Funkcionálne a iné vlastnosti jazyka Lua

Ing. Peter Drahoš, PhD.

Obsah

- Krátke predstavenie jazyka a jeho histórie
- Predstavenie syntaxe jazyka a základ. typov
- Funkcionálne vlastnosti jazyka
- Meta-tabuľky a ich použitie
- Záver

Čo je Lua ?

- Lua znamená *mesiac* v Portugalčine
- Minimalistický, multiparadigm. prog. jazyk
- Základná filozofia je: menej je viac
- Skriptovací jazyk s rozšíriteľnou sémantikou
- Často vnorený jazyk vo väčších aplikáciách
- MIT Licencia, Open Source, Closed Devel.
- Malý, Rýchly, Jednoduchý, Výkonný
- Vhodný na opis a uchovávanie dát

Odkiaľ pochádza Lua ?

- Brazílsky vývojový tím (akademický)
  - prof. Waldemar Celes
  - prof. Roberto Ierusalimschy
  - Luiz Henrique de Figueiredo

- Vytvorená v 1993, Univerzita PUC-Rio
- Konkrétne grafická skupina Tecgraf
- Následník jazykov SOL a DEL, fy. Petrobras
- Mailing list od 1997

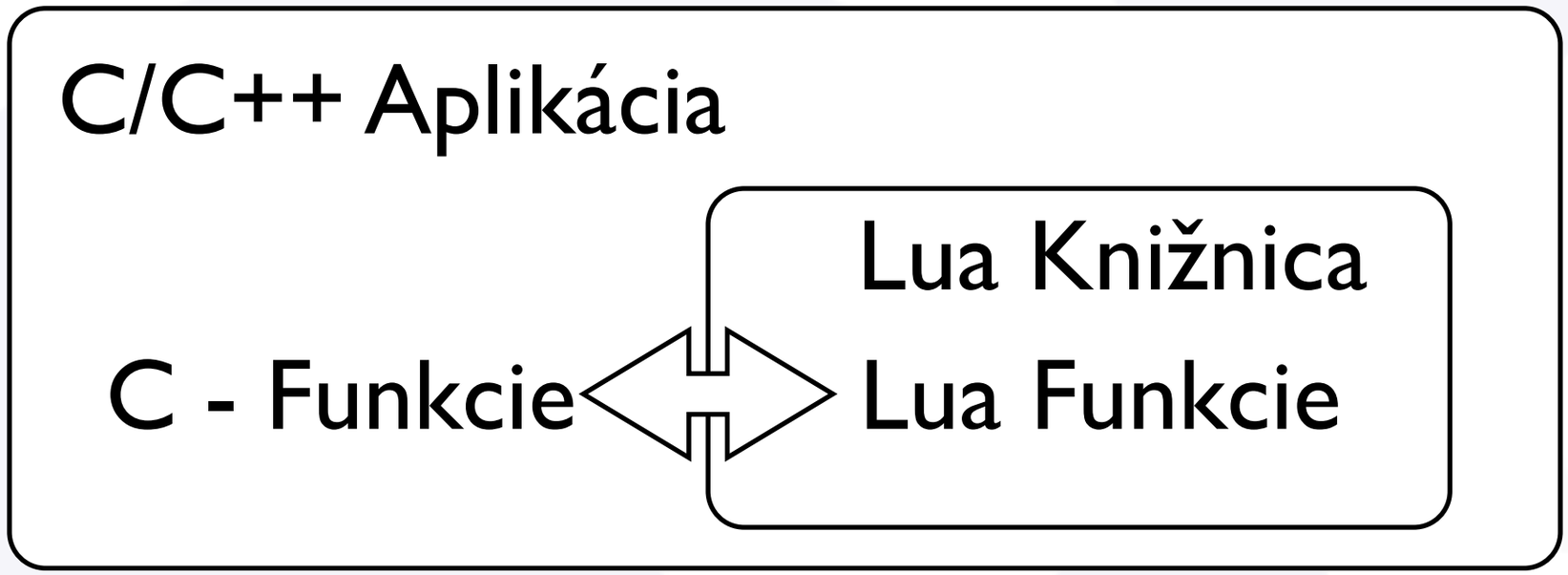
Prečo Lua ?

- Prenosnosť
  - Lua je C89 (ANSI C)
  - routre, mobilné app., mikropočítače, konzoly ...
- Rýchlosť
  - Lua je rýchla ako mnohé JIT jazyky. LuaJIT2 dosahuje rýchlosť C jazyka.
- Stabilita
  - Za celkovú históriu jazyka bolo identifikovaných len zopár chýb
- Veľkosť
  - Možno zvládnuť individuálne celý jazyk i s knižnicami

Ako malá ?

- Referenčný manuál má okolo 100 strán
- Gramatika opísatelná EBNF na 1 stranu
- Kód má zhruba 12500 riadkov v C
- Spustitelné súbory majú okolo 150kB
- Existuje len 8 typov
  - nil, boolean, number, string, **function**, userdata, thread, **table**
- Celkovo len 21 kľúčových slov
  - Žiadne definované funkcie
  - Dostupná len sada štandardných knižníc

Vnorený jazyk ?



- Lua je implementovaná ako C knižnica
- Možné previazať ju na existujúce C funkcie
- C funkcie môžu volať Lua funkcie
- Obdobne funguje vnorenie i do iných aplikácií

Kde sa používa ?

- Počítačové hry. Dominantný skript. jazyk
  - Wikipedia má pre ich zoznam samostatnú kategóriu
  - 1000ky iPhone a Android aplikácií (Corona SDK)
- Aplikácie
  - VLC, Apache, MySQL, PostgreSQL, nmap , Redis ...
- Firmware, SetTopBox, Routre
- Configuračný jazyk
- Dátové úložiská

Základy

- Volanie funkcie

```
print("Hello World!")
```

- Základné dátové typy

```
num1, num2 = 10, 3.1415

bool1, nil1 = true, nil

string1 = "Dáta môžu biť binárne"

array1 = { 1, "2", { 3.1 }, num1, string1}

table1 = { name = "Peter", surname = "Drahoš",
  address = { street = "...", city = "...", array1 }
}

print(num1*num2, string1, array1[2], table1["address"].street)
```

Funkcie a konštrukcie

- Faktoriál: 1, 1, 2, 6, 24, 120, 720, 5040 …
- Faktoriál, cyklus

```
function factorial(n)
        local a = 1
        for i = 1, n, 1 do
            a = a * i
        end
        return a
  end
print(factorial(7)) --> 5040
```

Funkcie a rekurzia

- Faktoriál, rekurzívne

```
function factorial(n)
    if n == 0 then
        return 1
    else
        return n * factorial(n - 1)
    end
end
print(factorial(7)) --> 5040
```

Lokálne Funkcie

- Faktoriál, iteratívne

```
function factorial(n)
    local function iter(prod, cnt)
        if cnt > n then
            return prod
        else
            return iter(cnt * prod, cnt + 1)
        end
    end
    return iter(1, 1)
end
print(factorial(7)) --> 5040
```

Funkcie ako premenné

- Všetky funkcie sú dáta, premenné určia názov

```
old_print = print
print = function(...)
  old_print(">>>",...,"<<<")
end
print("test") --> >>> test <<<
```

- Je možné ich ukladať do tabuliek

```
commands = { run = function() return "run" end,
  duck = function() return "duck" end,
  say = print,
}
commands.say("hello")
```

Funkcie z iných zdrojov

- Funkcie sú vo všeobecnosti vykonateľný kód

```
code_string = loadstring("print ('Hello String')")
code_file = loadfile(“hellofile.lua”)
code_string() --> Hello String
code_file() --> Hello File
```

- Funkcie ako atribút - lambda funkcie

```
function parrot( func, attr, times)
  for i = 1, times, 1 do
    func(attr)
  end
end
parrot ( print, "cracker", 3)
```

Iterátor

- Funkcie ako iterátor

```
function factorial() -- Konštruktor iterátora
  local cnt, prod = 1, 1
  return function() -- Nový iterátor
    prod = prod * cnt
    cnt = cnt + 1
    return prod
  end
end
for a in factorial() do -- Použi cyklus s iteračnou funkciou
  print (a)
  if a >= 5040 then break end
end
```

Klasické operácie

- Operácie typické pre Lisp možno v Lue replik.

```
-- foldr(function, default_value, table)

-- e.g: foldr(operator.mul, 1, {1,2,3,4,5}) -> 120

function foldr(func, val, tbl)
    for i,v in pairs(tbl) do
        val = func(val, v)
    end
    return val
end
```

- Iné typické funkcie

lua-users.org/wiki/FunctionalLibrary

Objekty ?

- Tabuľka + funkcie = OO v Lue

```
Account = {balance = 0}
function Account.withdraw (v)
  Account.balance = Account.balance - v
end
Account.withdraw(500)
```

- Nie je to ideálne ...

```
a = Account; Account = nil
a.withdraw(100.00)   -- ERROR!
function Account.withdraw (self, v)
  self.balance = self.balance - v
end
a = Account, a.withdraw(a, 100.00) -- lepšie
a:withdraw(100.00) -- Použitelné
```

Konštruktor objektov

- Dá sa to i bez *self*

```
function Account(bal) -- Konštruktor
  local balance = bal -- Private !
  local object = {} -- Proxy
  function object.withdraw (v)
    return balance = balance - v
  end
  return object
end
a = Account(100)
b = Account(500)
a.withdraw(10) -- 90
b.withdraw(20) -- 480
```

Meta-tabuľky

- Riadi prístup a prácu s tabuľkami

```
account1 = { balance = 10 }
account2 = { balance = 20 }
account3 = account1 + account2 -- ERROR !
```

- Definujeme operátor sčítania

```
mt = { __add = function(a,b) -- Definuje operátor +
    return { balance = a.balance + b.balance }
  end
}
setmetatable(account1, mt)
setmetatable(account2, mt)
account3 = account1 + account2
print(account3.balance) --> 30
```

Proxy tabuľka

- Chceme definovať obsah funkciou

```
factorials = {} -- Pole všetkých možných faktoriálov
mt = {
  __index = function(t, k)
        local ret = factor (k) -- Funkcia definujúca obsah
        t[k] = ret -- Cache
        return ret
  end
}
setmetatable(factorials, mt)
print(factorials[8]) --> 40320
```

Proxy tabuľka

- Práca s Databázou

```
persons = {} -- Prístup k osobám z SQL
mt = {
 __index = function(t, person)
        return sql_get_person_data(person) -- Vráti pole dát
    end
  __newindex = function(t,person, data)
        return sql_set_person_data(person, data) -- Nastaví dáta
    end
}
setmetatable(persons, mt)
print(persons.John.age, persons.Jane.address) -- Priamy prístup
persons.Jack = { age = 20, address = "New York" } -- Uchovanie
```

Ekosystém

- Implementácie jazyka
  - **Lua**, www.lua.org
  - **LuaJIT**, www.luajit.org
- Distribúcie
  - **LuaRocks**, www.luarocks.org
  - **LuaDist**, www.luadist.org
  - **Lua For Windows**, code.google.com/p/luaforwindows
- Knihy
  - **FREE Programming in Lua 1st ed.**, www.lua.org/pil
  - **Programming in Lua 2-4th ed**., www.lua.org/pil

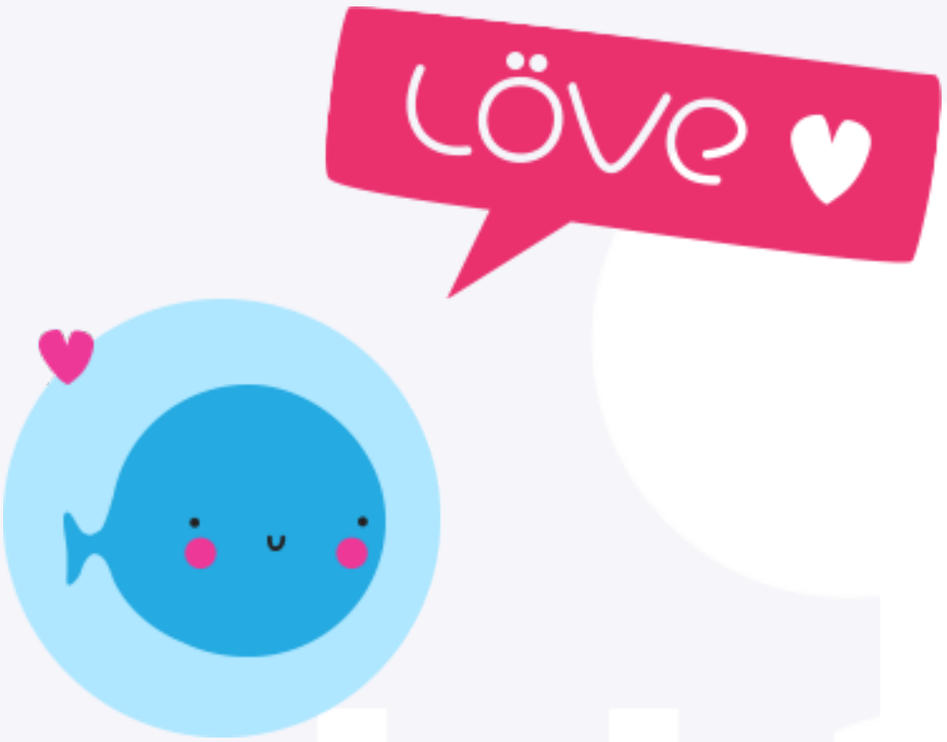
Výhody

- Jeden z najrýchlejších jazykov
- Jeden z najmenších jazykov
- Podporuje bytecode, dá sa vynechať parser
- Veľmy stabilný kód, skoro bez chýb
- Dostupnosť kódu a dokumentácie
- Dá sa individuálne úplne zvládnuť
- Vynikajúca a odborná komunita

Nevýhody

- Nie je to dominantný samostatný jazyk
- Štandardne nemá žiadne rozširujúce knižnice
- Nie je zaručená spätná kompatibilita
- Neobsahuje vlákna, len korutiny
- Tažko sa debugujú veľké aplikácie

Otázky ?



love2d.org

[lua.org](lua.org)