Lua 下的基础 OOP 框架实现

这个是在我并不了解 metatable（元表）这一个 Lua 本具有的十分良好的特性的时候苦苦构想出来的一个实现。

如同最初所想，这样一个框架，至少可以定义类（Class），并且可以生成类的具体实现对象（Object），然后，在类与类之间，还要有继承（Inheritance）等基础关系。另外由于曾看过 javascript 的一些特性，所以，想到了 Prototype（原型），也准备尝试加入以丰富其内容。

首先想到的是，要区分对象和类，必须从其在 Lua 中的内部类型表示来区分，那么根据大致的特点，就以 table 作为对象，以 function 作为类。

那么就得到了 new(TYPE)运算的类似实现：

```
--new()
local new = function (TYPE)
    if type(TYPE) == "table" then
        return Object(GetConstructor(TYPE))
    elseif type(TYPE) == "function" then
        return Object(TYPE)
    else
        assert(TYPE,"Error: Unknown TYPE.")
    end
end
```

（type 和 assert 为 Lua 内置的函数。）

new()函数会判断传入的参数类型，然后根据其类型选择返回相应的构造器，从而得到相应对象。Object()所需的参数就是相应的类名（即 Lua 中类型为 function 的值）：

```
local Object = function (TYPE)
    local OBJ = TYPE and TYPE() or { _type = "Object", prototype = "Object" }
    OBJ.prototype = TYPE and TYPE() or { _type = "Object", prototype =
"Object" }
    OBJ.constructor = TYPE
    return OBJ
end
```

（and、or 运算符支持短路原则。）

如上：若 TYPE 参数为空或者值为 nil（未定义），将返回一个统一的 OBJ：_type 值为"Object"，prototype 值为自身。

即：Object()方法仍然会返回一个对象，这个对象就是一切对象的基础（参考 Javascript 中的 Object）。

而 OBJ 的 constructor 值则会被赋为该类型，这样就可以通过 GetConstructor()

获取并返回。

而利用这个方法，就可以通过 new() 运算来使用对象创建对象：

```
local GetConstructor
    GetConstructor = function (TYPE)
    if TYPE.prototype == "Object" then
        return Object
    elseif TYPE.constructor then
        return TYPE.constructor
    else
        return GetConstructor(TYPE.prototype)
    end
end
```

（开始的先声明后赋值的是为了最后那句递归调用。）

嗯，似乎这样就差不多了。就是创建类貌似有些麻烦。可以通过下面的示例代码体会下：

```
local Animal = function ()
    return {
        _type = "Animal",
        walk = function () print("I am walking") end
    }
end

Tiger = new(Animal)
Tiger.walk()
```

嗯，有感觉是有感觉了，甚至可以通过 Animal().walk() 这种东西来调用方法。但是，问题总会出在各个方面：继承怎么办？Tiger 是否能够继续作为类使用？这种类定义方法感觉起来是否略显不方便了？

于是我想到了 C++中的 class 关键字。而如果想通过 Lua 实现类似的功能，那就要定义一个接受参数为 table 返回值为 function 的函数（其实还是 function）。

有了这个概念就基本上能够实现了。

```
--Class()
local Class = function (DEF)
    local CONS = function ()
        local BODY = {}
        for k,v in pairs(DEF) do
            BODY[k] = v
        end
        BODY.prototype = BODY.prototype or Object(CONS)
        return BODY
    end
    return CONS
end
```

Class内部定义了一个CONS临时函数，CONS通过for迭代获取DEF的值存放在BODY中，然后返回（这是构造对象）。然后Class 函数将CONS 返回（这是构造类）。

就这么简单。不然你真就想多了。

我们就进行再一次测试：

```
local Animal = Class({
    _type = "Animal",
    walk = function () print("I am walking.") end
})

Tiger = new(Animal)
Tiger.walk()

Tiger.run = function () print("I am running.") end
Tiger.run()

Tiger._type = "Tiger"
Tiger = Class(Tiger)
Sam = new(Tiger)

Sam.walk()
Sam.run()
```

输出结果：

```
I am walking.
I am running.
I am walking.
I am running.
```

于是Tiger 类就这样继承了Animal 类，Sam 成了 Tiger 类的一个实例对象。

就这么简单。

接上一段程序，下面是关于继承（原型）的测试：

```
print(Sam._type)    --Type of Sam
print(Sam.prototype._type) --Type of Tiger
print(Sam.prototype.prototype._type)   --Type of Animal
print(Sam.prototype.prototype.prototype._type) --Type of Animal.prototype
```

输出结果：

```
Tiger
Tiger
Animal
Object
```

如同所期待的那样。

这样我们就通过简单的几十行，800 多字符，接近 1KB 大小的代码量就完成了一个可以通过new()、Class()和Object()来构造简单 OOP 结构的基础对象构造系统。并且仅仅使用了三个 Lua 标准内置函数，可移植性绝对毋庸置疑。但是因为对于内存回收机制等

方面了解不太清楚，再加上多继承等其他方面的内容不太怎么好实现，于是就暂时停顿在了这个地方。但是，相对于可用性来说，整个实现过程是更有价值的。

本文代码在 Lua 5.1.4 下测试通过。

相关内容取自 KPSN #!/Required 项目。
详情访问项目网站。

版本：Feb./07/2012 v1
代码高亮效果由 Notepad++及 NppExport 提供。
PDF 版本由 WPS 文字导出