**调用其他 C 函数动态库**

Linux 下的动态库一般都以 .so 结束命名，而 Windows 下一般都以 .dll 结束命名。Lua 作为一种嵌入式语言，和 C 具有非常好的亲缘性，这也是 Lua 赖以生存、发展的根本，所以 Nginx + Lua=OpenResty，魔法就这么神奇的发生了。

NgxLuaModule 里面尽管提供了十分丰富的 API，但他一定不可能满足我们的形形色色的需求。我们总是要和各种组件、算法等形形色色的第三方库进行协作。那么如何在 Lua 中加载动态加载第三方库，就显得非常有用。

扯一些额外话题，Lua 解释器目前有两个最主流分支。

* Lua 官方发布的标准版[Lua](http://lua.org/)
* Google 开发维护的[LuaJIT](http://luajit.org/index.html)

LuaJIT 中加入了 Just In Time 等编译技术，使得 Lua 的解释、执行效率有非常大的提升。除此以外，还提供了[FFI](http://luajit.org/ext_ffi.html)。

什么是FFI？

The FFI library allows calling external C functions and using C data

structures from pure Lua code.

通过 FFI 的方式加载其他 C 接口动态库，这样我们就可以有很多有意思的玩法。

当我们碰到 CPU 密集运算部分，我们可以把他用 C 的方式实现一个效率最高的版本，对外导出 API，打包成动态库，通过 FFI 来完成 API 调用。这样我们就可以兼顾程序灵活、执行高效，大大弥补了 LuaJIT 自身的不足。

使用FFI判断操作系统

local ffi = require("ffi")

if ffi.os == "Windows" then

print("windows")

elseif ffi.os == "OSX" then

print("MAC OS X")

else

print(ffi.os)

end

调用zlib压缩库

local ffi = require("ffi")

ffi.cdef[[

unsigned long compressBound(unsigned long sourceLen);

int compress2(uint8\_t \*dest, unsigned long \*destLen,

const uint8\_t \*source, unsigned long sourceLen, int level);

int uncompress(uint8\_t \*dest, unsigned long \*destLen,

const uint8\_t \*source, unsigned long sourceLen);

]]

local zlib = ffi.load(ffi.os == "Windows" and "zlib1" or "z")

local function compress(txt)

local n = zlib.compressBound(#txt)

local buf = ffi.new("uint8\_t[?]", n)

local buflen = ffi.new("unsigned long[1]", n)

local res = zlib.compress2(buf, buflen, txt, #txt, 9)

assert(res == 0)

return ffi.string(buf, buflen[0])

end

local function uncompress(comp, n)

local buf = ffi.new("uint8\_t[?]", n)

local buflen = ffi.new("unsigned long[1]", n)

local res = zlib.uncompress(buf, buflen, comp, #comp)

assert(res == 0)

return ffi.string(buf, buflen[0])

end

-- Simple test code.

local txt = string.rep("abcd", 1000)

print("Uncompressed size: ", #txt)

local c = compress(txt)

print("Compressed size: ", #c)

local txt2 = uncompress(c, #txt)

assert(txt2 == txt)

自定义定义C类型的方法

local ffi = require("ffi")

ffi.cdef[[

typedef struct { double x, y; } point\_t;

]]

local point

local mt = {

\_\_add = function(a, b) return point(a.x+b.x, a.y+b.y) end,

\_\_len = function(a) return math.sqrt(a.x\*a.x + a.y\*a.y) end,

\_\_index = {

area = function(a) return a.x\*a.x + a.y\*a.y end,

},

}

point = ffi.metatype("point\_t", mt)

local a = point(3, 4)

print(a.x, a.y) --> 3 4

print(#a) --> 5

print(a:area()) --> 25

local b = a + point(0.5, 8)

print(#b) --> 12.5

Lua和LuaJIT对比

可以这么说，LuaJIT 应该是全面胜出，无论是功能、效率都是标准 Lua 不能比的。目前最新版 OpenResty 默认也都使用 LuaJIT。

世界为我所用，总是有惊喜等着你，如果哪天你发现自己站在了顶峰，那我们就静下心来改善一下顶峰，把他推到更高吧。