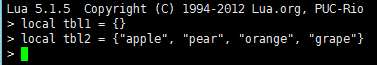
function 可以以匿名函数（anonymous function）的方式通过参数传递：





1. table

在 Lua 里，table 的创建是通过"构造表达式"来完成，最简单构造表达式是{}，用来创建一个空表。也可以在表里添加一些数据，直接初始化表:



Lua 中的表（table）其实是一个"关联数组"（associative arrays），数组的索引可以是数字或者是字符串，不同于其他语言的数组把 0 作为数组的初始索引，在 Lua 里表的默认初始索引一般以 1 开始。table 不会固定长度大小，有新数据添加时 table 长度会自动增长，没初始的 table 都是 nil。

1. userdata

userdata 是一种用户自定义数据，用于表示一种由应用程序或 C/C++ 语言库所创建的类型，可以将任意 C/C++ 的任意数据类型的数据（通常是 struct 和 指针）存储到 Lua 变量中调用。

1. thread

在 Lua 里，最主要的线程是协同程序（coroutine）。它跟线程（thread）差不多，拥有自己独立的栈、局部变量和指令指针，可以跟其他协同程序共享全局变量和其他大部分东西。

线程跟协程的区别：线程可以同时多个运行，而协程任意时刻只能运行一个，并且处于运行状态的协程只有被挂起（suspend）时才会暂停。

## 控制语句

1. If语句

if(布尔表达式)

then

--[ 在布尔表达式为 true 时执行的语句 --]

end

在布尔表达式为 true 时会if中的代码块会被执行，在布尔表达式为 false 时，紧跟在 if 语句 end 之后的代码会被执行。Lua认为false和nil为假，true 和非nil为真。要注意的是Lua中 0 为 true。

1. Ifelse

if(布尔表达式)

then

--[ 布尔表达式为 true 时执行该语句块 --]

else

--[ 布尔表达式为 false 时执行该语句块 --]

End

1. If elseif else

if( 布尔表达式 1)

then

--[ 在布尔表达式 1 为 true 时执行该语句块 --]

else if( 布尔表达式 2) then

--[ 在布尔表达式 2 为 true 时执行该语句块 --]

else if( 布尔表达式 3) then

--[ 在布尔表达式 3 为 true 时执行该语句块 --]

else

--[ 如果以上布尔表达式都不为 true 则执行该语句块 --]

end

## 循环语句

1. while循环

while(condition)

do

statements

end

1. for 循环

Lua 编程语言中 for语句有两大类：：

**数值for循环：**

for var=exp1,exp2,exp3 do

statements

end

var从exp1变化到exp2，每次变化以exp3为步长递增var，并执行一次"执行体"。exp3是可选的，如果不指定，默认为1。

**泛型for循环：**

泛型for循环通过一个迭代器函数来遍历所有值，类似java中的foreach语句。泛型for循环语法格式:

--打印数组a的所有值

for i,v in ipairs(a)

do print(v)

end

其中i是数组索引值，v是对应索引的数组元素值。ipairs是Lua提供的一个迭代器函数，用来迭代数组。

1. repeat...until 循环语法格式:

repeat

statements

**until** ( condition )

## 函数

Lua 编程语言函数定义格式如下：

optional\_function\_scope function function\_name( argument1, argument2, argument3..., argumentn)

function\_body

return result\_params\_comma\_separated

end

解析：

**optional\_function\_scope**

该参数是可选的制定函数是全局函数还是局部函数，未设置该参数末尾为全局函数，如果你需要设置函数为局部函数需要使用关键字**local**。

**function\_name:**

指定函数名称。

**argument1, argument2, argument3..., argumentn:**

函数参数，多个参数以逗号隔开，函数也可以不带参数。

**function\_body:**

函数体，函数中需要执行的代码语句块。

**result\_params\_comma\_separated:**

函数返回值，Lua语言函数可以返回多个值，每个值以逗号隔开。

Lua函数可以接受可变数目的参数，和C语言类似在函数参数列表中使用三点（...) 表示函数有可变的参数。

Lua将函数的参数放在一个叫arg的表中，**#arg** 表示传入参数的个数。

例如，我们计算几个数的平均值：

function average(...)

result = 0

local arg={...}

for i,v in ipairs(arg) do

result = result + v

end

print("总共传入 " .. #arg .. " 个数")

return result/#arg

end

print("平均值为",average(10,5,3,4,5,6))

## 运算符

Lua提供了以下几种运算符类型：

* 算术运算符
* 关系运算符
* 逻辑运算符
* 其他运算符
* 算术运算符

下表列出了 Lua 语言中的常用算术运算符，设定 A 的值为10，B 的值为 20：

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述** | **实例** |
| + | 加法 | A + B 输出结果 30 |
| - | 减法 | A - B 输出结果 -10 |
| \* | 乘法 | A \* B 输出结果 200 |
| / | 除法 | B / A w输出结果 2 |
| % | 取余 | B % A 输出结果 0 |
| ^ | 乘幂 | A^2 输出结果 100 |
| - | 负号 | -A 输出结果v -10 |

* 关系运算符

下表列出了 Lua 语言中的常用关系运算符，设定 A 的值为10，B 的值为 20：

|  |  |  |
| --- | --- | --- |
| 操作符 | 描述 | 实例 |
| == | 等于，检测两个值是否相等，相等返回 true，否则返回 false | (A == B) 为 false。 |
| ~= | 不等于，检测两个值是否相等，相等返回 false，否则返回 true< | (A ~= B) 为 true。 |
| > | 大于，如果左边的值大于右边的值，返回 true，否则返回 false | (A > B) 为 false。 |
| < | 小于，如果左边的值大于右边的值，返回 false，否则返回 true | (A < B) 为 true。 |
| >= | 大于等于，如果左边的值大于等于右边的值，返回 true，否则返回 false | (A >= B) is not true. |
| <= | 小于等于， 如果左边的值小于等于右边的值，返回 true，否则返回 false | (A <= B) is true. |

* 逻辑运算符：

下表列出了 Lua 语言中的常用逻辑运算符，设定 A 的值为10，B 的值为 20：

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述** | **实例** |
| and | 逻辑与操作符。 如果两边的操作都为 true 则条件为 true。 | (A and B) 为 false。 |
| or | 逻辑或操作符。 如果两边的操作任一一个为 true 则条件为 true。 | (A or B) 为 true。 |
| not | 逻辑非操作符。与逻辑运算结果相反，如果条件为 true，逻辑非为 false。 | !(A and B) 为 false。 |

* 其它运算符：

下表列出了 Lua 语言中的连接运算符与计算表或字符串长度的运算符：

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述** | **实例** |
| .. | 连接两个字符串 | a..b ，其中 a 为 "Hello " ， b 为 "World", 输出结果为 "Hello World"。 |
| # | 一元运算符，返回字符串或表的长度。 | #"Hello" 返回 5 |

* 优先级从高到低的顺序：

^

not - (unary)

\* /

+ -

..

< > <= >= ~= ==

and

or

除了^和..外所有的二元运算符都是左连接的。

## 字符串

字符串或串(String)是由数字、字母、下划线组成的一串字符。

Lua 语言中字符串可以使用以下三种方式来表示：

* 单引号间的一串字符。
* 双引号间的一串字符。
* [[和]]间的一串字符。

Lua 提供了很多的方法来支持字符串的操作：

|  |  |
| --- | --- |
| **序号** | **方法 & 用途** |
| 1 | **string.upper(argument):** 字符串全部转为大写字母。 |
| 2 | **string.lower(argument):** 字符串全部转为小写字母。 |
| 3 | **string.gsub(mainString,findString,replaceString,num)** 在字符串中替换,mainString为要替换的字符串， 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），如：  > string.gsub("aaaa","a","z",3);  zzza 3 |
| 4 | **string.strfind (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。  > string.find("Hello Lua user", "Lua", 1)  7 9 |
| 5 | **string.reverse(arg)** 字符串反转  > string.reverse("Lua")  auL |
| 6 | **string.format(...)** 返回一个类似printf的格式化字符串  > string.format("the value is:%d",4)  the value is:4 |
| 7 | **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.char(97,98,99,100)  abcd  > string.byte("ABCD",4)  68  > string.byte("ABCD")  65  > |
| 8 | **string.len(arg)** 计算字符串长度。  string.len("abc")  3 |
| 9 | **string.rep(string, n))** 返回字符串string的n个拷贝  > string.rep("abcd",2)  abcdabcd |
| 10 | **..** 链接两个字符串  > print("www.w3cschool".."cc")  www.w3cschoolcc |

## 数组

数组，就是相同数据类型的元素按一定顺序排列的集合，可以是一维数组和多维数组。

Lua 数组的索引键值可以使用整数表示，数组的大小不是固定的。

一维数组是最简单的数组，其逻辑结构是线性表。一维数组可以用for循环出数组中的元素，如下实例：

array = {"Lua", "Tutorial"}

for i= 0, 2 do

print(array[i])

end

以上代码执行输出结果为：

nil

Lua

Tutorial

正如你所看到的，我们可以使用整数索引来访问数组元素，如果知道的索引没有值则返回nil。

在 Lua 索引值是以 1 为起始，但你也可以指定 0 开始。

除此外我们还可以以负数为数组索引值：

array = {}

for i= -2, 2 do

array[i] = i \*2

end

for i = -2,2 do

print(array[i])

end

多维数组即数组中包含数组或一维数组的索引键对应一个数组。

以下是一个三行三列的阵列多维数组：

-- 初始化数组

array = {}

for i=1,3 do

array[i] = {}

for j=1,3 do

array[i][j] = i\*j

end

end

-- 访问数组

for i=1,3 do

for j=1,3 do

print(array[i][j])

end

end

## 迭代器

迭代器（iterator）是一种对象，它能够用来遍历标准模板库容器中的部分或全部元素，每个迭代器对象代表容器中的确定的地址。在Lua中迭代器是一种支持指针类型的结构，它可以遍历集合的每一个元素。

泛型 for 迭代器

泛型 for 在自己内部保存迭代函数，实际上它保存三个值：迭代函数、状态常量、控制变量。

泛型 for 迭代器提供了集合的 key/value 对，语法格式如下：

for k, v in pairs(t) do

print(k, v)

end

上面代码中，k, v为变量列表；pair(t)为表达式列表。

范性for的执行过程：

* 首先，初始化，计算in后面表达式的值，表达式应该返回范性for需要的三个值：迭代函数、状态常量、控制变量；与多值赋值一样，如果表达式返回的结果个数不足三个会自动用nil补足，多出部分会被忽略。
* 第二，将状态常量和控制变量作为参数调用迭代函数（注意：对于for结构来说，状态常量没有用处，仅仅在初始化时获取他的值并传递给迭代函数）。
* 第三，将迭代函数返回的值赋给变量列表。
* 第四，如果返回的第一个值为nil循环结束，否则执行循环体。
* 第五，回到第二步再次调用迭代函数

。在Lua中我们常常使用函数来描述迭代器，每次调用该函数就返回集合的下一个元素。Lua 的迭代器包含以下两种类型：

* 无状态的迭代器
* 多状态的迭代器
* 无状态的迭代器

无状态的迭代器是指不保留任何状态的迭代器，因此在循环中我们可以利用无状态迭代器避免创建闭包花费额外的代价。

每一次迭代，迭代函数都是用两个变量（状态常量和控制变量）的值作为参数被调用，一个无状态的迭代器只利用这两个值可以获取下一个元素。

* 多状态的迭代器

这种无状态迭代器的典型的简单的例子是ipairs，他遍历数组的每一个元素。

很多情况下，迭代器需要保存多个状态信息而不是简单的状态常量和控制变量，最简单的方法是使用闭包，还有一种方法就是将所有的状态信息封装到table内，将table作为迭代器的状态常量，因为这种情况下可以将所有的信息存放在table内，所以迭代函数通常不需要第二个参数

## Table

table 是 Lua 的一种数据结构用来帮助我们创建不同的数据类型，如：数字、字典等。

Lua table 使用关联型数组，你可以用任意类型的值来作数组的索引，但这个值不能是 nil。

Lua table 是不固定大小的，你可以根据自己需要进行扩容。

Lua也是通过table来解决模块（module）、包（package）和对象（Object）的。 例如string.format表示使用"format"来索引table string。

当我们为 table a 并设置元素，然后将 a 赋值给 b，则 a 与 b 都指向同一个内存。如果 a 设置为 nil ，则 b 同样能访问 table 的元素。如果没有指定的变量指向a，Lua的垃圾回收机制会清理相对应的内存。

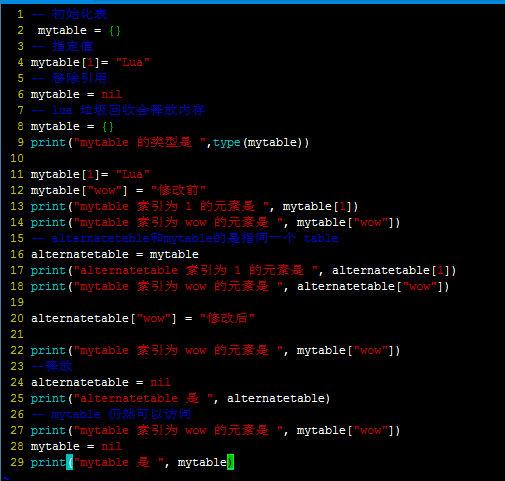
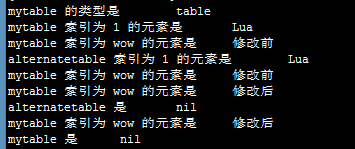
 

 Table 操作常用的方法：

|  |  |
| --- | --- |
| **序号** | **方法 & 用途** |
| 1 | **table.concat (table [, step [, start [, end]]]):**  concat是concatenate(连锁, 连接)的缩写. table.concat()函数列出参数中指定table的数组部分从start位置到end位置的所有元素, 元素间以指定的分隔符(sep)隔开。 |
| 2 | **table.insert (table, [pos,] value):**  在table的数组部分指定位置(pos)插入值为value的一个元素. pos参数可选, 默认为数组部分末尾. |
| 3 | **table.maxn (table)**  指定table中所有正数key值中最大的key值. 如果不存在key值为正数的元素, 则返回0。(**Lua5.2之后该方法已经不存在了,本文使用了自定义函数实现**) |
| 4 | **table.remove (table [, pos])**  返回table数组部分位于pos位置的元素. 其后的元素会被前移. pos参数可选, 默认为table长度, 即从最后一个元素删起。 |
| 5 | **table.sort (table [, comp])**  对给定的table进行升序排序 |

## 模块与包

模块类似于一个封装库，从 Lua 5.1 开始，Lua 加入了标准的模块管理机制，可以把一些公用的代码放在一个文件里，以 API 接口的形式在其他地方调用，有利于代码的重用和降低代码耦合度。

Lua 的模块是由变量、函数等已知元素组成的 table，因此创建一个模块很简单，就是创建一个 table，然后把需要导出的常量、函数放入其中，最后返回这个 table 就行。以下为创建自定义模块 module.lua，文件代码格式如下：