版本号	描述	日期(更新日期)/更新人
1.0	Android Studio Lua技术	2019/07/30 **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	

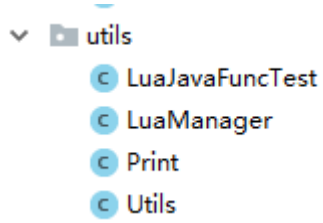
前面一节大致介绍了Lua和Java的基本交互,但是如何达到生成环境呢?

这个就要回顾前面一些LuaJava一些非常重要的方法:

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

让Lua直接使用Java的各种类,以及让Lua获取Java传递过来的复杂数据体才是真正重要的.

我们先在Java层写几个测试类:



里面的代码我就不贴了,github里面有.

首先初始化引用:

```
local bindClass = luajava.bindClass
local newClass = luajava.new
local newInstance = luajava.newInstance
```

然后开始使用:

```
local print = bindClass("pumpkin.org.angrypandalua.utils.Print")
```

```
print:debug()
print:show('yes this is OK')
--local log = print:show
-- log("hello world !")
local utils = bindClass("pumpkin.org.angrypandalua.utils.Utils")
utils:getVersion()
local utilsobj = newClass(utils)
utilsobj:getName()

 local util = newInstance("pumpkin.org.angrypandalua.utils.Utils")
 util:getName()

 local testStatic = bindClass("pumpkin.org.angrypandalua.utils.LuaJavaFunc
Test");
 -- new 通过class对象返回对应类的实例
 local testNew = newClass(testStatic);
 print:show(testNew:hello())
 -- print:show(testNew:testPrivate())
 print:show(testNew:testString("hello liuzhibao"))
```

整体看起来似乎很简单,不过操作过后的确非常的简单.

注意:

<1>:注意类中的静态方法,私有方法,公有方法;

<2>:注意普通类的调用,如果一个类引用其他类,其他类不需要绑定,这个可以减少程序很多,对于Lua使用Java类非常方便,比如Utils.java中使用了LuaManager.java这个单实例类,但是从上面程序来看,我们根本不需要在Lua程序里面去绑定LuaManager类,但是Java使用Lua程序就不行,必须所有的Lua文件都要事先加载,比如a.lua引用了b.lua的程序,那么Java层必须把a,b lua程序都需要加载进去.Lua的这种使用Java类操作,对于将Java业务逻辑搬到Lua来实现发挥了巨大的作用.

上面都是自定义的Java类,实际Java jdk中的以及Android中的类都可以被绑定.

比如下面我们从Java层传递Map和List数据给Lua层使用:

```
-- 获取List所有的参数
local list = bindClass("java.util.ArrayList");
local listObj = newClass(list)
listObj = testNew:testList()
print:show(listObj:get(0))
print:show(listObj:get(1))
--local len = listObj:size()
--print:show("list len : "+len)
```

```
9
10 -- 获取Map所有的参数
11 local map = bindClass("java.util.HashMap");
12 local mapObj = newClass(map)
13 mapObj = testNew:testMap()
14
15 if(mapObj:containsKey("A"))
16  then
17  print:show(mapObj:get("A"))
18 end
19 if(mapObj:containsKey("W"))
20 then
21  print:show(mapObj:get("W"))
22 else
23  print:show("can not find the value !")
24 end
```

自己慢慢的去体会,就会发现,基本上Java的类都可以被Lua直接拿来使用.达到举一反三的目的.这也就为什么很多Lua可以用来做界面,因为它可以拿到GroupView,View等类.

仔细体会前面几个所学的,试着做一些测试,会发现Lua还是非常好用的,灵活性很大.