lua 代码编写规范

注：* 表示最基本的，优先级最高，都应该做到；优先级越低，能力要求越高。

A+ (95~100]

A (90~95]

B+ (80~90]

B (70~80]

C (60~70]

D (40~60]

E [0 ~40]

1. 命名规范

1.1 * 文件命名使用小写字母，进程入口文件为 main.lua(5)

```
----- main.lua
local function main(arg1, arg2)
    print(arg1, arg2)
end

local arg1, arg2 = ...
main(arg1, arg2)

----- 命令行：
lua main.lua hello world
```

1.2 * 变量名、函数名风格在一个模块中要统一(10)

1.2.1 *小写字母下划线法（推荐）

```
local my_var = 1
local function my_func(...)
    -- TODO
end
```

1.2.2 * Java 命名法

```
local myVar = 1
local function myFunc(...)
    -- TODO
end
```

1.2.3 * 常量和枚举命名

枚举：以大写字母开始的驼峰式，如:

```
local StNew, StRun, StDead = 0, 1, 2
```

常量：全大写字母加下划线，如:

```
local CUR_VERSION = "v0.1"
```

1.2.4 * 单数与复数命名

```
local result, err = db:execute("delete from xxx")     -- 单个结果，表示成功或者失败
local results, err = db:select("select * from xxx")   -- 结果集，可以有 0 个或多个结果
```

或者使用其他模糊的命名，不用取分单数复数

1.2.5 *** 在第三方框架上写代码，新增的代码选择 2.1 或 2.2

1.3 * 变量名使用有意义的英文，禁止使用拼音(5)

1.4 ** i,k,v,t 常作临时变量，_作为可以忽略的变量(5)

```
local arr = {"a", "b", "c"}
for _, v in ipairs(arr) do      -- 不需要下标，用_忽略
    print(v)
end
```

2. 文件组织

2.1 *** 禁止把 lua 当其他语言使用(5)

每种语言都有自己的使用规范，与其把 lua 当作自己熟悉的语言使用，还不如不用。

2.2 * 禁止使用全局变量，所有的声明加上 local 限定词(10)

原因：全局便量赋值随意，模块越大越难维护

2.3 * 使用空行 (10)

2.3.1 * 方法之间

2.3.2 * 方法内部的逻辑段乱小节之间

2.3.3 * 在注释行之前

```
--- 良好风格：
local function xxx()
    -- TODO
end

-- 返回 arr 的最大值 （前面空一行）
-- @arr : 一个数字的数组
-- @return : 最大值
local function my_max(arr)
    local max = 0
    for _, n in ipairs(arr) do
        if n > max then
            max = n
        end
    end

    -- 前面是独立逻辑段，空一行
    return max
end

--- 良好风格：
local function chk_adtype()
    if not lfs.attributes(adtype_path) then
        log.error("missing %s", adtype_path)
        return
    end

    local s = read(adtype_path)                    assert(s)
    local cur_switch = s:find("webui") and 0 or 1
```

```
    if not lfs.attributes(cloud_path) then
        return
    end
end

--- 不良风格，难以阅读：
local function chk_adtype()
    if not lfs.attributes(adtype_path) then
        log.error("missing %s", adtype_path)
        return
    end
    local s = read(adtype_path)                         assert(s)
    local cur_switch = s:find("webui") and 0 or 1
    if not lfs.attributes(cloud_path) then
        return
    end
end
```

2.4 * 使用空格(10)

2.4.1 * 变量定义时

2.4.2 * 运算符前后

2.4.3 * 参数列表

2.4.4 * for 语句

```
a > b and a or b          -- 良好风格
a > b  and a or  b        -- 不良风格，有多余空格

local a, b, c, max            -- 良好的风格
local a,b,c,max               -- 不良的风格，没有空格

if a > b then             -- 良好的风格
    max = a
end
```

```lua
if a>b then max=a end      -- 不良的风格，操作符前后没有空格

data = data_table[index]   -- 良好的风格
data = data_table[ index ] -- 不良的风格，有多余空格

function(pos_x, pos_y)     -- 良好的风格
function (pos_x,pos_y)     -- 不良的风格，function 后面不需要空格，参数之间没有空格
```

2.5 * 缩进使用制表符(10)

代码缩进统一使用制表符，禁止制表符和空格混用

2.6 * 禁止使用; (5)

lua 中，;是无意义的，不要当 C 用。

2.7 *** 文件、函数声明前面、难理解的逻辑要加注释(10)

2.7.1 ** 文件注释要说明本文件的作用

2.7.2 ** 复杂函数要注释说明函数作用，参数和返回值

格式可以参考：

```lua
--[[
验证用户名密码是否正确
@param username ： 用户名，不能为 nil
@param password ： 密码，不能为 nil
@return ： 成功为 true，错误返回 nil，错误消息
]]
local function validate(username, password)
    assert(username and password)

    if not (username == "hello" and password == "world") then
        return nil, "invalid username password"
    end

    return true
```

```
end
```

2.8 ** 单行注释使用 --，间隔一个空格(5)

```
-- this is my comment
```

2.9 ** 块注释使用 --[[...]]，间隔一行(5)

```
--[[
this is my comment line 1
this is my comment line 2
]]
```

2.10 * 函数返回值(10)

函数出错时，返回 nil, error_msg，正确时，返回结果。

```
local function query(sql)
    local rs, e = mysql:select(sql)
    if not rs then
        return nil, e     -- 出错时的返回值
    end

    return rs             -- 正常的返回值
end
```

2.11 *** 提取公共函数(5)

使用超过两次的公共逻辑，要提取为函数。

```
--- 良好风格
local function reply(r, e)
    return js.encode({r = r, e = e})
end

local function validate_aux(param)
    local username, password = param.username, param.password

    if not username then
        return nil, "miss username"
```

```lua
    end

    if not password then
        return nil, "miss password"
    end

    if not (username == "hello" and password == "world") then
        return nil, "invalid username password"
    end

    return true
end

local function validate(param)
    local r, e = validate_aux(param)
    if not r, e then
        return reply(1, e)
    end

    return reply(0, "ok")
end

--- 不良风格
local function validate(param)
    local username, password = param.username, param.password

    if not username then
        return js.encode({r = 1, e = "miss username"})
    end

    if not password then
        return js.encode({r = 1, e = "miss password"})
    end

    if not (username == "hello" and password == "world") then
        return js.encode({r = 1, e = "invalid username password"})
    end

    return js.encode({r = 0, e = "ok"})
end
```

2.12 * 提取公共变量(10)

在一个函数内部，引用超过两次的应该提取为变量

```
--- 良好风格
local function check_user_timeout(user_map)
    local now = os.time() -- 公共变量

    local timeout_users = {}
    for username, userinfo in pairs(user_map) do
        local diff = userinfo.active - now -- 公共变量
        if diff > 3600 then
            table.insert(timeout_users, {username = username, diff = diff})
        end
    end

    return timeout_users
end

--- 不良风格
local function check_user_timeout(user_map)
    local timeout_users = {}
    for username, userinfo in pairs(user_map) do
        -- userinfo.active, os.time --使用都超过两次，造成重复计算
        if os.time() - userinfo.active > 3600 then
            table.insert(timeout_users, {username = username, diff = os.time() -
userinfo.active})
        end
    end

    return timeout_users
end
```

2.13 ** 使用短路写法，少用 else (5)

当条件不满足，后面的逻辑不需要再跑时，直接返回

```
--- 良好风格
local function validate(param)
    local username, password = param.username, param.password

    if not username then
```

```
        return nil, "miss username"
    end

    if not password then
        return nil, "miss password"
    end

    if not (username == "hello" and password == "world") then
        return nil, "invalid username password"
    end

    return true
end

--- 不良风格，很难理解
local function validate(param)
    local username, password = param.username, param.password

    local err
    if not username then
        err = "miss username"
    elseif not password then
        err = "miss password"
    elseif not (username == "hello" and password == "world") then
        err = "invalid username password"
    end

    if err then
        return nil, err
    else -- 这个 else 是没有用的
        return true
    end
end
```

2.14 ** 字符串拼接超过两次，要改用 string.format (5)

原因：字符串在 lua 中是不可变的，每次修改返回的都是重新分配的，拼接越多，分配内存的次数越多

```
local function print_msg(a, b)
    -- local s = "error: " .. a .. " " .. b  -- 不良风格：分配 3 次内存，中间的两次是无效的
    local s = string.format("error: %s %s", a, b)    -- 良好风格：分配 1 次内存
```

```
    print(s)
end
```

2.15 * 每个函数不超过 100 行 (10)

2.16 * 判断假使用 not (5)

```
local a, b = nil, false
if not a then     -- 不要写成 if (not a) then
    print("a is nil")
end

if not b then     -- 不要写成 if (not b) then
    print("b is false")
end
```

2.17 ** 多用短路赋值 (5)

```
-- 良好风格，简洁
local function set_default(cur, default)
    return cur and cur or default
end

-- 不良风格，太罗嗦
local function set_default(cur, default)
    if not cur then
        cur = default
    end
    return cur
end
```

2.18 * lua 文件函数定义的方法 (5)

2.18.1 * 前面定义局部函数，最后返回临时表(推荐)

```
--- url.lua
local function isurl()
```

```
    return s:find("^http://")
end

local function new()
    local obj = {map = {}}
    setmetatable(obj, mt)
    return obj
end

return {isurl = isurl, new = new}
```

2.18.2 ** 前面定义表，再定义表内容为函数

```
--- url.lua func_map 重复太多，不推荐
local func_map = {}
function func_map.isurl()
    return s:find("^http://")
end

function func_map.new()
    local obj = {map = {}}
    setmetatable(obj, mt)
    return obj
end

return func_map
```

2.19 * 函数内定义函数 (5)

```
local function validate(username, password)
    -- 使用定义变量的方式
    local reply = function(r, e)
        return js.encode({r = r, e = e})
    end

    if not (username == "hello" and password == "world") then
        return reply(1, "invalid param")
    end

    return reply(0, "ok")
```

```
end
```

2.20 *** 无效代码要删除，不能以注释方式保留 (5)

无效的注释也要删除

2.21 * 使用 t.xx 而不是 t["xx"]引用 table 的字段 (5)

如 t.account, t.account_id, **rows[1].account_id**
必须使用 t["xx"]的情况：
数字下标、数字开始、中间有[a-zA-Z0-9_]之外的字符，关键字，变量
如 t[1], t["123_444"], t["a-d"],t["end"], t[var]

```
--- 良好风格
local function get_account (db)
    local rs, e = db:select ("select * from tb_account")
    if not rs then
        return nil, e
    end

    local result = {}
    for _, r in ipairs (rs) do              -- 不需要 index 时，使用 ipairs/pairs
        table.insert (result, {             -- 可以使用.，就不要使用[]; 表初始化时，=对齐
            role         = r.role,
            account      = r.account,
            account_id   = r.account_id,
        })
    end

    return result
end

--- 不良风格
local function get_account (db)
    local rs, e = db:select ("select * from tb_account")
    if not rs then
        return nil, e
    end
```

```
    local result = {}
    for i = 1, #rs do
        result[i] = {
            ["account_id"] = rs[i]["account_id"],
            ["account"] = rs[i]["account"],
            ["role"] = rs[i]["role"],
        }
    end

    return result
end
```

table 赋值的风格必须统一，如果 table.xxx 不能满足所有的字段，改用 table["xxx"]

```
local cmd_map = {}
cmd_map.dbsync_ipgroup     = function() end
cmd_map.dbsync_timegroup   = function() end
cmd_map["/kv_get"]         = function() end
```

应当改为:

```
local cmd_map = {}
cmd_map["dbsync_ipgroup"]      = function() end
cmd_map["dbsync_timegroup"]    = function() end
cmd_map["/kv_get"]             = function() end
```

2.22 * table 初始化对齐 (5)

定义时，要么写成一行，要么每行一个：

```
-- 良好风格（每行一个）
local account_info = {
    role         = r.role,
    account      = r.account,
    account_id   = r.account_id,
}

-- 不良风格
local account_info = {
    account_id = r.account_id,
    role = r.role,
    account = r.account,
```

```
}

-- 良好风格（一行）
local account_info = {role = r.role, account = r.account, account_id = r.account_id}

-- 不良风格
local account_info = {role = r.role, account = r.account,
    account_id = r.account_id}
```

2.23 * 赋值对齐 (5)

```
-- 分块=对齐
local fp     = require("fp")
local ski    = require("ski")
local log    = require("log")
local nos    = require("luanos")
local js     = require("cjson.safe")
local lib    = require("authlib")
local batch  = require("batch")
local md5    = require("md5")
local rpccli = require("rpccli")
local simplesql   = require("simplesql")
local authlib     = require("authlib")
local cache       = require("cache")

local sumhexa     = md5.sumhexa
local set_module  = cache.set_module
local set_status, set_online   = nos.user_set_status, authlib.set_online
local keepalive, insert_online = authlib.keepalive, authlib.insert_online

-- 从短到长排列，中间用空格
local fp = require("fp")
local md5 = require("md5")
local ski = require("ski")
local log = require("log")
local nos = require("luanos")
local lib = require("authlib")
local batch = require("batch")
local cache = require("cache")
local js = require("cjson.safe")
```

```lua
local rpccli = require("rpccli")
local authlib = require("authlib")
local simplesql = require("simplesql")

local sumhexa = md5.sumhexa
local set_module = cache.set_module
local set_status, set_online = nos.user_set_status, authlib.set_online
local keepalive, insert_online = authlib.keepalive, authlib.insert_online

-- 不良风格
local fp = require("fp")
local ski = require("ski")
local log = require("log")
local nos = require("luanos")
local js = require("cjson.safe")
local lib = require("authlib")
local batch = require("batch")
local md5 = require("md5")
local rpccli = require("rpccli")
local simplesql = require("simplesql")
local authlib = require("authlib")
local cache  = require("cache")
```

3. lua 面向对象

3.1 *** lua 中只用到类的封装 (5)

在 C++中，类的特性是 封装、继承和多态，但是在 lua 中，我们只用到封装。lua 是动态语言，写得越复杂，运行越慢。不要使用继承和多态。

3.2 * 禁止在类定义的外部直接使用类内部的属性。(20)

```lua
--- 良好风格
----- cache.lua
local method = {}
local metatable = {__index = method}
function method:set(k, v)
    self.map[k] = v
```

```lua
end

function method:get(k)
    return self.map[k]
end

function method:foreach(cb)
    for k, v in pairs(self.map) do
        cb(k, v)
    end
end

local function new()
    local obj = {map = {}}
    setmetatable(obj, metatable)
    return obj
end

return {new = new}

--- main.lua
local cache = require("cache")
local c = cache.new()
c:set("k1", 1)
c:foreach(function(k, v)
    print(k, v)
end)

--- 不良风格，破坏了类的封装性
----- cache.lua
local method = {}
local metatable = {__index = method}
function method:set(k, v)
    self.map[k] = v
end

function method:get(k)
    return self.map[k]
end

local function new()
```

```
    local obj = {map = {}}
    setmetatable(obj, metatable)
    return obj
end

return {new = new}

--- main.lua
local cache = require("cache")
local c = cache.new()
c:set("k1", 1)
for k, v in pairs(c.map) do    -- c.map 使用了类内部的成员，破坏了封装
    print(k, v)
end
```

3.3 ** 类中的成员变量在 new 中先声明，并赋予初始值 (10)

不允许不声明直接使用。原因：禁止在运行过程中新增类成员变量，难以维护。

```
-- 不良风格
local cache = require("cache")
local c = cache.new()
c:set("k1", 1)
c.count = 1  -- 新增了类成员 count
c:set("k2", 2)
c.count = c.count + 1
```

3.4 *** 禁止滥用面向对象 (10)

```
-- 良好风格
---- url.lua
local method = {}
local mt = {__index = method}

local function isurl(s)
    return s:find("^http://")
end

function method:tostring()
    return js.encode(self.map)
end
```

```lua
function method:insert(url)
    if not isurl(url) then
        return
    end

    local map = self.map
    local count = map[url]
    if not count then
        map[url] = 1
        return
    end

    map[url] = count + 1
end

local function new()
    local obj = {map = {}}
    setmetatable(obj, mt)
    return obj
end

return {new = new, isurl = isurl}

--- main.lua
local url = require("url")
local ins = url.new()
local s = "http://www.haidu.com"
local _ = url.isurl(s) and ins:insert(s)
print(ins:tostring())

-- 不良风格
---- url.lua
local url = {}
-- 等价于 function.new(table_url), table_url 在这里等价于 self
-- 没有使用到类成员，滥用
function url:new()
    local obj = {}

    -- 这行的意思就是 obj 的元表是 url
    setmetatable(obj, {__index = self})
```

```
    return obj
end

-- 没有使用到类成员，滥用
function url:isurl(s)
    return s:find("^http://")
end

return {new = new, isurl = isurl}

--- main.lua
local url = require("url")
local ins = url:new() -- 等价于 url.new(url)
local r = ins:isurl("http://www.haidu.com")
```

4. 通用编码规范

所有语言通用

4.1 * 外部参数必须检查合法性(20)

外部参数包括：文件、外部网络，从这些地方获取的内容不能保证格式正确，或者没有被篡改，必须在使用前检查不否合法。
内部参数包括：进程内部，本机不同进程之间的数据传输，动态库调用等。这类参数使用时也要检查，但要求没那么严格。

4.2 * 必须检查函数返回值(20)

函数执行可能会出错，必须检查返回值，再决定后面的流程。
函数设计时，涉及到系统调用，一般都有返回值，并且统一风格。如果要简化设计，不返回错误，如直接退出，必须经过讨论并得到允许。