Tarantool team's experience with Lua developer tools

Yaroslav Dynnikov

Tarantool, Mail.Ru Group

3 March 2019

Tarantool

Tarantool is an open-source data integration platform

Tarantool = Database + Application server (Lua)

Tarantool

Tarantool is an open-source data integration platform

Tarantool = Database + Application server (Lua)

Core team

Focuses on the product development

Solution team

Implements projects for the Enterprise

Tarantool Solution Engineering

- `35` Lua developers
- `~ 50` Git repos
- `~ 300,000` SLoC

Customers

- IT, Banking, Telecom, Oil & Gas

Goals

- Develop projects **fast** and **well**

Writing better code

Development

- Runtime checks

The sooner code fails - the better

```lua
local function handle_stat_request(req)
  local stat = get_stat()
  return {
    status = 200,
    body = json.encode(stet),
  }
end

handle_stat_request()
-- returns: "null"
```

Development

- Runtime checks

The sooner code fails - the better

```
local function handle_stat_request(req)
  local stat = get_stat()
  return {
    status = 200,
    body = json.encode(stet),
  }
end

handle_stat_request()
-- returns: "null"
```

Strict mode restricts access to undeclared globals

```
require('strict').on()

handle_stat_request()
-- error: variable "stet" is not declared
```

Development

- Runtime checks

- Static analysis

Luacheck highlights mistakes in IDE

```lua
local function handle_stat_request(req)
  local stat, err = get_stat() -- unused variable 'err'
  if not stat then
    return {
      status = 500,
      body = errr -- accessing undefined variable 'errr'
    }
  end
end
```

Development

- Runtime checks

- Static analysis

Luacheck highlights mistakes in IDE

```lua
local function handle_stat_request(req)
  local stat, err = get_stat() -- unused variable 'err'
  if not stat then
    return {
      status = 500,
      body = errr -- accessing undefined variable 'errr'
    }
  end
end
```

It also takes part in CI/CD

```
$ luacheck server.lua
Checking server.lua                                1 warnings

  server.lua:3:15: unused variable err
  server.lua:8:28: accessing undefined variable errr
Total: 1 warnings / 0 errors in 1 file
```

Development

- Runtime checks

- Static analysis

- Type checking

Error messages should help in problem investigation

```lua
local function get_stat(uri, opts)
  return http.get('http://' .. uri .. '/stat', opts)
end

get_stat(req.uri) -- req.uri == nil
-- error: attempt to concatenate a nil value
-- Bad
```

Development

- Runtime checks

- Static analysis

- Type checking

Error messages should help in problem investigation

```lua
local function get_stat(uri, opts)
  return http.get('http://' .. uri .. '/stat', opts)
end

get_stat(req.uri) -- req.uri == nil
-- error: attempt to concatenate a nil value
-- Bad
```

Public API should be validated

```lua
local function get_stat(uri, opts)
  assert(type(uri) == 'string', 'uri must be a string')
end

get_stat(req.uri) -- req.uri == nil
-- error: uri must be a string
-- Better
```

Development

- Runtime checks

- Static analysis

- Type checking

Some mistakes are still hard to catch

```
get_stat('localhost', {timeuot = 1})
--                          ^^ typo
-- No error, but does not work as expected
-- Still bad
```

Development

- Runtime checks
- Static analysis
- Type checking

Some mistakes are still hard to catch

```lua
get_stat('localhost', {timeuot = 1})
--                         ^^ typo
-- No error, but does not work as expected
-- Still bad
```

Checks validates API conventions using `debug.getlocal`

```lua
require('checks')

local function get_stat(uri, opts)
  checks('string', {timeout = '?number'})
end

get_stat()
-- error: bad argument #1 to get_stat (string expected, got nil)

get_stat('localhost', {timeuot = 1})
-- error: unexpected argument opts.timeuot to get_stat
```

Development

- Runtime checks

- Static analysis

- Type checking

- Error handling

All problems are investigated by logs

```lua
local req = {}
local ok, resp = xpcall(handle_stat_request,
                        debug.traceback, req)
if not ok then
  log.error(resp)
end
```

```
$ tarantool server.lua
...
... E> stat.lua:14: bad argument #1 to get_stat (string
        expected, got nil)
stack traceback:
  [C]: in function 'error'
  checks.lua:140: in function 'checks'
  stat.lua:14: in function 'get_stat'
  handlers.lua:25: in function 'handle_stat_request'
  [C]: in function 'xpcall'
  server.lua:13: in main chunk
```

Development

- Runtime checks
- Static analysis
- Type checking
- Error handling

Business logic errors must be handled the other way

```lua
local function get_stat(uri)
  checks('string')

  if not stats[uri] then
    error('Unknown URI')
    -- Can not tell bad request from developer mistake
  end
end
```

```lua
local function get_stat(uri)
  checks('string')

  if not stats[uri] then
    return nil, 'Unknown URI'
    -- No stack trace
  end
end
```

Development

- Runtime checks

- Static analysis

- Type checking

- Error handling

```lua
function errors.new(str)
  return {
    str = str,
    stack = debug.traceback(),
    line = ...,
    file = ...,
    ...
  }
end
```

```lua
local errors = require('errors')

local function get_stat(uri)
  checks('string')

  if not stats[uri] then
    return nil, errors.new('Unknown uri')
  end
end
```

Testing

Testing

- Coverage

Luacov measures lines coverage

```
-- script.lua
-- ====================================
   function get_stat(uri, opts)
 1   checks('string')
 1   return http.get('http://' .. uri .. '/stat', opts)
   end

 1 get_stat('localhost:8080')

-- coverage 100%
```

Line is covered $\nRightarrow$ it won't raise

Testing

- Coverage

Luacov measures lines coverage

```lua
-- script.lua
-- ====================================
   function get_stat(uri, opts)
 1   checks('string')
 1   return http.get('http://' .. uri .. '/stat', opts)
   end

 1 get_stat('localhost:8080')

-- coverage 100%
```

Line is covered $\nRightarrow$ it won't raise

```lua
   get_stat('localhost:9') -- connection refused
   get_stat('google.com') -- 404
   -- what else can http.get return?
```

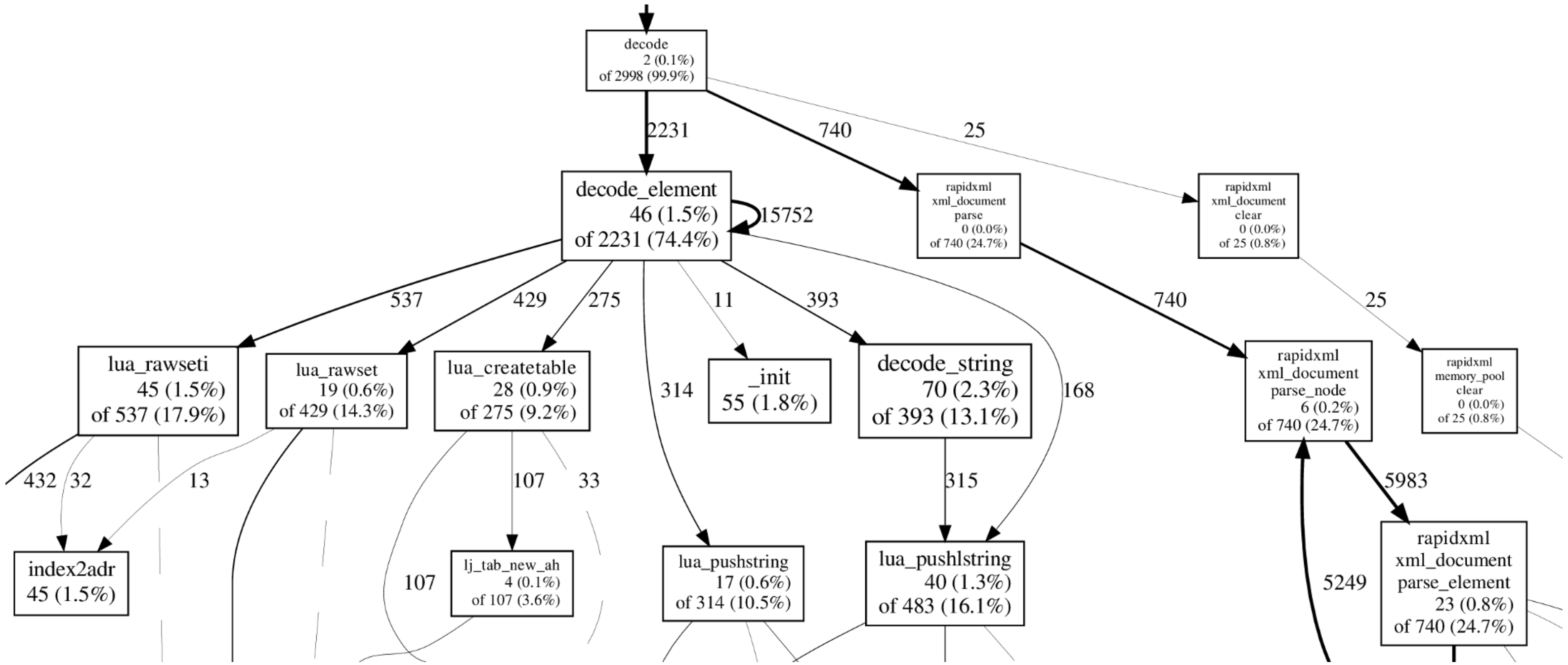
No condition coverage

Testing

- Coverage

- Performance

- `gperftools`, `callgrind` collect `C` stack traces
- `jit.p` captures `Lua` call graph
- No tool captures both



Code sharing

Code sharing

- Destinations

Within team

- LuaRocks

For customers and users

- RPM
- Deb
- tar.gz
- LuaRocks

Code sharing

- Destinations

- LuaRocks

For development:

- `luarocks make`
- `luarocks test` (looking forward to v3.0)

For shipping:

- `luarocks pack`
- `luarocks install`

For CI/CD:

- `luarocks write_rockspec`
- `luarocks new_version`

Conclusions

The sooner mistake is found - the better.

Use `strict` mode

Enable `luacheck` in IDE and in CI/CD

Use type `checks`

Log `debug.traceback`

Do not chase line coverage, be thoughtful

Links

- tarantool.io
- github.com/tarantool/tarantool

- checks: github.com/tarantool/checks
- luacheck: github.com/mpeterv/luacheck
- test-run: github.com/tarantool/test-run
- luaunit: github.com/bluebird75/luaunit
- pytest: https://pytest.org/

- gperftools: github.com/tarantool/gperftools
- jit.p https://blast.hk/moonloader/luajit/ext_profiler.html
- luarocks: https://luarocks.org/

Questions?