Lua 元表(Metatable)

在 Lua table 中我们可以访问对应的 key 来得到 value 值，但是却无法对两个 table 进行操作。

因此 Lua 提供了元表(Metatable)，允许我们改变 table 的行为，每个行为关联了对应的元方法。

例如，使用元表我们可以定义 Lua 如何计算两个 table 的相加操作 a+b。

当 Lua 试图对两个表进行相加时,先检查两者之一是否有元表,之后检查是否有一个叫"__add"的字段，若找到，则调用对应的值。"__add"等即时字段，其对应的值（往往是一个函数或是 table）就是"元方法"。

有两个很重要的函数来处理元表：

setmetatable(table,metatable): 对指定 table 设置元表(metatable)，如果元表(metatable)中存在__metatable 键值，setmetatable 会失败 。

getmetatable(table): 返回对象的元表(metatable)。

以下实例演示了如何对指定的表设置元表：

```
mytable = {}                          -- 普通表
mymetatable = {}                      -- 元表
setmetatable(mytable,mymetatable)     -- 把 mymetatable 设为 mytable 的元表
```

以上代码也可以直接写成一行：

```
mytable = setmetatable({},{})
```

以下为返回对象元表：

```
getmetatable(mytable)                 -- 这回返回 mymetatable
```

__index 元方法

这是 metatable 最常用的键。

当你通过键来访问 table 的时候，如果这个键没有值，那么 Lua 就会寻找该 table 的 metatable（假定有 metatable）中的__index 键。如果__index 包含一个表格，Lua 会在表格中查找相应的键。

我们可以在使用 lua 命令进入交互模式查看：

```
$ luaLua 5.3.0  Copyright (C) 1994-2015 Lua.org, PUC-Rio> other = { foo = 3 } > t = setmetatable({}, { __index = other }) > t.foo3> t.barnil
```

如果__index 包含一个函数的话，Lua 就会调用那个函数，table 和键会作为参数传递给函数。

__index 元方法查看表中元素是否存在，如果不存在，返回结果为 nil；如果存在则由 __index 返回结果。

```
mytable = setmetatable({key1 = "value1"}, {
  __index = function(mytable, key)
    if key == "key2" then
      return "metatablevalue"
    else
      return nil
    end
  end})
print(mytable.key1,mytable.key2)
```

实例输出结果为：

```
value1    metatablevalue
```

实例解析：

mytable 表赋值为 **{key1 = "value1"}**。

mytable 设置了元表，元方法为 __index。

在 mytable 表中查找 key1，如果找到，返回该元素，找不到则继续。

在 mytable 表中查找 key2，如果找到，返回 metatablevalue，找不到则继续。

判断元表有没有__index 方法，如果__index 方法是一个函数，则调用该函数。

元方法中查看是否传入 "key2" 键的参数（mytable.key2 已设置），如果传入 "key2" 参数返回 "metatablevalue"，否则返回 mytable 对应的键值。

我们可以将以上代码简单写成：

```
mytable = setmetatable({key1 = "value1"}, { __index = { key2 = "metatablevalue" } })print(mytable.key1,mytable.key2)
```

总结

Lua 查找一个表元素时的规则，其实就是如下 3 个步骤:

- 1.在表中查找，如果找到，返回该元素，找不到则继续
- 2.判断该表是否有元表，如果没有元表，返回 nil，有元表则继续。
- 3.判断元表有没有__index 方法，如果__index 方法为 nil，则返回 nil；如果__index 方法是一个表，则重复 1、2、3；如果__index 方法是一个函数，则返回该函数的返回值。

__newindex 元方法

__newindex 元方法用来对表更新，__index 则用来对表访问 。

当你给表的一个缺少的索引赋值，解释器就会查找__newindex 元方法：如果存在则调用这个函数而不进行赋值操作。

以下实例演示了 __newindex 元方法的应用：

```
mymetatable = {}

mytable = setmetatable({key1 = "value1"}, { __newindex = mymetatable })

print(mytable.key1)
```

```
mytable.newkey = "新值 2"print(mytable.newkey,mymetatable.newkey)


mytable.key1 = "新值 1"print(mytable.key1,mymetatable.key1)
```

以上实例执行输出结果为：

```
value1nil    新值 2 新值 1    nil
```

以上实例中表设置了元方法 __newindex，在对新索引键（newkey）赋值时（mytable.newkey = "新值 2"），会调用元方法，而不进行赋值。而如果对已存在的索引键（key1），则会进行赋值，而不调用元方法 __newindex。

以下实例使用了 rawset 函数来更新表：

```
mytable = setmetatable({key1 = "value1"}, {
  __newindex = function(mytable, key, value)
    rawset(mytable, key, "\""..value.."\"")


  end})


mytable.key1 = "new value"
mytable.key2 = 4
print(mytable.key1,mytable.key2)
```

以上实例执行输出结果为：

```
new value    "4"
```

为表添加操作符

以下实例演示了两表相加操作：

```
-- 计算表中最大值，table.maxn 在 Lua5.2 以上版本中已无法使用-- 自定义计算表中最大键值函数 table_maxn，即计算表的元素个数 function table_maxn(t)
    local mn = 0
    for k, v in pairs(t) do
        if mn < k then
            mn = k
        end
    end
    return mnend
-- 两表相加操作
mytable = setmetatable({ 1, 2, 3 }, {
  __add = function(mytable, newtable)
    for i = 1, table_maxn(newtable) do
      table.insert(mytable, table_maxn(mytable)+1,newtable[i])
    end
    return mytable
  end})

secondtable = {4,5,6}

mytable = mytable + secondtable
  for k,v in ipairs(mytable) doprint(k,v)end
```

以上实例执行输出结果为：

```
1  12  23  34  45  56  6
```

__add 键包含在元表中，并进行相加操作。 表中对应的操作列表如下：(**注意：**__是两个下划线)

模式	描述
__add	对应的运算符 '+'.
__sub	对应的运算符 '-'.
__mul	对应的运算符 '*'.
__div	对应的运算符 '/'.
__mod	对应的运算符 '%'.
__unm	对应的运算符 '-'.
__concat	对应的运算符 '..'.
__eq	对应的运算符 '=='.
__lt	对应的运算符 '<'.
__le	对应的运算符 '<='.

__call 元方法

__call 元方法在 Lua 调用一个值时调用。以下实例演示了计算表中元素的和：

```
-- 计算表中最大值，table.maxn 在 Lua5.2 以上版本中已无法使用-- 自定义计算表中最大键值函数 table_max
n，即计算表的元素个数 function table_maxn(t)

    local mn = 0

    for k, v in pairs(t) do

        if mn < k then

            mn = k

        end

    end

    return mnend

-- 定义元方法__call
```

```
mytable = setmetatable({10}, {
  __call = function(mytable, newtable)
  sum = 0
  for i = 1, table_maxn(mytable) do
    sum = sum + mytable[i]
  end
    for i = 1, table_maxn(newtable) do
    sum = sum + newtable[i]
  end
  return sum
  end})
newtable = {10,20,30}print(mytable(newtable))
```

以上实例执行输出结果为：

```
70
```

__tostring 元方法

__tostring 元方法用于修改表的输出行为。以下实例我们自定义了表的输出内容：

```
mytable = setmetatable({ 10, 20, 30 }, {
  __tostring = function(mytable)
    sum = 0
    for k, v in pairs(mytable) do
    sum = sum + v
  end
    return "表所有元素的和为 " .. sum
```

```
end})print(mytable)
```

以上实例执行输出结果为：

```
表所有元素的和为 60
```

从本文中我们可以知道元表可以很好的简化我们的代码功能，所以了解 Lua 的元表，可以让我们写出更加简单优秀的 Lua 代码。