前言

这篇博文将通过几个简单的实例演示，巩固对__index和__newindex的理解，同时加深对Lua中元表和元方法的理解，如果对Lua的元表和元方法还不是很熟悉的话，请参考这篇文章：《Lua中的元表与元方法 (http://www.jellythink.com/archives/511)》。

具有默认值的table

我们都知道，table中的任何字段的默认值都是nil，但是通过元表，我们可以很容易的修改这一规定，代码如下：

```
function setDefault(tb, defaultValue)
     local mt = {__index = function () return defaultValue end}
     setmetatable(tb, mt)
end

local tb1 = {x = 10, y = 20}
print(tb1.x, tb1.z)     --> 10 nil
setDefault(tb1, 100) -->设置默认值
print(tb1.x, tb1.z) --> 10 100 这里打印的就是默认值
```

可以看到，在代码中，setDefault函数为所有需要默认值的table创建了一个新的元表。如果准备创建很多需要默认值得table，这种方法的开销或许就比较大了。由于在元表中默认值defaultValue是与元方法关联在一起的，所以setDefault无法为所有table都使用同一个元表。如果要让具有不同默认值得table都使用同一个元表，那么就需要将每个元表的默认值存放在table本身中，可以使用一个额外的字段来存储默认值。例如以下代码：

```
local mt = {__index = function (t) return t.___ end}
function setDefault(tb, defaultValue)
     tb.___ = defaultValue       -- 非常谢谢hellowei犀利的review。具体请参见评论
     setmetatable(tb, mt)
end
```

上面代码中的“___”是为了防止名字冲突而起的名字；如果这样的话，你还担心名字冲突，确保key在table中的唯一性，只需要创建一个新的table，并用它作为key即可，每一个新创建的table都是一个唯一的地址，比如以下代码：

```
local key = {} -- 唯一的key
local mt = {__index = function (tb) return tb[key] end}

function setDefault(tb, defaultValue)
     tb[key] = defaultValue
     setmetatable(tb, mt)
end
```

记录table的访问

有的时候，一种特定的需求，我们需要记录对一个table的所有访问，不管是查询还是更新，我们都需要记录日志。这如何完成？我们都知道，元表中的__index和__newindex是在table中没有所需要访问的index时才发挥作用的，因此，只有将一个table保持为空，然后设置__index和__newindex元方法，才有可能记录下来所有对它的访问。

为了监视一个table的所有访问，就应该为真正的table创建一个代理。这个代理就是一个空的table，其中__index和__newindex元方法可用于跟踪所有的访问，并将访问重定义到原来的table上。这就是思路，接下来看代码：

```
local t = {} --原来的table

-- 保持对原table的一个引用
local _t = t

-- 创建代理
t = {}

-- 创建元表
local mt = {
__index = function (t, k)
print("access to element " .. tostring(k))
return _t[k]
end,

__newindex = function (t, k, v)
print("update of element " .. tostring(k))
_t[k] = v
end
}

setmetatable(t, mt)

t.x = 10 -- update of element x
print(t.x) -- access to element x
```

如果想要同时监视几个table，无须为每个table创建不同的元表；相反，只要以某种形式将每个代理与其原table关联起来，并且所有代理都共享一个公共的元表。这个问题与设置table默认值相关联的问题类似，也是将原来的table保存在代理table的一个特殊的字段中。代码如下：

```
-- 创建唯一索引
local index = {}

-- 创建元表
local mt = {
     __index = function (t, k)
          print("access to element " .. tostring(k))
          return t[index][k]
     end,

     __newindex = function (t, k, v)
          print("update of element " .. tostring(k))
          t[index][k] = v
     end
}

function track(t)
     local proxy = {}
     proxy[index] = t
     setmetatable(proxy, mt)
     return proxy
end

local t = {}
local proxy = track(t)
proxy.x = 10
print(proxy.x)
```

只读的table

通过代理的概念，可以很容易的实现只读的table。只需要跟踪所有对table的更新操作，并引发一个错误就好了，对于查询时，我们不用去馆，只需要管对table的更新操作，废话不说，来段简单的代码，自然而然的一目了然了。

```
function readOnly(t)
     local proxy = {}

     -- 创建元表
     local mt = {
          __index = t,
          __newindex = function (t, k, v)
               error("Attempt to update a read-only table", 2)
          end
     }

     setmetatable(proxy, mt)
     return proxy
end

local tbDemo = readOnly{1, 2, 3, 4, 5}
print(tbDemo[1])
tbDemo[1] = 20
```

元表中__index对应的是原来的table，而更新原来的table时，就会显示错误提示：Attempt to update a read-only table。

总结

这篇文章对Lua中的__index和__newindex的使用进行了详细的讲解和分析，并提供了实际的代码，主要是为了加深对Lua中元表和元方法的理解，元表和元方法在Lua中的地位太总要了，很多高级的编程技巧和特殊需求都是基于元表和元方法来实现了，所以，也希望大家能好好的阅读这篇文章，同时也希望我的文章对大家有帮助。

2014年7月19日 于深圳。

PS：有朋友在微信公众账号给我回消息，很抱歉，我刚刚才看到，对于你问的问题，我再这里做个简单的回答：
>>我是如何学习的？
每个人的学习方法都是不一样的，找到你最喜欢的学习方法就好了，也不要刻意的去模仿别。我就随便说说我的学习方法吧。看书->看博客->写博客（总结，心得）->项目实践。你学习的东西，最好要和你最近在做的项目相关，这样你就可以在项目中施展你最近学到的东西。

>>我看书么？
每天上班都很忙，下班到家，洗个澡，再忙些别的东西，基本就没有时间了，但是我还是抓紧一些可以利用的时间看书。

以上是我回答的你的问题。希望对你有用。最后，也希望继续关注“果冻想”。同时，也谢谢对我的关注和提出的意见。大家可以扫描以下二维码，关注“果冻想”。