and

the only thing missing is 'u'

Integrating Lua and Erlang with an overview of scripting options

Chad DePue

- Erlang/Ruby on Rails Consulting
- Buenos Aires, Argentina
- ErlangInside.com*
- twitter: @rubyrescue

inakanetworks

* Incidentally, I'm looking for collaborators, contact me if you're interested - chad@inakanetworks.com

What are we going to cover?

I **Embedded Scripting Languages**

II **Overview of the Lua Language**

III **Using Lua in Erlang**

IV **Erlmon, a system monitoring tool that uses Lua**

1 : Embedded Scripting Languages

Popular Languages

Language	Description
Javascript	Ubiquitous
Reia	Ruby+Python/Erlang
Lua	Fast, lightweight; popular with gamers

Popular Projects

Language	Project	Description	Author	License
Javascript	erlang_js	Linked-in driver, using SpiderMonkey, provides OTP interface; used in Riak	Basho/Kevin Smith	Apache
Reia	reia	Ruby-like language built on top of Erlang	Tony Arcieri	MIT
Lua	erlua	Erlang Port, provides integration with Lua.	Cyril Roman	GPLv3
Lua	erl-lua	Linked-in driver,can crash the Erlang runtime	Ray Morgan	None, public domain?

flowchart

Which one should I use?

If you...	you should use	because...
interact with C	Lua	it's designed to do this.
want to reuse node.js scripts	Javascript	it's written in JS
value safety over speed	Lua, via erlua	it's not a linked-in driver
want to stay in the Erlang VM	Reia	it's all Erlang baby

II : Lua Overview

Lua is a powerful, fast, lightweight, embeddable scripting language, often used for configuration, very popular in the gaming communities.

Things I would say about Lua In an Elevator With a Geek...

- Simple, Ruby-like syntax
- Designed to be embedded
- Stack-based interface to the host
- Compiles cleanly nearly everywhere - ANSI-C
- Tables are the key to Lua's power
- First class functions allow closures, coroutines
- The only real oddity is that indices start at **1**

or, you can read the poster...

Builds in all platforms with an **ANSI/ISO C** compiler

Fits into **128K ROM**, **64K RAM** per interpreter state[1]

Fastest in the realm of interpreted languages

Well-documented **C/C++ API** to extend applications

One of the fastest mechanisms for **call-out to C**

Incremental **low-latency garbage collector**

Sandboxing for restricted access to resources

Meta-mechanisms for language extensions,
e.g. class-based **object orientation** and inheritance

Natural datatype can be integer, float or double

Supports **closures** and cooperative **threads**

Open source under the **OSI-certified** MIT license

[1] Complete Lua SOC, practical applications in 256K ROM / 64K RAM

Designed, implemented and maintained at the
Pontifical Catholic University of Rio de Janeiro

www.lua.org

Lua Types

- nil
- boolean
- number
- string
- table
- function
- userdata
- thread

Lua Types

- Variables are typeless
- Only values have types
- All values are ‘first class values’
- Numbers are always floats

Tables

"Sets" and "Arrays" are both implemented via Tables. Most similar to Ruby Hashes, but fields addressable as methods on the table.

```
t = {}

t = { a=7;b=print;c="yahtzee" }

t.a     --   7

t['c'] -- "yahtzee"

print(type(t.b)) -- "function"
```

Functions

- Functions are first-class
- Can be anonymous
- Can take a variable number of parameters

Tables and Functions Together

```
1
2 function create_message(from,to,body)
3     local message= {from=from;to=to;body=body}
4     message.send = function()
5       print("Sending from " .. message.from .. " to " .. message.to)
6     end
7     return message
8 end
9
```

```
m = create_message(“chad@erlanginside.com”,
                   “chad@rubyrescue.com”,
                   “you have too many emails”)
> m.send()
  Sending from chad@erlanginside.com to chad@rubyrescue.com
```

Closures

- Functions in Lua have full access to the variables in the enclosing function, which allows use of functions as closures.
- These *external local variables* are called in Lua-world “upvalues”.

Closures

```
14 function newCounter ()
15   local i = 0
16   return function ()   -- anonymous function
17     i = i + 1
18     return i
19   end
20 end
21
22 c1 = newCounter()
23 print(c1())  -- 1
24 print(c1())  -- 2
25
```

Easy C interop : C->Lua

```
2  void remove_blanks (char *s)
3  {
4    lua_pushstring(s);  /* prepare parameter */
5
6    /* call Lua function */
7    lua_call("remove_blanks");
8
9    /* copy result back to 's' */
10   strcpy(s, lua_getstring(lua_getresult(1)));
11 }
12
```

Easy C interop: Lua-> C

C header

```
12
13 typedef int (*lua_CFunction) (lua_State *L);
14
```

C source

```
12
13 static int l_sin (lua_State *L) {
14     double d = lua_tonumber(L, 1);  /* get argument */
15     lua_pushnumber(L, sin(d));  /* push result */
16     return 1;  /* number of results */
17   }
18
```

Register Lua function in C

```
28 lua_pushcfunction(l, l_sin);
29 lua_setglobal(l, "mysin");
30
```

Metatables

```
 8 function _add_host(name)
 9   local host = {}
10   local mt = {}
11   mt.__index = function (table, key)
12     return Erlmon[key]
13   end
14   mt.__newindex = function (table, key)
15     return Erlmon[key]
16   end
17   setmetatable(host,mt)
18   Erlmon.hosts[name] = host
19   return host
20 end
```

Other Lua Features

- Tail-calls
- Coroutines

Quiz

- Why is Lua called “Lua”?
- What language is Lua’s biggest influence?

- SOL

- Scheme

III : Lua in Erlang

Things I would say about embedding Lua in Erlang in an Elevator with a geek...

- Lua is really fast
- Lua is designed to be embedded
- State is easy to keep in a process
- Tables can easily be converted to lists of terms

erl-lua

- Written by Ray Morgan, erl-lua was the starting point for my investigations into Lua interop
- Rewritten by Darrik Mazey - fork available on github
- Currently used in erlmon
- Linked-in driver

erl-lua limitations

- Currently can't call from Lua into Erlang
- Because it's a linked-in driver, you can crash the VM if you write bad code

erl-lua:getting Lua state

```
1> {ok, State} = lua:new_state()
```

```
{ok,{lua,#Port<0.1126>}}
```

erl-lua:using the stack

```
2> lua:dostring(State,"function myadd(a,b)
return a+b;end").
3> lua:getfield(State, global, "myadd").
4> lua:pushnumber(State,1).
5> lua:pushnumber(State,2).
6> lua:call(State,2,1).
```

erl-lua:using the stack

```
7> lua:pop(State)
```

{ok,number,3}

erl-lua:accessing a table

```
1> {ok, State} = lua:new_state(),
2> lua:dostring(State,”foo={bar = 1}”),
3> lua:gettable(State,global,foo).
```

returns

[{"bar",1}]

erlua (no dash, one “L”)

- Written by Cyril Roman
- License is GPL so we decided not to use for erlmon
- Port, so safer for use - a Lua error won’t take down the Erlang VM.

erlua limitations

- Only one lua state
- Slower because it's a gen_server and a Port
- GPL

erlua: gen_server

```
2> erlua:start().
3> erlua:set(“foo”,“bar”).
4> Bar = erlua:get(“foo”).
 “bar”.
```

erlua:checking syntax

```
2> erlua:start().
3> erlua:check_syntax("math.pow(3,2)").
   []
4> erlua:check_syntax("foo=").
   {syntax_error, Where}
```

IV: Erlmon

Erlmon

- System monitoring tool written in Erlang and Lua.
- Designed to perform basic system, process, file, and port monitoring, similar to monit or god.

Erlmon

Designed For...

- Systems in different geographies.
- Dynamic server volume.
- Existing Erlang environments.
- Ruby environment dependency.

...Because...

- We wanted a "turing complete" (like god) monitor with distributed monitoring (like m/monit).
- We wanted non-developers to be comfortable.
- We wanted to build something really robust.

How Erlmon works

- All monitoring code is in Erlang
- Configuration file is in Lua.
- Connected nodes share the same configuration file.

How Erlmon works

- Nodes monitor each other.
- Connected nodes share the same configuration file.
- Nodes monitor each other's health, and status of the system is available via any connected node.

How Erlmon uses Lua

- One of the specific applications for Lua listed by the language creators is as a ‘configuration language’.
- Extensive use of Lua closures.
- Ability to write event callbacks in Lua when a monitoring event occurs.

hooking up methods in Lua

```
 7
 8 function _add_monitors(host_monitors)
 9
10   mlist = {}
11   host_monitors.list = mlist
12
13   host_monitors.add = function(mtype,name,init)
14     -- create table if this is the first monitor of this type
15     if mlist[mtype] == nil then mlist[mtype] = {} end
16
17     -- you don't have to name your monitors
18     -- but we should probably hash them
19     if name == nil then
20       table.insert(mlist[mtype],init)
21     else
22       mlist[mtype][name] = init
23     end
24
25     return init
26   end
```

adding a monitor in Lua

```
29 memcached = monitor_port(11211)
30 -- or
31 memcached = Erlmon.monitors.monitor_port(11211)
32
33 memcached.start = "/etc/init.d/memcached start"
34 memcached.stop = "/etc/init.d/memcached stop"
35
```

becomes...

```
[{"monitor_port",function},
    {"remove",function},
      {"add",function},
        {"list",
          [{"tcp_port",
              [{1,
        [{"host","localhost"},
            {"port",11211},
{"name","memcached-11211"},
   {"start","memcached"},
    {"restart_grace",10},
     {"start_grace",10},
       {"stop","killall
 memcached"}]}]}]}]
```

gen_server holds Lua state

```
65
66 init([]) ->
67   {ok,State} = lua:new_state(),
68   {ok, #config_state{lua_state=State}}.
69
```

reload config

```
72
73 %% reload config
74 handle_call({reload,_ReloadType}, _From, _State) ->
75   debug:log("CONFIG: loading lua file"),
76   {ok, L} = lua:new_state(),
77   NewState = #config_state{lua_state=L},
78   Reply = lua:dofile(L,?CONFIG_FILE),
79   case Reply of
80     {error,_} ->
81       {reply, Reply, _State};
82     _ ->
83
84     Monitors = config:setting([monitors,list]),
85     apply_config_list(Monitors),
86     debug:log("CONFIG: reloaded"),
87     {reply, Reply, NewState}
88   end;
89
```

config:setting

```
44
45 setting(Type) when is_atom(Type) ->
46   setting([Type]);
47
48 setting(Types) ->
49   Settings = gen_server:call(config,erlmon),
50   find_setting(Settings,Types).
51
52 find_setting(Settings,[Type|Types]) ->
53   Result = lists:keyfind(atom_to_list(Type),1,Settings),
54   case Result of
55     false ->
56       Result;
57     {_Name,NewSettings} -> find_setting(NewSettings,Types)
58   end;
59
60 find_setting(Settings,[]) ->
61   Settings.
```

So...what about monitoring tool X?

	monit	m/monit	god	nagios	xymon
good at	low memory, reliable, easy to configure	start/stop services in one place	turing complete, ruby flavored syntax	huge community, lots of plugins	not being user friendly
not so good at	multiple hosts	helpful status ui	monitoring its own health	config is complicated	being user friendly

arbitrarily not listed tools include commercial systems like HP/Openview, Tivoli, Big Brother

and what about erlmon?

	erlmon	erlmon	erlmon	erlmon	erlmon
good at	being hard to build	having lots of open workitems	teaching us a lot about linked_in drivers	coexisting with other erlang apps	an open source project using lua and nitrogen
not so good at	installing easily	having helpful status ui	having users	advanced notification like SNMP	i'm sure we can think of more things

More Information

- Web UI built in Nitrogen
- Config handled by modified fork of erl-lua
- Would like to use erlua because it's a Port vs Linked-In Driver, but erlua is GPL
- Currently in production use.
- More information at http://github.com/darrikmazey/erlmon.com

Demo

Next Steps

- erl_lua : rewrite linked-in driver as nib?
- erlmon : additional lua support - goal is ‘god-like powers’
- erlmon: ‘pure lua’ callback and monitors
- erlmon: lots of bugs in config

Resources

- erlua - http://gitorious.org/erlua
- erl-lua - http://github.com/darrikmazey/erl-lua
- Lua Poster by Timm Müller - http://www.schulze-mueller.de/download/lua-poster-090207.pdf
- Lua Manual - http://www.lua.org/manual/5.1
- erlmon - http://github.com/darrikmazey/erlmon
- erlang_js - http://bitbucket.org/basho/erlang_js/
- Go get the “Blue PiL (Programming in Lua, Second Edition)”

Questions?

lua:push(L,”exit”).