Scripting with Luerl

Luerl

Luerl is an implementation of standard Lua written in Erlang/OTP.

Lua is a powerful, efficient, lightweight, embeddable scripting language common in games, machine learning, IoT devices and scientific computing research.

It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

https://github.com/rvirding/luerl https://luerl.org

Lua ...

The Lua Language

Lua was born out of necessity, initially developed as a result of strict trade barriers instituted by the Brazilian government.

Created in 1993 by Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes at the Pontifical Catholic University of Rio de Janeiro, in Brazil.

Lua is explicitly designed as scripting language to be embedded into applications build on other languages.

The Lua Language

Originally influenced by Tcl (scripting, embedded),
Modula, SNOBOL and AWK (associative arrays),
LISP with their single, data-structure mechanism (the list),
and Scheme; first class anonymous functions with lexical scope (closures).

The Lua Language

Lua is an embedded language: meaning that is not a stand-alone package, but a library that links to other applications to extend them.

The ability to be used as a library is what makes Lua an ***extension language***. Lua can register new functions in the environment; such functions are implemented in C (or another language) this add features that cannot be written directly in Lua.

The Lua Language

These two views of Lua the extension language and as an extensible language correspond to two kinds of interaction between C and Lua.

In the first kind, C its the application language and Lua is the library.

In the second kind, Lua its the application language and C is the library.

Both of them use the same API to communicate with Lua, the so called C API.

The Lua Language

Lua programs are not interpreted directly from the source code, but are compiled into bytecode, which runs into the Lua virtual machine.

The compilation process is typically invisible and is performed during run-time.

Uses of Lua

- Widely used in some niches not just game development
- Embedded systems; TVs(Samsung), routers(Cisco), mobiles(Huawei), ...
- Scripting for applications; Wikipedia, Wireshark, Nmap, VLC Media Player
- Very popular in systems infrastructure; nginx, haproxy, asterisk, kamailio
- Web development; Sailor, OpenResty, Lapis, Turbo
- Image manipulation; Adobe Lightroom
- API Gateways; Kong
- Deep Learning; Torch

Erlang ...

The Erlang Language

Erlang is a concurrent, functional language which runs on its own scalable, distributed, fault-tolerant, soft real-time, highly available VM called the BEAM.

The BEAM ecosystem

Multilingual from the start, the Erlang ecosystem with all its languages is build for and runs on the BEAM VM, the Erlang language and the OTP framework.

Erlang, Elixir, LFE, Luerl, Erlog, Efene, Bragful, Alpaca, Joxa ...

The BEAM internal properties

- Lightweight, massive concurrency
- Asynchronous communication
- Process isolation
- Error handling
- Hot code replacement
- Support for introspection and monitoring

We rarely have to worry about this properties, except for receiving messages!

The BEAM external properties

- Immutable data
- Pattern matching
- Functional language
- Predefined set of data types
- Modules
- No global data

These are what we “see” directly in the BEAM ecosystem of languages.

Uses of Erlang

- Widely used in some niches it’s not just a “telecom language”
- Web development; Cowboy (99s)
- Communication systems; MongooseIM(Erlang Solutions), WhatsApp(FB), ejabberd (Process One), ...
- Operations and monitoring; WombatOAM (Erlang Solutions)
- Message brokers; RabbitMQ (Pivotal)
- Very popular in cluster databases; Riak (R.I.P Basho), SimpleDB (Amazon)

So, why Luerl ? ...

Well… Robert likes to implement languages

Luerl features

Luerl should allow you to fast switch between Lua and Erlang, introducing a new way to use very small bits of logic programmed in Lua, inside an Erlang application, which means that:

- Programs are written in at least two languages; a scripting and system language.
- The system language implements the hard parts of the application; algorithms, data structures, there is little change at this level.
- Scripting glues together the hard parts; this level is flexible, easy to change.

Millions of processes

Luerl presents an environment that allows you to effortlessly run millions of Lua processes concurrently and in parallel, levering the famed processes implementation of the BEAM VM.

A note on coroutines

There are no coroutines in Luerl, this could seem counterintuitive coming from a solid Lua background, but they are not needed in the BEAM ecosystem, here we use processes; concurrent, lightweight asynchronous processes.

Uses of Luerl

Lua is used in various products build with Erlang and open-source projects around the world, including several games, API Gateways, and StarCraft AI bots.

The trade-offs

Lua is not good for:

- Concurrency and multi-core parallelism
- The standard library is not very large
- DIY approach to life (Do It Yourself)

Erlang is not good for:

- Heavy number crunching
- Global, shared, mutable state
- Scripting, yes, we know about escript but.. really?

Luerl: The goal

A proper implementation of the Lua language.

- It should look and behave the same as Lua
- It should include the standard libraries
- It should interface well with Erlang

Luerl is a complete reimplementation of the Lua language including a Luerl Erlang API that match the Lua C API available on standard Lua.

Luerl: The result

Luerl is a successful implementation of Lua in Erlang

- except goto, _ENV and coroutines

It, also has great interaction between Erlang and friends

- Easy for Erlang to call Lua and Lua to call Erlang
- Compatible with Erlang concurrency and error handling.

Luerl: The result

Luerl standard library implemented

- Functions
- Modules
- String manipulation
- Table mechanism
- Math functions
- Bitwise operators
- Very few I/O and OS facilities

Luerl: The result

- Extendable if required to call Lua from Erlang
- Straight-forward to call Erlang from Lua
- No C interface! - (Erlang application programming interface instead of the Lua C API.)

Luerl: The result - VM and compiler

A relative straight-forward VM

- Similar, but not the same, as the standard Lua one.

Compiler optimises the environment handling

- Separates purely local environment of blocks and functions from global env

Lots of “unnecessary” information compiled away

- Error messages are very basic, but no worries this is in the pipeline.

Luerl: The result - datatypes

Lua	Luerl
nil	Atom nil
true/false	Atom true/false
strings	strings binaries
numbers	floats
tables	#tables{} with arrays for keys we have 1...n and #dicts for the rest
functions	#function{} or {function, Fun}

Luerl: The result - Lua state

The main difficulty of Luerl implementation was the need to implement Lua’s mutable global data with Erlang’s immutable local data.

We keep all Lua state in one data structure explicitly threaded through everything, this “threading of stuff” is what you typically do in Erlang anyway.

Robert implement Luerl’s own garbage collector on top of Erlang’s collector for Lua state.

Community

Luerl embrace both #LuaLang and #Erlang the two of them are simple, diverse and complementary communities and language ecosystems.

https://luerl.org

https://github.com/rvirding/luerl

https://github.com/rvirding/luerl/wiki

Thanks!

Barbara Chassoul
@chassoula
https://nonsense.ws