

Linguagem de Programação Lua

Ueider F. de Oliveira

¹Centro Politécnico – Universidade Católica de Pelotas (UCPel)
Pelotas – RS – Brasil

ueiderfdo@ucpel.tche.br

Abstract. *The Lua programming language was developed by a development team at PUC-Rio, which is an interpreted programming language, procedural, imperative, scripting, with simple syntax, fast, light and designed for extending applications. Lua is dynamically typed, is interpreted from bytecodes to a virtual machine-based registers, and has automatic memory management with incremental garbage collection. These characteristics make it ideal for configuration, automation (scripting) and rapid prototyping.*

Resumo. *A linguagem de programação Lua foi desenvolvida por uma equipe de desenvolvimento na PUC-Rio, sendo esta uma linguagem de programação interpretada, procedural, imperativa, de script, com sintaxe simples, rápida, leve e projetada para estender aplicações. Lua é tipada dinamicamente, é interpretada a partir de bytecodes para uma máquina virtual baseada em registradores, e tem gerenciamento automático de memória com coleta de lixo incremental. Essas características fazem de Lua uma linguagem ideal para configuração, automação (scripting) e prototipagem rápida.*

1. Introdução

O objetivo deste artigo é mostrar um pouco da linguagem de programação Lua, sua história, características, como é seu processo de interpretação, onde é usada e demais assuntos pertinentes à linguagem.

A motivação para a criação de Lua no Tecgraf foi a necessidade crescente das suas aplicações serem configuráveis externamente pelos usuários. Isso quer dizer que diversos aspectos essenciais das aplicações podem ser modificados sem recompilar a aplicação. Desde o início, nas aplicações criadas no Tecgraf, esse tipo de configuração era muito mais do que simplesmente poder escolher a cor da janela ou o tipo de fonte de texto: era necessário poder tomar decisões em tempo de execução que somente os usuários sabiam quais eram. Sendo assim, era necessário fornecer algum tipo de programação para os usuários finais. Outro tipo de configuração era a descrição de 2 complexos relatórios e análises feitas pela Petrobras por encomenda ao Tecgraf. Mais uma vez, essa descrição não podia estar congelada dentro da aplicação pois cada usuário tinha uma necessidade diferente e que mudava a cada tarefa. O Tecgraf tinha, portanto (e ainda tem) forte demanda para aplicações que fossem configuráveis externamente, tanto descrevendo que decisões deveriam ser tomadas quanto descrevendo quais dados seriam usados e como eles seriam usados.

Após projetar e usar com sucesso duas pequenas linguagens específicas para cada uma dessas tarefas, o Tecgraf decidiu investir na criação de uma linguagem única que

puddesse atender a todas as necessidades de configuração das suas aplicações. Assim nasceu Lua, uma linguagem de programação de script, procedural, interpretada, pequena, reflexiva e leve, projetada para dar suporte à programação procedimental em geral e que oferece facilidades para a descrição de dados. A linguagem também oferece um bom suporte para programação orientada a objetos, programação funcional e programação orientada a dados [1].

Lua foi planejada para ser utilizada por qualquer aplicação que necessite de uma linguagem de script leve e poderosa. Lua é implementada como uma biblioteca, escrita em C limpo (isto é, no subconjunto comum de ANSI C e C++) [6].

2. História

Lua é inteiramente projetada, implementada e desenvolvida no Brasil, por uma equipe na PUC-Rio (Pontifícia Universidade Católica do Rio de Janeiro). Lua nasceu e cresceu no Tecgraf, o Grupo de Tecnologia em Computação Gráfica da PUC-Rio. Atualmente, Lua é desenvolvida no laboratório Lablua. Tanto o Tecgraf quanto Lablua são laboratórios do Departamento de Informática da PUC-Rio.

A primeira versão de Lua (1.0) foi lançada em julho de 1993, sendo a primeira versão pública (1.1) lançada em julho de 1994. A versão atual é a 5.0.2, lançada em março de 2004 para corrigir pequenas falhas na versão 5.0, de abril de 2003. A versão 5.1 está em desenvolvimento no momento. Este texto trata da versão oficial atual (5.0.2). A versão seguinte de Lua (2.1) foi lançada em fevereiro de 1995 e trouxe maior poder de expressão, introduzindo a noção de semântica extensível: passou a ser possível programar o que fazer em casos excepcionais, como quando um campo não existe numa tabela. Ter uma semântica extensível é desde então uma característica marcante de Lua.

A versão 2.1 também foi a primeira a ser completamente livre; as versões anteriores eram livres somente para aplicações acadêmicas. Aliada à portabilidade e à eficiência da implementação, a falta de restrições de uso foi um dos fatores importantes na adoção de Lua em inúmeros projetos no mundo todo.

A primeira notícia sobre o uso de Lua em jogos foi em 1997 quando a LucasArts adotou Lua como a sua linguagem de script no lugar de SCUMM no jogo "Grim Fandango". A adoção de Lua nesse jogo foi uma consequência direta da publicação de um artigo de divulgação sobre Lua na revista Dr. Dobbs's Journal, em junho de 1996. Desde então, Lua tem sido cada vez mais usada em jogos, uma área longe das áreas que motivaram a sua criação! Atualmente, Lua tem uma comunidade ativa de programadores. A lista de discussão tem mais de 750 assinantes do mundo todo. Além do site oficial de Lua (lua.org), há também um ativo site mantido por usuários (lua-users.org), uma sala de bate-papo, um *web forum* e um repositório (luaforge.net) [1].

3. Linguagem Lua

Lua é uma linguagem dinâmica, mas apresenta algumas particularidades em relação a outras linguagens desta mesma classe, como o uso de técnicas de programação funcional, uso ubíquo de tabelas como estruturas de dados para os mais variados fins, o uso de corrotinas e a comunicação com código escrito em C. Para definir uma linguagem como dinâmica ela precisa ter as seguintes características:

- Interpretação dinâmica: isso significa que a linguagem é capaz de executar trechos de código criados dinamicamente, no mesmo ambiente de execução do programa. Como exemplos dessa facilidade temos a função `loadstring` em Lua e a função `eval` em Scheme/Lisp e Perl.
- Tipagem dinâmica forte: tipagem dinâmica significa que a linguagem faz verificação de tipos em tempo de execução do programa. Linguagens com tipagem dinâmica em geral não possuem declarações de tipos no código e não fazem verificação de tipos em tempo de compilação. Tipagem forte significa que a linguagem jamais aplica uma operação a um tipo incorreto.
- Gerência automática de memória dinâmica (coleta de lixo): isso significa que não precisamos gerenciar memória explicitamente no nosso programa; em especial, não há necessidade de um comando para liberar memória após seu uso.

Em geral, linguagens dinâmicas são interpretadas, e não compiladas para código nativo da máquina; mas essa é uma característica das implementações dessas linguagens, não das linguagens em si. Obviamente, as características acima favorecem uma implementação via um interpretador e dificultam a construção de compiladores.

Lua se destaca de outras linguagens dinâmicas por ser uma linguagem de script. Uma linguagem de script é uma linguagem projetada para controlar e coordenar componentes geralmente escritos em outra linguagem. As primeiras linguagens de script foram as linguagens de shell do Unix, usadas para conectar e controlar a execução de programas. Apesar de várias linguagens dinâmicas poderem ser usadas para script, poucas foram projetadas para essa finalidade. Lua seguiu um caminho criado por Tcl, onde a linguagem é estruturada como uma biblioteca C com uma API que permite tanto código na linguagem chamar funções escritas em C como código C chamar funções escritas na linguagem. Lua se destaca de outras linguagens de script por sua simplicidade, portabilidade, economia de recursos e desempenho [4].

```

include "lua.h"
include "lauxlib.h"

int main (int argc, char **argv){
    luaState *L = luaLnewstate();
    if (luaLloadfile(L, argv[1]) != LUAOK)
        fprintf(stderr, "error:luatostring(L,-1));
    else if (lua_pcall(L, 0, 0, 0) != LUAOK)
        fprintf(stderr, "error:luatostring(L,-1));
    else{
        luagetglobal(L, "result");
        printf("resultado: %f\n, lua_tonumber(L, -1));
    }
    luaclose(L);
    return 0;
}

```

Figure 1. Código Lua embarcado em C

Lua é uma linguagem de extensão projetada para ser usada como linguagem de configuração, acoplada a um programa hospedeiro (escrito em outra linguagem de programação), como mostrado na Figure 1. Aplicações em geral podem acoplar códigos em Lua, permitindo prototipagem rápida e acesso programável pelo usuário à tecnologia implementada pela aplicação.

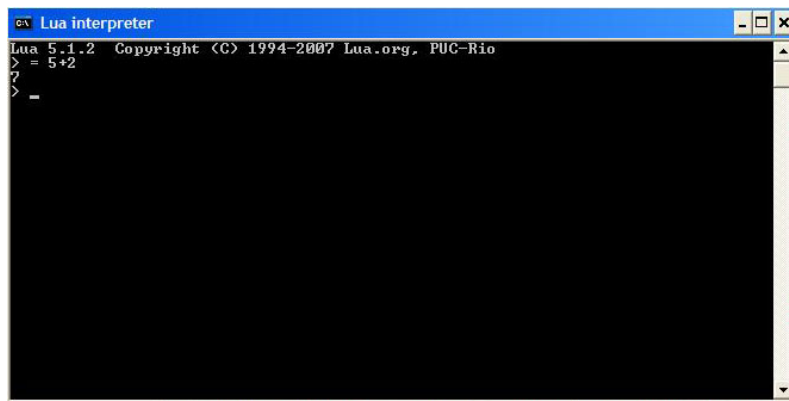


Figure 2. Interpretador stand-alone

No entanto é bastante comum o uso de Lua como um interpretador *stand-alone* (Figure 2). Nesse enfoque o código Lua é a linha de execução principal, o programador escreve todo o código da aplicação em Lua e não precisa ter conhecimento da API em C. O ambiente de programação é formado pelas funções pré-definidas de Lua, pelas bibliotecas padrão da linguagem e por eventuais pacotes de extensão adicionados.

Todos os comandos e construções de Lua são executados em um único ambiente global. Este ambiente, que guarda todas as variáveis globais e definições de funções, é automaticamente inicializado quando o interpretador é ativado e persiste enquanto o interpretador estiver em execução.

O ambiente global de Lua pode ser manipulado por códigos escritos em Lua ou por bibliotecas C que utilizem funções da interface C-Lua. Códigos Lua são compostos por comandos globais e definições de funções. Um módulo Lua é um arquivo ou uma cadeia de caracteres contendo códigos Lua. Quando se carrega um módulo Lua, os códigos são compilados para uma linguagem de máquina interna, para serem posteriormente executados. Os códigos globais (que não estão dentro de funções) são automaticamente executados ao fim da compilação do módulo. Os códigos de funções são armazenados e executados na chamada das respectivas funções. Em outras palavras, Lua usa uma estratégia híbrida de compilação e interpretação, que evita o custo de desempenho de interpretação textual direta, ao mesmo tempo em que mantém a versatilidade de um ambiente interpretativo [2].

Em linguagens de programação híbridas, o compilador tem o papel de converter o código fonte em um código chamado de *Bytecode* (código em bytes), que é um estágio intermediário entre o código-fonte (escrito numa linguagem de programação específica) e a aplicação final, como mostrado na Figure 3.

Para executar o *bytecode* utilizamos uma máquina virtual, que serve para interpretar e executar o *bytecode*.

As principais vantagens do *bytecode* são:

- Portabilidade (Independência de Sistema Operacional) - pois o *bytecode* roda na máquina virtual, e não no Sistema Operacional do equipamento.
- Não-necessidade do pré-processamento, típico dos compiladores.

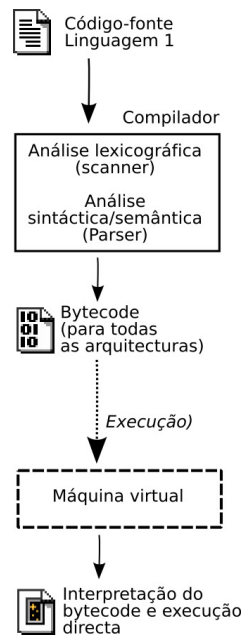


Figure 3. Estratégia híbrida usando bytecodes

O *bytecode* é encarado como um produto final, cuja validação da sintaxe e tipos de dados (entre outras funções dos compiladores) não será necessária [7].

Em Lua, não declaramos o tipo de uma variável. O tipo da variável é determinado dinamicamente, dependendo do valor que ela está armazenando. Sendo assim, uma variável pode armazenar valores de qualquer um dos tipos básicos de Lua. Existem oito tipos básicos em Lua: *nil*, *boolean*, *number*, *string*, *function*, *userdata*, *thread* e *table* [3].

Os mecanismos oferecidos para controle de fluxo pela linguagem são: *if*, *for*, *while* e *repeat*. Estes controladores agrupam diversos comandos em blocos que podem ou não ser executados uma ou mais vezes. Lua também permite a declaração de variáveis locais. Estas variáveis deixam de existir ao final do bloco onde foram declaradas.

A declaração de uma variável local pode ocorrer em qualquer lugar dentro de um bloco de comandos, e seu escopo termina quando termina o bloco no qual foi declarada. A declaração de uma variável local com mesmo nome de uma variável global obscurece temporariamente (i.e., dentro do bloco da declaração local) o acesso à variável global. Quando o programador escrever o nome da variável, estará se referindo à variável local.

As funções são tratadas como valores de primeira classe. Isto quer dizer que funções podem ser armazenadas em variáveis, podem ser passadas como parâmetros para outras funções e podem ser retornadas como resultados de funções. Quando definimos uma função em Lua, estamos armazenando na variável global cujo nome corresponde ao nome da função, o código “de máquina” interno da função, que pode posteriormente ser executado através de chamadas para a função. Funções em Lua podem ser definidas em qualquer lugar do ambiente global.

Tabelas em Lua são vetores associativos, isto é, vetores que podem ser indexados por valores numéricos, por cadeias de caracteres ou por qualquer valor dos demais tipos da linguagem. Vetores só não podem ser indexados pelo valor *nil*. Esta flexibilidade na

indexação de tabelas é a base do mecanismo existente em Lua para descrição de objetos.

Todos os erros que podem ser gerados em Lua são “bem comportados” e podem ser controlados pelo programador. Quando ocorre um erro na compilação ou execução de códigos Lua, a função cadastrada como *error method* é executada e a função da biblioteca Lua responsável pela execução do trecho de código é cancelada, retornando uma condição de erro [2].

Lua realiza gerenciamento automático da memória. Isto significa que você não precisa se preocupar com a alocação de memória para novos objetos nem com a liberação de memória quando os objetos não são mais necessários. Lua gerencia a memória automaticamente executando um coletor de lixo de tempos em tempos para coletar todos os objetos mortos (ou seja, objetos que não são mais acessíveis a partir de Lua). Toda memória usada por Lua está sujeita ao gerenciamento automático de memória: tabelas, userdata, funções, fluxos de execução, cadeias de caracteres, etc.

Lua implementa um coletor de lixo marca-e-limpa (*mark-and-sweep*) incremental. O coletor usa dois números para controlar o seu ciclo de coleta de lixo: a pausa do coletor de lixo e o multiplicador de passo do coletor de lixo. O valor de ambos é expresso de forma percentual (ou seja, um valor de 100 representa um valor interno de 1). A pausa do coletor de lixo controla quanto tempo o coletor espera antes de iniciar um novo ciclo. Valores maiores fazem o coletor ser menos agressivo. Valores menores do que 100 significam que o coletor não irá esperar para iniciar um novo ciclo. Um valor de 200 significa que o coletor irá esperar até que a memória total em uso dobre antes de iniciar um novo ciclo.

O multiplicador de passo controla a velocidade relativa do coletor em relação à alocação de memória. Valores maiores fazem o coletor ser mais agressivo mas também aumentam o tamanho de cada passo incremental. Valores menores do que 100 fazem com que o coletor seja muito lento e pode ocorrer que o coletor nunca termine um ciclo. O valor padrão, 200, significa que o coletor é executado a uma velocidade que é “duas vezes” a velocidade de alocação de memória.

É possível mudar estes números através de chamadas às funções `lua_gc` em C ou `collectgarbage` em Lua. Com estas funções você também pode controlar o coletor diretamente (e.g., pará-lo e reiniciá-lo) [6].

Em particular, Lua oferece um mecanismo de co-rotinas assimétrico, de primeira classe e com pilha. Nesta seção, vamos apresentar esse mecanismo e mostrar alguns de seus usos.

Co-rotinas, como implementadas por Lua, são bastante similares a linhas de execução (*threads*) cooperativas. Cada co-rotina em Lua representa uma linha de execução independente, com sua própria pilha de chamadas. Mas, ao contrário de um sistema *multithreading* convencional, não há preempção em um sistema de co-rotinas. Uma co-rotina só interrompe sua execução quando termina ou quando invoca explicitamente uma primitiva de suspensão (`yield`) [4].

3.1. Vantagens

A linguagem de programação Lua oferece uma série de vantagens aos programadores como:

- Robustez e estabilidade, sendo usada em muitas aplicações industriais, com ênfase em sistemas embutidos e jogos. Lua tem um sólido manual de referência e existem vários livros sobre a linguagem. Várias versões de Lua foram lançadas e usadas em aplicações reais desde a sua criação em 1993.
- Rapidez, podendo ser comprovado através de vários benchmarks que mostram ela como a linguagem mais rápida dentre as linguagens de script interpretadas.
- Portabilidade, ou seja, é distribuída via um pequeno pacote e compila sem modificações em todas as plataformas que têm um compilador ANSI/ISO C, rodando em diversas plataformas como Unix, Windows, BREW, Symbian, Pocket PC, Palm, Playstation II, ARM e Rabbit.
- Lua é embutível, ou seja, é uma engine rápida e pequena que você pode facilmente embutir na sua aplicação, permitindo uma integração forte com código escrito em outras linguagens. Lua é usada para estender programas escritos não só em C e C++, mas também em Java, C#, Smalltalk, Fortran, Ada, e mesmo outras linguagens de script, como Perl e Ruby.
- Lua é poderosa (e simples), um conceito fundamental no projeto de Lua é fornecer meta-mecanismos para a implementação de construções, em vez de fornecer uma multidão de construções diretamente na linguagem. Por exemplo, embora Lua não seja uma linguagem puramente orientada a objetos, ela fornece meta-mecanismos para a implementação de classes e herança. Os meta-mecanismos de Lua trazem uma economia de conceitos e mantêm a linguagem pequena, ao mesmo tempo que permitem que a semântica seja estendida de maneiras não convencionais.
- Lua é pequena, o pacote de Lua 5.1.4, contendo o código fonte, documentação e exemplos, ocupa 212K comprimido e 860K descompactado. O código-fonte contém cerca de 17000 linhas de C. No Linux, o interpretador Lua contendo todas as bibliotecas padrões de Lua ocupa 153K e a biblioteca Lua ocupa 203K.
- Lua é livre, software livre de código aberto, distribuída sob uma licença muito liberal (a conhecida licença MIT). Lua pode ser usada para quaisquer propósitos, incluindo propósitos comerciais, sem qualquer custo ou burocracia. Basta fazer um download e usá-la.
- Lua tem importância global: O projeto e a evolução de Lua foram apresentados em junho de 2007 na HOPL III, a 3ª Conferência da ACM sobre a História das Linguagens de Programação. Essa conferência ocorre a cada 15 anos (a primeira foi em 1978 e a segunda em 1993) e somente poucas linguagens são apresentadas a cada vez. A escolha de Lua para a HOPL III é um importante reconhecimento do seu impacto mundial. Lua é a única linguagem de programação de impacto desenvolvida fora do primeiro mundo, estando atualmente entre as 20 linguagens mais populares na Internet (segundo o índice TIOBE) [8].

3.2. Onde é usada?

Lua é usada em diversos projetos, softwares e games, veja abaixo alguns deles:

- Adobe Lightroom - software de gerenciamento de fotos;
- Gingga - middleware padrão brasileiro para TV digital;
- Wireshark - analisador de protocolos;
- Snort - *intrusion detection and prevention system*;
- Nmap - rastreador de redes para segurança;

- Eyeon's Digital Fusion - pós-produção de filmes;
- SimCity 4 - jogo da Maxis/Electronic Arts;
- TeCGraf: Laboratório de Pesquisa em Computação Gráfica;
- Grim Fandango e Escape from Monkey Island, dois jogos da LucasArts;
- Sistema de Pacotes RPM do Conectiva Linux [5].

Uma pesquisa realizada em setembro de 2003 pela gamedev.net-um importante site para programadores de jogos-revelou que a grande maioria dos jogos (72%) é desenvolvida com o auxílio de uma linguagem de script[1].

A mesma pesquisa mencionada acima revelou que Lua é atualmente a linguagem de script mais utilizada no desenvolvimento de jogos (20% dos jogos são desenvolvidos com Lua, enquanto somente 7% usam Python, a segunda linguagem de script mais citada na pesquisa). De fato, devido ao seu pequeno tamanho, bom desempenho, portabilidade e facilidade de integração, Lua tem sido amplamente utilizada na indústria de jogos. Empresas como LucasArts, BioWare, Microsoft, Relic Entertainment, Absolute Studios e Monkeystone Games desenvolvem jogos usando Lua [1].

4. Referências

[1] Celes, Waldemar; Figueiredo, Luiz Henrique de; Ierusalimschy, Roberto. A Linguagem Lua e suas Aplicações em Jogos. Disponível em WWW por <http://www.lua.org/doc/wjogos04.pdf>. Acessado em outubro/2009.

[2] Celes, Waldemar; Figueiredo, Luiz Henrique de; Ierusalimschy, Roberto. Noções de Lua 3.1. Disponível em WWW por <http://www.tecgraf.puc-rio.br/luafpt/nocoas-3.1.pdf>. Acessado em outubro/2009.

[3] Lua: Conceitos Básicos e API C. Disponível em WWW por http://www.keplerproject.org/docs/apostila_lua_2008.pdf. Acessado em outubro/2009.

[4] Ierusalimschy, Roberto. Uma Introdução à Programação em Lua. Disponível em WWW por <http://www.lua.org/doc/jai2009.pdf>. Acessado em outubro/2009.

[5] Ierusalimschy, Roberto. A Evolução de Lua. Disponível em WWW por <http://www.inf.puc-rio.br/roberto/talks/luapyconf.pdf>. Acessado em outubro/2009.

[6] Celes, Waldemar; Figueiredo, Luiz Henrique de; Ierusalimschy, Roberto. Manual de Referência de Lua 5.1 . Disponível em WWW por <http://www.lua.org/manual/5.1/pt/manual.html>. Acessado em outubro/2009.

[7] Introdução à Teoria dos Compiladores/Definições . Disponível em WWW por http://pt.wikiversity.org/wiki/Introdução_à_Teoria_dos_Compiladores/Definições. Acessado em outubro/2009.

[8] Celes, Waldemar; Figueiredo, Luiz Henrique de; Ierusalimschy, Roberto. A Linguagem de Programação. Disponível em WWW por <http://www.lua.org/portugues.html>. Acessado em outubro/2009.