Lua

Lua — Brazylijski księżyc

Łyk historii

- Bardzo dawno temu powstał język DEL dla PETROBRAS

```
:e      gasket              "gasket properties"
mat     s                    # material
m       f       0            # factor m
y       f       0            # settlement stress
:p
gasket.m>30
gasket.m<3000
gasket.y>335.8
gasket.y<2576.8
```

- W 1993 roku rosnące wymagania dały początek językowi Sol

```
type @track { x:number,y:number= 23, z=0}
type @line { t:@track=@track{x=8},z:number*}

t1 = @track { y = 9, x = 10, z="hi!"}
l = @line { t= @track{x=t1.y, y=t1.x}, z=[2,3,4] }
```

Początki kolejnego języka

- Rosnące wymagania dotyczące języka postawiło w roku 1993 twórców przed dylematem: Tcl? Forth? Perl?
- Nie są to języki opisu danych, nie były także projektowane jako języki rozszerzeń.
- Twórcy decydują sie na utworzenie języka Lua. O jego prostocie decydują:
  - Składnia przepływu zaczerpnięta z Moduli
  - Dynamiczne typowanie
  - Prosty system typów (7 typów) liczba, napis, słownik, nil, userdata (wskaźnik na dane przekazywane z C), funkcja Lua, funkcja C

Zastosowanie

- 1997 - Lua użyta w Grim Fandango z LucasArts.
- Przemysł „rozrywkowy” docenia język lua. Baldur's Gate, MDK2, Escape from Monkey Island, Homeworld 2, Far Cry to tylko niektóre tytuły.
- Napisana w Ansi C, używana w komercyjnych projektach na większości platform: i386, PPC, Playstation 2
- Smithsonian Astrophysical Observatory, CPC4400, DigiTest...
- Crazy Ivan wygrał RoboCup 2000 i 2001

Lua dzisiaj

Lua dzisiaj

(5.1 alpha dostępna od 1 października 2005)

Składnia

```
a = {} -- konstruktor słownika

c = { imie = "Jan", nazwisko = "Kowalski", wiek = 23 }

a.add = function (a, b)  -- function a.add (a, b)
  return a + b           --   return a + b
end                      -- end

a.b = 2                  -- a["b"] = 2

x, y = y, x      -- zamienia wartości x  i y

_G = _G._G       -- _G.a -- równoważne -- a
```

A co z OOP

- Cały system obiektowy byłby zbyt duży na potrzeby języka
- Udostępnia mechanizmy pozwalające łatwo go wyrazić

```
a = {money = 0}
function a:f(i) -- cukier dla -- function a.f(self,i)
self.money = self.money + i
end

a:f(2) -- cukier dla -- a.f(a,2)
```

- Dziedziczenie jest realizowane przez użycie metaslownikow

```
setmetatable(a, {__index = b})
```

Odczytanie nieistniejącego pola obiektu *a* spowoduje odczytanie pola obiektu *b*

Dzięki metatabelkom potrafimy dużo więcej!

- Przedefiniowanie operatorow
  - Arytmetycznych `__add`, `__sub`, `__mul`, `__div`, `__pow`, `__unm`
  - Porzadkowych `__eq`, `__lt`, `__le`
  - Sklejania `__concat`
  - Dostepu do tabelki `__index`, `__newindex`
  - Zwalniania zasobow `__gc`
- Pola prywante, publiczne, transakcje — wszystko nienajbrzydszą składnią
- Metatableke możemy również przypiąć do userdata, „ucząc” tym samym Lue jak ma postępować z obiektami przekazanymi z kodu natywnego

I to dzięki metatabelkom definiujemy interfejs .NET

- Lua.NET to projekt sponsorowany przez Microsoft Research. W jego skład wchodzą dwa podprojekty
  - Lua2IL — rekompilator z bajtokodu Luy na bibliotekę dynamiczną IL
  - LuaInterface — P/Invoke korzysta z oryginalnej implementacji Luy
- Nie implementują kompilatora
- Kod wygenerowany przez Lua2IL jest trzykrotnie szybszy od kodu Microsoft JScript.NET (bez optymalizacji typów)
- LuInterface jest lepiej udokumentowana, dlatego poniższe slajdy są na jej podstawie

LuaInterface

```
using LuaInterface;

//Tworzymy instancje interpretera Luy
Lua l = new Lua();
//otwieramy kilka standardowych bibliotek
l.OpenBaseLib();
l.OpenIOLib();
l.OpenStringLib();
l.OpenMathLib();
l.OpenTableLib();
l.OpenDebugLib();
l.OpenLoadLib();

l.DoFile("plik.lua");  //  wykonuje plik
l.DoString("return 1"); // wywołuje kawałek kodu
```

LuaInterface — podstawy

- Instancja l zachowuje sie tak jak _G

```
l["a"] = 3;        // a=3
l.NewTable["t"];   // t={}
l["t.a"] = "a";    // t.a = a // t["a"] = "a"
```

- Dostęp do typów Luy

```
Console.WriteLine(((LuaTable)l["Person"])["Name"]);
Console.WriteLine(l["Person.Name"]);
```

- Przypisanie instancji klasy .NET

```
l["obj"] = new StringBuilder()
```

LuaInterface — Rozszerzanie o CLR

- Rejestrowanie metod

```
l.RegisterFunction("stat",null,typeof(Test).GetMethod("stat")); // statycz
l.RegisterFunction("fun",a,typeof(Test).GetMethod("fun")); // Test.fun
```

- i wywoływanie

```
l.DoString("return f(a)");
l.GetFunction("f").Call("ala");
```

zwracają one `object[ ]` zawierająca wyniki wywołań funkcji

luanet

- Dostep do CLR z poziomu luy został zaimplementowany przy użyciu typu userdata (obecnego w Lua od samego jej początku).
- Do danych użytkownika dołączone są metatabelki pozwalające na dość wygodny (jak dla języka i implementacji która nie ma pojęcia że współpracuje z .NET) sposób interakcji z obiektami
- Aby użyć Assembly, należy je załadować

```
require("luanet")
luanet.load_assembly("System.Windows.Forms")
luanet.load_assembly("System.Drawing")
```

- Lua nie dołącza wszystkich typów (pamiętajmy, ze pojęcie typu obiektu w Lua jest dość ulotne... to porostu archetyp obiektu), musimy dodać je sobie explicite

```
Form = luanet.import_type("System.Windows.Forms.Form")
Point = luanet.import_type("System.Drawing.Point")
```

luanet — iterakcja z CLR

- Tworzenie obiektów wygląda juz znajomo

```
mainForm = Form()
```

- Używamy metod i pól

```
mainForm:ShowDialog()
button1.Text = "OK"
```

- Dodajemy zdarzenia

```
function handle_mouseup(sender,args)
  print(sender:ToString() .. "MouseUp!")
  button.MouseUp:Remove(handler)
end
handler = button.MouseUp:Add(handle_mouseup)
```

Drobne problemy z typami

- Lua dość swobodnie konwertuje liczby i łańcuchy. Majac

```
-- 1. void SomeType(int,T1)
-- 2. void SomeType(int,T2)

-- Nie ma sposobu na wywołanie 2 deklaracji, wszystko to userdata
a = T2()
o = SomeType(1,a) -- wywoła new SomeType(int,(T1)a)
```

- Należy jawnie wyłuskać odpowiadający nam konstruktor czy metodę

```
Int32 = luanet.import_type("System.Int32")

SomeTypeT2 = get_constructor_bysig(SomeType, Int32, a:GetType())
SomeMethod = luanet.get_method_bysig(obj, "SomeMethod", a:GetType())
```

A co z parametrami *ref* i *out*

- Lua przekazuje typy proste przez wartość, nie ma więc out i ref, dlatego wywołania wyglądają następująco

```
-- wywołując int obj::OutMethod1(int,out int,
retVal, out1, out2 = obj:OutMethod1(inVal)

-- wywołując void obj::OutMethod2(int,out int)
retVal, out1 = obj:OutMethod2(inVal) -- retVal bedzie nilem

-- wywołując int obj::RefMethod(int,ref int)
retVal, ref1 = obj:RefMethod(inVal,ref1)
```

Ostatnio chcieliśmy zdążyć opowiedzieć o:

- SML / NJ, SML.NET,
- OCaml, OCamlIL, F#,
- Nemerle,

a tym razem zdążyć opowiedzieć o:

- Nemerle cd.
- Ada, A#,
- Boo,
- Lua.

Podziękowania

Mamy nadzieję, że nasze seminarium przybliżyło idee „Jedna platforma — wiele języków”.
Dziękujemy za uwagę!