TURING 图灵原创

■ codedump ◎著

Lua 设计与实现

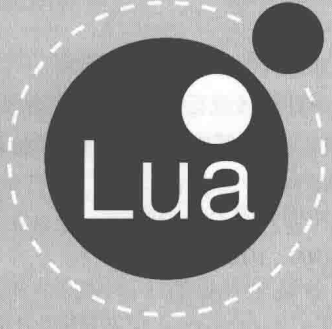
★ 第一本揭示Lua实现原理的图书
★ 经典的纯C语言项目分析
★ 一线开发人员倾力打造

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵原创

Lua 设计与实现

codedump ◎著



人民邮电出版社
北京

图书在版编目（CIP）数据

Lua设计与实现 / 李创著. -- 北京 : 人民邮电出版社, 2017.8
（图灵原创）
ISBN 978-7-115-46537-5

Ⅰ. ①L… Ⅱ. ①李… Ⅲ. ①游戏程序－程序设计
Ⅳ. ①TP317.6

中国版本图书馆CIP数据核字(2017)第185067号

内 容 提 要

本书基于Lua 5.1.4版本讨论了Lua语言的设计原理，全书共分三部分：第一部分讲解数据结构（如通用数据是如何表示的）、字符串以及表类型的实现原理；第二部分是本书最重要的部分，主要讨论了虚拟机的实现；第三部分讨论了垃圾回收、模块实现、热更新、协程等的实现原理。

本书适合对Lua内部实现感兴趣以及对编程语言实现原理感兴趣的人阅读。

◆ 著　　codedump
责任编辑　王军花
责任印制　彭志环

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷

◆ 开本：800×1000　1/16
印张：12.25
字数：290千字　　2017年8月第1版
印数：1－3 000册　　2017年8月北京第1次印刷

定价：49.00元

读者服务热线：(010)51095186转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字20170147号

站在巨人的肩上

Standing on Shoulders of Giants

iTuring.cn

站在巨人的肩上

Standing on Shoulders of Giants

iTuring.cn

前　言

接触 Lua 是在很早之前，但是那时仅限于基本的学习，没有在项目中使用，也没有意识到这门语言真正的魅力。

时间来到 2011 年，那时我在从事网页游戏的开发工作。游戏开发有其独有的特点——上线周期短，经常一两周就要有一个版本上线，而这个过程中需要实现的功能并不见得少。简单地说，就是时间紧、任务重。

假如采用 C++这样的编译型语言来开发游戏，那么典型的开发流程大致是这样的：撸起袖子来写了一大段代码，然后编译、解决调试编译的错误，中间可能还要处理类似崩溃、段错误、内存泄露等问题。另外，由于重新编译了代码，又需要重启服务器，而重启过程中势必涉及数据的加载。总而言之，采用纯编译型语言开发的情况下，相当一部分时间并没有用在真正的业务逻辑开发中。

当时项目采用的是 C++编写的核心引擎模块，暴露核心接口给 Lua 脚本层，网络数据的收发等都在 C++层完成，而业务逻辑采用 Lua 实现。这个架构也是很多游戏服务器采用的经典架构。使用这个架构来开发游戏服务时，不再会把大量的精力放在语言本身的问题上，而可以集中精力来做业务逻辑。另外，借助于 Lua 的热更新能力，整个开发过程中需要重启服务的次数并不多。

可以说，这个项目经历打开了一扇新的窗口，开阔了我的视野。由于在项目开发过程中尝到了甜头，并且自己也对编程语言的实现很感兴趣，所以业余时间就开始慢慢阅读 Lua 解释器的实现原理。那时候在互联网上能找到的分析 Lua 实现的文章并不多，加上自己编译原理等相关知识的基础薄弱，大部分时候只能硬啃代码。我一边摸索，一边开始逐步整理相关的文章并将其放在网上，希望对其他有类似需求的朋友有一些帮助。

Lua 作为一门诞生已经超过 20 年的语言，在设计上是非常克制的。以本书讲解所涉及的 Lua 5.1.4 版本来说，这个版本是 Lua 发展了十几年之后稳定了很长时间的版本，其解释器加上周边的库函数等不过就是一万多行的代码量，而如果再进行精简，只需要吃透最核心的几千行代码就可以了。这样一门广泛使用的工业级别的脚本语言，只需要吃透几千行代码就能明白其核心原理，这个性价比极高的诱惑对当时的我来说无疑是巨大的。

Lua 在设计上，从一开始就把简洁、高效、可移植性、可嵌入型、可扩展性等作为自己的目

标。打一个可能不是太恰当的比方，Lua 专注于做一个配角，作为胶水语言来辅助像 C、C++这样的主角来更好地完成工作。当其他语言在前面攻城拔寨时，它在后方完成自己辅助的作用。在现在大部分主流编程语言都在走大而全的路线，在号称学会某一门语言就能成为所谓的“全栈工程师”的年代，Lua 始终恪守本分地做好自己胶水语言的本职工作，不得不说是一个异类的存在。

“上善若水，水善利万物而不争”，简单、极致、强大的可扩展性，大概是我能想到的最适合用来描述 Lua 语言设计哲学的句子。

本书将对 Lua 语言的设计原理做一些分析讨论，采用的是 Lua 5.1.4 版本，在引用到该版本中的代码时，会在引用代码的同时加上代码所在的文件以及行号，方便读者对应到具体的代码中一起跟着阅读。另外，我也把自己在阅读 Lua 代码中做了一些注释的代码版本放在了 GitHub 上，地址是：https://github.com/lichuang/Lua-5.1.4-codedump。

本书适用于以下读者。

- 希望能够进一步了解 Lua 实现的内部原理的 Lua 语言用户。
- 对程序语言设计感兴趣的读者。

阅读本书，读者至少需要具备以下的基础知识。

- 扎实的 C 语言功底，Lua 虚拟机采用纯 C 编写。在我看过不算少的纯 C 语言完成的项目中，Lua 虚拟机的代码质量是最高的。
- 一定的编译原理知识，比如了解词法分析、语法分析、递归下降分析、BNF 规则等，如果不清楚这些原理，阅读 Lua 虚拟机实现时会遇到很多问题。

本书按照如下方式组织。

- 第一部分讲解 Lua 中的数据结构，如通用数据是如何表示的，Lua 的字符串以及表类型的实现原理。
- 第二部分是本书最重要的部分，主要讨论了 Lua 虚拟机的实现。另外，这里分类讲解了 Lua 虚拟机中的一些重点指令。
- 第三部分的内容比较杂，这部分讨论垃圾回收、模块实现、热更新、协程等的实现原理。

本书的完成要特别感谢以下几个人。

感谢图灵公司的王军花编辑，在茫茫的互联网中找到我在网上开源的 Lua 分析系列文章（这也是本书写作的基础），并且鼓励我整理出版，在多次跳票的情况下给予了我很多的鼓励和帮助。没有她的发掘和鼓励，就不会有本书。

感谢云风在百忙之中抽空对本书初稿进行了审阅，给予了很多中肯的意见。有一些我听取并进行了改进，而有一些因为各种原因很遗憾没能进行完善。

感谢我太太对我工作的理解，家人的理解和支持是一切的基础。

最后，由于本人能力有限，在很多问题的讨论上可能还存在一些误区，希望读者不吝赐教。请从这里开始您的旅程……

目　录

第三部分 独立功能的实现

第 1 章
概述

在这一章中，我们首先对Lua语言的历史进行简单回顾，了解其发展经历及设计哲学。接下来，会对Lua的源码组织、函数命名等做一些介绍。Lua是C语言项目的典范，在这方面也做得很规范、整齐，这给我们阅读源码带来了便利。最后，简单介绍一下Lua虚拟机整体的工作流程。

1.1 前世今生

Lua语言于1993年诞生于巴西里约热内卢Pontifical Catholic大学（简称PUC-Rio）的Tecgraf实验室，作者是Roberto Ierusalimschy、Luiz Henrique de Figueiredo和Waldemar Celes。Tecgraf实验室创立于1987年，主要专注于图形图像相关的工具研发。创立之后，该实验室的工作就是向客户提供基本的图形相关的软件工具，比如图形库、图形终端等。

从1977年到1992年，巴西政府实施了“市场保护”政策，这使得计算机软硬件存在巨大的贸易壁垒。在这种大环境下，Tecgraf实验室的很多客户由于政治和经济上的原因，都不能向国外公司购买定制化的软件。这些原因都驱使Tecgraf实验室的工作人员从头开始构建面向本国用户的软件工具。

Petrobras（巴西石油公司）是Tecgraf的最大客户之一，Tecgraf为其开发了两门语言，分别是DEL和SOL，这两门语言是Lua语言的前身。

Petrobras的工程师每天要处理的一个问题是，输入大量的数据文件到数值模拟器上，每个文件有行和列，有点类似今天的Excel文件。这个过程琐碎且繁杂，因为模拟程序会对数据进行严格的检查，人工输入数据时常会导致错误。Petrobras向Tecgraf实验室提出需求，要求他们开发用于输入这种类型数据的图形终端给工程师们使用，这些终端的输入是可交互的，可以由使用者定义规则，然后自动化地生成模拟器能识别的正确数据。除了按照规则生成数据之外，这个终端还

需要提供数据校验等功能。

为了简化这个终端的开发，Figueiredo和另一位工程师Luiz Cristovao Gomes Coelho决定为这个终端开发一种称为DEL（Data Entry Language）的语言，它更像现在的DSL（domain-specific language，领域特定的语言）。

DEL在Petrobras获得了广泛的使用，Petrobras对它提出了更高的要求，要求能够提供控制处理等特性，这更像一门程序语言了。

几乎在DEL被创建的同一时间，由Ierusalimschy和Celes领导的另一个团队开始在PGM上面工作，这是一个可配置的用于生成岩石属性文件的生成器，这个产品的客户同样也是Petrobras公司。

PGM生成的报告包含多个列，其中的数据是高度可配置的：用户可以选择颜色、字体、标签等，同时这些配置信息还可以保存下来重用。于是这个开发团队决定为PGM开发一门语言，称为SOL（Simple Object Language，简单对象语言）。

因为PGM会处理很多不同的对象，每个对象都可能有许多的属性，所以开发团队决定为这门语言加上类型声明的特性，比如：

```
type @track{ x:number, y:number=23, id=0 }
type @line{ t:@track=@track{x=8}, z:number* }
T = @track{ y=9, x=10, id="1992-34" }
L = @line{ t=@track{x=T.y, y=T.x}, z=[2,3,4] }
```

这段代码定义了两种类型track和line，创建了两个对象，分别是track的类型对象T以及line类型的对象L。track类型有两个数值属性，分别是x和y，以及一个没有类型的属性id。line类型有一个track类型的属性t，其中x的默认值是8，以及一个number列表的类型z。

SOL团队在1993年完成了初期的开发，但是并没有发布这个版本，原因是此时PGM要求在这门语言中支持过程编程的一些特性，这要求SOL需要进行扩展了。

与此同时，前面提到的DEL语言也遇到了类似的需求。于是在1993年，Figueiredo和Celes坐在一起讨论了这两门语言面对的问题和挑战，它需要满足以下当时考虑到的需求。

- 需要是一门真正的编程语言，提供赋值、控制结构、子函数等编程语言的特性，而不仅仅是一门数据描述语言。
- 与SOL一样，对数据描述提供便利。
- 因为面向的用户很多都是没有编程经验的人，所以这门语言需要足够地简单和易于上手。
- 因为Petrobras公司的很多设备运行在不同的平台上，所以要求这门语言的可移植性和便携性要足够好。

当时，满足这几个需求的语言还不存在。Tcl和Perl语言只能运行在Unix平台，它们的语法对外行人也不够友好。

于是他们决定创造一门更强大的编程语言来代替它们。因为这门语言的前身之一是SOL语言，在葡萄牙语中这个单词的意思是“太阳”，他们决定给这门新的语言起名为Lua，葡萄牙语的意思是“月亮”。Lua语言就这样诞生了。

Lua语言继承了SOL语言对列表和记录的表示方式，但是从一开始就将两者统一起来。这就是到现在都能看到的，Lua的表既能作为散列表，也可以作为数组。

1996年对Lua来说是很重要的一年，Lua开始在国际上获得了关注，迎来国际用户。在这一年，作者在*Software: Practice & Experience*杂志上发表了一篇关于Lua的论文，引来了不少的关注。同年12月，Lua 2.5版本发布，*Dr. Dobb's Journal*杂志也专门针对Lua做了报告。由于这本杂志在程序员圈子里受众非常多，吸引了软件业中不少从业者注意，这其中包括当时任职于Lucas艺术旗下Grim Fandango游戏项目的主管Bret Mogilefsky。由于Lua的良好特性，他在自己的项目中使用Lua替换掉了项目原来用的脚本语言，后来又在Game Developers Conference（简称为GDC，是游戏程序员最重要的会议之一）分享了自己使用Lua的成功经验。从此，Lua在游戏圈就开始流行起来了，这其中包括了后来大获成功的WOW等。如今Lua语言已经是游戏领域使用最广泛的脚本语言之一。

Lua虽然起源于巴西，也是从巴西公司的项目中受需求驱动而开发的，但是从一开始这门语言的设计者就把眼光投向世界。在很长一段时间里，Lua的文档只有英语版本，而不是作者的母语葡萄牙语。前面提到的1996年发表的论文，同样也可以看作Lua作者们国际化视野的一个标志。

Lua语言从一开始就将自己定位成一个“嵌入式的脚本语言”，提供了如下的特性。

- **可移植性**：使用clean C编写的解释器，可以在Mac、Unix、Windows等多个平台轻松编译通过。
- **良好的嵌入性**：Lua提供了非常丰富的API，可供宿主程序与Lua脚本之间进行通信和交换数据。
- **非常小的尺寸**：Lua 5.1版本的压缩包，仅有208KB，解压缩之后也不过是835KB，一张软盘就可以装下。Lua解释器的源代码只有17 000多行的C代码，编译之后的二进制库文件仅有143KB，这些都决定了使用Lua的设备并不会因为添加了它导致非常明显的空间占用。
- **Lua的效率很高，是速度最快的脚本语言之一**：为了提高Lua的性能，作者们将最初使用Lex、Yacc等工具自动生成的代码都变成了自己手写的词法分析器和解析器。

这意味着，用户使用C、C++等语言进行主要功能的开发，而一些需要扩展、配置等会频繁动态变化的部分使用Lua语言来进行开发。Lua语言的以上几个特性，都决定了它能很好地完成这些辅助作用。Lua的作者甚至戏称这门语言是一门能穿过针孔的语言（Passing a Language through the Eye of a Needle），“小而精”大概是对Lua语言最好的描述了。

作为一门从发展中国家起源的语言，在一开始的选择和定位上，Lua都做了现在看来正确的选择：面向国际，老老实实做好辅助作用。在一个点上做精做细，而不是走大而全的路线去与类似背景的语言进行竞争，这是Lua后来取得巨大成功的原因。这也能理解为什么过去了这么多年，至今Lua解释器的代码只有非常少的代码量（以本书中分析的5.1.4版本来看，全部C代码只有17 193行。如果只算核心部分，那就更少了）。

这也是我一直很推崇Lua解释器源码，并且决定将这门语言进行分析的原因：Lua解释器的代码是殿堂级的C语言代码范本，Lua作者对语言特性、设计目标、受众的取舍值得我们学习。从一万多行的源码中，就能学习到一门工业级脚本语言的实现，性价比是极高的。

除了在游戏领域的广泛使用，Lua在其他领域也获得了运用。

- OpenResty使用Lua来扩展Nginx服务器的功能，使用者仅需要编写Lua代码就能轻松完成业务逻辑。值得一提的是，这个项目的作者是中国人章亦春。
- Redis服务提供Lua脚本。
- Adobe的Lightroom项目使用Lua来编写插件。

还有很多非游戏领域的成功项目，在此不一一列举了。

那么，如何在你的项目中使用Lua语言呢？以笔者比较熟悉的游戏服务器领域来说，一般是这样组织和分工的。

- C\C++语言实现的服务器引擎内核，其中包括最核心的功能，比如网络收发、数据库查询、游戏主逻辑循环等。以下将这一层简称为引擎层。
- 向引擎层注册一个Lua主逻辑脚本，当接收到用户数据时，将数据包放到Lua脚本中进行处理，主逻辑脚本主要是一个大的函数表，可以根据接收到的协议包的类型，调用相关的函数进行处理。以下将这一层简称为脚本层。
- 引擎层向脚本层提供很多API，能方便地调用引擎层的操作，比如脚本层处理完逻辑之后调用引擎层的接口应答数据等。

可以看到，在这个架构中，引擎层实现了游戏服务的核心功能，这部分的变动相对而言不那么频繁；而游戏的逻辑、玩法是变动很频繁的，这部分使用脚本来完成。这个组合架构的优势在于如下几点。

- **编码效率高**：由于引擎层相对稳定，而脚本不需要进行编译就能直接运行，省去了很多编译的时间。
- **开发效率高**：大部分脚本，包括Lua在内都支持热更新功能，这意味着在调试开发期间，可以不用停服务器就能调试新的脚本代码，这省去了重启服务的时间，比如加载数据库数据、静态配置文件等的耗时。

- **对人员素质要求相对低**：一般的游戏服务器团队配置，都是由主程级别的人来把控引擎的质量，其他的成员负责编写脚本玩法逻辑，即使出错，大部分时候并不会导致服务器宕机等严重问题。

1.2 源码组织

本书是基于Lua 5.1.4版本进行分析的，打开src目录下的Makefile文件，可以看到这样一段代码：

```
25 LUA_A=  liblua.a
26 CORE_O= lapi.o lcode.o ldebug.o ldo.o ldump.o lfunc.o lgc.o llex.o lmem.o \
27   lobject.o lopcodes.o lparser.o lstate.o lstring.o ltable.o ltm.o  \
28   lundump.o lvm.o lzio.o
29 LIB_O=  lauxlib.o lbaselib.o ldblib.o liolib.o lmathlib.o loslib.o ltablib.o \
30   lstrlib.o loadlib.o linit.o
31
32 LUA_T=  lua
33 LUA_O=  lua.o
34
35 LUAC_T= luac
36 LUAC_O= luac.o print.o
```

从中能看到，Lua源码大体分为三个部分：虚拟机核心、内嵌库以及解释器、编译器。

虚拟机核心的文件列表如表1-1所示。需要补充说明的是，Lua解释器中，内部模块对外提供的接口、数据结构都以“lua模块名简称_”作为前缀，而供外部调用的API则使用“lua_”前缀。

表1-1 虚拟机核心相关文件列表

文 件 名	作 用	对外接口前缀
lapi.c	C语言接口	lua_
lcode.c	源码生成器	luaK_
ldebug.c	调试库	luaG_
ldo.c	函数调用及栈管理	luaD_
ldump.o	序列化预编译的Lua字节码	
lfunc.c	提供操作函数原型及闭包的辅助函数	luaF_
lgc.c	GC	luaC_
llex.c	词法分析	luaX_
lmem.c	内存管理	luaM_
lobject.c	对象管理	luaO_
lopcodes.c	字节码操作	luaP_
lparser.c	分析器	luaY_

（续）

文 件 名	作　用	对外接口前缀
lstate.c	全局状态机	luaE_
lstring.c	字符串操作	luaS_
ltable.c	表操作	luaH_
lundump.c	加载预编译字节码	luaU_
ltm.c	tag方法	luaT_
lzio.c	缓存流接口	luaZ_

内嵌库文件如表1-2所示。

表1-2　内嵌库相关文件列表

文 件 名	作　用
lauxlib.c	库编写时需要用到的辅助函数库
lbaselib.c	基础库
ldblib.c	调试库
liolib.c	IO库
lmathlib.c	数学库
loslib.c	OS库
ltablib.c	表操作库
lstrlib.c	字符串操作库
loadlib.c	动态扩展库加载器
linit.c	负责内嵌库的初始化

解析器、字节码编译器的相关文件如表1-3所示。

表1-3　解析器、字节码编译器相关文件列表

文 件 名	作　用
lua.c	解释器
luac.c	字节码编译器

1.3　Lua虚拟机工作流程

关于Lua虚拟机工作流程的详细分析，第5章会专门介绍，但是这里还是先大概了解一下。

Lua代码是通过翻译成Lua虚拟机能识别的字节码运行的，以此它主要分为两大部分。

- **翻译代码以及编译为字节码的部分**。这部分代码负责将Lua代码进行词法分析、语法分析等，最终生成字节码。涉及这部分的代码文件包括llex.c（用于进行词法分析）和lparser.c（用于进行语法分析），而最终生成的代码则使用了lcode.c文件中的功能。在lopcodes.h、lopcodes.c文件中，则定义了Lua虚拟机相关的字节码指令的格式以及相关的API。
- **Lua虚拟机相关的部分**。在第一步中，经过分析阶段之后，生成了对应的字节码，第二步就是将这些字节码装载到虚拟机中执行。Lua虚拟机相关的代码在lvm.c中，虚拟机执行的主函数是luaV_execute，不难想象这个函数是一个大的循环，依次从字节码中取出指令并执行。Lua虚拟机对外看到的数据结构是lua_State，这个结构体将一直贯穿整个分析以及执行阶段。除了虚拟机的执行之外，Lua的核心部分还包括了进行函数调用和返回处理的相关代码，主要处理函数调用前后环境的准备和还原，这部分代码在ldo.c中，垃圾回收部分的代码在lgc.c中。Lua是一门嵌入式的脚本语言，这意味着它的设计目标之一必须满足能够与宿主系统进行交互，这部分代码在lapi.c中。