===================== ==================

===================== lua tip ==================

===================== ==================

Test.lua

function Global()

print("----- Global -----")

end

全局函数

调用： require "Test"

Global()

\*另外只要在前面require之后，后面则可直接调用 无需再require一次

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test.lua

module("Test",package.seeall)

TestTable = { number = 10}

function ModuleFunc()

print("----- ModuleFunc ------")

end

function TestTable:ModuleTest()

print(self.number)

end

module模块 此处 ModuleFunc 函数则被丢进 Test 模块当中

调用： require "Test"

Test.ModuleFunc()或者Test:ModuleFunc()

\*两者区别于self

Test.TestTable.ModuleTest()或者Test.TestTable:ModuleTest()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

有如下Manager.cs脚本 需进行tolua操作生成相对应的wrap文件

namespace LuaFramwork

{

public class Manager

{

Manager inst = new Manager();

public void TestFunc()

{

Debug.Log("------- ComeOn -------")

}

}

}

然后则可调用：ma = LuaFramwork.Manager

ma.TestFunc()

或者：ma = LuaFramwork.Manager.inst

ma:TestFunc()

\*\* 相当于实例的话用 ：调用

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

有如下两个脚本

TestA.lua

module("Common",package.seeall)

function F1()

end

function F2()

end

TestB.lua

module("Common",package.seeall)

function F3()

end

function F4()

end

其中定义Common模板，类似于扩展，所以require其中任意一个

则可进行调用： Common.F1() ...

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

for循环 两种

1.数值for循环

for var=a,b,c do

print("-------- For循环 -------")

end

从 a 到 b 的过程,首先看 c 的正负

如果 c 为正,则当 a <= b 时,执行循环体, a 递增 c 的值

如果 c 为负,则当 a >= b 时,执行循环体, a 递减 c 的值

当 c 未指定时，默认为 1

如： for i=1,10 do for i=10,1,-1 do

print(i) print(i)

end end

输出结果为：1,2,3,4,5,6,7,8,9,10 10,9,8,7,6,5,4,3,2,1

for循环的三个表达式 在循环开始前将会一次性求值 之后则不会再进行求值

function f(x)

print("我只执行一次!")

return x\*2

end

for i=1,f(5) do

print(i)

end

输出结果为：我只执行一次 1,2,3,4,5,6,7,8,9,10 \*\* 函数f(x)只执行一次,其结果将被用在循环中执行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.泛型for循环

i 为数组索引值 v 是对应索引的数组元素值 a 为表

for i,v in ipairs(a)

do print(v)

end

for k,v in pairs(a)

do print(v)

end

注意两者的区别：ipairs 不能返回 nil 只能返回 0 如果遇到 nil 则会退出 也就是说他只能遍历到表中出现的第一个不是整数的key

pairs 可以遍历表中所有的key 而且还能返回 nil

eg:

ipairs pairs

local TestTable = { [1] = "TAT",num = 100 } TAT TAT 100

local TestTable = { [2] = "TAT"} 不输出任何值 TAT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

string 模块相关操作

1. string.byte 把字符串中第 i 个字符转为 ASCII 编码

默认是输出第一个字符

string.byte("abc") -----> 输出a的ASCII 97

string.byte("abc",2) -----> 输出b的ASCII 98

2. string.char 把一个 ASCII 编码转换为对应的字符

string.char(97) -----> 输出为 a

string.char(99,100,101) ---------> 输出为 cde

3. string.dump 把一个函数序列化 必须为lua函数 可以用 load 函数反序列回来直接调用

function test()

print("----- test -----")

end

local SD = string.dump(test)

local LS = load(SD)

LS()

4. string.find 查找字符串中第一个符合条件的位置 返回其开始和结束的位置 如果没有找到则返回 nil

string.find("It is a test","st") ------> 11,12 \*lua中索引从 1 开始 同时要注意空格的存在

string.find("It is a test","15") ------> nil

5. string.match 查找字符串中第一个符合匹配的字符串 若没找到 则返回 nil

string.match(str, pattern, init) 参数init可选 指定搜寻的起点 默认为 1

string.match("Hello World !","Hello") ----> "Hello"

6. string.sub 从字符串中截取一个从第 i 个字符到第 j 个字符的字符串

string.sub("ABCDEF",2,5) -------> BCDE

lua中索引可为负数 当为负数时便从字符串后面向前计数 所以

string.sub("ABCDEF",2,-2) --------> BCDE

string.sub("ABCDEF",-4,-2) --------> CDE

string.sub("ABCDEF",-2,-4) --------> 返回空的字符串 因为从 -2 ~ -4的字符串是空

也就是说 i 不能比 j 大

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table = {}

#Table 返回的是一个表的长度 但只对于其 key 为连续的数值时才有意义

也就是说当 key 为不连续的数值 或 存在 nil #的使用就无意义

#"Table" 也可返回一个字符串的长度

and or 借用 与 或 逻辑记忆

A and B 看 A 是否为 假 若为假 则返回 A 若为真 则返回 B

A & B 只有两个都为真 则为真 即 A 为 0 则不管 B 为何情况都为 0 所以直接 返回 A

若 A 为 1 还要看 B 但不管 B 是什么 都是返回 B 的结果 所以直接返回 B

A or B 看 A 是否为真 若为真 则返回 A 若为假 则返回 B

A | B 只有两个都为假 则为假 即 A 为 1 则不管 B 为何情况都为 1 所以直接返回 A

若 A 为 0 还要看 B 但不管 B 是什么 都是返回 B 的结果 所以直接返回 B

所以当存在多个 and 或 or 的表达式时 如 A and B and C ...

and 就返回第一个为假的表达式

or 就返回第一个为真的表达式

% 取模操作符 lua 中对该操作的定义为 a % b = a - floor(a/b)\*b

所以可以知道 x % 1 结果是 x 的小数部分 x - x % 1 结果为 x 的整数部分 x - x%0.01 精确到小数点后两位

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

table 函数库

1. table.concat(table, sep, start, end)

用于列出指定table的数组部分从start 到 end 所有元素 元素间以指定的分隔符 sep 分开

sep 默认为 空字符 start 默认为 1 end 默认为数组部分的总长

\*\* sep start end 三个参数是顺序读入的 所以如果要指定后面的参数 则必须同时指定前面的参数

eg：

> tbl = {"alpha", "beta", "gamma"}

> print(table.concat(tbl, ":")) ---- alpha:beta:gamma

> print(table.concat(tbl, nil, 1, 2)) ---- alphabeta

> print(table.concat(tbl, "\n", 2, 3)) ---- beta (换行)

gamma

2. table.insert(table, pos, value)

用于在指定table的数组部分的指定位置 pos 插入值 value pos默认为数组部分末尾

eg:

> tbl = {"alpha", "beta", "gamma"}

> table.insert(tbl, "delta")

> table.insert(tbl, "epsilon")

> print(table.concat(tbl, ", ") ---- alpha, beta, gamma, delta, epsilon

> table.insert(tbl, 3, "zeta")

> print(table.concat(tbl, ", ") ----alpha, beta, zeta, gamma, delta, epsilon

3. table.maxn(table)

用于返回指定table中 所有正数key值中的最大key值 如果不存在key值为正数 则返回 0

\*\*此函数不限于table的数组部分

eg：

> tbl = {[1] = "a", [2] = "b", [3] = "c", [26] = "z"}

> print(#tbl)

3 -- 因为26和之前的数字不连续, 所以不算在数组部分内

> print(table.maxn(tbl))

26

> tbl[91.32] = true

> print(table.maxn(tbl))

91.32

4. table.remove(table, pos)

用于删除指定table数组部分中位于pos位置的元素 删除后该位置以后的元素则会前移 默认为table的长度 即从最后一个元素删起

5. table.sort(table, comp)

用于对指定的table进行 升序排序 comp可省略

\*\* comp 这个参数 必须是一个 函数 并且可以接收表的两个元素

默认规则是 第一个元素 小于 第二个元素 返回 true 其他情况返回 false

并且 table 的元素不能为 nil

也就是说 当我们反着写 sortFunc = function(a, b) return b < a end 可以达到降序的效果

tbl = {"alpha", "beta", "gamma", "delta"}

table.sort(tbl, sortFunc)

print(table.concat(tbl, ", ")) ----- gamma, delta, beta, alpha

可以这么理解： 默认是按照 a < b 的顺序 依次排列 加了 sortFunc 函数 则就是按照 b < a 的顺序依次排列

所以可以有更复杂的操作：

guild = {}

table.insert(guild, {

　name = "Cladhaire",

　class = "Rogue",

　level = 70,

})

table.insert(guild, {

　name = "Sagart",

　class = "Priest",

　level = 70,

})

table.insert(guild, {

　name = "Mallaithe",

　class = "Warlock",

　level = 40,

})

对这个table进行排序时, 应用以下的规则: 按等级升序排序, 在等级相同时, 按姓名升序排序.

可以写出这样的排序函数:

function sortLevelNameAsc(a, b)

　if a.level == b.level then

　　return a.name < b.name

　else

　　return a.level < b.level

　end

end

测试功能如下:

> table.sort(guild, sortLevelNameAsc)

> for idx, value in ipairs(guild) do print(idx, value.name) end

1, Mallaithe

2, Cladhaire

3, Sagart

6. table.foreachi(table, function(i,v))

用于对指定table 从数字 1 开始的连续整数 范围内 遍历table的 key值 和 value值 进行function(i,v)的操作

t1 = {2, 4, 6, language="Lua", version="5", 8, 10, 12, web="hello lua"};

table.foreachi(t1, function(i, v) print (i, v) end) ; --等价于foreachi(t1, print)

输出结果：

1 2

2 4

3 6

4 8

5 10

6 12

7. table.foreach(table, function(i,v))

与 foreachi 不同 , 此函数会对整个表进行迭代

t1 = {2, 4, 6, language="Lua", version="5", 8, 10, 12, web="hello lua"};

table.foreach(t1, function(i, v) print (i, v) end) ;

输出结果：

1 2

2 4

3 6

4 8

5 10

6 12

web hello lua

language Lua

version 5

8. table.getn(table)

返回指定table中元素的个数 也要有序

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

lua 中的 魔法字符

魔法字符包括 . % + \* - ? [] ^ () $

模式符号 ：

. 所有字符

%a 字母

%c 控制字符

%d 数字

%l 小写字母

%p 标点符号

%s 空白字符

%u 大写字母

%w 字母或数字

%x 十六进制字符

%z 内部表示为 0 的字母

若是大写形式 则表示其自身的补集 例如 %A 表示非字母

local str = "Today is 13/10/2017"

print(string.match(str, "%d")) --- 1

print(string.match(str, "%D")) --- T

修饰符号 ：

+ 重复 1 次或者多次

\* 重复 0 次或者多次

- 重复 0 次或者多次

? 可选部分 (重复 0 次或 1 次)

其他符号 ：

[] 匹配范围的集合 ---- [0-9] 0~9的范围

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

lua 的 Math 库

1. pi 圆周率

math.pi --- 3.1415926535898

2. abs 取绝对值

math.abs(-2017) --- 2017

3. ceil 向上取整

math.ceil(9.1) --- 10

4. floor 向下取整

math.floor(9.9) --- 9

5. max 取参数最大值

math.max(2,4,6,8) --- 8

6. min 取参数最小值

math.min(2,4,6,8) --- 2

7. pow 计算次方

math.pow(10,5) --- 10^5

8. sqrt 开平方

math.sqrt(65536) --- 256

9. mod 取模

math.mod(65535,2) --- 1

\*\* \*\*

关于取模 和 取余 的区别

取余，遵循尽可能让商向0靠近的原则

取模，遵循尽可能让商向负无穷靠近的原则

符号相同时，两者不会冲突

例如 4/(-3) 约等于 -1.3

取余运算 则往 0 方向舍弃小数位 结果为 -1

取模运算 则往 负无穷 方向舍弃小数位 结果为 -2

10. modf 取整数和小数部分

math.modf(20.12) --- 20 0.12

11. random 取随机数

math.random(5,90) --- 范围5 ~ 90

12. randomseed 设随机数种子

math.randomseed(os.time()) --- os.time() 系统时间

13. 还有一些三角函数,log 等 不一一赘述...

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.proto 文件语法

定义一个消息类型 :

message SearchRequest

{

required string query = 1;

optional int32 page\_number = 2;

optional int32 result\_per\_page = 3;

}

该消息请求包含 一个查询字符串 结果所在页数 每页多少条查询结果

&& 在消息定义中 每个字段都有唯一的一个标识符 这些标识符是用来在消息的二进制格式中识别各个字段的

一旦开始使用就不能 改变 ！

[1,15]之内的标识符 在编码时 占用一个字节 [16,2047]之内的标识符占用 两个字节

所以 频繁出现的消息元素 应保留[1,15] 同时也应该考虑到将来会添加的 预留一些标识符

&& 指定字段规则

所指定的消息字段修饰符 必须是如下之一 :

@ required 表示该值是必须要设置的 一个格式良好的消息 一定要含有 1 个这种字段

@ optional 消息格式中该字段可以有 0 或 1 个值 不超过 1 个

@ repeated 表示该值可以重复 在一个格式良好的消息中 这种字段可以重复任意多次 包括 0 次

重复的值的顺序会被保留

protobuf 数据类型 描述 打包 C++语言映射

bool 布尔类型 1字节 bool

double 64位浮点数 N double

float 32为浮点数 N float

int32 32位整数 N int

uint32 无符号32位整数 N unsigned int

int64 64位整数 N \_\_int64

uint64 64为无符号整 N unsigned \_\_int64

sint32 32位整数 N int32

处理负数效率更高

sing64 64位整数 N \_\_int64

处理负数效率更高

fixed32 32位无符号整数 4 unsigned int32

fixed64 64位无符号整数 8 unsigned \_\_int64

sfixed32 32位整数 4 unsigned int32

能以更高的效率处理负数

sfixed64 64位整数 8 unsigned \_\_int64

string 只能处理 ASCII字符 N std::string

bytes 用于处理多字节的语言字符,如中文 N std::string

enum 一个用户自定义的枚举类型 uint32N(uint32) enum

message 一个用户自定义的消息类型 N object of class

**元表**

顾名思义 本身是一个table 存放 元方法 的table

元表的作用是修改一个值的行为

Lua中每一个值都有一个元表 只是 table 和 userdata 可以有各自独立的元表

而其他类型的值则是共享其类型所属的单一元表

Lua代码中只能设置 table的元表 其他类型的则只能通过 C 代码设置

多个table可以共享一个通用的元表 但是每个table只能有一个元表

\_index 元方法 用于改变 访问 table中不存在的key时的行为

当访问一个table中不存在的key值，如果该table中定义了\_index 字段 那么lua就会调用该元方法 规则如下：

当 \_index元方法 是一个table时，lua就会用不存在的key值去访问这个table

当 \_index元方法 是一个函数时，lua就会用table 和 不存在的key值作为参数 来调用该函数

Eg. mytable = setmetatable({key1 = "value1"}, {

\_\_index = function(mytable, key)

if key == "key2" then

return "metatablevalue"

else

return mytable[key]

end

end

})

print(mytable.key1,mytable.key2)

输出为：value1 metatablevalue

\_newindex 元方法 用于改变 设置 table中不存在key值时的行为

当对一个table中不存在的key赋值时，如果该table中定义了\_newindex字段 那么lua 就会调用该元方法 规则基本类似于 \_index