Lua 快速参考

表、函数、元表、协程、模块、模式匹配

基础

Hello World

```lua
print("Hello, Lua!")
```

变量与赋值

```lua
local name = "Lua"      -- local variable
x = 10                  -- global (avoid)
local a, b = 1, 2      -- multiple assignment
a, b = b, a            -- swap values
```

注释

```lua
-- single line comment
--[[ multi-line
     comment ]]
```

运算符

`+ - * / %`	算术运算符
`//`	整除（5.3+）
`^`	幂运算
`..`	字符串拼接
`#`	长度运算符
`== ~=`	等于 / 不等于
`and or not`	逻辑运算符

类型

数据类型

`nil`	无值；假值
`boolean`	true 或 false
`number`	双精度浮点数（5.3+ 也支持整数）
`string`	不可变字节序列
`table`	关联数组（唯一的复合类型）
`function`	一等公民的闭包
`userdata`	封装给 Lua 使用的 C 数据
`thread`	协程句柄

类型检查

```lua
print(type(42))        -- "number"
print(type("hi"))      -- "string"
print(type(nil))       -- "nil"
print(type({}))        -- "table"
```

表（Table）

数组风格的表

```lua
local fruits = {"apple", "banana", "cherry"}
print(fruits[1])          -- "apple" (1-indexed)
table.insert(fruits, "date")
table.remove(fruits, 2)   -- remove "banana"
print(#fruits)            -- length
```

字典风格的表

```lua
local user = {name = "Alice", age = 30}
user.email = "a@b.com"    -- add field
user["name"] = "Bob"      -- bracket access
user.age = nil            -- remove field
```

表函数

`table.insert(t, v)`	向数组末尾追加值
`table.insert(t, i, v)`	在位置 i 插入
`table.remove(t, i)`	删除位置 i 的元素
`table.sort(t [,cmp])`	就地排序数组
`table.concat(t, sep)`	将数组元素连接为字符串
`table.move(t,a,b,c)`	将 a..b 范围的元素移到位置 c

函数

函数定义

```lua
local function add(a, b)
  return a + b
end
local mul = function(a, b) return a * b end
print(add(2, 3))  -- 5
```

可变参数与多返回值

```lua
local function sum(...)
  local s = 0
  for _, v in ipairs({...}) do s = s + v end
  return s
end
local function swap(a, b) return b, a end
local x, y = swap(1, 2)
```

闭包

```lua
local function counter()
  local n = 0
  return function()
    n = n + 1; return n
  end
end
local c = counter()
print(c(), c())  -- 1  2
```

控制流

条件

```lua
if x > 0 then
  print("positive")
elseif x == 0 then
  print("zero")
else
  print("negative")
end
```

循环

```lua
for i = 1, 10 do print(i) end
for i = 10, 1, -1 do print(i) end
for k, v in pairs(tbl) do print(k, v) end
for i, v in ipairs(arr) do print(i, v) end
```

While 与 Repeat

```lua
while x > 0 do x = x - 1 end
repeat
  x = x + 1
until x >= 10
```

字符串

字符串函数

`string.len(s) / #s`	字节长度
`string.sub(s, i, j)`	截取 i 到 j 的子串
`string.upper(s)`	转为大写
`string.lower(s)`	转为小写
`string.rep(s, n)`	重复字符串 n 次
`string.reverse(s)`	反转字符串
`string.format(fmt, ...)`	printf 风格格式化
`string.find(s, pat)`	查找模式，返回索引
`string.gsub(s, pat, rep)`	全局替换
`string.gmatch(s, pat)`	遍历所有模式匹配

模式字符

`.`	任意字符
`%a / %A`	字母 / 非字母
`%d / %D`	数字 / 非数字
`%w / %W`	字母数字 / 非字母数字
`%s / %S`	空白 / 非空白
`%p`	标点符号
`* + - ?`	贪婪、贪婪、懒惰、可选

元表

设置元表

```lua
local mt = {}
mt.__add = function(a, b)
  return {val = a.val + b.val}
end
local a = setmetatable({val=1}, mt)
local b = setmetatable({val=2}, mt)
local c = a + b  -- c.val == 3
```

常用元方法

`__index`	查找缺失键（表或函数）
`__newindex`	拦截新键赋值
`__add / __sub / __mul`	算术运算符
`__eq / __lt / __le`	比较运算符
`__tostring`	自定义字符串表示
`__len`	自定义 # 运算符
`__call`	将表作为函数调用
`__concat`	自定义 .. 运算符

用元表实现 OOP

```lua
local Dog = {}; Dog.__index = Dog
function Dog.new(name)
  return setmetatable({name=name}, Dog)
end
function Dog:bark() print(self.name.." says Woof") end
local d = Dog.new("Rex"); d:bark()
```

协程

协程生命周期

`coroutine.create(f)`	从函数创建协程
`coroutine.resume(co, ...)`	启动或继续协程
`coroutine.yield(...)`	挂起执行，返回值
`coroutine.status(co)`	"running"、"suspended"、"dead"
`coroutine.wrap(f)`	创建可调用的协程包装器

Lua 快速参考

协程示例

```
local function gen(max)
  for i = 1, max do coroutine.yield(i) end
end
local co = coroutine.wrap(gen)
print(co(5))  -- 1
print(co())   -- 2
```

模块

创建模块

```
-- mylib.lua
local M = {}
function M.greet(name)
  return "Hello, " .. name
end
return M
```

使用模块

```
local mylib = require("mylib")
print(mylib.greet("World"))
```

标准库

math	数学函数（sin、random、huge 等）
string	字符串操作与模式匹配
table	表操作（insert、sort 等）
io	文件 I/O 操作
os	操作系统接口（时间、时钟、执行）
debug	调试接口（慎用）

常见模式

三元运算习惯用法

```
-- Lua has no ternary; use and/or idiom
local val = condition and "yes" or "no"
-- Caution: fails if "yes" is false/nil
```

安全的表访问

```
local function get(t, ...)
  for _, k in ipairs({...}) do
    if type(t) ~= "table" then return nil end
    t = t[k]
  end
  return t
end
get(config, "db", "host")  -- safe nested access
```

ipairs 与 pairs 的区别

```
-- ipairs: array part, stops at first nil
for i, v in ipairs(arr) do print(i, v) end
-- pairs: all keys (unordered)
for k, v in pairs(tbl) do print(k, v) end
```