# 数据类型

* nil，空值
* boolean，取值true和false（nil视为false）
* number，数字，lua使用浮点型，没有整形类型
* string，字符串，“..”连接两个字符串，以“\0”结尾
* function，函数对象
* userdata，用户数据
* thread，线程对象
* table，表，可以用作数组和字典，相当于面向对象中的类

# 操作符

## 算数操作符

* +，加
* -，减
* \*，乘
* /，除
* %，取余
* ^，指数运算
* 一元-，取负

## 逻辑运算符

* and，且，相当于&&
* or，或，相当于||

## 关系操作符

* <、<=
* >、>=
* =、~=

## 其他操作符

* #，一元操作符，返回字符串或表的长度，#“Hello”返回5

# 控制流

## 代码块

do

--代码

end

## if判断

if 2 > 1 then

--代码

elseif 2 > 2 then

--代码

else

--代码

end

## while控制流

在条件为true时，让程序重复地执行某些语句。执行语句前会先检查条件是否为true。

while 2 > 1 do

--代码

end

## repeat控制流

重复执行循环，直到指定的条件为真时为止。

local i = 1

repeat

--代码

i = i + 1

until i > 100

## for语句

重复执行指定语句，重复次数可在for语句中控制。

### 数字递增方式

--初始值为1，最大值为100，以步长值2递增

for local i = 1, 100, 2 do

--代码

end

### 迭代器方式

local v = [1, 2, 3]

for i in v do

--代码

end

# 关键字

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

* lua中的变量全是全局变量，哪怕是语句块或是函数里，除非用local显式声明为局部变量

# table操作

## 用作数组

local v = {“a”, “b”, “c”}

--等同于

local v = {}

v[1] = “a”

v[2] = “b”

v[3] = “c”

## 用作字典

local v = {[“a”] = 1, [“b”] = 2, [“c”] = 3}

v[“a”]

--等同于

v.a

# lua\_state

## lua堆栈

简单来说，lua和C/C++语言通信的主要方法是一个无处不在的虚拟栈。栈的特点是先进后出。索引方式可以是正数也可以是负数，区别：正数索引1永远表示栈底，负数索引-1永远表示栈顶。



C与栈的转换：

//1.创建一个state

lua\_State \*L = luaL\_newstate();

//2.入栈操作

lua\_pushstring(L, "Hello World! ");

lua\_pushnumber(L, 20);

//3.取值操作

if (lua\_isstring(L, 1))

{

lua\_tostring(L, 1);

}

if (lua\_isnumber(L, 2))

{

lua\_tonumber(L, 2);

}

//4.关闭state

lua\_close(L);

C++调用lua：

str = "Hello World!"

tbl = {name = "LinY", id = 19940511}

function add(a,b)

return a + b

end

//1.创建一个state

lua\_State \*L = luaL\_newstate();

if (L == NULL)

{

return;

}

//2.加载lua文件

int bRet = luaL\_loadfile(L, "hello.lua");

if (bRet)

{

std::cout << "load file error" << std::endl;

return;

}

//3.运行lua文件

bRet = lua\_pcall(L, 0, 0, 0);

if (bRet)

{

std::cout << "pcall error" << std::endl;

return;

}

//4.读取变量

lua\_getglobal(L, "str");

std::string str = lua\_tostring(L, -1);

std::cout << "str = " << str.c\_str() << std::endl;

//5.读取table

lua\_getglobal(L, "tbl");

lua\_getfield(L, -1, "name");

str = lua\_tostring(L, -1);

std::cout << "tbl:name = " << str.c\_str() << std::endl;

//6.读取函数

lua\_getglobal(L, "add");

lua\_pushnumber(L, 10);

lua\_pushnumber(L, 20);

int iRet = lua\_pcall(L, 2, 1, 0);//2-参数个数，1-返回结果个数

if (lua\_isnumber(L, -1))

{

double fValue = lua\_tonumber(L, -1);

std::cout << "Result is = " << fValue << std::endl;

}

//7.关闭state

lua\_close(L);

## lua.h函数

### state操作

LUA\_API lua\_State \*(lua\_newstate) (lua\_Alloc f, void \*ud);

LUA\_API void (lua\_close) (lua\_State \*L);

LUA\_API lua\_State \*(lua\_newthread) (lua\_State \*L);

LUA\_API lua\_CFunction (lua\_atpanic) (lua\_State \*L, lua\_CFunction panicf);

### 基本栈操作

LUA\_API int (lua\_gettop) (lua\_State \*L);//返回栈顶索引（即栈长度）

LUA\_API void (lua\_settop) (lua\_State \*L, int idx);//将栈顶设置为一个指定的位置，即修改栈中元素的数量。如果值比原栈顶高，则高的部分nil补足，如果值比原栈低，则原栈高出的部分舍弃。可以用lua\_settop(0)来清空栈

LUA\_API void (lua\_pushvalue) (lua\_State \*L, int idx);//将idx索引上的值的副本压入栈顶

LUA\_API void (lua\_remove) (lua\_State \*L, int idx);//移除idx索引上的值

LUA\_API void (lua\_insert) (lua\_State \*L, int idx);//弹出栈顶元素，并插入索引ind位置

LUA\_API void (lua\_replace) (lua\_State \*L, int idx);//弹出栈顶元素，并替换索引idx位置的值

LUA\_API int (lua\_checkstack) (lua\_State \*L, int sz);

LUA\_API void (lua\_xmove) (lua\_State \*from, lua\_State \*to, int n);

### stack -> C

LUA\_API int (lua\_isnumber) (lua\_State \*L, int idx);

LUA\_API int (lua\_isstring) (lua\_State \*L, int idx);

LUA\_API int (lua\_iscfunction) (lua\_State \*L, int idx);

LUA\_API int (lua\_isuserdata) (lua\_State \*L, int idx);

LUA\_API int (lua\_type) (lua\_State \*L, int idx);

LUA\_API const char \*(lua\_typename) (lua\_State \*L, int tp);

LUA\_API int (lua\_equal) (lua\_State \*L, int idx1, int idx2);

LUA\_API int (lua\_rawequal) (lua\_State \*L, int idx1, int idx2);

LUA\_API int (lua\_lessthan) (lua\_State \*L, int idx1, int idx2);

LUA\_API lua\_Number (lua\_tonumber) (lua\_State \*L, int idx);

LUA\_API lua\_Integer (lua\_tointeger) (lua\_State \*L, int idx);

LUA\_API int (lua\_toboolean) (lua\_State \*L, int idx);

LUA\_API const char \*(lua\_tolstring) (lua\_State \*L, int idx, size\_t \*len);

LUA\_API size\_t (lua\_objlen) (lua\_State \*L, int idx);

LUA\_API lua\_CFunction (lua\_tocfunction) (lua\_State \*L, int idx);

LUA\_API void \*(lua\_touserdata) (lua\_State \*L, int idx);

LUA\_API lua\_State \*(lua\_tothread) (lua\_State \*L, int idx);

LUA\_API const void \*(lua\_topointer) (lua\_State \*L, int idx);

### C -> stack

LUA\_API void (lua\_pushnil) (lua\_State \*L);

LUA\_API void (lua\_pushnumber) (lua\_State \*L, lua\_Number n);

LUA\_API void (lua\_pushinteger) (lua\_State \*L, lua\_Integer n);

LUA\_API void (lua\_pushlstring) (lua\_State \*L, const char \*s, size\_t l);

LUA\_API void (lua\_pushstring) (lua\_State \*L, const char \*s);

LUA\_API const char \*(lua\_pushvfstring) (lua\_State \*L, const char \*fmt,

va\_list argp);

LUA\_API const char \*(lua\_pushfstring) (lua\_State \*L, const char \*fmt, ...);

LUA\_API void (lua\_pushcclosure) (lua\_State \*L, lua\_CFunction fn, int n);

LUA\_API void (lua\_pushboolean) (lua\_State \*L, int b);

LUA\_API void (lua\_pushlightuserdata) (lua\_State \*L, void \*p);

LUA\_API int (lua\_pushthread) (lua\_State \*L);

### Lua -> stack

LUA\_API void (lua\_gettable) (lua\_State \*L, int idx);

LUA\_API void (lua\_getfield) (lua\_State \*L, int idx, const char \*k);

LUA\_API void (lua\_rawget) (lua\_State \*L, int idx);

LUA\_API void (lua\_rawgeti) (lua\_State \*L, int idx, int n);

LUA\_API void (lua\_createtable) (lua\_State \*L, int narr, int nrec);

LUA\_API void \*(lua\_newuserdata) (lua\_State \*L, size\_t sz);

LUA\_API int (lua\_getmetatable) (lua\_State \*L, int objindex);

LUA\_API void (lua\_getfenv) (lua\_State \*L, int idx);

1. stack -> Lua

LUA\_API void (lua\_settable) (lua\_State \*L, int idx);

LUA\_API void (lua\_setfield) (lua\_State \*L, int idx, const char \*k);

LUA\_API void (lua\_rawset) (lua\_State \*L, int idx);

LUA\_API void (lua\_rawseti) (lua\_State \*L, int idx, int n);

LUA\_API int (lua\_setmetatable) (lua\_State \*L, int objindex);

LUA\_API int (lua\_setfenv) (lua\_State \*L, int idx);

### load and run Lua code

LUA\_API void (lua\_call) (lua\_State \*L, int nargs, int nresults);

LUA\_API int (lua\_pcall) (lua\_State \*L, int nargs, int nresults, int errfunc);

LUA\_API int (lua\_cpcall) (lua\_State \*L, lua\_CFunction func, void \*ud);

LUA\_API int (lua\_load) (lua\_State \*L, lua\_Reader reader, void \*dt,

const char \*chunkname);

LUA\_API int (lua\_dump) (lua\_State \*L, lua\_Writer writer, void \*data);

### 协同函数

LUA\_API int (lua\_yield) (lua\_State \*L, int nresults);

LUA\_API int (lua\_resume) (lua\_State \*L, int narg);

LUA\_API int (lua\_status) (lua\_State \*L);

### 废弃协同函数

#define LUA\_GCSTOP 0

#define LUA\_GCRESTART 1

#define LUA\_GCCOLLECT 2

#define LUA\_GCCOUNT 3

#define LUA\_GCCOUNTB 4

#define LUA\_GCSTEP 5

#define LUA\_GCSETPAUSE 6

#define LUA\_GCSETSTEPMUL 7

LUA\_API int (lua\_gc) (lua\_State \*L, int what, int data);

### 其他函数

LUA\_API int (lua\_error) (lua\_State \*L);

LUA\_API int (lua\_next) (lua\_State \*L, int idx);

LUA\_API void (lua\_concat) (lua\_State \*L, int n);

LUA\_API lua\_Alloc (lua\_getallocf) (lua\_State \*L, void \*\*ud);

LUA\_API void lua\_setallocf (lua\_State \*L, lua\_Alloc f, void \*ud);

### 一些有用的宏

#define lua\_pop(L,n) lua\_settop(L, -(n)-1)

#define lua\_newtable(L) lua\_createtable(L, 0, 0)

#define lua\_register(L,n,f) (lua\_pushcfunction(L, (f)), lua\_setglobal(L, (n)))

#define lua\_pushcfunction(L,f) lua\_pushcclosure(L, (f), 0)

#define lua\_strlen(L,i) lua\_objlen(L, (i))

#define lua\_isfunction(L,n) (lua\_type(L, (n)) == LUA\_TFUNCTION)

#define lua\_istable(L,n) (lua\_type(L, (n)) == LUA\_TTABLE)

#define lua\_islightuserdata(L,n) (lua\_type(L, (n)) == LUA\_TLIGHTUSERDATA)

#define lua\_isnil(L,n) (lua\_type(L, (n)) == LUA\_TNIL)

#define lua\_isboolean(L,n) (lua\_type(L, (n)) == LUA\_TBOOLEAN)

#define lua\_isthread(L,n) (lua\_type(L, (n)) == LUA\_TTHREAD)

#define lua\_isnone(L,n) (lua\_type(L, (n)) == LUA\_TNONE)

#define lua\_isnoneornil(L, n) (lua\_type(L, (n)) <= 0)

#define lua\_pushliteral(L, s) \

lua\_pushlstring(L, "" s, (sizeof(s)/sizeof(char))-1)

#define lua\_setglobal(L,s) lua\_setfield(L, LUA\_GLOBALSINDEX, (s))

#define lua\_getglobal(L,s) lua\_getfield(L, LUA\_GLOBALSINDEX, (s))

#define lua\_tostring(L,i) lua\_tolstring(L, (i), NULL)