LUA - WEB PROGRAMMING

Lua is a highly flexible language and it is often used in multiple platforms including web applications. The Kepler community that was formed in 2004 to provide open source web components in Lua.

Even though, there are other web frameworks using Lua that have been developed, we will be primarily focusing on the components provided by Kepler community.

Applications and Frameworks

- **Orbit** is an MVC web framework for Lua, based on WSAPI.
- **WSAPI** is the API that abstracts the web host server from Lua web applications and is the base for many projects.
- **Xavante** is a Lua Web server that offers a WSAPI interface.
- **Sputnik** is a wiki/CMS developed over WSAPI on Kepler Project used for humor and entertainment.
- **CGILua** offers LuaPages and LuaScripts web page creation, based on WSAPI but no longer supported. Use Orbit, Sputnik or WSAPI instead.

In this tutorial, we will try to make you understand what Lua can do and to know more about its installation and usage, refer kepler the website

Orbit

Orbit is an MVC web framework for Lua. It completely abandons the CGILua model of "scripts" in favor of applications, where each Orbit application can fit in a single file, but you can split it into multiple files if you want.

All Orbit applications follow the WSAPI protocol, so they currently work with Xavante, CGI and Fastcgi. It includes a launcher that makes it easy to launch a Xavante instance for development.

The easiest way to install Orbit is using LuaRocks. Luarocks install orbit is the command for installing. For this, you need to install LuaRocks first.

If you haven't installed all the dependencies, here are the steps to be followed to setup Orbit in Unix/Linux environment.

Installing Apache

Connect to your server. Install Apache2, its support modules and enable required Apache2 modules using −

```
$ sudo apt-get install apache2 libapache2-mod-fcgid libfcgi-dev build-essential
$ sudo a2enmod rewrite
$ sudo a2enmod fcgid
$ sudo /etc/init.d/apache2 force-reload
```

Install LuaRocks

```
$ sudo apt-get install luarocks
```

Install WSAPI, FCGI, Orbit, and Xavante

```
$ sudo luarocks install orbit
$ sudo luarocks install wsapi-xavante
$ sudo luarocks install wsapi-fcgi
```

Setting up Apache2

```
$ sudo raj /etc/apache2/sites-available/default
```

Add this following section below the <Directory /var/www/> section of the config file. If this section has an 'AllowOverride None' then you need to change the 'None' to 'All' so that the .htaccess file can override the configuration locally.

```
<IfModule mod_fcgid.c>

   AddHandler fcgid-script .lua
   AddHandler fcgid-script .ws
   AddHandler fcgid-script .op

   FCGIWrapper "/usr/local/bin/wsapi.fcgi" .ws
   FCGIWrapper "/usr/local/bin/wsapi.fcgi" .lua
   FCGIWrapper "/usr/local/bin/op.fcgi" .op

   #FCGIServer "/usr/local/bin/wsapi.fcgi" -idle-timeout 60 -processes 1
   #IdleTimeout 60
   #ProcessLifeTime 60

</IfModule>
```

Restart the server to ensure the changes made comes into effect.

To enable your application, you need to add +ExecCGI to an .htaccess file in the root of your Orbit application − in this case, /var/www.

```
Options +ExecCGI
DirectoryIndex index.ws
```

Simple Example − Orbit

```
#!/usr/bin/env index.lua

-- index.lua
require"orbit"

-- declaration
module("myorbit", package.seeall, orbit.new)

-- handler

function index(web)
   return my_home_page()
end

-- dispatch
myorbit:dispatch_get(index, "/", "/index")

-- Sample page

function my_home_page()

   return [[
      <head></head>
      <html>
         <h2>First Page</h2>
      </html>
   ]]

end
```

Now, you should be able to launch your web browser. Go to http://localhost:8080/ and you should see the following output −

```
First Page
```

Orbit provides another option, i.e., Lua code can generate html.

```
#!/usr/bin/env index.lua

-- index.lua
require"orbit"

function generate()
   return html {
      head{title "HTML Example"},

      body{
         h2{"Here we go again!"}
      }
   }
end

orbit.htmlify(generate)

print(generate())
```

Creating Forms

A simple form example is shown below −

```
#!/usr/bin/env index.lua
require"orbit"

function wrap (inner)
   return html{ head(), body(inner) }
end

function test ()
   return wrap(form (H'table' {
      tr{td"First name",td( input{type='text', name='first'})},
      tr{td"Second name",td(input{type='text', name='second'})},
      tr{ td(input{type='submit', value='Submit!'}),
         td(input{type='submit',value='Cancel'})
      },
   }))
end

orbit.htmlify(wrap,test)

print(test())
```

WSAPI

As mentioned earlier, WSAPI acts as the base for many projects and have multiple features embedded in it. You can use WSAPI and support the following platforms,

- Windows
- UNIX-based systems

The supported servers and interfaces by WSAPI includes,

- CGI
- FastCGI
- Xavante

WSAPI provides a number of libraries, which makes it easier for us in web programming using Lua. Some of the supported features in Lua includes,

- Request processing
- Output buffering
- Authentication
- File uploads
- Request isolation
- Multiplexing

A simple example of WSAPI is shown below −

```
#!/usr/bin/env wsapi.cgi

module(..., package.seeall)
function run(wsapi_env)
   local headers = { ["Content-type"] = "text/html" }

   local function hello_text()
      coroutine.yield("<html><body>")
      coroutine.yield("<p&gtHello Wsapi!</p>")
      coroutine.yield("<p&gtPATH_INFO: " .. wsapi_env.PATH_INFO .. "</p>")
      coroutine.yield("<p&gtSCRIPT_NAME: " .. wsapi_env.SCRIPT_NAME .. "</p>")
      coroutine.yield("</body></html>")
   end

   return 200, headers, coroutine.wrap(hello_text)
end
```

You can see in the above code a simple html page is formed and returned. You can see the usage of coroutines that makes it possible to return statement by statement to calling function. Finally, html status code200, headers and html page is returned.

Xavante

Xavante is a Lua HTTP 1.1 Web server that uses a modular architecture based on URI mapped handlers. Xavante currently offers,

- File handler
- Redirect handler
- WSAPI handler

File handler is used for general files. Redirect handler enables URI remapping and WSAPI handler for handing with WSAPI applications.

A simple example is shown below.

```
require "xavante.filehandler"
require "xavante.cgiluahandler"
require "xavante.redirecthandler"

-- Define here where Xavante HTTP documents scripts are located
local webDir = XAVANTE_WEB

local simplerules = {

   { -- URI remapping example
      match = "^[^%./]*/$",
      with = xavante.redirecthandler,
      params = {"index.lp"}
   },
```

```
   { -- cgiluahandler example
      match = {"%.lp$", "%.lp/.*$", "%.lua$", "%.lua/.*$" },
      with = xavante.cgiluahandler.makeHandler (webDir)
   },

   { -- filehandler example
      match = ".",
      with = xavante.filehandler,
      params = {baseDir = webDir}
   },
}

xavante.HTTP{
   server = {host = "*", port = 8080},

   defaultHost = {
      rules = simplerules
   },
}
```

To use virtual hosts with Xavante, the call to xavante.HTTP would be changed to something like as follows −

```
xavante.HTTP{
   server = {host = "*", port = 8080},

   defaultHost = {},

   virtualhosts = {
      ["www.sitename.com"] = simplerules
   }
}
```

Lua Web Components

- **Copas**, a dispatcher based on coroutines that can be used by TCP/IP servers.
- **Cosmo**, a "safe templates" engine that protects your application from arbitrary code in the templates.
- **Coxpcall** encapsulates Lua native pcall and xpcall with coroutine compatible ones.
- **LuaFileSystem**, a portable way to access the underlying directory structure and file attributes.
- **Rings**, a library which provides a way to create new Lua states from within Lua.

Ending Note

There are so many Lua based web frameworks and components available for us and based on the need, it can be chosen. There are other web frameworks available which include the following −

- **Moonstalk** enables efficient development and hosting of dynamically generated web-based projects built with the Lua language; from basic pages to complex applications.
- **Lapis**, a framework for building web applications using MoonScript *orLua* that runs inside of a customized version of Nginx called OpenResty.
- **Lua Server Pages**, a Lua scripting engine plug-in that blows away any other approach to embedded web development, offers a dramatic short cut to traditional C server pages.

These web frameworks can leverage your web applications and help you in doing powerful operations.