Lua是一种轻量小巧的脚本语言 用标准C语言编写并以源代码的形式开发

设计目的：嵌入应用程序中 从而为应用程序提供灵活的扩展和定制功能

Lua语言的特性：

1 轻量级 用C语言编写 编译后仅仅100多k 很方便嵌入到应用程序中

2 可扩展 扩展接口的机制比较简单方便 由宿主语言(C C++ C#)提供功能 Lua使用它们如同内置功能一样

3 支持面向过程编程和函数式编程

4 自动的内存管理

5 通用类型表(table) 用它可以实现数组 哈希表 集合 对象

6 函数可以当作是一个值来使用

7 通过闭包和table可以实现面向对象编程的关键机制（抽象 虚函数 重载）

8 提供多线程（协同程序 不是操作系统的线程）

Lua应用场景：游戏开发 独立应用开发 Web应用脚本 数据库扩展和插件 安全系统（入侵检测系统）

输出语句

```lua
print("HelloWorld") --HelloWorld
print('HelloWorld') --HelloWorld
print("你好".."123") --你好123
print("23"+"123")   --146
print("2"+6)        --8

print(type("HelloWorld")) --string
print(type('H'))            --string
print(type(100))          --number
print(type(10.4*3))       --number
print(type(print))        --function
print(type(true))         --boolean
print(type(nil))          --nil
```

通过输出语句可以看出

1 ..是字符串的连接符号 +永远是计算和

2 不管是整数还是小数 都被看做是number类型

3 print是一个方法 在Lua中 方法也是一种数据类型 function

4 nil是空 和null一样

5 ""和''都表示string lua中没有char

数据类型

Lua是动态类型语言 变量不需要类型定义 只需要为变量赋值

值可以存储在变量中 作为参数传递或者结果返回

Lua中有8个基本类型 分别是:

nil boolean number string function thread table和userdata

nil 空 相当于null

nil类型表示没有一种有效值 只能为nil

例如打印一个没有赋值的变量 便会得到一个nil

```
print(a) --nil
```

boolean 布尔

lua把false和nil看作假 其他都为真

```
print(type(true)) --boolean
print(type(false)) --boolean
print(type(nil)) --nil

if false or nil then
```

```
    print("有一个为真")
else
    print("两个都为假") --打印
end

```

number 数字

Lua中没有双精度或单精度浮点数 所有的数值都被当做是number类型

```
print(type(2)) --number
print(type(2.2)) --number
print(type(0.22222)) --number
```

string 字符串

字符串由一对双引号或者一对单引号来表示

```
string1="this is string1"
string2='this is string2'
```

用2个中括号表示一段字符串

```
html=[[
<html>
    <head>dasfa</head>
    <body>
        <a href="http://baidu.com/">百度首页</a>
    </body>
</html>
]]

```

Lua变量

变量在使用前 必须在代码中进行声明 即创建该变量

Lua变量有三种类型：局部变量 全局变量 表中的域

Lua中的变量默认都是全局变量 即使是在语句块或者函数中

表示局部变量就必须在变量前加上local关键字

局部变量的作用域从声明位置开始到所在语句块结束

变量的默认值均为nil

```
function main()
    c=5         --全局变量
    local d=6   --局部变量
end

main()          --方法调用
print(c,d)      --5 nil

a=5           --全局变量
local b=5     --局部变量

do
    local a=6 --局部变量
    b=6       --全局变量
    print(a,b)--6 6
end

print(a,b) --5 6
```

赋值语句

Lua可以对多个变量同时赋值

变量列表和值列表各元素之间逗号分开

赋值语句右边的值依次赋值给左边变量

当变量个数和值个数不同 Lua一直变量个数为基础采取策略

1 变量个数>值的个数 按变量个数补足nil

2 变量个数<值的个数 多余的值会被忽略

```
a,b=10,2 --a=10  b=2

a,b,c=0,1
a,b=a+1,b+1,b+2
```

交换变量的值

```
a=10
b=20
a,b=b,a
print(a,b)
```

Lua循环

while死循环

```
while(true)
do
    print("死循环")
end
```

函数

```
--比较两个数的大小
function fun(a,b)
    if a>b then
```

```
        result=a;
    else
        result=b
    end
    return result
end

print("两个数的最大值是",fun(4,5))
print("两个数的最大值是",fun(10,5))
```

Lua中 将函数作为参数传递给另一个函数

```

myPrint=function(param)
    print("这是一个打印的函数",param)
end


myPrint(100)


function add(num1,num2,functionPrint)
    result=num1+num2
    --调用传递的函数参数
    functionPrint(result)
end

--myPrint 函数作为参数进行传递
add(2,5,myPrint)

```

Lambda表达式 匿名函数

```
function FF(tab,fun) --tab是表  fun是个函数
    for k,v in pairs(tab) do
```

```
        print(fun(k,v))
    end
end

tab={k1="val1",k2="val2",k3="val3"} --定义1个table

FF(tab,function(k,v)--直接写出fun方法体实现Lambda
            return k.."$$$"..v
       end)
```

多返回值

Lua函数可以返回多个结果 比如 string.find 其返回汽配开始到结束的下标

```
s,e=string.find("www.baidu.com","baidu")
print(s,e) --5,9
```

Lua函数中 在return后返回的值可以通过一个逗号隔开的方式返回多个

例:

获取一个数组的最大值和最大值索引

```
function max(a)
    local mi=1 --最大值索引
    local m=a[mi] --最大值

    for i,v in ipairs(a) do
        if v>m then
            mi=i
            m=v
        end
    end
    return m,mi
end

print(max({100,277777,1,3,1,3123}))
```

可变参数列表

Lua函数可以接受和支持可变数目的参数

函数的参数放在一个叫做arg的表中 ...表示参数个数可变 #arg 表示传入参数的个数

计算平均数

```lua
function average(...)
    result=0
    local arg={...}

    for i,v in ipairs(arg) do
        result=result+v
    end
    print("一共传入了"..#arg.."个参数")
    return result/#arg
end

print("平均数是",average(10,5,3,4,5,6))
```

把函数当做变量赋值

因为函数本身被看作是一种数据类型 所以可以使用函数给另一个函数进行赋值 实际上地址的赋值

```lua
function fn(n)
    if n==0 then
        return 1
    else
        return 100
    end
end

print(fn(5))
fn2=fn
print(fn2(0))
```

运算符

算数运算符

设 A=10 B=20

+ A+B 30

- A-B -10

* A*B 200

/ B/A 2

% B%A 0

^ 乘幂 A^2 100

- 负号-A -10

关系运算符

== 等于

~= 不等于

> 大于

< 小于

>=大于等于

<=小于等于

逻辑运算符

and 逻辑与

or 逻辑或

not 逻辑

单目运算符 一元运算符

#hello 结果 5

数组

Lua数组可以是一维数组 也可以是多维数组

数组的大小是不固定的

一维数组

逻辑结构的线性表 一维数组使用for循环遍历元素

```
array={"Lua","C#"}
for i=0,2 do
    print(array[i]) -- nil lua c#
end
```

索引不到值时 返回nil

所以在Lua中索引值默认是从1开始 但是我们可以指定从0开始

也可以指定从负数开始

```
array={} --定义空表

for i=-2,2 do
    array[i]=i*2
end
for i=-2,2 do
    print(array[i])
end
```

多维数组

```
array={} --定义空表
for i=1,3 do
    array[i]={} --每一个元素定义一个空表
    for j=1,3 do
        array[i][j]=i*j
    end
end

```

```lua
--访问多维数组
for i=1,3 do
    for j=i,3 do
        print(array[i][j])
    end
end
```

Lua Table表

table是Lua中的一种数据结构 用来帮助我们创建不同的数据类型

Lua也是通过table来解决 模块(module) 包(package)和对象(object)

table的构造

构造器是创建和初始化表的表达式

最简单的构造函数{} 用来创建空表

直接初始化数组

```lua
myTable={}      --初始化表
mytable[1]="Lua" --指定值使用表
myTable=nil     --析构 Lua垃圾回收会自动释放内存
```

当table a设置元素 将a赋值给b 则 a和b都指向同一个内存区域

如果a设置成了nil b能访问表中的元素

如果没有指定的变量指向a lua的垃圾回收机制会自动清理内存

```lua
Mytable={}

print(type(Mytable)) --table

Mytable[1]="Lua"
Mytable["abc"]="修改前"
```

```lua
print(Mytable[1]) --lua
print(Mytable["abc"]) --修改前

--Mytable2和Mytable都指向同一个表的内存
Mytable2=Mytable
print(Mytable2[1]) --lua
print(Mytable2["abc"]) --修改前

Mytable2["abc"]="修改后"
print(Mytable["abc"]) --修改后

Mytable2=nil --释放
print(Mytable2) --nil

--当Mytable2移除引用后 Mytable依然可以访问
print(Mytable["abc"]) --修改后

Mytable=nil
print(Mytable) --nil

--Lua GC自动清理内存
```

Lua标准库 Table

1 table.concat 连接

2 table.insert 插入

3 table.maxn 所有整数key值中最大的key

4 table.remove 移除 默认从最后位置删除一个元素

5 table.sort 升序排序

```lua

fruits={"banana","orange","apple"}

print("连接",table.concat(fruits)) --把所有元素合并到一个string

print("连接",table.concat(fruits,"&")) --指定连接字符把所有元素合并到一个string
```

```
6
7  print("连接",table.concat(fruits,"&",2,3)) --指定连接字符和索引把所有元素
   合并到一个string
```

```
1  fruits={"banana","orange","apple"}
2
3  table.insert(fruits,"mango") --默认在末尾插入数据
4  print(fruits[4])--mango
5
6  table.insert(fruits,2,"grapes") --指定索引值插入数据
7  print(fruits[2]) --grapes
8
9  print(fruits[5]) --mango
10
11 table.remove(fruits)
12 print(fruits[5]) --nil
```

```
1  fruits={"banana","orange","apple"}
2
3  table.sort(fruits) --升序排列
4
5  for k,v in ipairs(fruits) do
6      print(k,v)
7  end
8
9  tb1={[1]="a",[2]="b",[3]="c",[26]="z"}
10 print("tb1最大的key",table.maxn(tb1))
```

Lua模块和包

模块类似于一个封装库 从Lua5.1开始 加入了标准模块管理机制

把一些公用的代码放在一个文件中 以API接口的形式在其他地方调用

有利于代码的重用和解耦合

Lua的模块是由变量 函数等已知元素组成的table

因此创建一个模块很简单 就是创建一个table 把需要的常量 函数 放入其中 最后返回这个table

Module.lua

定义模块

```lua

--定义一个名为module的模块
module={}

--定义一个变量
module.x="这是个变量"

--定义一个函数
function module.func1()
    print("这是一个公有函数")
end

--定义一个私有方法  不能加module.
local function func2()
    print("这是一个私有函数")
end

--定义一个公有方法 负责访问模块中私有成员
function module.func3()
    func2()
end

return module --返回模块
```

text.lua

接受模块 设定模块别名 访问模块

```lua

require("module") --加载模块
print(module.x)   --访问成员变量
```

```lua
module.func1()    --访问共有方法
module.func3()

--给加载的模块定义一个别名变量 方便调用
--module模块 可以设定一个别名叫做m

local b=require("module")
print(b.x)
```

模块的结构就是一个table

table中的私有变量和私有函数 外部不能访问

必须通过模块中其他的共有函数来调用

私有函数不能添加模块名.的标记

如果设定模块别名 需要在模块实现后返回模块