Lua 模块与包 Lua 协同程序(coroutine)

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，set metatable 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：

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

```
3
> t.bar
nil
```

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。

该部分内容来自作者寰子：https://blog.csdn.net/xocoder/article/details/9028347

__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：

```
value1
nil    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 mn
end

-- 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) do
print(k,v)
end
```

以上实例执行输出结果为：

```
1    1
2    2
```

```
3    3
4    4
5    5
6    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_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 mn
end

-- 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。

← Lua 模块与包 Lua 协同程序(coroutine) →

实现 `__index` 元方法:

```
text = { }
text.defaultValue = { size = 14, content = "hello" }
text.mt = { }  -- 创建元表

function text.new( a )
  setmetatable( a, text.mt )
  return a
end

text.mt.__index = function( tb, key )
  return text.defaultValue[key]
end

local x = text.new{ content = "bye" }
print( x.size )   --> 14
```

毕月乌 10个月前 (05-24)

Lua 算术运算的 Metamethods

这一部分我们通过一个简单的例子介绍如何使用 `metamethods`。假定我们使用 **table** 来描述结合，使用函数来描述集合的并操作，交集操作，**like** 操作。我们在一个表内定义这些函数，然后使用构造函数创建一个集合：

```
Set = {}
Set.mt = {}   --将所有集合共享一个metatable
function Set.new (t)   --新建一个表
    local set = {}
    setmetatable(set,Set.mt)
    for _, l in ipairs(t) do set[l] = true end
    return set
end
function Set.union(a,b)   --并集
    local res = Set.new{}  --注意这里是大括号
    for i in pairs(a) do res[i] = true end
    for i in pairs(b) do res[i] = true end
    return res
end
function Set.intersection(a,b)   --交集
    local res = Set.new{}   --注意这里是大括号
    for i in pairs(a) do
        res[i] = b[i]
    end
    return res
end
function Set.tostring(set)  --打印函数输出结果的调用函数
    local s = "{"
```

```
    local sep = ""
    for i in pairs(set) do
        s = s..sep..i
        sep = ","
    end
    return s.."}"
end
function Set.print(set)   --打印函数输出结果
    print(Set.tostring(set))
end
--[[
Lua中定义的常用的Metamethod如下所示：
算术运算符的Metamethod：
__add（加运算）、__mul（乘）、__sub(减)、__div(除)、__unm(负)、__pow(幂)，__concat（定义连接行为）。
关系运算符的Metamethod：
__eq（等于）、__lt（小于）、__le（小于等于），其他的关系运算自动转换为这三个基本的运算。
库定义的Metamethod：
__tostring（tostring函数的行为）、__metatable（对表getmetatable和setmetatable的行为）。
]]
Set.mt.__add = Set.union

s1 = Set.new{1,2}
s2 = Set.new{3,4}
print(getmetatable(s1))
print(getmetatable(s2))
s3 = s1 + s2
Set.print(s3)

Set.mt.__mul = Set.intersection   --使用相乘运算符来定义集合的交集操作
Set.print((s1 + s2)*s1)
```

如上所示，用表进行了集合的并集和交集操作。

Lua 选择 **metamethod** 的原则：如果第一个参数存在带有 **__add** 域的 **metatable**，Lua 使用它作为 **metamethod**，和第二个参数无关；

否则第二个参数存在带有 **__add** 域的 **metatable**，Lua 使用它作为 **metamethod** 否则报错。

tianqixin 10个月前 (05-24)