Programski jezik Lua

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Prvi autor, drugi autor, treći autor, četvrti autor
kontakt email prvog, drugog, trećeg, četvrtog autora

26. mart 2019

Sažetak

U ovom tekstu je ukratko prikazana osnovna forma seminarskog rada. Obratite pažnju da je pored ove .pdf datoteke, u prilogu i odgovarajuća .tex datoteka, kao i .bib datoteka korišćena za generisanje literature. Na prvoj strani seminarskog rada su naslov, apstrakt i sadržaj, i to sve mora da stane na prvu stranu! Kako bi Vaš seminarski zadovoljio standarde i očekivanja, koristite uputstva i materijale sa predavanja na temu pisanja seminarskih radova. Ovo je samo šablon koji se odnosi na fizički izgled seminarskog rada (šablon koji *morate* da koristite!) kao i par tehničkih pomoćnih uputstava.

Sadržaj

1 Uvod

Kada budete predavali seminarski rad, imenujete datoteke tako da sadrže redni broj teme, temu seminarskog rada, kao i prezimena članova grupe. Precizna uputstva na temu imenovnja će biti data na formi za predaju seminarskog rada. Predaja seminarskih radova biće isključivo preko veb forme, a NE slanjem mejla. Link na formu će biti dat u okviru obaveštenja na strani kursa. Vodite računa da prilikom predavanja seminarskog rada predate samo one fajlove koji su neophodni za ponovno generisanje pdf datoteke. To znači da pomoćne fajlove, kao što su .log, .out, .blg, .toc, .aux i slično, **ne treba predavati**.

2 Nastanak i istorijski razvoj, mesto u razvojnom stablu, uticaji drugih programskih jezika

3 Osnovna namena i mogućnosti programskog jezika Lua

Iako je Lua prvenstveno razvijena za potrebe dva projekta, danas se, zbog svoje jednostavnosti, efikasnosti i portabilnosti, koristi u najrazličitijim oblastima: ugradnim sistemima, mobilnim uređajima, veb serverima i igricama.

Lua se obično koristi na jedan od sledeća tri načina: kao skript jezik u sastavu aplikacija pisanih na nekom drugom jeziku, kao samostalan jezik ili zajedno sa C-om.[2]

Ako se Lua upotrebljava u aplikaciji kao ugrađeni jezik, za njeno konfigurisanje u skladu sa domenom date aplikacije potrebno je koristiti Lua-C API. Pomoću njega se mogu, na primer, registrovati nove funkcije, praviti novi tipovi i vršiti izmene u ponašanja nekih operacija. Jedan od primera gde se Lua upotrebljava kao ugrađeni jezik je CGILua, alat za pravljenje dinamičkih veb stranica i manipulisanje podacima prikupljenih putem veb formi. Kao ugrađeni jezik, Lua je našla i široku primenu u igricama.

Sve češće, Lua se koristi i kao samostalan jezik, čak i za veće projekte. U te svrhe su razvijene biblioteke koje nude različite funkcionalnosti. Na primer, postoje biblioteke za rad sa stringovima, tabelama, fajlovima, modulima, itd.

Treća mogućnost za upotrebu Lue jeste u okviru programa pisanih u C-u. Lua se tada importuje kao C biblioteka. Programeri koji se opredele za ovakav način rada uglavnom najveći deo programa pišu u C-u, ali moraju dobro da poznaju i Luu kako bi kreirali jednostavne interfejse lake za upotrebu.

Lua je i tzv. jezik za spajanje (eng. glue language). Ona podržava razvoj softvera zasnovan na komponentama. U tom slučaju, aplikacija se kreira spajanjem postojećih komponenti višeg nivoa - komponenti koje su napisane u kompajliranom, statički tipiziranom jeziku kao što je C ili C++. One obično predstavljaju koncepte niskog nivoa koji se neće mnogo menjati tokom razvoja programa jer oduzimaju mnogo procesorskog vremena. Takve komponente se spajaju pomoću Lue. Dakle, Lua se koristi za pisanje onih delova koda koji će se verovatno menjati mnogo puta i na taj način ubrzava proces razvoja programa.

Kao i mnogi drugi jezici, Lua teži tome da bude fleksibilna. Ali, takođe teži tome da bude mali jezik - i u pogledu implementacije i u terminima specifikacije. Za ugrađne jezike ovo je veoma bitna osobina pošto se veoma često koriste u uređajima koji imaju ograničene hardverske resurse. Kako bi se postigla ova dva suprotstavljena cilja, dodavanju novih karakteristika u jezik se pristupa ekonomično. Zbog toga Lua koristi malo mehanizama. A pošto ih je malo, oni moraju biti efikasni. Neki od takvih mehanizama su, na primer, tabele (koje su u suštini asocijativni nizovi), funkcije prvog reda, korutine i refleksivne mogućnosti. Takođe, da bi jezik bio što manji, umesto hijerarhije numeričkih tipova (realni, racionalni, celi), Lua ima samo double floating-point (prevesti?) tip vrednosti.[3]

4 Najpoznatija okruženja (framework) za korišćenje ovog jezika i njihove karakteristike

5 Instalacija i uputstvo za pokretanje na Linux/Windows operativnim sistemima

6 Paradigme i koncepti

6.1 Tabele i objekti

Jedina struktura podataka koju Lua pruža je tabela. Tabele su dovoljno moćne da pomoću njih može da se implementira druga struktura podataka, kao sto su liste, redovi ili stekovi. Lua nije objektno-orjentisani jezik, te nema podršku za objekte. Medjutim, koristeći tabele i metatabele, sistem objekata i klasa može biti implementiran iz temelja.

6.1.1 Tabele

Tabele su first-class, dinamički kreirani asocijativni nizovi [An associative array is an array that can be indexed not only with numbers, but also with strings or any other value of the language, except nil.]. One su u osnovi rečnik ili niz. Tabele se sastoje iz parova ključ-vrednost. Ako su ključevi tabele numerički, tabela predstavlja niz. Ako su ključevi nenumeričke ili mešovite vrednosti, tabela je rečnik. Kao ključ u tabeli se može koristiti bilo šta, osim nil. Bilo Šta, uključujuci i nil, može biti vrednost. Tabele se skladište po referenci, a ne po vrednosti, i program jedino preko referenci (ili pokazivača) njima manipuliše. Nema skrivenih kopija ili kreiranja novih tabela iza scene.

6.1.2 Kreiranje tabela

Tabela se kreira putem konstruktora, koji u svojoj najjednostavnijoj formi izgleda ovako . Nakon što je tabela kreirana, ona treba biti dodeljena promenljivoj. Ako se tabela ne dodeli promenljivoj, nije moguće na nju referisati. Sledeći kod kreira novu tabelu i dodeljuje je promenljivoj a:

```
a = {}     -- kreira tabelu i smešta njenu referencu u promenljivu a
```

6.1.3 Skladištenje vrednosti

Tabela je relaciona struktura podataka koja skladisti vrednosti. Da bi se promenljiva sačuvala u tabeli, koristi se sledeća sintaksa:

```
table[key] = value
```

Naredni primer demonstrira kako napraviti tabelu, sačuvati vrednost sa ključem x, i kako vratiti (izvaditi) tu vrednost:

```
k = "x"
a[k] = 10        -- nov ulaz, sa ključem key="x" i vrednošću value=10
a[20] = "great"  -- nov ulaz, sa ključem key=20 i vrednošću value="great"
print(a["x"])    -- > 10
k = 20
print(a[k])      -- > "great" - vraća vrednost (izvadi vrednost iz tabele)
a["x"] = a["x"] + 1     -- uvećava ulaz "x"
print(a["x"])    -- > 11
```

Kada u programu ne postoji ni jedna referenca na neku tabelu, Lua menadzer (upravljač) memorije će (eventualno) obrisati tu tabelu (i osloboditi memoriju koju je zauzimala), tako da memorija koju je ta tabela zauzimala kasnije može biti ponovo upotrebljena. Ključ tabele moze biti bilo kog tipa (čak i druga tabela), osim nil, i svaka tabela može da čuva vrednosti različitih tipova indeksa. Razmotrimo sledeći primer:

```
a = {}     -- prazna tabela
-- kreira se 1000 novih ulaza
for i=1,1000 do a[i] = i*2 end
print(a[9])    --> 18
a["x"] = 10
print(a["x"])  --> 10
print(a["y"])  --> nil
```

Primetite poslednju liniju: Baš kao globalne promenljive, polja tabele dobijaju vrednost nil ako nisu inicijalizovana (ako ne dodelite vrednost ključu u tabeli, podrazumevana (defalut) vrednost je nil). Takođe kao kod globalnih promenljivih, polju tabele možete dodeliti vrednost nil ako želite da ga obrišete. Ako je ključ za tabelu tipa string, za pristup tabeli se može koristiti tačka sintaksa (eng. dot syntax). Lua podržava ovu reprezentaciju pružajući a.name kao lepši zapis za ["name"]. Dakle, poslednje tri linije prethodnog koda (primera) mogu se napisati i ovako:

```
    a.x = 10                  -- isto što i a["x"] = 10
    print(a.x)                -- isto što i print(a["x"])
    print(a.y)                -- isto što i print(a["y"])
```

6.2 Meta tabele i meta metodi

Meta tabele menjaju ponašanje tabela koristeći meta metode. Te meta metode su funkcije sa specifičnim imenom koje utiče na to kako se tabela ponaša. Na primer, koristeći meta tabele, možemo definisati kako Lua računa izraz a + b, gde su a i b tabele. Kad god Lua proba da sabere dve tabele, prvo proverava da li svaka od njih ima meta tabelu i da li ta meta tabela ima __add polje. Ako Lua pronadje to polje, zove odgovarajuću funkciju da bi izračunala sumu. Da bi se postavila ili promenila meta tabela bilo koje tabele koristi se funkcija setmetatable:

```
t1 = {}
    setmetatable(t, t1)
    assert(getmetatable(t) == t1)
```

Bilo koja tabela može biti meta tabela bilo koje druge tabele; grupa povezanih tabela može da deli zajedničku meta tabelu (koja opisuje njihovo zajedničko ponašanje); tabela može biti svoja sopstvena meta tabela (tako da opisuje svoje individualno ponašanje). Bilo koja od ovih konfiguracija je validna.

6.2.1 Kreiranje meta tabele

Za kreiranje meta tabele neophodno je prvo kreirati običnu tabelu nazvanu meta. Ova tabela će imati funkciju koja se zove __add[__add je rezervisano ime za funkciju]. __add funkcija će primati dva argumenta. Levi argument će biti tabela sa poljem koje se zove value, a desni argument će biti broj:

```
meta = { } -- kreira tabelu
meta.__add = function(left, right) -- dodaje meta metod
return left.value + right -- pretpostavlja se da je left tabela
end
```

Dalje, treba napraviti tabelu zvanu container. Container tabela će imati promenljivu zvanu value, koja će imati vrednost 5:

```
container = {
value = 5
}
```

Pokušajte da dodate broj 4 container tabeli; Lua će izbaciti sintaksnu grešku. Ovo je zato što ne možete da dodate broj tabeli. Kod koji uzrokuje grešku izgleda ovako:

```
result = container + 4 -- ERROR
print ("result: " .. result)
```

Dodavanjem meta tabele container tabeli, koja ima __add meta metod, mozemo napraviti da ovaj kod radi. Funkcija setmetatable se koristi da dodeli meta tabelu. Kod koji omogućava da sve ovo radi izgleda ovako:

```
setmetatable(container, meta) -- postavlja meta tabelu
result = container + 4 -- RADI!
print ("result: " .. result)
```

DOPUNITI... Meta tabele su možda najmoćnija karakteristika Lua-e.

6.3 Iteratori i zatvoranja

Iterator, konstrukcija koje omogćava prolazak kroz kolekciju, predstavlja se pomoću funkcije. Pri svakom pozivu ta funkcija vraća naredni element iz kolekcije. Na listingu 1 dat je primer iteratora nad listom koji vraća vrednost elemenata liste.

```
1000 function values (t)
       local i = 0
1002   return function () i = i + 1; return t[i] end
     end
1004
     t = {10, 20, 30}
1006 for element in values(t) do
       print(element)
1008 end
```

Listing 1: Primer iteratora nad listom

Između uzastopnih poziva, iterator mora da pamti stanje u kom se nalazi kako bi znao da nastavi dalje odatle. U tu svrhu se koriste zatvorenja (eng. closure). Zatvorenje je funkcija koja se nalazi unutar neke druge funkcije. Pri tome, unutrašnja funkcija može da pristupa promenljivama spoljašnje funkcije. Ta spoljašnja funkcija, koja se zove fabrika (eng. factory), zapravo pravi zatvorenje. [1]

U listingu 1 fabrika je values(). Pri svakom pozivu ona pravi zatvorenje (koje predstavlja sam iterator). To zatvorenje čuva svoje stanje u spoljašnjim promenljivama (t, i, n) tako da, svaki put kada se pozove, vraća naredni element iz liste t. Kada više nema vrednosti u listi, vraća nil.

7 Zaključak

Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.

Literatura

[1] Roberto Ierusalimschy. Lua.org, 2003-2004. on-line at: https://www.lua.org/pil/7.1.html.

[2] Roberto Ierusalimschy. *Programming in Lua*. Lua.org, Rio de Janeiro, 2006.

[3] Roberto Ierusalimschy. Programming with multiple paradigms in Lua. *Proceedings of the 18th International Workshop on Functional and (Constraint) Logic Programming*, page 5–13, 2009.

A Dodatak

Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe.