Lua热更新原理(1) - require机制

在本篇里，我教大家实现一种最简单，但也是问题最多的一种热更新，它是我后面文章的基础。

require 做了什么？

把下面的代码写到一个lua文件里，比如叫`example.lua`

```
local function print_some()
    print("something")
end
return print_some
```

那么`require "example"`可以用下面的语句实现

```
if package.loaded["example"] == nil then    --package是默认就有[
    local function f()
        local function print_some()         --注意只有这四行是
            print("something")             --example.lua的内容
        end
        return print_some
    end
    local result = f()
    if result == nil then
        package.loaded["example"] = true
    else
        package.loaded["example"] = result
    end
end
return package.loaded["example"]
```

根据这个语义，多次执行`require "exmaple.lua"`，也只有第一次执行exmaple里的内容，执行之后的返回值会缓存到`package.loaded`里，再次require就不会执行`example.lua`，而是直接返回`package.loaded["example"]`。

如果你改写了example的内容，然后想require改写之后的example，那怎么办呢？只需要先执行`package.loaded["example.lua"] = nil`，那么再`require "exmaple"`时，就会得到新的example。如下面的代码

```
package.loaded["example"] = nil
local func = require "example"
```

每次调用这两句，func都会是新的，这相当于实现了热更新，别高兴，这只适用于非常简单的情况。工程上没法这么用，别的不说，首先每次require都要先调用`package.loaded["example"] = nil`，这需要修改逻辑代码。其次因为example的内容会重新执行一次，会重新执行你不想要执行的代码，假如`example.lua`是这么写的

```
global_var = 0
local function print_some()
    print("something")
end
return print_some
```

那么每一次“热更新”都会把全局变量global_var置为0，这是你想要结果吗？当然这种热更新还有一个严重的问题，就是upvalue被重置了，这在lua热更新原理(2) - upvalue讨论。

Written on January 23, 2016