

版本号	描述	日期(更新日期)/更新人
1.0	Android Studio Lua技术	2019/07/26 刘志保
Github地址	https://github.com/MMLoveMeMM/AngryPandaLua	
	java工程参考上面工程根目录下LuaJava.zip工程	

前期阅读学习,Lua语言的基本语法等学习可以参考:

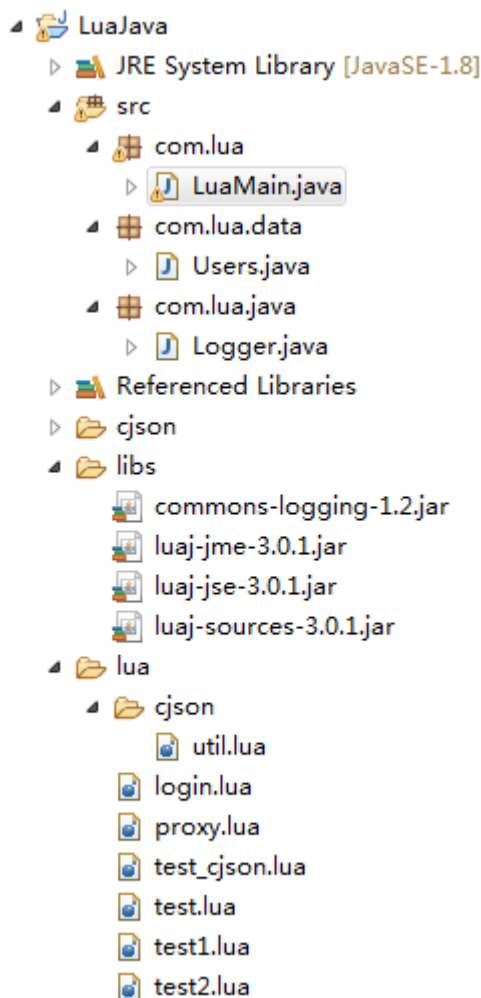
<https://www.runoob.com/lua/lua-tutorial.html>

<http://www.lua.org/manual/5.3/>

然后用上一章的文档中Lua开发环境运行测试即可.

如何在java中使用,纯Java工程和Android工程有点不一样,但是大部分都是差不多.

<1> : 先看看Java工程开发环境下,开始使用LuaJava如下:




















初始化Lua:

```
1 Globals globals = JsePlatform.standardGlobals();
```

一句话就搞定了.

通过Globals类来操作Lua和Java之间的交互:

- ▲  **Globals**
 - STDIN : InputStream
 - STDOUT : PrintStream
 - STDERR : PrintStream
 - finder : ResourceFinder
 - running : LuaThread
 - baselib : BaseLib
 - package_ : PackageLib
 - debuglib : DebugLib
 - loader : Loader
 - compiler : Compiler
 - undumper : Undumper
 -  Globals()
 -  checkglobals() : Globals
 - loadfile(String) : LuaValue
 - load(String, String) : LuaValue
 - load(String) : LuaValue
 - load(String, String, LuaTable) : Lua
 - load(Reader, String) : LuaValue
 - load(Reader, String, LuaTable) : Lu
 - load(InputStream, String, String, L
 - loadPrototype(InputStream, String
 - compilePrototype(Reader, String)
 - compilePrototype(InputStream, St
 - yield(Varargs) : Varargs
 - ▷   BufferedStream
 - ▷   UTF8Stream
 - ▷   AbstractBufferedStream
 - ▷   StrReader
 - ▷   Undumper
 - ▷   Compiler
 - ▷   Loader

Globals类中的方法.加载Lua脚本基本上使用load(...)或者loadfile(...)方法.

然后简单的加载Lua脚本语言程序运行:

```
1 String luaStr = "print 'hello,world!';  
2 LuaValue chunk = globals.load(luaStr);  
3 chunk.call();
```

运行就会打印:hello,world!

那么如何加载Lua程序文件运行呢?比如加载login.lua文件:

```
1 public static void loadLoginScript() {
2     String luaPath = "lua/login.lua"; //lua脚本文件所在路径
3     Globals globals = JsePlatform.standardGlobals();
4     //加载脚本文件login.lua, 并编译
5     globals.loadfile(luaPath).call();
6     //获取无参函数hello
7     LuaValue func = globals.get(LuaValue.valueOf("hello"));
8     //执行hello方法
9     func.call();
10    //获取带参函数test
11    LuaValue func1 = globals.get(LuaValue.valueOf("test"));
12    //执行test方法,传入String类型的参数参数
13    String data = func1.call(LuaValue.valueOf("I'am from
14    Java!")).toString();
15    //打印lua函数回传的数据
16    System.out.println("data return from lua is:"+data);
17 }
```

看了上面的,感觉整个Lua和Java交互的世界就用两个类完成了,Globals和LuaValue(Lua变量交互类)

```
1 --无参函数
2 function hello()
3     print 'hello'
4 end
5 --带参函数
6 function test(str)
7     print('data from java is: '..str)
8     return 'haha'
9 end
```

上面的程序是Java调用纯Lua的代码.

那如果Lua如何调用Java的类呢?

luajava 所提供了5种方法分明名

为: bindClass、newInstance、new、createProxy、loadLib

[使用方法可以参考:<https://blog.csdn.net/lgj123xj/article/details/81677036>]

bindClass	bindClass方法的作用是检索与className对应的Java类, 并返回一个可用于访问相应类中静态字段和静态方法的对象
newInstance	newInstance方法帮助开发者通过Java类的全名创建一个新的Java对象并返回一个Java对象的引用, 开发者可以在Lua脚本中以面向对象的方式操作该引用
new	new方法通过接受一个Java类实例化一个新的对象。他的工作原理与newInstance方法相近, 不同的一点是new方法所接受的第一个参数是Java类的Lua实例(通过

	bindClass方法获得的Java类),newInstance方法是通过包含完整类名的字符串查找到的对应Java类,而new则是从类实例中直接获取了对应Java类
createProxy	createProxy方法可以用来实现Java接口中定义的方法,createProxy方法接收两个参数,分别是接口的全称和一个包含于接口相同方法的Lua对象
loadLib	loadLib方法的用途类似于Lua中的loadLib函数。通过loadLib方法可以将开发者用Java编写的库加载到Lua中去

这个几个方法非常重要,可以非常使Lua很快的使用Java中的类和对象.

比如Java工程中新建一个test1.lua的文件,java程序中调用:

```
1 public static void luajavaCreateInstance1() {
2     Globals globals = JsePlatform.standardGlobals();
3     globals.loadfile("lua/test1.lua").call();
4 }
```

然后新建一个被Lua脚本使用的类:

```
1 package com.lua.java;
2
3 import org.apache.commons.logging.Log;
4 import org.apache.commons.logging.LogFactory;
5
6 public class Logger {
7     public static String TAG = "Logger";
8     private static Log logger = LogFactory.getLog(Logger.class);
9     public Logger(){
10         if(logger == null){
11             logger = LogFactory.getLog(Logger.class);
12         }
13     }
14
15     public void testLogger(String str) {
16         logger.info(str);
17     }
18
19     public static void info(String content){
20         logger.info(content);
21     }
22 }
```

然后test1.lua脚本:

```
1 --使用luajava绑定一个java类
2 local logger_method = luajava.bindClass("com.lua.java.Logger");
3 --调用类的静态方法/变量
4 logger_method:info("test call static java function in lua")
```

```
5 print(logger_method.TAG)
6 -- 使用绑定类创建类的实例（对象）
7 local logger_instance = luajava.new(logger_method)
8 -- 调用对象方法
9 logger_instance:testLogger("Test call java in lua1")
```

其他的参考github上的样例.