Lua

1 引言

- **特点：** 轻量级、可扩展

  Lua是用标准C语言编写并以源代码形式开放，编译后仅仅一百余K，可以很方便的嵌入别的程序里；Lua提供了非常易于使用的扩展接口和机制：由宿主语言(通常是C或C++)提供这些功能，Lua可以使用它们，就像是本来就内置的功能一样。

- **下载源码**

  http://www.lua.org/download.html 当前版本是**3.3.5**。

- **生成lua编译器和命令行终端**

  打开lua源码目录如图1所示，包括两个文件夹**doc**和**src**, 两个文件**Makefile**和**README**，图中其他文件夹是我为编译执行文件添加的，**bin**文件夹存放编译好的**lua.exe**和**luac.exe**可执行程序。如果是Linux系统，可以直接使用make工具生成可执行文件。

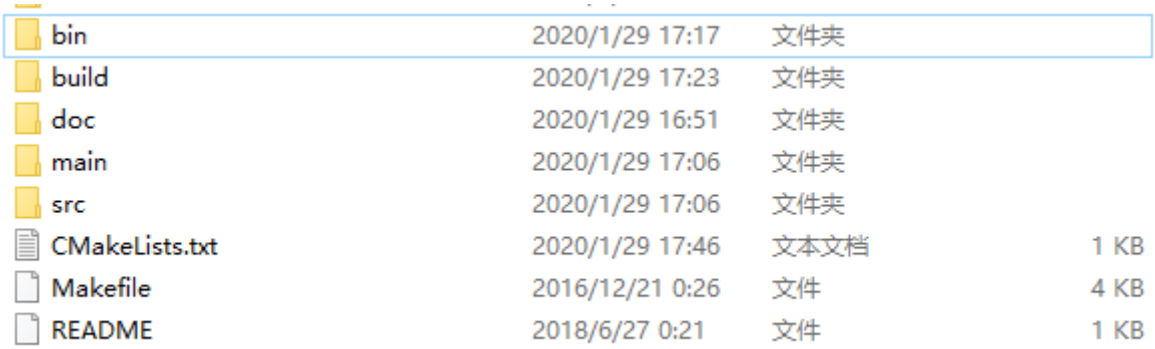

图1.1 lua源码的目录

为了生成可执行文件，需要先编写下**CMakeLists.txt**文件。在lua源码目录下新建txt文件，命名为**CMakeLists.txt**，打开该文件，写入如下内容：

```
cmake_minimum_required(VERSION 3.1.1)
project(lua)

include_directories("${PROJECT_SOURCE_DIR}/src")
file(GLOB lua_src "${PROJECT_SOURCE_DIR}/src/*.c"
                  "${PROJECT_SOURCE_DIR}/main/lua.c")
file(GLOB luac_src "${PROJECT_SOURCE_DIR}/src/*.c"
                  "${PROJECT_SOURCE_DIR}/main/luac.c")
```

```
  set(EXECUTABLE_OUTPUT_PATH "${PROJECT_SOURCE_DIR}/bin")  # 设置输出可执行文
件路径

  add_executable(lua ${lua_src})    # 生成lua命令行终端
  add_executable(luac ${luac_src})  # 生成lua编译器
```

同时，需要将**src**文件夹中的**lua.c**和**luac.c**文件移动到**main**文件夹下，然后打开cmd终端执行以下命令：

```
cd ./build
cmake ../     # 选择默认的编译器生成文件，  本人使用Visual Studio 2019
# 如果安装了mingw，可以指定gcc，即：cmake -G "MinGW Makefiles" ../
```

这样就生成了编译器项目了，如果编译器是Visual Studio的，进入**build**目录下，打开**lua.sh**项目，即可进行编译啦；如果是编译器是mingw (gcc)的，进入**build**目录执行make命令即可编译；编译完成后，在**bin**文件夹下会生成两个可执行文件，如下图所示：

图1.2 生成的可执行文件

注：如果没有安装cmake，cmake下载地址：https://cmake.org/files/

- **测试编译器和命令行终端**
  - **测试编译器**

    luac.exe的作用是将 **.lua**脚本变成二进制码，即完成加密。

    Step1: 新建一个名为**test.lua**的文件，写入如下的代码：

    ```
    print("Hello World!")
    print("This is a test!")
    ```

    Step2: 将**lua.exe**和**test.lua**放在同一个目录下，执行：

    ```
    .\luac -o test1.lua test.lua
    ```

    即可生成二进制文件**test1.lua**了，它可以直接被**lua.exe**执行，如：

```
PS E:\2019work\lua\lua-5.3.5\bin\Release> .\lua .\test1.lua
Hello World!
This is a test!
```

图1.3 luac生成的二进制文件被lua执行的结果

- **测试命令行终端**

  lua.exe是命令行终端，类似于cmd和shell，直接运行进行终端。

```
E:\2019work\lua\lua-5.3.5\bin\Release>lua.exe
Lua 5.3.5  Copyright (C) 1994-2018 Lua.org, PUC-Rio
>
>
> 1 + 1
2
>
```

图1.4 lua命令行终端

当然，也可以直接用终端运行 **.lua**脚本，如运行上面未加密的**test.lua**脚本：

```
E:\2019work\lua\lua-5.3.5\bin\Release>
E:\2019work\lua\lua-5.3.5\bin\Release>lua.exe test.lua
Hello World!
This is a test!

E:\2019work\lua\lua-5.3.5\bin\Release>
```

图1.5 lua命令行终端

2 基本语法

2.1 数据类型

Lua 是动态类型语言，变量不要类型定义，只需要为变量赋值。值可以存储在变量中，作为参数传递或结果返回。Lua 中有 8 个基本类型分别为：**nil**、**boolean**、**number**、**string**、**userdata**、**function**、**thread** 和 **table**。

数据类型	说明
nil	表示无效值，只有nil值，有些类似c语言中的NULL，但是它不是0
boolean	与c语言中的bool类型一样
number	表示双精度类型的实浮点数
string	字符串由一对双引号或单引号来表示
userdata	由 C 或 Lua 编写的函数
function	表示任意存储在变量中的C数据结构
thread	表示执行的独立线路，用于执行协同程序

数据类型	说明
table	Lua 中的表（table）其实是一个"关联数组"（associative arrays），数组的索引可以是数字、字符串或表类型。在 Lua 里，table的创建是通过"构造表达式"来完成，最简单构造表达式是{}，用来创建一个空表。

例：

```
print(type("Hello world"))      --> string
print(type(10.4*3))             --> number
print(type(print))              --> function
print(type(type))               --> function
print(type(true))               --> boolean
print(type(nil))                --> nil
print(type(type(X)))            --> string
```

执行结果：

```
E:\2019work\lua\lua-5.3.5\bin\Release>lua test.lua
string
number
function
function
boolean
nil
string

E:\2019work\lua\lua-5.3.5\bin\Release>
```

图2.1 执行结果

- **重点介绍下string、table和function类型**
  - **string(字符串)**

```
-- 字符串由单引号或双引号表示
string1 = "this is string1"
string2 = 'this is string2'

-- 跨行字符串用[[]]表示
sting3 = [[
  hello
  world
]]

-- 字符串与数字进行运算，会将字符串转换成数字，如果不能转换，则报错
print("2" + 6)  --> 8.0
print("2 + 6")  --> 2 + 6
print("-2e2" * "6")  --> -1200.0
print("error" + 1)  --> 报错

-- 使用..连接字符或者数字
print("a" .. 'b')  --> ab
```

```
print(157 .. 428)  --> 157428
print("11" .. 99)  --> 1199

-- 用#放在字符串前，表示计算字符串的长度
string4 = "hello"
print(#string4)  --> 5
print(#"world")  --> 5
```

- **table(表)**

```
-- table使用{}创建，创建一个空表
local tbl1 = {}  -- local 表示tbl1为局部变量，lua中默认变量都是全局变量，这一点与c语言完全不同，要注意

-- 直接初始化表
local tbl2 = {"apple", "pear", "orange", "grape"}

-- table中每个元素都默认一个key，例如上面"apple"的key=1, "pear"的key=2, 这与c++的map和python中的字典类似
local tbl = {"apple", "pear", "orange", "grape"}
for key, val in pairs(tbl) do
    print("Key", key)
end
--[[ 输出结果：
Key     1
Key     2
Key     3
Key     4
--]]

-- table不会固定长度大小，有新数据添加时table长度会自动增长，没初始的 table 都是nil
tbl3 = {}
for i = 1, 10 do
    tbl3[i] = i
end
tbl3["key"] = "val"
print(tbl3["key"])
print(tbl3["none"])
--[[ 输出结果：
val
nil
--]]

-- 对table进行遍历，输出顺序并不是赋值的顺序，而是按照key值的hash值排序后的顺序
tbl4 = {"Hello","World",a=1,b=2,z=3,x=10,y=20,"Good","Bye"}
for k,v in pairs(tbl4) do
    print(k.."  "..v)
end
--[[ 输出结果：
1  Hello
```

```
2  World
3  Good
4  Bye
x  10
y  20
z  3
b  2
a  1
--]]
```

- **function(函数)**

```
-- function的格式为: function 函数名(参数...)
function func_name(param)
    return param + 1
end

-- function可以通过匿名方式作为参数传递
-- func = function(param)
function testFun(tab,fun)
  for k ,v in pairs(tab) do
          print(fun(k,v))
  end
end
tab={key1="val1",key2="val2"};
testFun(tab,
function(key,val)--匿名函数
        return key.."="..val
end
)
--[[ 输出结果：
key1 = val1
key2 = val2
--]]
```

2.2 循环和条件语句

- **循环语句**

循环类型	说明	举例
while	与C语言相同，在条件为true时，执行循环，知道条件为false或者break跳出	
for		
repeat...until		
嵌套循环		

- **条件语句**
  未完待续