

Lua

2021年5月26日 14:53

葡萄牙语 Lua

嵌入式脚本语言 底层使用 C 效率最高

客户端

2002 云风 《梦幻西游》

Unreal 蓝图./Lua

2004 《魔兽世界》 脚本层

服务端

nginx → Openresty → Lua

redis → Lua 原子操作

生活

罗技宏

lua的特点

2021年5月26日 15:10

效率高，充分利用C的性能

Clean C C和C++不冲突的解法

Lua 5.1

Lua → Lua 官方 5.4 ..

→ Lua JIT. (Just in time)
5.1

① curl -O 下载镜像

② tar xfv

③ make libreadline-dev (sudo apt install) **sudo make linux**
libncurses-dev ④ sudo make install

Lua的特定

2021年5月26日 15:23

```
[liao@ubuntu ~/36_lua/day01]$ lua  
Lua 5.1.5 Copyright (C) 1994-2012 Lua.org, PUC-Rio  
>
```

1. C 编译式。
Lua 解释性。 运行慢 开发快
2. 动态类型 运行时再检查 类型是否检查
3. 不需要声明。 类型由值决定
DD-000
4. 垃圾回收

lua的第一行代码

2021年5月26日 15:33

1. 行式

```
[liao@ubuntu ~/36_lua/day01]$ lua
Lua 5.1.5  copyright (C) 1994-2012 Lua.org, PUC-Rio
> print("hello world")
hello world
```

2. 脚本语言文件

```
[liao@ubuntu ~/36_lua/day01]$ chmod u+x 0_hello.lua
[liao@ubuntu ~/36_lua/day01]$ ./0_hello.lua
Hello world!
[liao@ubuntu ~/36_lua/day01]$ cat 0_hello.lua
#!/usr/local/bin/lua
print("Hello world!")
```

看脚本的单位

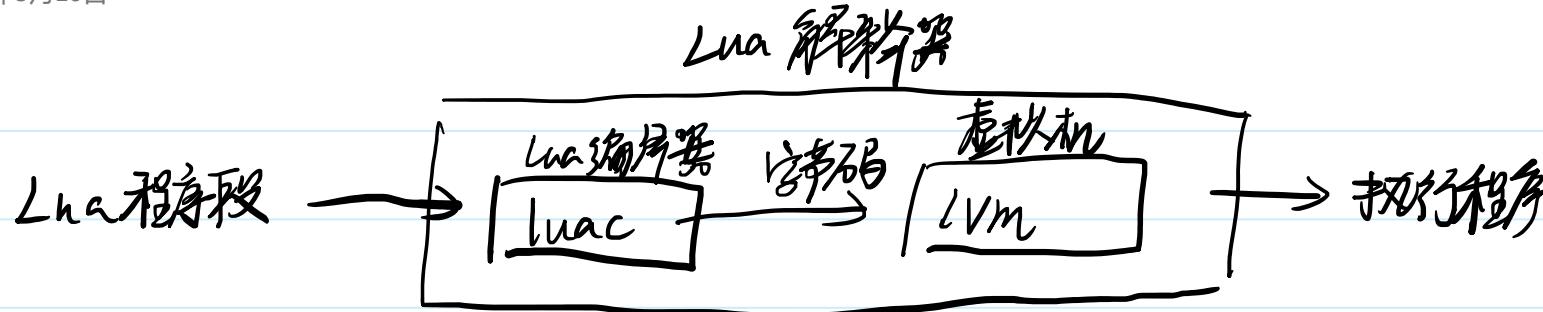
一个chunk
程序段

3. 用lua命令加载文件

```
[liao@ubuntu ~/36_lua/day01]$ lua 1_hello.lua
Hello world!
[liao@ubuntu ~/36_lua/day01]$ cat 1_hello.lua
print("Hello world!")
```

Lua解释器的结构

2021年5月26日 15:40



把字符串根据底层硬件平台.

转换成对应的二进制指令

加载指令到机器中执行

跨平台

Luac 编译器 编译原理.

```
[liao@ubuntu ~/36_lua/day01]$ file luac.out
luac.out: Lua bytecode, version 5.1
```

① 提高效率

② 保护源代码

输出所有的命令行参数

2021年5月26日 16:42

```
for i = -2,#arg do  
    print(i,arg[i])  
end
```

钩变量

下标是从1开始 没有就是nil

块分解 do end

语句不用加分号。

```
[liao@ubuntu ~/36_lua/day01]$ lua 2_args.lua hello 123
```

```
-2      nil  
-1      lua  
0      2_args.lua  
1      hello  
2      123
```

Lua的详细语法

2021年5月26日 16:51

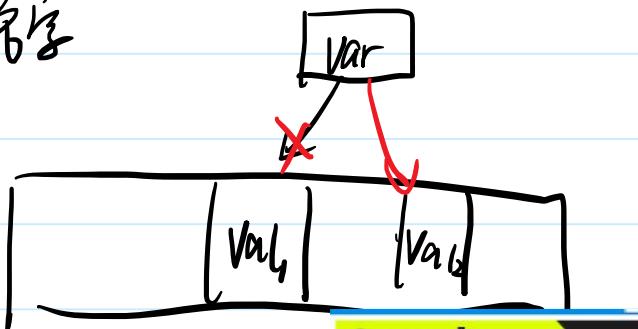
标识符 - 小写/大写字母 形
- 1位/大写字母 数 例

关键字

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

变量：是一个值的名字

值决定了数据类型



Var = Val₁

Var = Val₂

未初始化的变量默认对应nil

```
3_var.lua
1 print(wangdao)
2 wangdao = "biancheng"
3 do
4     --wangdao = "wangdao" 所有的变量默认是全局变量
5     --local wangdao = "wangdao"
6     local wangdao = wangdao --用全局变量的值对局部变量赋值
7     print(wangdao)
8 end
9 print(wangdao)
```

lua的八种数据类型

2021年5月26日 17:04

nil 空类型

boolean 布尔类型

type (v)

Returns the type of its only argument, coded as a string. The possible results of this function are "nil" (a string, not the value nil), "number", "string", "boolean", "table", "function", "thread", and "userdata".

nil

false

会使条件表达式为假

0 / "" 都会使条件表达式为真

print(0 and 123)	123
print(nil and 123)	nil
print(false and 123)	false
print(123 or nil)	123
print(false or nil)	nil
print(not nil)	true
print(not 0)	false

and lhs and rhs

如果lhs为真，返回rhs

如果lhs为假，返回lhs

or lhs or rhs

如果lhs为真，返回lhs

如果lhs为假，返回rhs.

not

not 假 → true

not 真 → false

一个特殊的表达式

2021年5月26日 17:15

$$x = x_1 \text{ or } \{ \}$$

x_1 存在 x 为 x_1
 x 不存在 nil x 为 $\{ \}$

↳ 初级语义

$$a = a_1 \text{ or } 0$$

$$A \text{ and } B \text{ or } C \rightarrow A ? B : c ;$$

A 和 B 为 真 .

$$A \text{ 真 } \quad A \text{ and } B \rightarrow B$$

$$B \text{ or } C \rightarrow B$$

$$A \text{ 假 } \quad A \text{ and } B \rightarrow A$$

$$A \text{ or } C \rightarrow C$$

数值类型

2021年5月26日 17:18

不区分整型和浮点型

32位 64位

+ - * / ^
乘方

```
print(type(123))
print(type(3.14))
print(type(2.99e10))
```

print(1+1.3)	2.3	< <=
print(1/2)	0.5	
print(2^3)	8	> >=
print(2^0.5)	1.4142135623731	
print(2^0.5 == 1.414)	false	==
print(2^0.5 == 1.4142135623731)	false	
print(12345 == 1.2345e4)	true	
print(1.0 == 1.00000)	true	

< <=

> >=

== ≈

tostring (e)

Receives an argument of any type and converts it to a string in a reasonable format.

字符串

2021年5月26日 17:26

单引号 / 双引号

```
print("I'm strong")
print("\\"how are you\"she said")
print('"how are you"she said')
```

根据内容选择合适的界定符

```
1 print("I'm strong")
2 print("\\"how are you\"she said")
3 print('"how are you"she said')
4 str = [[--方括号的右边要接一个换行
5 "I am 18",she'said]]
6 print(str)
7 code = [=====[
8 a[b[i]] = 1]=====]--在方括号之间加入相等数量的=，可以自定义界定符i
9 print(code)
```

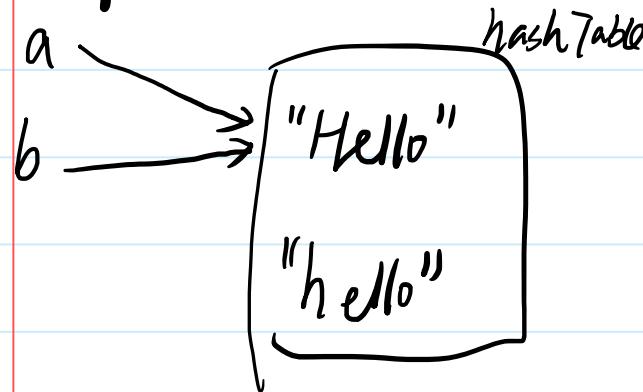
..

```
print(#code)
print("how are you"..str)
```

lua字符串的实现

2021年5月27日 9:29

哈希表. 集中存储了所有的字符串值.



① 字符串值不可修改, 一旦修改, 新值就插入哈希表

② 一个变量可以在不同阶段, 指向不同的字符串值

③ 假如多个变量指向同一个值, 那么就指向同一个链表

1. 容量大小

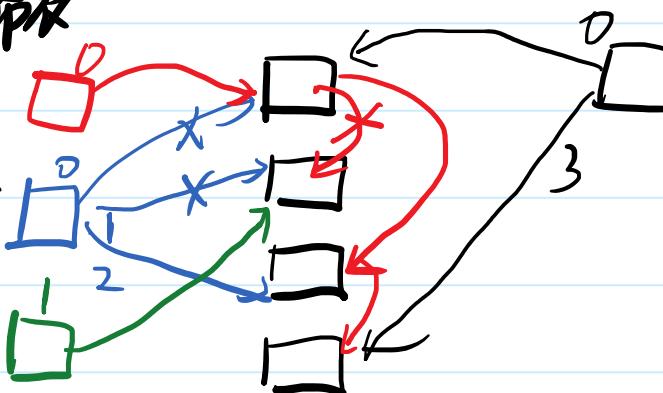
Capacity - 初始比较小,

当插入数据达到 capacity - 半. 将 Capacity * 2.

重建哈希表

2. 解决冲突

飞虫与青蛙
内存分配



两种字符串使用的差距

2021年5月27日 9:51

```
table.concat (table [, sep [, i [, j]]])
```

Given an array where all elements are strings or numbers, returns `table[i]..sep..table[i+1] .. sep.. table[j]`.

```
local time_beg = os.clock()
local str = ""
local t = {}
for i = 1,300000 do
    t[#t+1] = 'a'
end
str = table.concat(t,"")
local time_end = os.clock()
print(time_end - time_beg)
```

```
local time_beg = os.clock()
local str = ""
for i = 1,300000 do
    str = str .. 'a'
end
local time_end = os.clock()
print(time_end - time_beg)
```

字符串标准库

2021年5月27日 9:56

string
string.byte
string.char
string.dump
string.find
string.format
string.gmatch
string.gsub
string.len → 長度.
string.lower
string.match → 大→小
string.rep → 亂
string.reverse
string.sub → 子串
string.upper → 小→大

反转 ←

string.format (formatstring, ...)

Returns a formatted version of its variable number of arguments following the description given in its first argument (which must be a string). The format string follows the same rules as the `printf` family of standard C functions. The only differences are that the options/modifiers *, l, L, n, p, and h are not supported and that there is an extra option, q. The q option formats a string in a form suitable to be safely read back by the Lua interpreter: the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written. For instance, the call

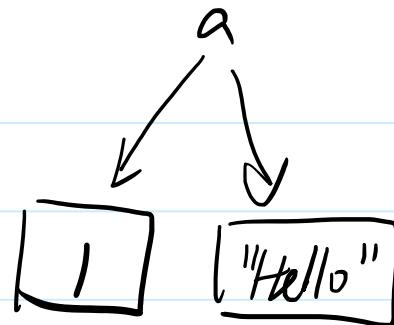
```
str = "a\b"
print(string.format("luaStr = %s, luaStr = %q", str, str))
```

赋值

2021年5月27日 10:13

$a = 1$

$a = "Hello"$



```
local x = 1
local y = 2
x,y = y,x --先取出值，再赋值给变量
print(x,y)
a = 1,2,3
print(a)
a,b = 1,2,3
print(a,b)
a,b,c = 1,2
print(a,b,c)
```

作用域

2021年5月27日 10:19

默认是全局。

local 是局部 这界是 end.

```
for i = 1,5 do ← 局部的
    print(i)
end
print(i)
```

选择结构

2021年5月27日 10:21

if 条件 then
语句块
end

if 条件 then
TR1
else
TR2
end .

if 条件1 then
TR1
elseif
TR2
:

else
TRn
end.

循环

2021年5月27日 10:23

```
local cnt = 0
local i = 1
while i <= 100 do
    cnt = cnt + i
    i = i+1
end
print("cnt = ",cnt)
```

```
local cnt = 0
local i = 1
repeat
    cnt = cnt + i
    i = i + 1
until i > 100
print("cnt = ",cnt)
```

```
local x = 2
local sqrt = x/2
repeat
    sqrt = (sqrt + x/sqrt)/2
    local error = math.abs(sqrt^2 - x)
until error < 0.000001 -- error的作用域包括until后面的条件语句
print(sqrt)
```

for

2021年5月27日 10:53

数值 for

for v = e1, e2, e3 do block end

遍历值 $\leftarrow e_1 e_2 e_3$ 只执行一次

```
--[[local cnt = 0
for i = 1,math.huge do
    cnt = cnt + i
    if cnt > 1000 then
        break
    end
end
print("cnt = ",cnt)
]]
function f(x)
    return 5*x
end
local num = 1
for i = 1,f(num) do
    print("hello")
    num = 2
end
```

逐个取

f(num) 逐个用 5 次

泛型 for iterator

for var_1, ..., var_n in expList do block end

table 列表/表

2021年5月27日 11:04

唯一的数据结构 由键值对构成的集合

key/键/索引 $\xrightarrow{\text{table}}$ 值/value/元素

表 行 [] = [newkey] = [old] = nil
查 找 修改 增加 拆分

```
local t = {}  
t["x"] = 3 -- ("x", 3)  
local k = "x"  
print(t[k]) -- 变量k得到值"x"作为t的键, 访问到3
```

运算符是[]的语法糖 $t["x"] \rightarrow 3$. $t.x = 3$

```
local x = "y"  
t[x] = 4  
print(t.x) -- t["x"] . 运算符后面的内容, 不能看成是一个变量的名字, 而是一个字符串的内容  
t["k"] = 5  
t[k] = 6 -- t["x"] = 6  
print(t.x) -- t["x"] --> 6  
print(t.k) -- t["k"] --> 5  
print(t[k]) -- t["x"] --> 6
```

编码规范

2021年5月27日 11:17

①尽量不在[]中写入字符串。

②尽量使用，采用字符串值作为键。

```
local player = {}  
player.minAtk = 50  
player.maxAtk = 80  
player.armor = 6  
player.health = 800  
player.mana = 200  
print(player.mana) -- 推荐用法  
print(player["mana"]) -- 恶心人用法
```

player.100 = 30

不符合编码规范。

```
--player.100 = 30  
player["100"] = 30  
print(player["100"])
```

数字类型的键

2021年5月27日 11:24

```
local t = {}  
t[1] = 1  
t["1"] = 2  
print(t[1]) -- 1  
print(t[2/2]) -- 只要数值相同就是同一个键，数值和字符串是不兼容  
print(t[3.14^0])  
print(t[1.0])  
print(t[tostring(1)]) -- 2
```

表构造器

2021年5月27日 11:28

列表式模式：看成一数组

```
local t = {"SUN", "MON", "TUE", "WED", "THUR", "FRI", "SAT"}  
--采用列表，直接写值的列表，键默认从1开始  
for i = 1,7 do  
    print(i,t[i])  
end
```

1	SUN
2	MON
3	TUE
4	WED
5	THUR
6	FRI
7	SAT

如果表的键全部都是正整数(不包含0)，那么这个表称为数组。

空洞 数组中间有值放nil
规则 无空洞 (井)

记录式表构造器

2021年5月27日 11:35

如果表的键全都是字符串类型，那么这个表称为**记录**：

```
local player = {  
    minAtk = 50, -- 左边应该是一个符合标识符规范的字符串内容  
    maxAtk = 80,  
    armor = 6,  
    health = 800,  
    mana = 200,  
    ["100"] = 30, -- 左边可以是方括号里面写键的真实值  
}  
print(player.mana) -- 推荐用法  
print(player["mana"]) -- 恶心人用法  
print(player["100"])
```

混合式构造器

2021年5月27日 11:40

```
local t = {
    len = 3,
    1, -- 只有值的内容，会自动从1开始编号
    2,
    3,
    ["1"] = 4,
    -- [3] = 5, 一旦发生冲突，忽视键=值版本的
    [100] = 5,
    {x = 1, y = 2}
}
print(t.len)
print(t[1])
print(t[3])
print(t[100])
print(t[4].x)
```

迭代器

2021年5月27日 14:31

pairs (t)

Returns three values: the `next` function, the table `t`, and `nil`, so that the construction

```
for k,v in pairs(t) do body end
```

will iterate over all key-value pairs of table `t`.

```
for k,v in pairs(_G) do
    print(k,v)
end
print("-----")
for k,v in pairs(_G.os) do
    print(k,v)
end
```

ipairs (t)

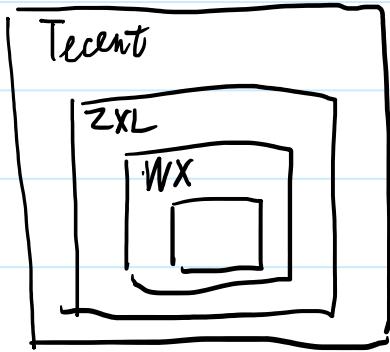
Returns three values: an iterator function, the table `t`, and 0, so that the construction

```
for i,v in ipairs(t) do body end
```

will iterate over the pairs $(1, t[1]), (2, t[2]), \dots$, up to the first integer key absent from the table.

嵌套表的安全访问

2021年5月27日 14:41



Company

leader

department

employee

((company or {}). leader or {}). department or {}

..

表支持的标准库函数

2021年5月27日 14:44

table.concat (table [, sep [, i [, j]]])

Given an array where all elements are strings or numbers, returns `table[i]..sep..table[i+1] .. sep.. table[j]`.

table.insert (table, [pos] value) *插入*

table.remove (table [, pos]) *弹出*

```
local t = {}  
for line in io.lines() do  
    table.insert(t,line)  
end  
print(#t)  
for line in io.lines() do  
    table.remove(t)  
    print('-----')  
    for k,v in pairs(t) do  
        print('\t',k,v)  
    end  
    print('-----')  
end
```

列表的底层实现

2021年5月27日 14:52

混合式 { 数组 union { number, boolean, nil , pointer }
哈希表 }

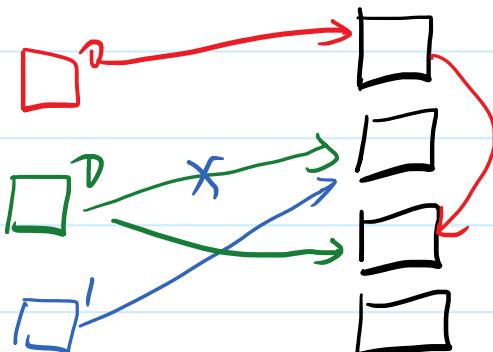
查找

key 是非整数/负数 → 哈希表

key 是正整数. key < n 行数组部分

key > n 行哈希表

哈希表 混合键表和开放寻址.



重建哈希表. 使用率 $> 50\%$

- ① 增添扩容数组 - 直行到新增部分不是最利用部分
- ② 将可以放到数组里的数据放入数组
- ③ 扩容. 重建哈希表

函数

2021年5月27日 15:07

函数是一等公民

① 函数是一个值

② 函数内部可以嵌套定义函数。

函数式编程

定义 创建一个函数值

调用 执行函数的代码

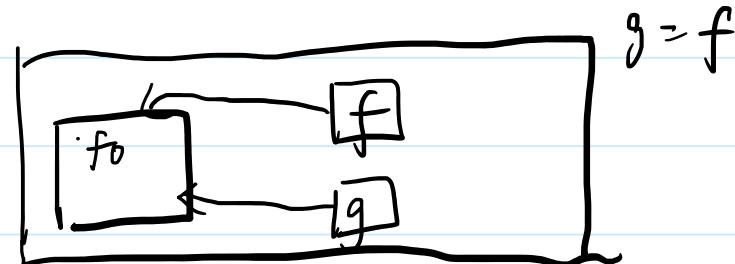
函数名(实参列表)

定义

function 函数名 (形参列表)
函数体
end



函数名 = function 形参 语句体 end
匿名函数



:运算符默认添加了一个self参数

2021年5月27日 15:25

```
function f(arg)
    return arg
end
print(f(1))
print(f("hello"))
g = f -- 函数是一等公民，可以像普通值一样作为右值
print(g(2))
local object = {}
object.attack = 50
object.func = function (self) print(self.attack) end
object.func(object)
object:func() -- 相当于给func传递了一个self参数
```

```
--object.func = function (self) print(self.attack) end
function object.func(self)
    print(self.attack)
end
--function object:func() print(self.attack) end
```

多返回值

2021年5月27日 15:49

```
```  
print(x,y,z) -- 5 7 nil 多返回值只能作为赋值、传参、表构造器和return表达式最右边一部分
-- 如果不满足，只返回第一个
x,y,z = 7,g() -- 赋值
print(x,y,z)
x,y,z = 1+g() -- 算术只返回一个
print(x,y,z)
print(g(),1) -- 传参
print(1,g())
local t1 = {g(),2} -- 表构造器，最右时返回多个
local t2 = {2,g()}
print(#t1,#t2)
function g1()
 return 3,g() -- return语句
end
function g2()
 return g(),3
end
print(g1())
print(g2())
```

# 递归

2021年5月27日 15:59

```
function sort(arr, left, right)
 if left < right then
 local pivot = partition(arr, left, right)
 sort(arr, left, pivot-1)
 sort(arr, pivot+1, right)
 end
end
function partition(arr, left, right)
 local j = left
 local pivot = arr[right]
 for i = left, right-1 do
 if arr[i] < pivot then
 arr[i], arr[j] = arr[j], arr[i]
 j = j + 1
 end
 end
 arr[j], arr[right] = arr[right], arr[j]
 return j;
end
```

在使用时，函数名一定要能识别对  
象的函数值

# 可变参数

2021年5月27日 16:08

... 单独使用 / 极右参数

↳ { ... }

```
function add(x,...)
 local cnt = x or 0
 for i,v in ipairs({...}) do
 cnt = cnt+v
 end
 return cnt
end
print(add())
print(add(1))
print(add(1,2))
print(add(1,2,3,4))
```

```
function present(a)
 return unpack(a)
end
x,y,z = present({1,2,3})
print(x,y,z)
```

unpack (list [, i [, j]])

解包.

Returns the elements from the given table. This function is equivalent to

```
return list[i], list[i+1], ..., list[j]
```

# 自己实现unpack

2021年5月27日 16:21

```
function myUnpack(a)
 -- {1,2,3} --> 1,{2,3}
 local ret = a[1]
 if #a > 0 then
 table.remove(a,1)
 else
 return
 end
 return ret,myUnpack(a)
end
x,y,z = myUnpack({3,5,7})
print(x,y,z)
```

# 自己实现printf

2021年5月27日 16:22

```
function printf(fmt,...)
 return io.write(string.format(fmt,...))
end
local name = "Panzi"
local age = 65
printf("I am %s. I am %d years old\n",name, age)
```

# select函数

2021年5月27日 16:25

`select (index, ...)`

If index is a number, returns all arguments after argument number index. Otherwise, index must be the string "#", and select returns the total number of extra arguments it received.

$\text{select}(3, 1, 2, 3, 4, 5) \rightarrow 3, 4, 5$

index 是一个数字：从第 index 个参数返回。

index 是一个 "#", 返回可变参数个数

```
x,y,z = select(3,1,2,3,4,5) -- 返回第3个（包括第三个）之后所有参数
print(x,y,z)
x = select("#",1,2,3,4,5,6,7)
print(x)
```

```
function add(...)
 local cnt = 0
 for i = 1,select("#",...) do
 cnt = cnt + select(i,...)
 end
 return cnt
end
print(add())
print(add(1,2,3,4,5))
```

# 函数是一等公民

2021年5月27日 16:33

function f()  
 local fa = 1  
 local fb = 2  
 local fc = 3

function g(x)  
 return x\*fc

end

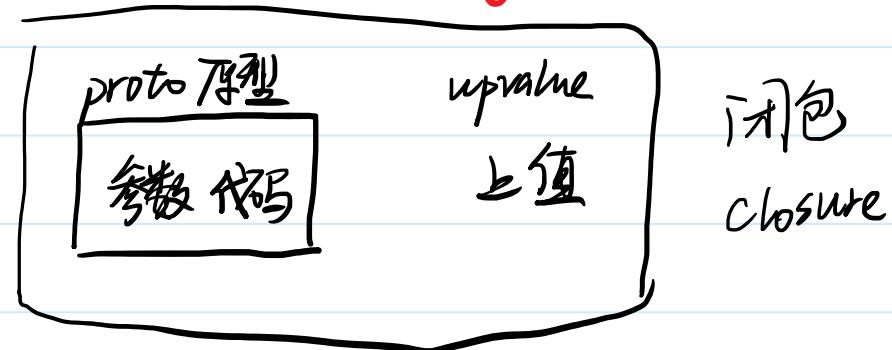
return g(fa)

end

print(f())

定义函数的本质是用匿名函数对变量赋值

g



语法作用域 → 静态作用域

在嵌套中，可以像访问局部变量一样去访问外层函数的局部变量

上值 upvalue

# 多重外围函数

2021年5月27日 17:05

```
function f()
 local fa = 1
 local fb = 2
 local fc = 3
 function g()
 local ga = 4
 function h()
 return ga*fc --upvalue是可以跨越多重外围函数的
 end
 return h()
 end
 return g()
end
print(f())
```

main是默认存在的

2021年5月27日

17:08

function main(args)

local a = 1

local b = 2

function f()

    return a+b ← a,b是f的 upvalue

end

print(f())

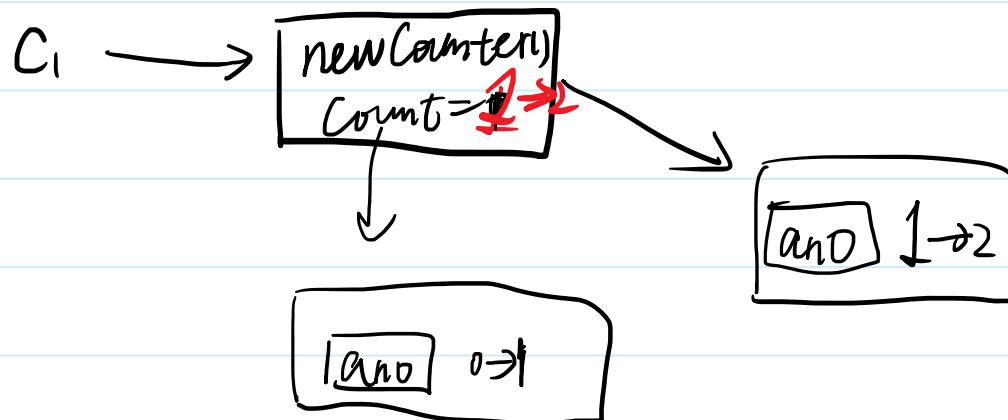
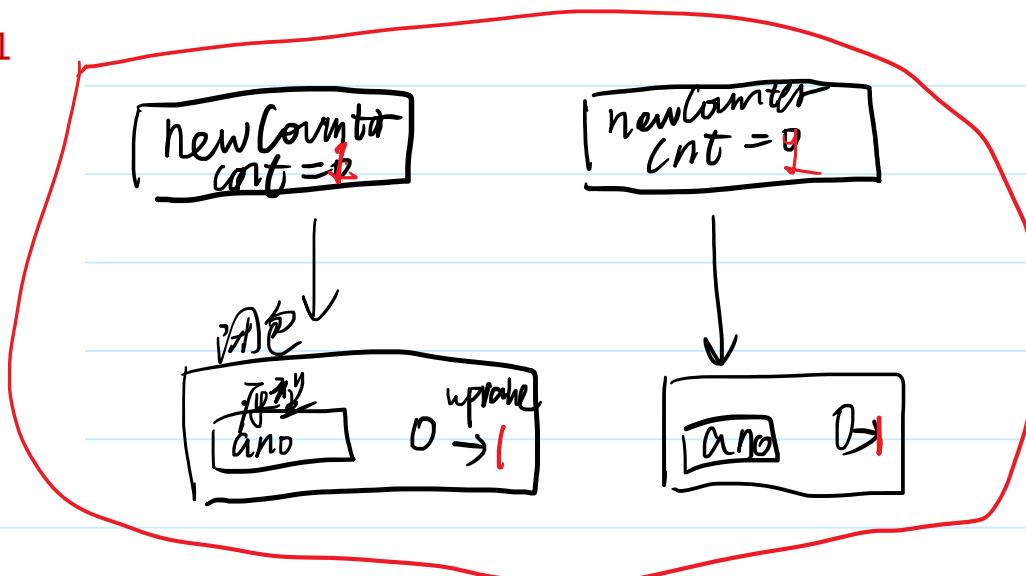
end

# 闭包的生命周期

2021年5月27日 17:10

```
function newCounter()
 local count = 0
 return function ()
 count = count + 1
 return count
 end
end
print(newCounter()())
print(newCounter()())
local c1 = newCounter()
print(c1())
print(c1())
```

gc 负责回收内存。再也没有机会被给了0  
值。...



# 迭代器的实现

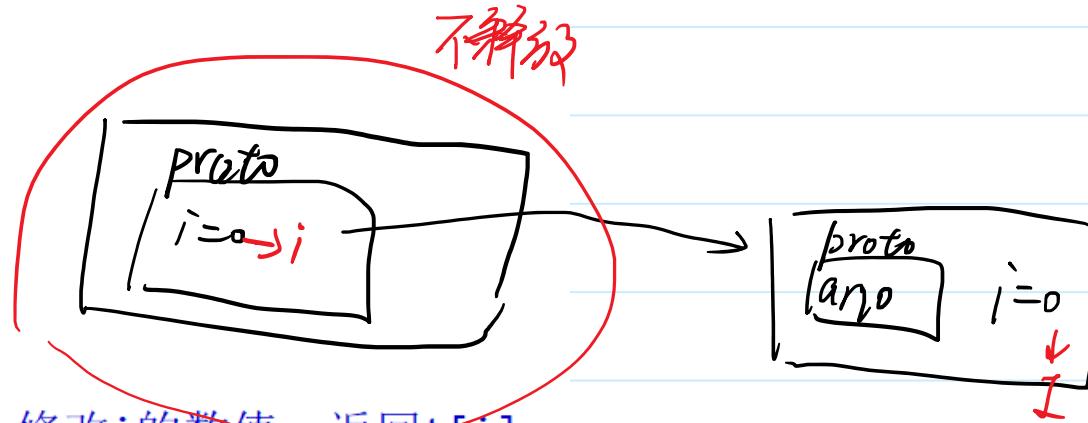
2021年5月27日 17:22

```
function factory(t)
 local i = 0
 return function()
 i = i + 1
 return t[i]
 end
end
```

-- 调用factory(t)发生了什么?

-- 调用匿名函数, 匿名会捕获i, 修改i的数值, 返回t[i]

```
local tb = {1,2,3,4,5}
for element in factory(tb) do
 print(element)
end
```



# luac -l 看字节码

2021年5月27日 17:32

main <31\_iter.lua:0,0> (19 instructions, 76 bytes at 0x5593cd8ac870)

0+ params, 7 slots, 0 upvalues, 5 locals, 7 constants, 1 function

|    |      |           |       |                  |
|----|------|-----------|-------|------------------|
| 1  | [7]  | CLOSURE   | 0 0   | ; 0x5593cd8aca50 |
| 2  | [1]  | SETGLOBAL | 0 -1  | ; factory        |
| 3  | [10] | NEWTABLE  | 0 5 0 |                  |
| 4  | [10] | LOADK     | 1 -2  | ; 1              |
| 5  | [10] | LOADK     | 2 -3  | ; 2              |
| 6  | [10] | LOADK     | 3 -4  | ; 3              |
| 7  | [10] | LOADK     | 4 -5  | ; 4              |
| 8  | [10] | LOADK     | 5 -6  | ; 5              |
| 9  | [10] | SETLIST   | 0 5 1 | ; 1              |
| 10 | [11] | GETGLOBAL | 1 -1  | ; factory        |
| 11 | [11] | MOVE      | 2 0   |                  |
| 12 | [11] | CALL      | 1 2 4 |                  |
| 13 | [11] | JMP       | 3     | ; to 17          |
| 14 | [12] | GETGLOBAL | 5 -7  | ; print          |
| 15 | [12] | MOVE      | 6 4   |                  |
| 16 | [12] | CALL      | 5 2 1 |                  |
| 17 | [11] | TFORLOOP  | 1 1   |                  |
| 18 | [12] | JMP       | -5    | ; to 14          |
| 19 | [13] | RETURN    | 0 1   |                  |

```
function factory(t)
 local i = 0
 return function()
 i = i + 1
 return t[i]
 end
end
```

-- 调用factory(t)发生了什么?  
-- 调用匿名函数，匿名会捕获i，修改i的数值，返回t[i]  
local tb = {1,2,3,4,5}  
for element in factory(tb) do  
 print(element)  
end

function <31\_iter.lua:1,7> (6 instructions, 24 bytes at 0x5593cd8aca50)

1 param, 3 slots, 0 upvalues, 2 locals, 1 constant, 1 function

|   |     |         |      |                  |
|---|-----|---------|------|------------------|
| 1 | [2] | LOADK   | 1 -1 | ; 0              |
| 2 | [6] | CLOSURE | 2 0  | ; 0x5593cd8ace90 |
| 3 | [6] | MOVE    | 0 1  |                  |
| 4 | [6] | MOVE    | 0 0  |                  |
| 5 | [6] | RETURN  | 2 2  |                  |
| 6 | [7] | RETURN  | 0 1  |                  |

# 嵌套函数

2021年5月27日 17:36

```
function factory(t)
 local i = 0
 return function()
 i = i + 1
 return t[i]
 end
end
```

---调用factory(t)发生了什么?

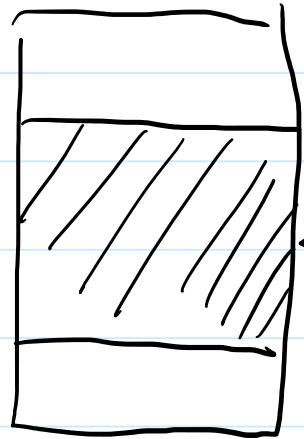
-- 调用匿名函数, 匿名会捕获i, 修改i的数值, 返回t[i]

```
local tb = {1,2,3,4,5}
for element in factory(tb) do
 print(element)
end
```

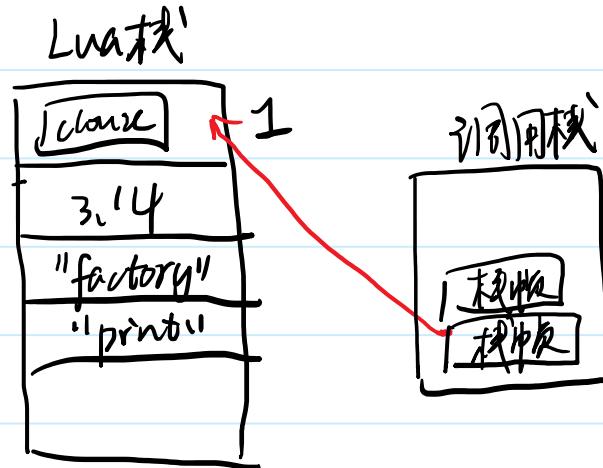
```
function <31_iter.lua:3,6> (8 instructions, 32 bytes at 0x5593cd8ace90)
0 params, 2 slots, 2 upvalues, 0 locals, 1 constant, 0 functions
 1 [4] GETUPVAL 0 0 ; i
 2 [4] ADD 0 0 -1 ; - 1
 3 [4] SETUPVAL 0 0 ; i
 4 [5] GETUPVAL t 0 1 ; t
 5 [5] GETUPVAL i 1 0 ; i
 6 [5] GETTABLE t[i] 0 0 1
 7 [5] RETURN 0 2
 8 [6] RETURN 0 1
```

# lua的函数调用内存布局

2021年5月27日 17:40



c malloc / tc malloc  
← 堆空间 → 虚拟机  
lua-state



主调函数在调用被调函数时，所有数据失去价值  
用被调函数栈帧覆盖新调函数。  
△△

尾调用优化

```
function f(i)
 print(i)
 return (f(i+1))
end
f(1)
```