CLOUDFLARE

Lua: the world's most infuriating language

October 17, 2013

The simplest way to a safer, faster and smarter website

My background

- Lots of...
  - Assembly
  - C, C++ and Go
  - Perl, Tcl
  - LISP (and relations)
- And lately...
  - Lua

CHDK on Canon A560

CHDK Lua Interface

- Lua's extensibility means it can control the camera
- http://chdk.wikia.com/wiki/Lua
- Functions added to Lua like
  - `shoot()` – takes a picture
  - `press()`, `release()` – press the shutter button (allows a partial press)
  - `get_orientation_sensor()` – figure out camera orientation
  - `get_temperature()` – get camera internal temperatures

```
-- Now enter a self-check of the manual mode settings

log( "Self-check started" )

assert_prop( 49, -32764, "Not in manual mode" )
assert_prop(  5,      0, "AF Assist Beam should be Off" )
assert_prop(  6,      0, "Focus Mode should be Normal" )
assert_prop(  8,      0, "AiAF Mode should be On" )
assert_prop( 21,      0, "Auto Rotate should be Off" )
assert_prop( 29,      0, "Bracket Mode should be None" )
assert_prop( 57,      0, "Picture Mode should be Superfine" )
assert_prop( 66,      0, "Date Stamp should be Off" )
assert_prop( 95,      0, "Digital Zoom should be None" )
assert_prop( 102,      0, "Drive Mode should be Single" )
assert_prop( 133,      0, "Manual Focus Mode should be Off" )
assert_prop( 143,      2, "Flash Mode should be Off" )
assert_prop( 149,    100, "ISO Mode should be 100" )
assert_prop( 218,      0, "Picture Size should be L" )
assert_prop( 268,      0, "White Balance Mode should be Auto" )
assert_gt( get_time("Y"), 2009, "Unexpected year" )
assert_gt( get_time("h"), 6, "Hour appears too early" )
assert_lt( get_time("h"), 20, "Hour appears too late" )
assert_gt( get_vbatt(), 3000, "Batteries seem low" )
assert_gt( get_jpg_count(), ns, "Insufficient card space" )

log( "Self-check complete" )
```

https://github.com/jgrahamc/gaga/tree/master/gaga-1/camera

```
if ( ok == 1 ) then
  sleep(s)
  log( "Starting picture capture" )

  n = 0

  while ( 1 )  do
    tc = c
    while ( tc > 0 ) do
      shoot()
      n = n + 1
      log( string.format("Picture %i taken", n ))
      tc = tc - 1
    end
    log( string.format("Temperatures: %i, %i, %i",
        get_temperature(0), get_temperature(1), get_temperature(2) ))
    log( string.format("Battery level %i", get_vbatt()))
    sleep(i)
  end
end

log( "Done" )
```

Initial Irritants

- Seemed awfully verbose (bit like BASIC!)
  - `if... then... else... end`
- Little shortcuts were missing
  - `x += 1`
  - `p?a:b`
- Made it feel like a toy language

Irritant: +=

- Happy not to have `x++`
- But why no `+=`?

Irritant: ternary operator

- Super useful to be able to do

```
local bird = duck?"It's a duck":"Not a duck"
```

- Can't.
- Solution is the ugly

```
local bird = duck and "It's a duck" or "Not a duck"
```

- And it's a bad solution

```
local thing = question and returnsfalse() or crash()
```

Irritant: `not` and `~=`

- Not equals is `~=`
  - Which Perl programmers will confuse with `=~`

```
if x ~= 5 then print "Not five" end
```

- But not is not `~` it's `not`!

✖
```
if not x == 5 then print "Not five" end
```

✖
```
if ~(x == 5) then print "Not five" end
```

```
if not (x == 5) then print "Not five" end
```

Next Lua Program

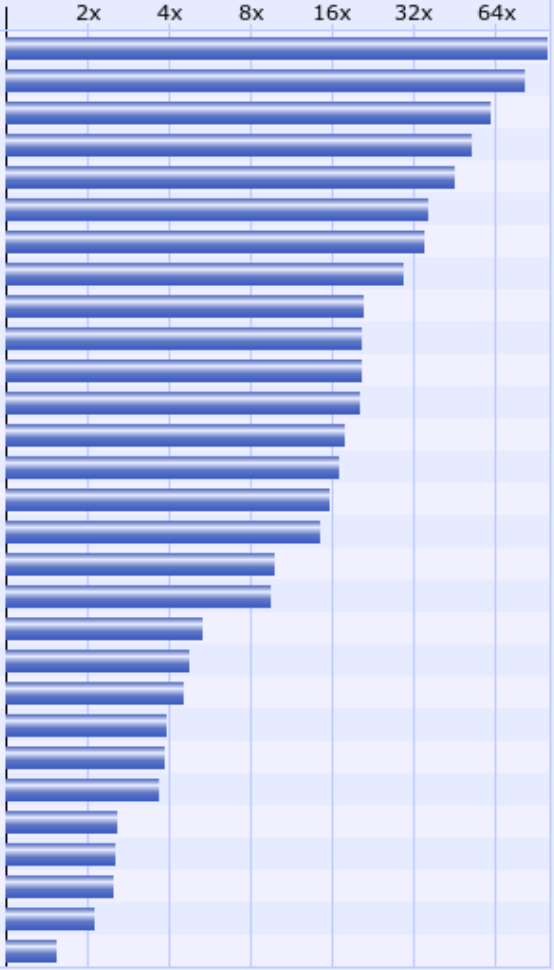
- Web Application Firewall
- About 2,000 lines of Lua
  - Replaced 37,000 line C program!
  - Plus 3,000 of automatically generated Lua
- Used by CloudFlare to process HTTP requests
  - Checks for XSS, CSRF, SQL injection
  - Bad browsers
  - Custom rules
- Turns out Lua was great for this!

Really, really fast: 1 to 2ms per request

CLOUDFLARE

LuaJIT

Benchmark▼	N	Ratio▼
md5	20000	**112.09**
array3d	300	**84.00**
euler14-bit	2e7	**62.02**
mandelbrot-bit	5000	**53.57**
scimark-lu	5000	**45.66**
scimark-fft	50000	**36.84**
scimark-sor	50000	**35.66**
nsieve-bit	12	**29.60**
spectral-norm	3000	**21.09**
fannkuch	11	**20.94**
ray	9	**20.89**
nbody	5e6	**20.43**
mandelbrot	5000	**18.10**
recursive-ack	10	**17.24**
scimark-sparse	15e4	**15.97**
recursive-fib	40	**14.60**
pidigits-nogmp	5000	**10.02**
nsieve-bit-fp	12	**9.56**
k-nucleotide	5e6	**5.38**
binary-trees	16	**4.79**
fasta	25e6	**4.60**
nsieve	12	**3.97**
coroutine-ring	2e7	**3.89**
partialsums	1e7	**3.73**
revcomp	5e6	**2.61**
chameneos	1e7	**2.55**
life		**2.51**
series	10000	**2.16**
sum-file	5000	**1.56**



LuaJIT FFI

```lua
local ffi = require 'ffi'
local C = ffi.C

ffi.cdef[[
  typedef long time_t;

  typedef struct timeval {
    time_t tv_sec;
    time_t tv_usec;
  } timeval;

  int gettimeofday(struct timeval* t, void* tzp);
]]

local gettimeofday_struct = ffi.new("timeval")

-- _gettimeofday_: wrapper function that calls the C gettimeofday
-- function via FFI and returns a value in microseconds
local function gettimeofday()
   C.gettimeofday(gettimeofday_struct, nil)
   return tonumber(gettimeofday_struct.tv_sec) * 1000000 + tonumber(gettimeofday_struct.tv_usec)
end
```

Lulip

- Line level profiler for Lua in Lua: https://github.com/jgrahamc/lulip/

```
local profiler = require 'lulip'
local p = profiler:new()
p:dont('some-module')
p:maxrows(25)
p:start()

-- execute code here

p:stop()
p:dump(output_file_name)
```

Lulip Output

file:line	count	elapsed (ms)	line
wr.lua:1129	2	822.455	`hash = ngx_sha1_bin(value)`
wr.lua:1172	428	470.849	`captures, err = ngx_re_match(v, p)`
wr.lua:1197	3762	207.487	`x = string_find(v, f)`
wr.lua:212	157	154.386	`string_gsub(v, "//([^/]+)//", "%1")`
wr.lua:1196	3788	87.475	`for i=1,g() do`
wr.lua:1158	1563	52.906	`if not f() then`

Lulip Core

- Nice things
  - Lua's debug library
  - LuaJIT's FFI for `gettimeofday()`
  - Closures

```
-- start: begin profiling
function start(self)
   self:dont('lulip.lua')
   self.start_time = gettimeofday()
   self.current_line = nil
   self.current_start = 0

   debug.sethook(function(e,l) self:event(e, l) end, "l")
end
```

Performance Tricks

- Wait til you've finished; Measure; Fix the slow things
- Locals way faster than globals

```
local rand = math.random

local len = #t
for i=1,len do
   ...
end
```

- `.` syntax faster than `:`

```
local slen = string.len
s:len() vs. slen(s)
```

- Minimize closures

Autogenerated Code

```
local waf_vars = waf.vars
local waf_streq = waf.streq
local waf_setvar = waf.setvar
local waf_msg = waf.msg
local waf_drop = waf.drop
local waf_disabled_ids = waf.disabled_ids
local waf_deny = waf.deny
local waf_activate = waf.activate
local t1_1 = {}
if not waf_disabled_ids['00001'] and waf_streq(waf, v2_5, '2_5', t1_1, '1_1', 'b783efc191a7c066c1d87068f63a84a39f9830bb', false) then
  waf_vars['RULE']['ID'] = '00001'
  waf_activate(waf, rulefile)
  waf_msg(waf, 'CloudFlare Test Rule (drop) activated')
  waf_setvar(waf, {{'TX:ANOMALY_SCORE', '+100'},{'TX:%{RULE:ID}', 'CloudFlare unique hash test rule (drop)'}})
  waf_drop(waf, rulefile)
end
if not waf_disabled_ids['00002'] and waf_streq(waf, v2_5, '2_5', t1_1, '1_1', '4709edce126971876b547523778fa7b942ec14b5', false) then
  waf_vars['RULE']['ID'] = '00002'
  waf_activate(waf, rulefile)
  waf_msg(waf, 'CloudFlare Test Rule (deny) activated')
  waf_setvar(waf, {{'TX:ANOMALY_SCORE', '+100'},{'TX:%{RULE:ID}', 'CloudFlare unique hash test rule (deny)'}})
  waf_deny(waf, rulefile)
end
```

nginx + Lua and OpenResty

- http://wiki.nginx.org/HttpLuaModule
- Control over all phases of nginx inside Lua
- Very fast, very flexible, non-blocking (without callbacks!)
- Sockets, coroutines, subrequests, shared in-memory caching

- http://openresty.org/
- Complete package of nginx, Lua and a set of libraries
- Access to Redis, memcached, MySQL, Postgresql
- JSON parsing, DNS, locks, ...

- CloudFlare HTTP almost entirely nginx + Lua / OpenResty

CloudFlare WAF

```
location / {
    set $backend_waf        "WAF_CORE";
    default_type            'text/plain';

    access_by_lua '
        local waf = require "waf"
        waf.execute("")
    ';

    log_by_lua_file "lua/metrics/waf_metrics_main.lua";

    content_by_lua 'ngx.say("")';
    error_page 500 =200 @error;
}
```

Irritant: Arrays

- Index starts at 1
- `#` doesn't do what you think

```
> x = {"a", "b", "c"}
> =#x
3
> x = {"a", "b", nil, "c"}
> =#x
4
> x = {"a", "b", nil, "c", nil}
> =#x
2
```

Irritant: escape characters

- `string.find`, `string.match`
- Lots of special characters just like regular expressions
  - Character classes: `[a-z]`, `[^a-z]`, `.`
  - Repetition: `*`, `+`, `?`
  - Anchors: `^` and `$`
  - Groups and alternation: `(foo|bar)`
- And then... `%`
  - `%d`, `%u`, `%D`, `%s`
- Although `%b` and `%f` are cool

Awesome: tables

- Anything can be a key; anything can be a value
  - Tables as values for nesting
  - Functions as values

```
x = {“a”, “b”, “c”}
y = {subtable=x, double=function(v) return v*2 end}
```

- Tables are references

```
function f(t, v) t.subtable = v end
f(y, {“1”, “2”, “3”})
```

Tables aren’t cool... metatables are cool

CLOUDFLARE

metatables to make a table read-only

```
local t = {a = "A"}
local _t = t

t = {}
local ro = {
    __index = _t,
    __newindex = function() error("R/O") end
}

setmetatable(t, ro)


print(t.a)
t.a = "B"
```

CLOUDFLARE

metatables for lazy loading

```
local t = {}
local loader = {
   __index=function(_t, _v)
      _t[_v] = docostlyload(_v)
      return _t[_v]
   end}
setmetatable(t, loader)


print(t.expensive)
```

metatables to make objects

```
local C = {}
C.__index = C

function C.new(d)
   local newObject = {data=d}
   return setmetatable(newObject, C)
end

function C.get_data(self)
   return self.data
end

local o = C.new("hello")
print(o:get_data())
```

metatables to sandbox code #1

```
local env = { print = print }
local envmeta = { __index={}, __newindex=function() end }
setmetatable(env, envmeta)


function run(code)
   local f = loadstring(code)
   setfenv(f, env)
   pcall(f)
end

run([[
  local x = “Hello, World!”
  print(x)
  local y = string.len(x)
]])
```

CLOUDFLARE

metatables to sandbox code #2

```
local env = { print = print, string = { len = string.len } }
local envmeta = { __index={}, __newindex=function() end }
setmetatable(env, envmeta)


function run(code)
   local f = loadstring(code)
   setfenv(f, env)
   pcall(f)
end

run([[
  local x = “Hello, World!”
  print(x)
  local y = string.len(x)
]])
```

metatables for undefined variable detection

```
function doubler(x) return X*2 end
print(doubler(2))
t.lua:1: attempt to perform arithmetic on global 'X' (a nil value)
```

```
function doubler(x) return X*2 end
catcher={__index=function(t,v) error(v .. " undefined") end,
         __newindex=function(t,k,v) error(v .. " undefined") end}
setmetatable(_G, catcher)
print(doubler(2))
lua: t.lua:2: Tried to read undefined X
```

Conclusion

- Get past initial irritation!
- Lua is a GREAT language for embedding in large systems
  - Fast
  - Lots of functionality
  - Good standard library
  - Small
  - Extensible both from C and to C