Aurora

Lua lernen ✓
Entwickeln üben ✓
IRC Bot haben ✓

Julius Roob

Chaostreff Kaiserslautern

11. Juni 2010

Übersicht

Einleitung

Lua

IRC

Aurora

Lua

Inhalt

- Kurze Beschreibung
- Datentypen
- Tables
- Funktionen
- Sandboxing
- OOP
- Coroutines
- Luarocks
- JSON
- Copas

Lua

Kurze Beschreibung - Wikipedia lässt grüßen

- 1993 in Brasilien entwickelt
- klein
- Bytecode
- LuaJIT
- Verwendet in:
  - World of Warcraft
  - Grub2
  - Adobe Photoshop Lightroom
  - Awesome
  - ...

Datentypen

- Number
- String
- Boolean
- Nil
- Table
- Function
- Thread
- „Userdata“

Tables

Lua Programme werden hauptsächlich durch Tables organisiert. Tables lassen sich wie Hashtables und Arrays nutzen. Metatables (und Metamethoden) bestimmen Verhalten von LUA

_G

```
> print(_G.print)
function: 0x9402020
```

__index

```
> A = {a = 1, b = 2}
> B = {b = 3}
> setmetatable(B, {__index = A})
> print(B.a)
1
> print(B.b)
3
```

Funktionen

Schreibweise

```
function <Name>(<Parameter>) ... end
-- ist das gleiche wie:
<Name> = function(<Parameter>) ... end
```

Mehrere Rückgabewerte

```
local a, b, c = unpack({"hallo", "welt", "!"})

print(a, b, unpack({"!", "wie geht es dir?"}))
```

OOP

```
welt:hallo("!")
-- ist kurz fuer:
welt.hallo(welt, "!")
```

Proper Tail Calls

```
function f (x)
  return g(x)
end
```

Sandboxing

Sehr komfortabel in Lua.

Scope `setfenv()`

Prozessor

```
function e() error("CPU limit reached!") end
debug.sethook(e, "", 1000)
```

Speicher `setrlimit()` und `pcall()`

OOP

Alternative zur bereits beschriebenen OOP: *Closures for Scopes*

```
1  function factory()
2    local self = {kaffee=0}
3    local interface = {}
4    function interface.set(n)
5      self.kaffee = n
6    end
7    function interface.get()
8      return self.kaffee
9    end
10   return interface
11 end
```

Coroutines

```
function test()
  local a = 0
  while true do
    a = a + 1
    coroutine.yield(a)
  end
end
```

```
> a = coroutine.create(test)
> print(coroutine.resume(a))
true 1
> print(coroutine.resume(a))
true 2
> print(coroutine.resume(a))
true 3
> print(coroutine.resume(a))
true 4
> print(coroutine.resume(a))
true 5
```

Luarocks

Einfaches installieren von zusätzlichen Bibliotheken bzw. Paketen.

```
# luarocks search pcre
# luarocks install lrexlib-pcre
```

Kann von einem Benutzer im Homedir installiert werden.

Skript mit Rocks ausführen:

```
# lua -lluarocks.loader aurora.lua
```

JSON

Einfach und schön

```
{
  "wasser":{"julius":1},
  "africola":{"bnerd":1,"julius":0},
  "baileys":{"julius":1},
  "cappu":{"lucy":2,"julius":7,"mullej":7},
  "mate":{"julius":0},
  "kaffee":{"julius":2,"lastofthewolves":1}
}
```

`json.encode()` und `json.decode()` sind wohl die einfachste Möglichkeit, Tables zu serialisieren.

Copas

Coroutine Oriented Portable Asynchronous Services for Lua

Primitiver Chatserver

```
 3 users = {}
 4 function handle(client)
 5   copas.send(client, "name? ")
 6   local name = copas.receive(client, "*l")
 7   users[name] = client
 8   while true do
 9     local line = copas.receive(client, "*l")
10     for _,sock in pairs(users) do
11       copas.send(sock, name .. ": " .. line .. "\n")
12     end
13   end
14 end
15 serv = socket.bind("0.0.0.0", 1025)
16 copas.addserver(serv, handle)
17 copas.loop()
```

IRC

Inhalt

- Kurze Beschreibung
- Nachrichtenformat
- Bleistifte
- CTCP

Kurze Beschreibung (aka Wikipedia lite)

- Gibt es als **Internet** Relay Chat seit 1988.
- Es existiert eine unüberschaubare Anzahl an Clients, Bots und Servern
- Thin Clients

Nachrichtenformat

Die Kommunikation zwischen Server und Client findet ausschließlich durch max. 510 Byte lange mit einem CRLF getrennte Nachrichten statt.

```
NICK julius_
USER julius julius irc.hackint.org :Julius
:irc.hackint.org NOTICE AUTH :*** No Ident response
:irc.hackint.org NOTICE AUTH :*** Couldn't look up
    your hostname
:irc.hackint.org 001 julius :Welcome to the hackint
    Internet Relay Chat Network julius

PING LAG2126774742
:irc.hackint.org PONG irc.hackint.org :LAG2126774742
PRIVMSG julius :blub!
:irc.hackint.org 301 julius_ julius :auf nach hause

:eBrnd!~eBrnd@ein.anderer.server PRIVMSG #c3kl
    :bitte irgendwas :)
```

Bleistife

- `PRIVMSG <Channel> <Nachricht>`
- `JOIN <Channel>(,<Channel>)*`
- `QUIT <Nachricht>`
- `NICK <Name>`

CTCP

Client-To-Client Protocol

Format

”\001<Befehl> <Parameter>\001”

Beispiele

```
\001ACTION demonstriert mal eben CTCP Action\001
-> 10:10  *julius demonstriert mal eben CTCP ACTION

\001VERSION\001
-> 10:16 [freenode] CTCP VERSION reply from julius:
    irssi v0.8.12

\001DCC SEND chess.lua 2213957892 48042 1682\001
```

Aurora

Inhalt

- Motivation
- Architektur
- Module
- Probleme

Motivation

Warum einen eigenen IRC Bot schreiben?

- Sprache Lua lernen
- Eventbasierende Socket IO verinnerlichen
- IRC Bots sind immer praktisch

Motivation

Warum einen eigenen IRC Bot schreiben?

- Sprache Lua lernen
- Eventbasierende Socket IO verinnerlichen
- IRC Bots sind immer praktisch
- Spaß

Architektur

Uh - ja...

- Module
- ???
- IRC Netzwerke

Module

- Konstruktor
- Destruktor
- Step
- Handler
  - DISCONNECT
  - CONNECT
  - PRIVMSG
  - ... (siehe RFC :)

Probleme

- Module organisieren?
- Zwischen Modulen und Netzwerken herrscht Anarchie
- Authentifizierung im IRC ist unschön

Fragen?

Vorschläge, Anregungen...

`http://www.github.com/Zottel/Aurora`

:-)

Quellen

- #lua auf Freenode
- LUA-Users Wiki
- LUA Reference Manual
- Projektseite von Copas
- Wikipedia
- RFC 1459 (Internet Relay Chat Protocol)
- Julius Kopf