Luerl - Lua in Erlang

Scripting mechanisms for the BEAM ecosystem

Jean Chassoul

FOSDEM 2019

Luerl: Lua in Erlang

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

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

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

The Lua Language

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

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

Designed to be embedded into applications build on other languages.

Lua: Goals

- Portable
- Small < 10000 LoC
- Simple
- Emphasis on scripting

Simplicity

- Few but poweful mechanisms
- Associative arrays implements all data structures
- Closures
- Coroutines*

Scripting

A language used to control other languages and programs

A program is written in at least two different languages the scripting language and the system language usually C/C++

The system language implements the hard parts:

- Reasonable stable parts

The scripting language connects those parts

- Flexible, easy to change

Erlang

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

Created by Mike Williams, Robert Virding and Joe Armstrong at Ericsson.

The problem domain

- Lightweigh, massive concurrency
- Fault-tolerance must be provided
- Timing constraints
- Continuous operation for a long time
- Continuous maintenance/evolution of the system
- Distributed system

The problem domain

THEY WERE JUST TRYING TO SOLVE THE PROBLEM!

- Erlang is NOT an academic implementation of a functional language
- Erlang is NOT an academic implementation of the actor model

The BEAM virtual machine

A virtual machine to run Erlang.

The BEAM internal properties

We seldom have to worry about this...

- Lightweigh, massive concurrency
- Asynchronous communication
- Process isolation
- Error handling
- Hot-code reloading
- Support introspection and monitoring

...except for receiving messages!

The BEAM external properties

These are what we “see”

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

The OTP framework

OTP is a collection of useful middleware, libraries, and tools written in Erlang.

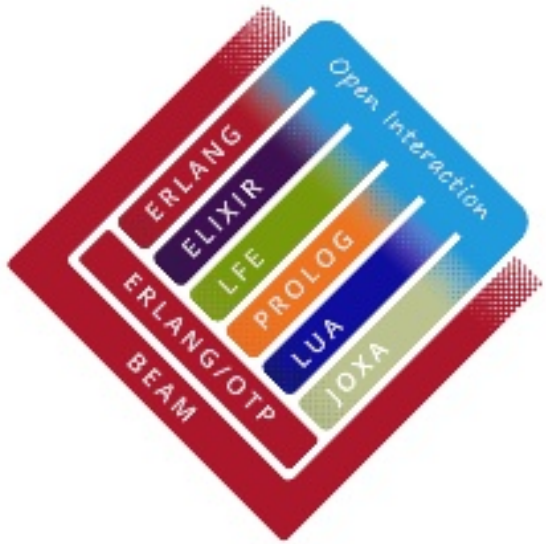
- An application concept
- Behaviours (design patterns)

The BEAM ecosystem

Erlang Ecosystem

Languages built/running on top of the BEAM, Erlang and OTP.

By following "the rules" the languages openly interact with, and support, each other making the whole system more powerful than any individual language can ever be.



28

Mechanisms Not Policies

Lua is powerful (but simple): A fundamental concept in the design of Lua is to provide meta-mechanisms for implementing features, instead of providing a host of features directly in the language.

Lua ecosystem

Just no name a few implementations

- PUC Lua (*oficial standard*)
- LuaJIT
- RaptorJIT
- MoonScript*
- Luerl
- ...

And many, many more!

Luerl: Goals

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: Implementation

Like standard Lua, Luerl is implemented as a library. The Luerl VM is a mixture of interpreting Lua VM instructions using Erlang directly to implement function calls.

Luerl: Features

Fast Language Switch: Luerl allow you to switch between Erlang and Lua incredibly fast, introducing a new way to use very small bits of logic programmed in Lua inside an Erlang application, with good performance.

Multicore: Luerl provides a way to transparently utilize multicores. The underlying VM takes care of the distribution.

Multiprocesses: It give you a Lua environment that allows you to effortlessly run concurrent Lua processes in parallel, leveraging the famed BEAM VM.

Scripting: Scripting emphasizes inter-language communication in at least two languages a scripting and a system language.

Luerl: Fast switch

Luerl SHOULD allow fast switch between Lua and Erlang, introducing a new way to use very small bits of logic programming 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 usually the *hard* parts of the application; algorithms, data structures.
- Scripting glues together the *hard* parts; this level is flexible, easy to change.

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

Luerl: Some trade-offs

Lua is not good for:

- Concurrency and multi-core parallelism
- The standard library is not very large
- DIY approach to life

Erlang is not good for:

- Heavy number crunching
- Global, shared, mutable state
- Desktop GUI applications.

Luerl: The result

Luerl is a successful implementation of Lua in Erlang.

It also has great interaction between Elixir, LFE and friends.

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

We keep our 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.

Luerl: Community

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

Thanks!

Thank you for listening!

Jean Chassoul

@jchassoul

https://github.com/rvirding/luerl

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

https://luerl.org