

Lua 教程

Lua 教程

Lua 环境安装

Lua 基本语法

Lua 数据类型

Lua 变量

Lua 循环

Lua 流程控制

Lua 函数

Lua 运算符

Lua 字符串

Lua 数组

Lua 迭代器

Lua table(表)

Lua 模块与包

Lua 元表(Metatable)

Lua 协同程序(coroutine)

Lua 文件 I/O

Lua 错误处理

Lua 调试(Debug)

Lua 垃圾回收

Lua 面向对象

Lua 数据库访问

Lua5.3 参考手册

Lua 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；pairs(t)为表达式列表。
查看以下实例：

实例

```
array = {"Google", "Runoob"}

for key,value in ipairs(array)
do
    print(key, value)
end
```

以上代码执行输出结果为：

```
1  Google
2  Runoob
```

以上实例中我们使用了 Lua 默认提供的迭代函数 ipairs。
下面我们看看泛型 for 的执行过程：

- 首先，初始化，计算 in 后面表达式的值，表达式应该返回泛型 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。这种无状态迭代器的典型的简单的例子是 ipairs，它遍历数组的每一个元素。

以下实例我们使用了一个简单的函数来实现迭代器，实现 数字 n 的平方：

实例

```
function square(iteratorMaxCount,currentNumber)
if currentNumber<iteratorMaxCount
then
    currentNumber = currentNumber+1
    return currentNumber, currentNumber*currentNumber
end
end

for i,n in square,3,9
do
    print(i,n)
end
```

以上实例输出结果为：

```
1  1
2  4
3  9
```

迭代的状态包括被遍历的表（循环过程中不会改变的状态常量）和当前的索引下标（控制变量），ipairs 和迭代函数都很简单，我们在 Lua 中可以这样实现：

实例

```
function iter (a, i)
    i = i + 1
    local v = a[i]
    if v then
        return i, v
    end
end

function ipairs (a)
    return iter, a, 0
end
```

当 Lua 调用 ipairs(a) 开始循环时，他获取三个值：迭代函数 iter、状态常量 a、控制变量初始值 0；然后 Lua 调用 iter(a,0) 返回 1, a[1]（除非 a[1]=nil）；第二次迭代调用 iter(a,1) 返回 2, a[2].....直到第一个 nil 元素。

多状态的迭代器

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。

以下实例我们创建了自己的迭代器：

实例

```
array = {"Google", "Runoob"}

function elementIterator (collection)
    local index = 0
    local count = #collection
    -- 闭包函数
    return function ()
        index = index + 1
        if index <= count
        then
            -- 返回迭代器的当前元素
            return collection[index]
        end
    end
end

for element in elementIterator(array)
do
    print(element)
end
```

以上实例输出结果为：

```
Google
Runoob
```

以上实例中我们可以看到，elementIterator 内使用了闭包函数，实现计算集合大小并输出各个元素。

+

3 篇笔记

写笔记



在线实例

- [HTML 实例](#)
- [CSS 实例](#)
- [JavaScript 实例](#)
- [Ajax 实例](#)
- [jQuery 实例](#)
- [XML 实例](#)
- [Java 实例](#)

字符集&工具

- [HTML 字符集设置](#)
- [HTML ASCII 字符集](#)
- [HTML ISO-8859-1](#)
- [HTML 实体符号](#)
- [HTML 拾色器](#)
- [JSON 格式化工具](#)

最新更新

- [PostgreSQL 删除...](#)
- [PostgreSQL 选择...](#)
- [PostgreSQL 创建...](#)
- [PostgreSQL 数据...](#)
- [C 语言整数与字...](#)
- [Vue.js Ajax\(ax...](#)
- [PostgreSQL 语法](#)

站点信息

- [意见反馈](#)
- [合作联系](#)
- [免责声明](#)
- [关于我们](#)
- [文章归档](#)

关注微信

