

## 第27讲 | 如何使用脚本语言编写周边工具?

2018-07-26 蔡能

从0开始学游戏开发

[进入课程 >](#)



讲述：蔡能

时长 07:57 大小 3.65M



上一节，我们讲了脚本语言在游戏开发中的应用，我列举了很多 C 语言代码，这些代码做了这样一些事情：

1. 使用 C 语言和 Lua 语言进行沟通；
2. 在 C 语言代码里，使用了宏和结构，方便批量注册和导入 C 语言函数；
3. Lua 代码如何传输内容给 C 语言；
4. Lua 虚拟机堆栈的使用。

这一节，我们要用 Lua 脚本来编写一个游戏周边工具 Makefile。游戏周边工具有很多种，并没有一个统一的说法，比如在线更新工具、补丁打包工具、人物模型编辑工具、游戏环境设置工具等等。

你或许就会问了，那我为什么选择 Makefile 工具来编写，而不选择别的周边工具来编写呢？


因为这个工具简单、小巧，我们可以将 Lua 脚本语句直接拿来用作 Makefile 语句，而在这个过程中，我们同时还可以通过 Lua 语句来了解 Lua 的工作机理。而且这个编写过程我们一篇文章差不多就可以说清楚。

而别的周边工具编写起来可能会比较复杂，比如如果要编写类似 Awk 的工具的话，就要编写文本解析和文件查找功能；如果编写游戏更新工具的话，就必须涉及网络基础以及压缩解压缩的功能。

简单直白地说，Makefile 是一种编译器的配置脚本文件。这个文件被 GNU Make 命令读取，并且解析其中的意义，调用 C/C++（绝大部分时候）或者别的编译器（小部分）来将源代码编译成为执行文件或者动态、静态链接库。

我们可以自己定义一系列的规则，然后通过顺利地运行 gcc、cl 等命令来进行源代码编译。

我们先定义一系列函数，来固定我们在 Lua 中所使用的函数。

 复制代码

```
1 int compiler(lua_State*);
2 int linker(lua_State*);
3 int target(lua_State*);
4 int source_code(lua_State*);
5 int source_object(lua_State*);
6 int shell_command(lua_State*);
7 int compile_param(lua_State*);
8 int link_param(lua_State*);
9 int make(lua_State*);
```

这些都是注册到 Lua 内部的 C/C++ 函数。我们现在要将这些函数封装给 Lua 使用，但是在这之前，我们要将大部分的功能都在 C/C++ 里编写好。

随后，我们来看一下，在 Lua 脚本里面，具体是怎么实现 Make 命令操作的。

```
1 target("test.exe");
2 linker("c:\\develop\\dm\\bin\\dmc.exe");
3 compiler("c:\\develop\\dm\\bin\\dmc.exe");
4
5 source_code("c.cpp", "fun.cpp", "x.cpp");
6 source_object("c.obj", "fun.obj", "x.obj");
7
8 compile_param( "$SRC", "-c",
9               "-Ic:/develop/dm/stlport/stlport",
10              "c:/develop/dm/lib/stlp45dm_static.lib");
11
12 link_param("$TARGET", "$OBJ");
13 make();
14 shell_command("del *.obj");
```

首先，第一行对应的就是目标文件 target 函数，后续的每一个 Lua 函数都能在最初的函数定义里找到。


在这个例子当中，我们使用的是 DigitalMars 的 C/C++ 编译器，执行文件叫 dmc.exe。我们可以看到，在 linker 和 compiler 函数里都填写了 dmc.exe，说明编译器和链接器都是 dmc.exe 文件。

现在来看一下在 C/C++ 里面是如何定义这个类的。

```
1 struct my_make
2 {
3     string target;
4     string compiler;
5     string linker;
6     vector<string> source_code;
7     vector<string> source_object;
8     vector<string> c_param;
9     vector<string> l_param;
10 };
```

为了便于理解，我将 C++ 类声明改成了 struct，也就是把成员变量改为公有变量，你可以通过一个对象直接访问到。

随后，我们来看一下如何将 target、compiler 和 linker 传入到 C 函数里面。


 复制代码

```
1 int compiler(lua_State* L)
2 {
3     string c = lua_tostring(L, 1);
4     get_my_make().compiler = c;
5     return 0;
6 }
7 int linker(lua_State* L)
8 {
9     string l = lua_tostring(L, 1);
10    get_my_make().linker = l;
11    return 0;
12 }
13 int target(lua_State* L)
14 {
15     string t = lua_tostring(L, 1);
16     get_my_make().target = t;
17     return 0;
18 }
19
```

在这三个函数里面，我们看到，get\_my\_make 函数就是返回一个 my\_make 类的对象。这个具体就不进行说明了，因为返回对象有多种方式，比如 new 一个对象并且 return，或者直接返回一个静态对象。

随后，我们直接使用了 Lua 函数 lua\_tostring，来得到 Lua 传入的参数，比如如果是 target 的话，我们会得到“test.exe”，并且将这个字符串传给 my\_make 对象的 string target 变量。后续的 compiler、linker 也是一样的道理。

我们接着看下面两行。

 复制代码

```
1 source_code("c.cpp", "fun.cpp", "x.cpp");
2 source_object("c.obj", "fun.obj", "x.obj");
```

这两行填入了 cpp 源文件以及 obj 中间文件，这些填入的参数并没有一个固定值，可能是 1 个，也可能是 100 个，那在 C/C++ 和 Lua 的结合里面，我们应该怎么做呢？

我们看到一个函数 lua\_gettop。这个函数是取得在当前函数中，虚拟机中堆栈的大小，所以返回的值，就是堆栈的大小值，比如我们传入 3 个参数，那么返回的就是 3。

接下来可以看到，使用 Lua 的计数方式，从 1 开始计数，并且循环结束的条件是和堆栈大小一样大，然后就在循环内，将传入的参数字符串，压入到 C++ 的 vector 中。

随后的 source\_object、compile\_param 和 link\_param 都是相同的方法，将传入的参数压入到 vector 中。

你可能要问了，我在 Lua 的代码中看到了 *TARGET*、OBJ、\$SRC 等字样的字符串，这些字符串的处理在哪里，这些字符串又是做什么的呢？

这些字符串是替代符号，你可以理解为 C 语言中 printf 函数的格式化符号，例如 “%d %s” 等等，虽然在这里，这些符号都是自己定义的，但是我们仍然需要解析它们。

其实解析的步骤并不难，我们只需要将 vector 内的内容提取出来，对比是不是字符串 \$TARGET 等，如果是的话，就被替代为前面我们在 target 函数或者 source\_code 函数中所定义的内容。

我们拿 source\_code 部分来举例，来看一下部分代码。

 复制代码

```
1 void run()
2     {
3         string command_line;
4         string src = "$SRC";
5         string tar = "$TARGET";
6         string obj = "$OBJ";
7         for(int i = 0; i < source_code.size(); i++)
8         {
9             .....
10        for(int j=0; j<c_param.size(); j++)
11        {
12            if(c_param[j] == src)
13            {
14                command_line += source_code[i];
15                .....
```

```
16         }
17     }
18 }
```

在这部分的代码里面可以看到，我们将压入的 `source_code` 内容进行循环。在循环之后，必须对 `c_param` (`compile_param`)，也就是编译参数进行循环。当我们发现编译参数里面出现了 `$SRC` 这个替代字符串的时候，就将 `source_code` 的内容（其实就是源代码文件）合并到 `command_line`（命令行）里面去，然后整合成为一个完整的、可以运行的命令行。

随后我再贴一部分代码，可以看到别的可替代字符串是怎么做的。

 复制代码

```
1 else if(c_param[j] == obj)
2 {
3     command_line += source_object[i];
4 }
5 else if(c_param[j] == tar)
6 {
7     command_line += target;
8 }
```

我们对替代字符串做了相同的比较，如果是一致的话，就将被替代内容添加到 `command_line` 变量里面，组成一个完整的可运行命令行。

这个 `run` 函数其实就是在 `make` 的时候调用的函数。至于如何调用这一串 `command` 命令，在 C 里面最简单的方式就是调用 `system` 函数，或者使用 `execl` 函数系列。注意，这个 `execl` 并不是来自微软的 `excel` 表格，而是 C 语言的函数。

我们封装完了 Lua 部分的代码之后，就需要将 Lua 的函数注册到 Lua 虚拟机里面，这个我上一节已经具体说过了。

最后，由于我们的 Lua 源代码本身就是一个 `Makefile` 文件，所以我们不需要做过多的解析，直接将这个源代码输入给 Lua 虚拟机即可。

```
1 string makefile;
2 ifstream in("my_makefile");
3 makefile = "my_makefile";
4 if(!in.is_open())
5 {
6 in.close();
7 }
8 else luaL_dofile(L, makefile.c_str());
```

在这段代码里面，我们首先使用 C++ 的 fstream 库中的 ifstream 来尝试读取是不是有这个 my\_makefile 文件，如果没有的话，就跳过，并且关闭文件句柄，如果存在的话，就把这个文件填入到 Lua 虚拟机中，让 Lua 虚拟机直接运行这个源文件。所以这种方式是最简单快捷的。

代码有点多，不要担心，我带你梳理一下今天的内容。

1. **利用 C/C++ 语言和 Lua 源代码进行交互，从 Lua 代码中获取数据并且在 C 语言里面进行算法的封装和计算，最后将结果返回给 Lua。** 我们在 C/C++ 语言里面进行大量的封装和算法提取，并且也利用 C/C++ 进行调用和结果的呈现，这是一种常用的方式，也就是 C 语言占比 60%~70%，Lua 代码占比 30%~40%。
2. 另一种比较好的方式是，**使用 C/C++ 编写底层实现逻辑，随后将数据传输给 Lua，让 Lua 来做逻辑运算，最终将结果返回给 C 语言并且呈现出来。**这是很多人在游戏开发中都会做的事情，比如我们编写地图编辑器，先在 Lua 中编写好逻辑，用 C 语言在界面中呈现出来即可。如果反过来做的话，那就会出现大量的硬代码，是很不合适的。所以这种情况下，C 语言占比 30%~40%，Lua 代码占比 60%~70%。
3. **Lua 可以是一种胶水语言。严谨地说，像 Python、Ruby 等脚本语言，都是合格的胶水语言。** 在这种情况下，胶水语言起到的作用就是粘合系统语言（C/C++）和上层脚本逻辑。所以，使用胶水语言，就像是一种动态的配置文件。  
按照普通的配置文件来讲，你需要手工解析比如类似 INI、XML、JSON 等配置文件，随后按照这些文件的内容来做出一系列的配置，但是胶水语言不需要，它本身就是一种动态的语言。  
你也可以把它当作一种配置的文件，就像今天讲的 Makefile，它可以不需要你检测语法问题，这些问题在 Lua 虚拟机本身就已经做掉了，你需要做的就是将我们脑海里想让它做的事情，通过 C 和 Lua 的库代码进行整合，直接使用就可以了。所以，**胶水语言的本身就是一个配置文件，同时它也是一个脚本语言源代码。**



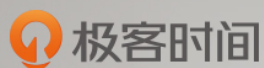
## 小结

在使用 C/C++ 结合脚本语言的时候，需要梳理这些内容，比如哪些是放在 C/C++ 硬代码里写的，那些可以放到脚本语言里写，梳理完后，就可以将脚本语言和 C/C++ 结合起来，编写出易于修改脚本逻辑（如果有不同需求，可以很方便地改写脚本而不需要动 C/C++ 硬代码）、易于使用的工具。

现在给你留一个小问题吧。

在 Lua 当中有 table 表的存在，如何在 C 语言中，给 Lua 源代码生成一个 table 表，并且可以在 Lua 中正常使用呢？

欢迎留言说出你的看法。我在下一节的挑战中等你！



# 从 0 开始学游戏开发

你的游戏开发入门第一课

蔡能

原网易游戏引擎架构师  
资深游戏底层技术专家



新版升级：点击「 请朋友读」，10位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 第26讲 | 脚本语言在游戏开发中有哪些应用？

下一篇 第28讲 | 热点剖析（七）：谈谈微信小游戏的成功点





**放羊大王**

2018-08-08



第一次见这样的make file 感觉更像makeList.txt， 听的懂， 但实践太难了。 😊

---



**OCEAN**

2018-07-30



table表对应c中数据， Lua中给出访问接口， 并构造这个table表

展开 ▾