Lua Application Programming: Starting a conversation

Hisham Muhammad

https://hisham.hm/
https://mastodon.social/@hisham_hm

LuaConf 2017

Rio de Janeiro, 2017-06-03

Chapter 1

Programming in the Large

But what is Lua for, anyway?

Lua, an extension language for configuration

- **Lua 5.0 (2003)**
  - Lua is an extension programming language designed to support general procedural programming with data description facilities. It also offers good support for object-oriented programming, functional programming, and data-driven programming. Lua is intended to be used as a powerful, light-weight configuration language for any program that needs one.

Lua, an extension language for scripting

- **Lua 5.1 (2006)**
  - Lua is an extension programming language designed to support general procedural programming with data description facilities. It also offers good support for object-oriented programming, functional programming, and data-driven programming. Lua is intended to be used as a powerful, light-weight scripting language for any program that needs one.

Lua, an extension language for scripting

- **Lua 5.2 (2011)**
  - Lua is an extension programming language designed to support general procedural programming with data description facilities. It also offers good support for object-oriented programming, functional programming, and data-driven programming. Lua is intended to be used as a powerful, light-weight embeddable scripting language for any program that needs one.

Lua, a scripting language

- **Lua 5.3 (2015)**
  - Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

Lua, a scripting language

- **Lua 5.3 (2015)**
  - Lua is a powerful, efficient, lightweight, embeddable scripting language. It supports procedural programming, object-oriented programming, functional programming, data-driven programming, and data description.

    (...)

    Lua is intended to be used both as a powerful, lightweight, embeddable scripting language for any program that needs one, and as a powerful but lightweight and efficient stand-alone language.

So let’s write stand-alone programs with Lua!

- Games!
- Servers!
- Theorem provers!
- Package managers!

Awesome, where do I start?

Writing applications

- A lot goes into writing an application:

Writing applications

- A lot goes into writing an application:
  - The language itself

Writing applications

- A lot goes into writing an application:
  - The language itself
  - The language environment

Writing applications

- A lot goes into writing an application:
  - The language itself
  - The language environment
  - The development tools

Writing applications

- A lot goes into writing an application:
  - The language itself
  - The language environment
  - The development tools
  - The deployment

Writing applications

- A lot goes into writing an application:
  - The language itself

    Which version of Lua? (Does it matter?)
  - The language environment
  - The development tools
  - The deployment

Writing applications

- A lot goes into writing an application:
  - The language itself

    Which version of Lua? (Does it matter?)
  - The language environment

    Platforms? Libraries? Frameworks?
  - The development tools
  - The deployment

Writing applications

- A lot goes into writing an application:
  - The language itself

    Which version of Lua? (Does it matter?)
  - The language environment

    Platforms? Libraries? Frameworks?
  - The development tools

    Editors? Static checkers? Testing? CI?
  - The deployment

Writing applications

- A lot goes into writing an application:
  - The language itself

    Which version of Lua? (Does it matter?)
  - The language environment

    Platforms? Libraries? Frameworks?
  - The development tools

    Editors? Static checkers? Testing? CI?
  - The deployment

    How will users install and run your program?

Programming in the Large

- Usually involves dealing with:
  - Teamwork
  - Long-term maintenance

Programming in the Large

- **Usually involves dealing with:**
  - **Teamwork**
  - **Long-term maintenance**

- **How to go about it:**
  - **Coding for collaboration**
  - **Architecture: handling complexity**

Chapter 2
Which Lua?

Lua 5.1

Lua 5.2

Lua 5.3

LuaJIT 2.0

LuaJIT 2.1-beta

Lua 5.1

Lua 5.2

Lua 5.3

LuaJIT 2.0

LuaJIT 2.1-beta

(also: MoonScript, Terra, Ravi, (and soon!) Typed Lua)

	5.1	5.2	5.3	LJ2.0	LJ2.1b
setfenv and getfenv	✓			✓	✓
math.log10	✓			✓	✓
module	✓	✓ depr.		✓	✓
package.loaders	✓	✓ depr.		✓	✓
goto		✓	✓	✓	✓
xpcall(f, err, [args...])		✓	✓	✓	✓
bit32		✓	✓ depr.		
_ENV		✓	✓		
package.searchers		✓	✓		✓
table.pack and table.unpack		✓	✓	✓ compat.	✓ compat.
Ephemeron tables		✓	✓		
__pairs and __ipairs		✓	✓ depr.	✓ compat.	✓ compat.
os.execute detail return values		✓	✓	✓ compat.	✓ compat.
io.read without *			✓		✓
table.move			✓		✓
coroutine.isyieldable			✓		✓
Bitwise operators			✓		
64-bit integer subtype			✓		
ffi				✓	✓
bit				✓	✓
continue					

	5.1	5.2	5.3	LJ2.0	LJ2.1b
setfenv and getfenv	✓			✓	✓
math.log10	✓			✓	✓
module	✓	✓ depr.		✓	✓
package.loaders	✓	✓ depr.		✓	✓
goto		✓	✓	✓	✓
xpcall(f, err, [args...])		✓	✓	✓	✓
bit32		✓	✓ depr.		
_ENV		✓	✓		
package.searchers		✓	✓		✓
table.pack and table.unpack		✓	✓	✓ compat.	✓ compat.
Ephemeron tables		✓	✓		
__pairs and __ipairs		✓	✓ depr.	✓ compat.	✓ compat.
os.execute detail return values		✓	✓	✓ compat.	✓ compat.
io.read without *			✓		✓
table.move			✓		✓
coroutine.isyieldable			✓		✓
Bitwise operators			✓		
64-bit integer subtype			✓		
ffi				✓	✓
bit				✓	✓
continue					

Solution:
Write to the common subset

- **Easier than it seems**
  - **LuaRocks does it!**
- **Many compatibility libraries:**
  - **https://luarocks.org/modules/luarocks/luabitop**
  - **https://luarocks.org/modules/siffiejoe/bit32**
  - **https://luarocks.org/modules/hisham/compat52**
  - **https://luarocks.org/modules/hisham/compat53**
  - **https://github.com/facebook/luaffifb**

Chapter 3

The Perl Paradox

Programming languages and their mottos

- **Perl**
  - **“There’s More Than One Way To Do It”**
- **Python**
  - **“There should be one—an preferrably only one—obvious way to do it” - *Zen of Python***
- **Lua**
  - **“Mechanisms, not policies”**

Programming languages and their mottos

- Perl – maximalist: n ways to do it
  - "There's More Than One Way To Do It"
- Python
  - "There should be one—an preferrably only one—obvious way to do it" - *Zen of Python*
- Lua
  - "Mechanisms, not policies"

Programming languages and their mottos

- **Perl – maximalist: n ways to do it**
  - **“There’s More Than One Way To Do It”**
- **Python – opinionated: 1 way to do it**
  - **“There should be one—an preferrably only one—obvious way to do it” - *Zen of Python***
- **Lua**
  - **“Mechanisms, not policies”**

Programming languages and their mottos

- **Perl – maximalist: n ways to do it**
  - “There’s More Than One Way To Do It”
- **Python – opinionated: 1 way to do it**
  - “There should be one—an preferrably only one—obvious way to do it” - *Zen of Python*
- **Lua – minimalist: 0 ways to do it**
  - “Mechanisms, not policies”

Programming languages and their mottos

- **Perl – maximalist: n ways to do it**
  - “There’s More Than One Way To Do It”
- **Python – opinionated: 1 way to do it**
  - “There should be one—an preferrably only one—obvious way to do it” - *Zen of Python*
- **Lua – minimalist: 0 ways to do it**
  - “Mechanisms, not policies”
    - Corollary: “There’s More Than One Way To Do It”

Create the world

GOOD:

BAD:

Create the world

GOOD:

- Control over all components
- Tailored for your application
- Everything is exactly the way you want

BAD:

Create the world

GOOD:

- Control over all components
- Tailored for your application
- Everything is exactly the way you want

BAD:

- Control over all components
- Tailored for your application
- Everything is exactly the way you want

Create the world

GOOD:

- Control over all components
- Tailored for your application
- Everything is exactly the way you want

BAD:

- Responsibility over all components
- Tailored for your application only
- The way you want may not be what others want

Create the world

GOOD:

- Control over all components
- Tailored for your application
- Everything is exactly the way you want

BAD:

- Responsibility over all components
- Tailored for your application only
- The way you want may not be what others want
- A lot of work!

Solution:
Model your app with libraries

- Structure as much of your application as possible as libraries
  - LuaRocks doesn’t do it :-(
- Split concerns into libraries
  - Favor reusing existing ones
- Don’t go overboard writing libraries
  - ...or you’ll never get to the app!

Libraries!? I just want to write a program!
What about the KISS principle?

Why not monolithic design

- Lua is a dynamic language: the structure of your tables is not written anywhere in the program

Why not monolithic design

- Lua is a dynamic language: the structure of your tables is not written anywhere in the program
- It’s too tempting to “create tables as you go” and just “pass tables around”

Why not monolithic design

- Lua is a dynamic language: the structure of your tables is not written anywhere in the program
- It’s too tempting to “create tables as you go” and just “pass tables around”
- Eventually things get out of hand
  - What is the lifetime of this field in this table?
  - Which parts of the code are responsible for keeping this field up-to-date?
  - Is this the only place where this part of the table is used?

Library-oriented design

- Helps coding for collaboration
  - Well-defined programming interfaces
  - Well-defined responsibility boundaries

- Helps taming complexity
  - Divide and conquer
  - Each piece is small and simple!

Libraries avoid tricks

- The library mindset helps you avoid tricks
- Avoid whenever possible:
  - Global metatable magic
  - Global variables
  - Global environment tricks
  - Debug library tricks
  - Implicit coroutine use
- These things do not compose

Don’t create another incompatible world

- Your main program becomes a client of well-behaved libraries

    …and not its own little custom world

- Compatible with development tools

    …some of which need to use the tricks

Don’t create another incompatible world

- Your main program becomes a client of well-behaved libraries

  …and not its own little custom world

- Compatible with development tools

  …some of which need to use the tricks

(Lua embedded in a non-Lua app *is* a little custom world!
In this case other criteria apply)

Chapter 4
A Brief Tour of Tools

Tools matter

- The Lua interpreter alone can only take you so far
  - Written in pure ISO C for maximum portability
  - A complete language VM in a 250kB lib!
  - Almost no OS facilities: can’t even list a directory by itself
- You almost certainly will need more
  - Interaction with the system: Additional libraries
  - Development tools

Platforms

- Desktop
  - Bindings of GUI libraries and their object models: lgi (GTK+), WxLua, NLua (.NET), LuaJ (JVM)
  - Löve2D for games
- Mobile
  - Cross-platform: Corona, Gideros... (Löve2D too!)
- Web
  - Lapis, Sailor
  - OpenResty

Development tools

- Editors:
  - ZeroBraneStudio, LDT for Eclipse, editor plugins
- Static checker:
  - luacheck (preferrably integrated with your editor!)
- Testing:
  - Busted for unit testing, luacov for coverage analysis
- Package management: LuaRocks
- Documentation: LDoc
- Lua version management: hererocks, luaver
- CI: Great talk by Enrique García on Travis integration

Deployment

- How will users install and run your program?
  - Via the package manager (Unix)
    - LuaRocks supports pure-Lua applications
    - Native distro packages?
  - Make self-contained Lua packages (Windows)
    - luabuild
    - wxFreeze in wxLua
- Unfortunately we don’t have (yet?) a simple one-size-fits-all solution

Chapter 5
In short...

Lua application programming is a reality

- Lua is not only for scripting
- Lua application programming is a reality
- Programming in the large requires its own mindset
- Works great in some environments
  - Examples in this conf!
- Still some rough edges in some scenarios

To be continued!