1 Basics

Luay is a hard fork of LuaJ for (selfish) purposes

- since luaj 3.0.2 seems to remain static, a new evolution is needed.
- scripting under java is fun and has applications
- jme is dead
- several extensions added in

Refer: https://github.com/luaj/luaj

Refer: https://www.lua.org/manual/5.2/manual.html#2

1.1 What has been done

- import LuaJ 3.0.2 sources
- maven-ize build system
- remove Java ME-isms
- move to luay namespace
- 64-bit ints, with bit64 operator library (check also bitop below ↓)
- array-only tables
- hash-only tables
- hack coerce for java Map and List interface objects to pose as native
- hack module loader to allow loading using serviceloader bader libraries
- implement lua 5.3 – io – only partially – implemented file:read options without '*', but keeping compat with 5.2
- implement lua 5.3 – utf8 – only partially – see docs
- extend the core modules with functions from ZDF – ONGOING.
- lua-stringy – Fast lua string operations (should we move that to core ?)

2 Introduction

Refer: https://www.lua.org/manual/5.2/manual.html#1

3 The Language

Refer: https://www.lua.org/manual/5.2/manual.html#3

4 Java API

TODO

5 lua aux lib

TODO

6 Standard Libraries

All libraries are implemented and provided as separate Java classes modules. Currently, Lua-5.2 has the following standard libraries:

- basic library (extended by Luay)
- coroutine library
- package library
- string manipulation
- table manipulation (extended by Luay)
- mathematical functions
- bitwise operations (extended by Luay)
- input and output
- operating system facilities
- debug facilities

Except for the basic and the package libraries, each library provides all its functions as fields of a global table or as methods of its objects.

In addition to the extensions of the standard libraries, Luay provides the following additional libraries:

- hash library (md5, sha1, ...)

6.1 Basic Functions

Refer: http://www.lua.org/manual/5.2/manual.html#6.1

- `assert (v [, message])`
- `collectgarbage ([opt [, arg]])`
- `dofile ([filename])`
- `error (message [, level])`
- `_G`
- `ipairs (t)`
- `load (ld [, source [, mode [, env]]])`
- `loadfile ([filename [, mode [, env]]])`
- `next (table [, index])`
- `pairs (t)`
- `pcall (f [, arg1, ···])`
- `print (···)`
- `rawequal (v1, v2)`
- `rawget (table, index)`
- `rawlen (v)`
- `rawset (table, index, value)`
- `select (index, ···)`
- `setmetatable (table, metatable)`
- `tonumber (e [, base])`
- `tostring (v)`
- `type (v)`
- `_VERSION`
- `xpcall (f, msgh [, arg1, ···])`

6.1.1 Luay Extensions

- `stringify (string[, type])`
- `printf (···)`

6.2 Coroutine Manipulation

```
local coroutine = require('coroutine');
```

Refer: http://www.lua.org/manual/5.2/manual.html#6.2

- `coroutine.create (f)`
- `coroutine.resume (co [, val1, ···])`
- `coroutine.running ()`
- `coroutine.status (co)`
- `coroutine.wrap (f)`
- `coroutine.yield (···)`

6.3 Modules

Refer: http://www.lua.org/manual/5.2/manual.html#6.3

- `require (modname)`
- `package.config`
- `package.loaded`
- `package.loadlib (libname, funcname)`
- `package.path`
- `package.preload`
- `package.searchers`
- `package.searchpath (name, path [, sep [, rep]])`

6.4 String Manipulation

```
local string = require('string');
```

Refer: http://www.lua.org/manual/5.2/manual.html#6.4

- `string.byte (s [, i [, j]]) -> bytei[, ..., bytej ]`
- `string.char (···) -> charstring`
- `string.dump (function) -> string`
- `string.find (s, pattern [, init [, plain]]) -> index1, index2 [, catures ...]`
- `string.format (formatstring, ···) -> string`
- `string.gmatch (s, pattern) -> iterator`

```
s = "hello world from Lua"
for w in string.gmatch(s, "%a+") do
    print(w)
end
```

```
t = {}
s = "from=world, to=Lua"
for k, v in string.gmatch(s, "(%w+)=(%w+)") do
    t[k] = v
end
```

- `string.gsub (s, pattern, repl [, n]) -> string`
- `string.len (s) -> int`
- `string.lower (s)`
- `string.match (s, pattern [, init])`
- `string.rep (s, n [, sep])`
- `string.reverse (s)`
- `string.sub (s, i [, j])`
- `string.upper (s)`

Refer: http://www.lua.org/manual/5.3/manual.html#6.4

TODO

- `string.pack (fmt, v1, v2, ···)`
- `string.packsize (fmt)`
- `string.unpack (fmt, s [, pos])`

6.4.1 Patterns

Refer: http://www.lua.org/manual/5.2/manual.html#6.4.1

6.4.2 Format Strings for Pack and Unpack

TODO

Refer: http://www.lua.org/manual/5.3/manual.html#6.4.2

6.5 UTF-8 Support

```
local utf8 = require('utf8');
```

Refer: http://www.lua.org/manual/5.3/manual.html#6.5

- `utf8.char (···) -> string`
- `utf8.charpattern` (NOT IMPLEMENTED)
- `utf8.codes (s)` (NOT IMPLEMENTED)
- `utf8.codepoint (s [, i [, j]]) -> list`
- `utf8.len (s [, i [, j]])` (NOT IMPLEMENTED)
- `utf8.offset (s, n [, i])` (NOT IMPLEMENTED)

6.5.1 Luay Extensions

- `utf8.length (s) -> int`
- `utf8.substr (s, n [, i]) -> string`
- `utf8.indexof (s, n [, i]) -> int/nil`
- `utf8.lastindexof (s, n [, i]) -> int/nil`
- `utf8.lower (s) -> string`
- `utf8.upper (s) -> string`

6.6 Table Manipulation

```
local table = require('table');
```

Refer: http://www.lua.org/manual/5.2/manual.html#6.5

- `table.concat (list [, sep [, i [, j]]])`
- `table.insert (list, [pos,] value)`
- `table.pack (···)`
- `table.remove (list [, pos]) -> value`
- `table.sort (list [, comp])`
- `table.unpack (list [, i [, j]])`

Refer: http://www.lua.org/manual/5.3/manual.html#6.6

TODO

table.move (a1, f, e, t [,a2])

Moves elements from table a1 to table a2, performing the equivalent to the following multiple assignment: a2[t],··· = a1[f],···,a1[e]. The default for a2 is a1. The destination range can overlap with the source range. The number of elements to be moved must fit in a Lua integer.

6.7 Mathematical Functions

```
local math = require('math');
```

Refer: http://www.lua.org/manual/5.2/manual.html#6.6

- `math.abs (x)`
- `math.acos (x)`
- `math.asin (x)`
- `math.atan (x)`
- `math.atan2 (y, x)`
- `math.ceil (x)`
- `math.cos (x)`
- `math.cosh (x)`
- `math.deg (x)`
- `math.exp (x)`
- `math.floor (x)`
- `math.fmod (x, y)`
- `math.frexp (x)`
- `math.huge`
- `math.ldexp (m, e)`
- `math.log (x [, base])`
- `math.max (x, ···)`
- `math.min (x, ···)`
- `math.modf (x)`
- `math.pi`
- `math.pow (x, y)`
- `math.rad (x)`
- `math.random ([m [, n]])`
- `math.randomseed (x)`
- `math.sin (x)`
- `math.sinh (x)`
- `math.sqrt (x)`
- `math.tan (x)`
- `math.tanh (x)`

6.8 Bitwise Operations

```
local bit32 = require('bit32');
```

Refer: http://www.lua.org/manual/5.2/manual.html#6.7

- `bit32.arshift (x, disp)`
- `bit32.band (···)`
- `bit32.bnot (x)`
- `bit32.bor (···)`
- `bit32.btest (···)`
- `bit32.bxor (···)`
- `bit32.extract (n, field [, width])`
- `bit32.replace (n, v, field [, width])`
- `bit32.lrotate (x, disp)`
- `bit32.lshift (x, disp)`
- `bit32.rrotate (x, disp)`
- `bit32.rshift (x, disp)`

6.8.1 Luay Extension

```
local bit64 = require('bit64');
```

- `bit64.arshift (x, disp)`
- `bit64.band (···)`
- `bit64.bnot (x)`
- `bit64.bor (···)`
- `bit64.btest (···)`
- `bit64.bxor (···)`
- `bit64.extract (n, field [, width])`
- `bit64.replace (n, v, field [, width])`
- `bit64.lrotate (x, disp)`
- `bit64.lshift (x, disp)`
- `bit64.rrotate (x, disp)`
- `bit64.rshift (x, disp)`

6.9 IO Functions

```lua
local io = require('io');
```

Refer: http://www.lua.org/manual/5.2/manual.html#6.8

- `io.close ([file])`
- `io.flush ()`
- `io.input ([file])`
- `io.lines ([filename ···])`
- `io.open (filename [, mode]) -> file`
- `io.output ([file])`
- `io.popen (prog [, mode])`
- `io.read (···)`
- `io.tmpfile () -> file`
- `io.type (obj)`
- `io.write (···)`
- `file:close ()`
- `file:flush ()`
- `file:lines (···)`
- `file:read (···)`
- `file:seek ([whence [, offset]])`
- `file:setvbuf (mode [, size])`
- `file:write (···)`

6.10 Operating System Facilities

```
local os = require('os');
```

Refer: http://www.lua.org/manual/5.2/manual.html#6.9

- `os.clock ()`
- `os.date ([format [, time]])`
- `os.difftime (t2, t1)`
- `os.execute ([command])`
- `os.exit ([code [, close])`
- `os.getenv (varname)`
- `os.remove (filename)`
- `os.rename (oldname, newname)`
- `os.setlocale (locale [, category])`
- `os.time ([table])`
- `os.tmpname ()`

7 Luay Extension Libraries

7.1 hash Library

```
local hash = require('hash');
```

- hash.md5 (bytes) -> bytes
- hash.sha1 (bytes) -> bytes
- hash.sha256 (bytes) -> bytes
- hash.sha512 (bytes) -> bytes
- hash.to_hex (bytes) -> string
- hash.from_hex (string) -> bytes
- hash.to_b32 (bytes) -> string
- hash.from_b32 (string) -> bytes
- hash.to_b64 (bytes) -> string
- hash.from_b64 (string) -> bytes
- hash.to_b26 (bytes) -> string
- hash.to_b36 (bytes) -> string
- hash.to_b62 (bytes) -> string
- hash.to_uuid (bytes) -> string
- hash.to_xid (bytes) -> string
- hash.to_xxid (bytes, bytes) -> string

7.1.1 hash object

```
local _md = hash.new_hash('MD5');
_md:update('this');
_md:update('hello','world');
print('md5 hex =',hash.to_hex(_md:finish()))
```

7.1.2 mac object

```
local _mh = hash.new_mac('HMACMD5', 's3cr3t');
_mh:update('this');
_mh:update('hello','world');
print('hmacmd5 hex =',hash.to_hex(_mh:finish()))
```

7.2 LuaZDF inspired Libraries

7.2.1 array

```
local array = require('array');
```

- `array.array() -> array-only-table`
- `array.list() -> list-only-table`
- `array.push (list, value[, value ...]) -> list`
  Appends the values onto the list, returns the list.
- `array.pop (list) -> value`
  Removes the last element from the list and returns it.
- `array.unshift (list, value[, value ...]) -> list`
  Prepends the values to the list, returns the list.
- `array.shift (table) -> value`
  Removes the first element from the list and return it.
- `array.append (list, value[, value ...]) -> table`
  Alias for push.
- `array.appendall (table, tbl1[, ..., tblN]) -> table`
  Append all k,v from following tables to the first one, return the first.
- `array.chunk(list, size) -> list(lst1, ..., lstN)`
  Function to split array into /size/ chunks.
- `array.split(list, size) -> lst1, lst2`
  Function to split array in two at /size/.
- `array.collect (table[, ..., tableN]) -> list(v1, ..., vX)`
  Function to collect all values for the list of tables.
- `array.count (table, fv) -> table`
  Sorts a list into groups and returns a count for the number of objects in each group. Similar to groupBy, but instead of returning a list of values, returns a count for the number of values in that group.
- `array.difference (list1, list2) -> list`
  Computes the values of list1 that not occure in list2.
- `array.find( list, value [, init] ) -> v, i`
- `array.findif( list, fv [, init] ) -> v, i`
  Looks through an array table and returning the first element that matches.
- `array.indexof( list, value [, init] ) -> i`
- `array.indexof( list, fv [, init] ) -> i`
  Looks through an array table and returning the index of the first element that matches.

7.2.2 map

```
local map = require('map');
```

- `map.collectk (table[, ..., tableN]) -> list(k1, ..., kX)`
  Function to collect all keys for the list of tables.
- `map.collectv (table[, ..., tableN]) -> list(v1, ..., vX)`
  Function to collect all values for the list of tables.
- `map.collect (table[, ..., tableN]) -> table`
  Function to collect all k,v for the list of tables.
- `map.count (table, fv) -> table`
  Sorts a list into groups and returns a count for the number of objects in each group. Similar to groupBy, but instead of returning a list of values, returns a count for the number of values in that group.

8 Modules port/inpired from LuaRocks

8.1 stringy module

- `stringy.split(hay, sep) -> list`
- `stringy.strip(hay) -> trimmed`
- `stringy.startswith(hay, needle) -> bool`
- `stringy.endswith(hay, needle) -> bool`
- `stringy.count(hay, needle [, start]) -> offset of nil`
- `stringy.find(hay, needle [, start]) -> offset of nil`

8.1.1 luay extension

- `stringy.join(sep, hay1, ..., hayN) -> string`

9 Extending the Runtime with Libraries

These are my observations on how to add additional libraries to the runtime that can be `require`'d.

9.1 The old LuaJ Way.

In LuaJ each available library comes in 3 forms:

- A `.lua` file on the filesystem or classpath (ie. searchpath), using the standard lua way.
- A predefined/preloaded Java Class that extends from TwoArgFunction that is provided to the runtime at setuptime. This is the most restrictive option as the library must provide its own setup and the executer the registration code.
- A Java Class that extends from TwoArcFunction available on classpath, with exact naming conventions. This can either return a library or a single function.

9.1.1 Library Case

```
public class extlib extends TwoArgFunction
{
    public extlib() {}

    @Override
    public LuaValue call(LuaValue modname, LuaValue env) {
        // setup code
        LuaTable _extlib = new LuaTable();
        _extlib.set('extfunction', new _extfunction());
        env.set("extlib", _extlib);
        if (!env.get("package").isnil())
            env.get("package").get("loaded").set("extlib", _extlib);
        return _extlib;
    }

    static class _extfunction extends VarArgFunction
    {
        @Override
        public Varargs invoke(Varargs args) {
            // function code
            return ...;
        }
    }
}
```

9.1.2 Single Function Case

```
public class jtest extends TwoArgFunction
{
    @Override
    public LuaValue call(LuaValue _name, LuaValue _env) {
        return AbstractLibrary._zeroArgFunctionWrapper.from(_name.tojstring(),jtest::
    }

    public static LuaValue _call()
    {
        return LuaValue.TRUE;
    }
}
```

9.2 LuayBuilder support for preload.

```
luayContext = LuayBuilder.create()
    .inputStream(System.in)
    .outputStream(System.out)
    .errorStream(System.err)
    .baseLibraries()
    .extensionLibrary(new extlib())
    .build();
```

9.3 The Luay Way

Luay extends the library loader with a serviceloader, so classes can have arbitrary names and paths.

9.3.1 LuaySimpleLibraryFactory

This is the simplest option. You implement a glue-factory class that provides proper library name and object instance, the java object will be auto-coerced and all methods be exposed.

```
public class ExtLibFactory implements LuaySimpleLibraryFactory
{
    @Override
    public String getName() {
        return "extlib";
    }

    @Override
    public Object getInstance() {
        return Class.forName("ext.ExtLib").newInstance();
    }
}

public class ExtLib
{
    // library methods
    // ...
}
```

9.3.2 LuayLibraryFactory

If you need more control over which methods get exposed, use this option. Note: your class needs to extend AbstractLibrary and some setup code.

```
public class ExtLibFactory implements LuayLibraryFactory
{
    @Override
    public String getName() {
        return "extlib";
    }

    @Override
    @SneakyThrows
    public AbstractLibrary getInstance() {
        return (AbstractLibrary) Class.forName("ext.ExtLib").newInstance();
    }
}

public class ExtLib extends AbstractLibrary
{
    @Override
    public List<LuaFunction> getLibraryFunctions() {
```

```
        return toList(
                _varArgFunctionWrapper.from("extfunc", ExtLib::extFunc)
        );
    }

    public static LuaValue extFunc(Varargs args) {
        // function code ...
    }

}
```

10 Extending the Parser

These are my observations on how to add parse tokens and extend the language.

10.1 The Parser.jj file