lua命令：

```
#enter shell
lua
#excute script file
lua xxx.lua
```

lua脚本：

```
#!/usr/local/bin/lua
```

核心概念：

As a extension language, Lua has no notion of a 'Main' program: it only works embedded in a host client, called the embedding program or simply the host. The host program can invoke functions to execute a piece of Lua code, can write and read Lua variables, and can register C functions to be called by Lua code.

Lua distrubition includes a sample host program called lua, which uses the Lua library to offer a complete, standalone Lua interpreter, for interactive or batch use.

Lua is a dynamically typed language. This means that variables do not have types; only values do. There are no type definitions in the language. All values carry their own type.

Lua can call functions written in Lua and functions written in C. Both are represented by the type function.

特性：

轻量级

可配置，可扩展

兼容性

类C的语言，大小写敏感

在Lua中，一切都是变 量，除了关键字

特点：

1. 变量名没有类型，值才有类型，变量名在运行时可与任何类型的值绑定。
2. 语言只提供唯一一种数据结构，称为表（Table），它混合了数组、哈希，可以用任何类型的值作为key和value；提供了一致且富有表达力的表构造语法，使得Lua很适合描述复杂的数据。
3. 函数是一等类型，支持匿名函数和正则尾递归（proper tail recursion）。
4. 支持词法界定（lexical scoping）和闭包（closure）。
5. 提供thread类型和结构化的协程（coroutine）机制，在此基础上可方便实现协作式多任务。
6. 运行期能编译字符串形式的程序文本并载入虚拟机执行。

7. 通过元表（meta-table）和元方法（meta-method）提供的动态元机制（dynamic meta-mechanism），从而允许程序运行时根据需要改变或扩充语法设施的内定语义。
8. 能方便地利用表和动态元机制实现基于原型（prototype-based）的面向对象模型。
9. 从5.1版开始提供了完善的模块机制，从而更好地支持开发大型的应用程序。

注释：

单行：--

多行：--[[   ]]

关键字：

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

元表与元方法:

Every value in Lua can have a metetable. This meatball is an ordinary Lua table that define the behavior of the original value under certain special operations. You can change several aspects of the behavior of operations over a value by setting specific fields in its metatable. The keys in a meatball are derived from the event names; the corresponding values are called metamethods.

You can query the metatable of any value using the getmetatable function. You can replace the metatable of tables using the setmetatable function. You can not change the metatable of other types from Lua code(except by using the debug library); you must use the C API for that.

Tables and full userdata have individual metatables. Values of all other types share one single meatball per type; that is, there is one single metatable for all numbers, one for all strings, etc. By default, a value has no metatable, but the string library sets a metatable for the string type.

变量类型： type(<var-name>)

Nil   Boolean   Number   String   Function   Table   Thread   Userdata

==

Nil：

nil

Number：double

例子：

```
num = 234

num = 0.4

num = 4.57e-3

num = 0.3e12
```

num = 0xef45

函数：

tostr:

tostring(123)

String：‘’ “” [[ ]]

例子：

```
'This is a "string".\n'
"This is a string.\n"
[[
string in line 'one'
string in line "two"
]]
[=[
string in line [[one]]
string in line [[two]]
]=]
```

函数：

拼接：

‘abc’..’def’

tonum:

tonumber(‘123’)

len:

print(#’12345')

Boolean：

true，false

Condition - False：

false，nil

Function：

定义：

固定参数：

```
function <func-name>(<args>)
    ……
end
```

变长参数：

```
function <func-name>(…)
    local args = {…}
    …...
end
```

限制：

在Lua 里函数定义必须在调用之前执行，这是因为Lua里的函数定义本质上是变量赋值。

```
function foo() …… end
====
foo = function () …… end
```

参数传递：

在常用基本类型中，除Table是按地址传递外， 其它都是按值传递的。

返回值：

Lua中函数可以返回多个值，如下：

定义：

```
function swap(a, b)
    return b, a
end
```

使用：

```
a, b = 1, 2
a, b = swap(a, b)
```

虚变量：

当一个方法返回多个值时，有些返回值有时候用不到，要是声明很多变量来一一接收并不太合适，于是Lua提供了一个虚变量（dummy variable），以单个下划线（“_”）来命名，用它来丢弃不需要的数据，仅仅起到占位符的作用。

```
local start, finish = string.find(‘Hello’, ‘he’)

local start = string.find(‘Hello’, ‘he’)

local _, finish = string.find(‘Hello’, ‘he’)
```

函数与方法：

静态方法和函数没有区别：

```
func1(……)

table.static_method(……)
```

实例方法和静态方法的唯一区别在于实例方法的第一个参数：

```
table.instance_method(table, ……)

table:instance_method(……)
```

动态调用：

定义：

```
local function doAction(method, …)

    local args = {…} or {}

    mothed(unpack(args, 1, table.maxn(args))

end
```

使用：

```
local function run(x, y)

    ngx.say(‘run’, x, y)

end

doAction(run, 1, 2)
```

Table：{} 关联数组

说明：

Table是关联数组，不仅可以使用整数来索引，还可以使用除了nil之外的其他值进行索引。

Lua 中的模块、包、对象，元表均是使用table来表示的。

在Lua中，数组下标从1开始计数。

例子：

```
T1 = {[1]=‘one’, [2]=’two’}

one = T1[1]
```

T1[3] = ’three’

T1[‘four’] = 4

T1.four = 4

print(T1.four)

方法：

Thread：

Userdata：

作用域：

在默认情况下，函数外的变量总被认为是全局的，除非你在前面加上”local”；函数内的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。

local a,b,c = 1,2,3 -- a,b,c都是局部变量

迭代：

迭代文件：

io.lines()

迭代Table：

pairs(<value-of-table>)

逻辑：

算数：

+ - * / ^ %

..

逻辑：

== ~= > < >= <=

and or not

分组：

()

语句分割：

在Lua中，语句之间可以用分号";"隔开，也可以用空白隔开。

结构：

if－else：

```
    if <condition> then
        ……
    elseif <condition> then
        …….
    else
        …….
    end
for：
    for i = <start>,<end>,<step> do
        ……
        break
    end

    for i, v in pairs(<value-of-table>) do
        …...
    end
while：
    while <condition> do
        ……
    end
until：
    repeat
        ……
    until <condition>
function：
    def:
        function <func-name>(<args>)
            ……
        end
```

call:

    <func-name>(<args>)

closure:

return:

    return <value>

multiple assign:

    a, b, c = 1, 2, 3

模块：

Lua 的模块是由变量、函数等已知元素组成的 table。

引用模块：

函数 require 有它自己的文件路径加载策略，它会尝试从 Lua 文件或 C 程序库中加载模块。

lua模块：app/utils.lua

    local utils = require “app.utils"

创建模块：

lua模块：app/utils.lua（/path/to/folder）

    module("app.utils", package.seeall)

设置搜索路径：以；为分割符

.lua - package.path - package.loadfile

    package.path的默认值为LUA_PATH环境变量的值。

    package.path = [[/path/to/folder/?.lua;]] .. package.path

.so - package.cpath - package.loadlib

    package.cpath的默认值为LUA_CPATH环境变量的值。

互操作与嵌入：

与C语言的互操作：

在C函数中调用Lua脚本

在Lua脚本中调用C函数

在Host 程序中嵌入Lua：

Lua的官方实现完全采用ANSI C编写，能以C程序库的形式嵌入到宿主程序中。

标准库：

basic:

环境：Lua将其所有的全局变量保存在一个常规的table中，这个table称为“环境”。

_G — 全局环境表(全局变量)

_VERSION — 返回当前Lua的版本号

函数：

type (v)

print (…)

getmetatable (object)

setmetatable (table, metatable)

getfenv (f)

setfenv (f, table)

collectgarbage (opt [, arg])

require (modname)

module(modname, package.seeall)

rawlen (v)

rawequal (v1, v2)

rawget (table, index)

rawset (table, index, value)

next (table [, index])

ipairs (table)

pairs (table)

dofile (filename)

load (func [, chunkname])

loadfile ([filename])

loadstring (string [, chunkname])

pcall (f, arg1, ···)

xpcall (f, err)

select (index, ···)

- tonumber (e [, base])
- tostirng (e)
- unpack (list [, i [, j]])
- assert (v [, message])
- error (message [, level])

io: (.:)

- io.close
- io.flush
- io.input
- io.lines
- io.open
- io.output
- io.popen
- io.read
- io.stderr
- io.stdin
- io.stdout
- io.tmpfile
- io.type
- io.write
- file:close
- file:flush
- file:lines
- file:read
- file:seek
- file:setvbuf
- file:write

string: (.)

- string.upper

- string.lower

table: (.)

- table.insert
- table.remove
- table.sort

utf8: (.)

- utf8.char
- utf8.len

coroutine: (.)

- coroutine.create
- coroutine.status

debug: (.)

- debug.debug
- debug.getinfo

os: (.)

- os.date
- os.time
- os.rename
- os.remove
- os.execute

package: (.)

- package.path
- package.cpath
- package.loaded

math: (.)

- math.abc
- math.max
- math.min
- math.pi

math.random

C API:

辅助库: