基本语法

Lua 像 C 和PASCAL 一样几乎支持所有的传统语句，同时他还支持非传统多变量赋值和局部变量声明

赋值语句

单变量赋值

```
a = "hello"
b = b + 1     --这是改变变量数值的最朴素的方法
```

多变量同时赋值

```
a , b = 10 , 2*c  --方法1
a = 10; b = 2*c   --方法2
--方法1与方法2效果一样
```

函数调用

```
a , b = function_A() --函数返回值的第一个传给a，第二个传给b
```

Lua赋值语句操作原理

首先Lua会先运算出所有右边的值，再依次给予左边。这个计算出的值**不依赖于原变量**

因此我们可以十分方便的进行变量交换

```
x , y = y , x  --是的，这样看似在C中十分错误的写法，在Lua中是正确有效的，但两个变量必须在同一行(一个Chunk)
```

异常处理

- 变量个数 > 数值个数 --多余的变量值为nil
- 变量个数 < 数值个数--忽略多余值(间接有表示是先出值，在依次赋给变量)

Tips:

*Lua*是一一对应的

```
a , b , c = 0 --ERROR! b与c都会是nil
a , b , c = 0,0,0 --RIGHT!
```

局部变量与代码块(block)

在创建变量时在变量名前加 *local* 完成局部变量定义

局部变量仅在被定义的那个代码块中有效！

代码块

- 一个控制结构内

```
if i > 20 then
   local x     --这个x仅在这个if当中有效(else都不算if的控制结构)
   x = 20
else
   local x     --这个x仅在else中有效
   x = 3
end
```

- 一个循环体

```
while i<10 do
   local x = 2*i --x仅在这个while的这一次循环中生效
   i = i + 1
end
```

- 一个函数体

```
function Aoo(x , y)
   local x          --这个x仅在这个里面function生效
……
end
```

- 当前这一个文件中(就是这一个.lua文件中，如果多文件调用跳到其他文件则不生效，但是全局变量生效(好像就优化了extern))
- 一个chunk（在使用交互式编程中，一个chunk使用的local下一个chunk不再可用）

作用域性质

- 如果一个域中有同名的变量，则优先使用local，而不是全局变量
- 可以编写do……end来划分出一个block，这样可以更好的控制局部变量

```lua
local y = 0
do
  local x
  x = 2*16
  y = x
end --使用do……end指令完成对于local局部变量的作用控制，x在end之后就被释放掉了
print(y)
```

一个大型示例

```lua
x = 10
local i = 1
while i <= x do --此时的x是这个x=10的全局x
   local x = i*2
   print(x)     --此时的x是local x = i*2的这个x(局部优先全局)
   i = i + 1
end

if i > 20 then
   local x
   x = 20
   print(x + 2) --此时是local x(局部优先全局)
else
   print(x)     --此时是全局x=10这个x(优先级排到了全局)
end

print(x)        --此时是全局x
```

局部变量的好处

1. 避免变量命名冲突
2. 访问局部变量会快于全局变量，程序跑更快

控制结构语句

Lua认为nil和false才是假，0为真哦

if语句

```lua
--句式1
if 条件判断 then
```

```
    执行语句(条件判定为true执行)
end

--句式2
if 条件判断 then
    执行语句(条件判断为true执行)
else
    执行语句(条件判定为false执行)
end

--句式3
if 条件判断 then
    执行语句(条件判断为true执行)
elseif 条件判断 then
    执行语句(满足elseif执行)
..... --可以多个elseif操作
else
    执行语句(条件判定为false执行)
end
```

while语句

在满足条件时执行

```
while 判断条件 do
    执行语句(满足判断条件时执行)
end;
```

repeat-until语句

在满足条件后停止

```
repeat
    执行语句
until 判断条件(满足条件时停止);
```

for语句

数值for循环

```
for 控制变量=exp1,exp2,exp3 do --控制变量就像一个执行for的i，【exp1是初始值，exp2是终止值，exp3是每次要变化的数值】
   循环执行语句
end
```

如果还不清楚，可以进行C语言循环对照

```
for(i = 0;i <= 10;i++){ //i = 0这个初始值就是exp1，i <= 10的10这个终止值是exp2，i++可转换为i = i + 1这个1就是递归值就是exp3

}
```

注意，这个*exp2*终止值是对应的有等于号的等于这个终止值之后才会截止

Tips

- 这个exp2和exp3如果是被函数的返回值取的，则这个函数只会在开始时调用一次

```
for i = 1,f(x) do --这个f(x)仅会在初始时调用一次，并且返回值赋予exp2与exp3，而就算之后loop-part中还有f(x)，调用的依然是最开始的f(x)的值
   loop-part
end
```

- 控制变量是局部变量，且会自动声明，只在循环内有效

```
local x = 30
for x = 1 , 20 , 2 do --这个x是自动生成的局部变量，与上面的x无关(局部优先)
   print(x)           --调用的是循环的x 不是local x
end
```

- 不要在循环体中操作控制变量，那样结果不可预知！！！！

```
for i = 1, 10, 2 do
   if x > 5 then --这个操作是无效的，操作的x是循环的x，而操作结果仅在if中有效跳出之后还是会在循环体时还原
      x = x + 5
   end
end
```

范型**for**语句

用于遍历表中的数据

```
table1 = {……}                    --定义一个table表
for i , v in pairs(table1) do --遍历这个表(i是索引v是内容),又体现了先计算后赋值的特点,
如果只有i则只会给索引

end
```

泛型**for**查找顺序

首先·，Lua会寻找正常的数字序列中的最前者，之后才会找被自定义的索引（例如："+","niu"等）

泛型使用之对换表

```
local day = {……}
local jingxiang_day = {}
for i , v in pairs(day) do
   jingxiang_day[v] = i     --local的值是一个指代，如果存入这种可以带出去的变量中，则值就
带出去了
end
```

两个**for**相同之处

- for中的变量都是局部变量，仅在循环体中生效
- 在循环体内都不要修改控制变量的值

break 和 return 语句

break

用于退出当前循环，可退出的循环有：

- for
- repeat……until
- while

return

用来返回函数的值

在函数中会有一个默认的return(类似C++的析构函数)

特殊的语法要求

Lua中break和return只能出现在block的最后一句

- 也是作为Chunk的最后一句
- 或是在end，else，until之前
- do……end可以强制拉出return或break

有时候我们可以利用这个特性3来更好的调试

```
function f00 ()
    ……        --这一部分正常执行
    do
        return
    end
    ……          --这一部分将不会执行到，成为一个新的block延续之前的部分执行，这样好像可以输出更多的return?
end
```