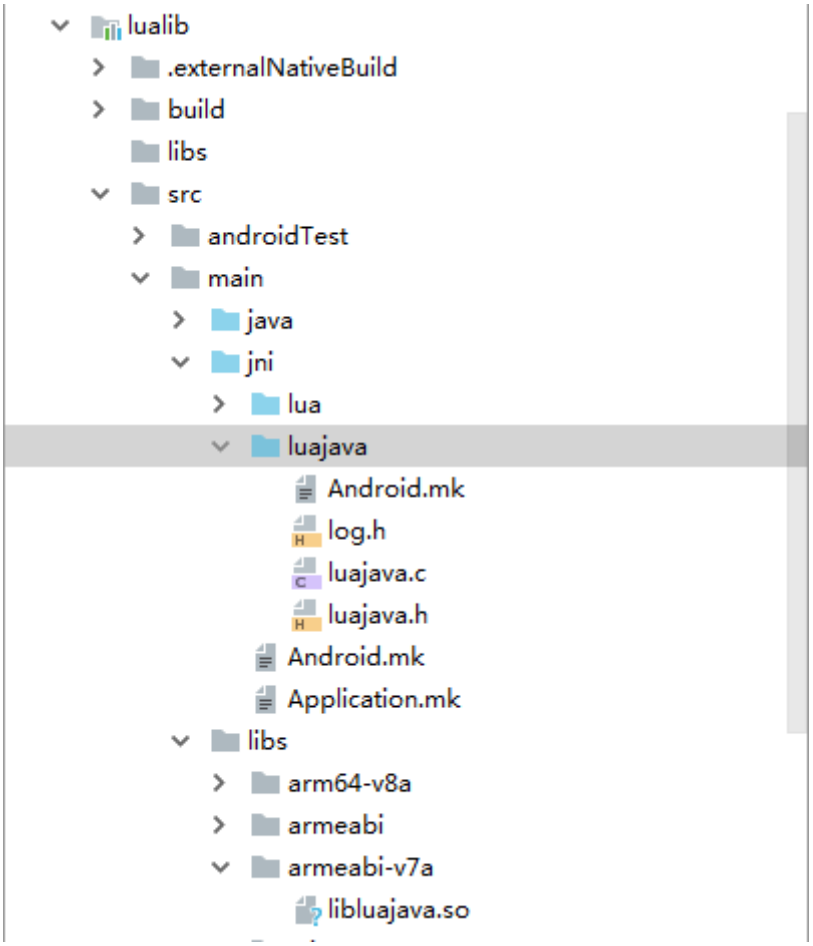
版本号	描述	日期(更新日期)/更新人
1.0	Android Studio Lua技术	2019/07/30 **刘志保**
Github地址	https://github.com/MMLoveMeMM/AngryPandaLua	
	java工程参考上面工程根目录下LuaJava.zip工程	

无论是Java工程中调用还是Android 中调用Lua,都是从LuaState开始.所以开发流程基本上都差不多.

Android 工程可以适当自己获取Lua源码库进行封装jni,通过NDK编译so出来使用,仍然以上面github的工程为例:



编译so,直接在命令终端切换到jni目录下,执行ndk-build命令即可.对应jni接口的java层基本类都在lualib module中了.

那么如何使用这个库呢.基本步骤如下:

<1>:初始化Lua,获取LuaState对象:

```
lua = LuaStateFactory.newLuaState();
if (lua == null) {
 return;
}
lua.openLibs();
```

<2>：加载脚本:

```
// 直接加载lua源码,这种方式适合加载少量的或者启动代码
lua.LdoString(readAssetsTxt(this, "test.lua"));
```

这个是从asset目录下加载test.lua脚本文件.但是这个地方稍微注意就会发现,这个只是加载一个test.lua的脚本,但是生产环境下往往会有大量的lua脚本.显然上面的方式不合适.

那么加载多个Lua脚本如下:

```
private void loadLua(){
 // 这种可以批量加载代码
 int ret = lua.LdoFile("/sdcard/libmodule.lua");
 Log.d("LdoFile","ret : "+ret);
 lua.LdoFile("/sdcard/test.lua");
 initLuaPath(lua);
}
// 加载lua文件后还要添加环境变量,否则无法一个lua调用另外一个lua的时候会提示无法找到文件
private void initLuaPath(LuaState L) {
  L.getGlobal("package");
  L.pushString("/sdcard/libmodule.lua;");
  L.setField(-2, "path");
  /*L.pushString(getLuaCpath());
  L.setField(-2, "cpath");*/
 }
```

当然还可以加载c文件和so文件,类似.

即加载lua脚本一般使用:

LdoString	加载脚本程序
LdoFile	加载脚本文件

<3>：加载脚本后即可以开始使用,新建test.lua脚本,程序如下:

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

```lua
local newInstance = luajava.newInstance
v1 = "this is value from lua"

table = {}
table["name"] = "Lily"
table["age"] = 18
table["sex"] = "female"

function extreme(a, b, c)

 -- local String = bindClass("java.lang.String")

 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 utilsobj = newClass(utils)
 -- utilsobj:getName()

 local testStatic = bindClass("pumpkin.org.angrypandalua.utils.LuaJavaFu
ncTest");
 -- 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"))

 -- 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)

 local map = bindClass("java.util.HashMap");
 local mapObj = newClass(map)
 mapObj = testNew:testMap()

 if(mapObj:containsKey("A"))
 then
 print:show(mapObj:get("A"))
 end
 if(mapObj:containsKey("W"))
 then
 print:show(mapObj:get("W"))
 else
 print:show("can not find the value !")
 end

 local max = a
 local min = a
 if(b>max) then
 max = b
 elseif(b<min) then
 min = b
 end
 if(c>max) then
 max = c
 elseif(c<min) then
 min = c
 end

 return max, min
end

function luaCallback(tv)
 -- http 函数是由 java `AsyncJavaFunction` 类注入的
 -- http function was injected by java `AsyncJavaFunction`
 http(function(result, time)
 tv:setText(string.format("result: %s\ntime: %dms", result, time));
 end
```

```
  )
end
```

<4>：Java调用Lua脚本,其实还是很显然的,和Java使用LuaJava套路基本差不多,都是通过Globals进行操作,比如:

```
/**
 * 从lua中获取变量值
 * @param key : 变量名
 */
private void getLuaVar(String key){
 lua.getGlobal(key);
 mText.setText(lua.toString(-1));
 lua.pop(1);
}

/**
 * 给lua中变量赋值
 * @param key
 */
private void pushLuaVar(String key){
 lua.pushString("value from java");
 lua.setGlobal(key);
}
```

调用:

```
getLuaVar("v1");
pushLuaVar("v2");
```

Table是lua一个比较重要的数据体,操作Lua的Table如下:

```
/**
 * 获取Table值
 */
private void getLuaTable(){
 StringBuilder s = new StringBuilder();
 lua.getGlobal("table");
 s.delete(0, s.length());
 if (lua.isTable(-1)) {
 lua.pushNil();
  while (lua.next(-2) != 0) {
```

```
 s.append(lua.toString(-2)).append(" = ")
 .append(lua.toString(-1)).append("\n");
 lua.pop(1);
 }
 lua.pop(1);
 mText.setText(s.toString());
 }
}

/**
 * 设置Table中的值,注意这个不是追加
 */
private void setLuaTable(){
 lua.newTable();
 lua.pushString("from");
 lua.pushString("java");
 lua.setTable(-3);
 lua.pushString("value");
 lua.pushString("Hello lua");
 lua.setTable(-3);
 lua.setGlobal("table");
 getLuaTable();
}
```

Java传递对象给Lua使用:

```
/**
 * 将java对象传递给lua调用
 * lua调用方式:"obj:obj's method name(parameters)"
 * @param textView
 */
private void pushJavaObj2Lua(TextView textView){
 lua.pushJavaObject(textView);
 // luaText这个是textView在lua中对应的变量名
 lua.setGlobal("luaText");
 // 这个时候java对象可以在lua中使用了
 lua.pushInteger(Color.GREEN);
 lua.setGlobal("red");
 lua.LdoString("luaText:setTextColor(red)");
}
```

上面看了还是非常简单.

重点是Lua和Java两种方法的互相交互调用.

需要让Lua支持Java方法调用,即Lua调用Java方法,流程如下:

<a>：首先实现一个类,并且该类继承JavaFunction,比如:

```
1  package pumpkin.org.angrypandalua.lua;
2
3  import org.keplerproject.luajava.JavaFunction;
4  import org.keplerproject.luajava.LuaException;
5  import org.keplerproject.luajava.LuaState;
6
7  import java.text.SimpleDateFormat;
8  import java.util.Date;
9
10 /**
11  * @ProjectName: AngryPandaLua
12  * @ClassName: TestJavaFunction
13  * @Author: 刘志保
14  * @CreateDate: 2019/7/5 20:23
15  * @Description: java类作用描述
16  */
17 public class TestJavaFunction extends JavaFunction {
18  /**
19  * Constructor that receives a LuaState.
20  *
21  * @param L LuaState object associated with this JavaFunction object
22  */
23  public TestJavaFunction(LuaState L) {
24  super(L);
25  }
26
27  @Override
28  public int execute() throws LuaException {
29  // 获取Lua传入的参数，注意第一个参数固定为上下文环境。
30  // Getting the parameters passed in by Lua
31  // Notice that the first argument is lua context.
32  String str = L.toString(2);
33
34  SimpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd H
H:mm:ss");
35  Date date = new Date(System.currentTimeMillis());
```

```
36   L.pushString(simpleDateFormat.format(date) + str);
37   return 1; // 返回值的个数
38   }
39
40   public void register() {
41   try {
42   // 注册为 Lua 全局函数
43   // Register as a Lua global function
44   // 在lua中可以调用getTime方法,但是这个方式是java语言实现的
45   register("getTime");
46   } catch (LuaException e) {
47   e.printStackTrace();
48   }
49   }
50
51 }
```

阅读上面的类,需要实现JavaFunction类的两个方法,一个是execute,一个是register方法.

execute	这个是Lua调用java方法,java方法实现的逻辑处
register	这个是Lua调用java方法之前需要先注册,即通过这个方法注册进去,注册带参数即为Lua调用Java的函数名,上面调用的方法名为"getTime"

具体使用如下:

```
1  /**
2   * lua调用Java方法
3   */
4  private void luaUsingJavaFunction(){
5   /**
6   * 首先注册:需要依赖JavaFunction子类进行,这里是TestJavaFunction类
7   * 然后使用lua语言调用注册的方法,并且传入一个参数进去(实际上是两个)
8   */
9   new TestJavaFunction(lua).register();
10   lua.LdoString("return getTime(' - passing by lua')");
11   mText.setText(lua.toString(-1));
12   lua.pop(1);
13
14 }
```

这个是方便一部分使用java程序能够更好更容易实现其逻辑,类似于jsp页面里面嵌入java程序代码.

那么Java如何调用Lua函数方法呢?

```
/**
 * java调用Lua方法
 */
private void callLuaFunction(){
 StringBuilder s = new StringBuilder();
 lua.getGlobal("extreme"); // 获取到函数入栈
 lua.pushNumber(15.6); // 依次压入三个参数
 lua.pushNumber(0.8);
 lua.pushNumber(189);
  /**
  * 第一个参数是函数传入的参数个数;
  * 第二个是返回的结果数量
  * 返回状态值
  */
  lua.pcall(3, 2, 0);
  /**
  * 返回值:
  * lua.toString(-1) 其中-1代表其次返回值
  * lua.toString(-2) 其中-2代表最前面的一个返回值
  * 这个顺序标号是逆序的
  */
  s.append("max : " + lua.toString(-2)+" min : "+lua.toString(-1));
  mText.setText(s);
}
```

似乎看完上面的,Lua能够调用Java程序,Java能够调用Lua程序,似乎两者互通了,但是上面的都只是一些基本的操作而已,要达到生成环境的使用还远远不够.