Lua with Database

About

Lua scripts allow one to connect to databases and external resources. Two Lua options are available if you are developing a FreeSWITCH application interfacing with a DBMS.

Accessing a Database

There are two ways to access databases from Lua: LuaSQL and freeswitch.Dbh. The Dbh method is preferred because it uses database handle pooling which results in greater efficiency and speed.

With LuaSQL, you can query not only an ODBC data source, but also PostgreSQL, Oracle, or MySQL via native drivers. The LuaSQL module itself will make the connection to your database. LuaSQL does not provide connection pooling as is possible with freeswitch.Dbh.

LuaSQL

The most common alternative for Lua developers is to use LuaSQL (http://keplerproject.org/luasql). With LuaSQL you can easily connect to ODBC, ADO, Oracle, MySQL, SQLite and PostgreSQL databases.

LuaSQL Example

```lua
-- this example of how to connect to postgres from Lua
local luasql = require "luasql.postgres"
--
-- Database connection settings
--
DBHOST = '127.0.0.1'
DBNAME = 'dbname'
DBUSER = 'dbusername'
DBPASS = 'dbpassword'
DBPORT = 5432

local env = assert (luasql.postgres())
local dbcon = env:connect(DBNAME, DBUSER, DBPASS, DBHOST, DBPORT)
sql = 'SELECT pin, name FROM customer'
res, serr = dbcon:execute(sql)
dbcon:close()
env:close()
```

freeswitch.Dbh

The FreeSWITCH Database Handler

The second solution is using the freeswitch.Dbh. Advantage of this method is that it makes use of connection pooling provided by FreeSWITCH which gives a nice increase in speed when compared to creating a new TCP connection for each LuaSQL env:connect().

It works as follows:
local dbh = freeswitch.Dbh("dsn","user","pass") -- when using ODBC (deprecated)
-- OR --
local dbh = freeswitch.Dbh("core:my_db") -- when using sqlite (deprecated, if you have to use this to make it
work you should upgrade your FS installation)
-- OR --
local dbh = freeswitch.Dbh("sqlite://my_db") -- sqlite database in subdirectory "db"
-- OR --
local dbh = freeswitch.Dbh("odbc://my_db:uname:passwd") -- connect to ODBC database
assert(dbh:connected()) -- exits the script if we didn't connect properly

dbh:test_reactive("SELECT * FROM my_table",
   "DROP TABLE my_table",
   "CREATE TABLE my_table (id INTEGER, name VARCHAR)"
) dbh:query("INSERT INTO my_table VALUES(1, 'foo')") -- populate the table
dbh:query("INSERT INTO my_table VALUES(2, 'bar')") -- with some test data

dbh:query("SELECT id, name FROM my_table", function(row)
   stream:write(string.format("%5s : %s
", row.id, row.name))
end)

dbh:query("UPDATE my_table SET name = 'changed'"
stream:write("Affected rows: " .. dbh:affected_rows() .. "\n")

dbh:release() -- optional

* freeswitch.Dbh("odbc://my_db:uname:passwd") gets an ODBC db handle from the pool.
* freeswitch.Dbh("sqlite://my_db") gets a core db (sqlite) db handle from the pool (this automatically creates the db if it didn't exist yet).
* dbh:connected() checks if the handle is still connected to the database, returns true if connected, false otherwise.
* dbh:test_reactive("test_sql", "drop_sql", "reactive_sql") performs test_sql and if it fails performs drop_sql and reactive_sql (handy for initial table creation purposes).
* dbh:query("query", function()) takes the query as a string and an optional Lua callback function that is called on each row returned by the db.
  * The callback function is passed a table representation of the current row for each iteration of the loop.
  * Syntax of each row is: [{"column_name_1"} = "value_1", {"column_name_2"} = "value_2"].
  * If you (optionally) return a number other than 0 from the callback-function, you'll break the loop.
* dbh:affected_rows() returns the number of rows affected by the last run INSERT, DELETE or UPDATE on the handle. It does not respond to SELECT operations.
* dbh:release() (optional) releases the handle back to the pool so it can be re-used by another thread. This is also automatically done when the dbh goes out of scope and is garbage collected (for example when your script returns). Useful for long-running scripts to release the connection sooner.

freeswitch.Dbh example

-- this is an example of how to connect to Postgres using ODBC from Lua
local dbh = freeswitch.Dbh("odbc://my_db:uname:passwd") -- connect to ODBC database
assert(dbh:connected()) -- exits the script if we didn't connect properly
dbh:query("SELECT pin, name FROM customer", function(row)
   stream:write(string.format("%5s : %s\n", row.pin, row.name))
end)

dbh:release() -- optional

See Also

Lua FreeSWITCH Dbh