LUA - METATABLES

A metatable is a table that helps in modifying the behavior of a table it is attached to with the help of a key set and related meta methods. These meta methods are powerful Lua functionality that enables features like, −

- Changing/adding functionalities to operators on tables.
- Looking up metatables when the key is not available in the table using __index in metatable.

There are two important methods that are used in handling metatables which includes −

- **setmetatable***table, metatable* − This method is used to set metatable for a table.
- **getmetatable***table* − This method is used to get metatable of a table.

Let's first look at how to set one table as metatable of another. It is shown below.

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

The above code can be represented in a single line as shown below.

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

_index

A simple example of metatable for looking up the meta table when it's not available in table is shown below.

```
mytable = setmetatable({key1 = "value1"}, {
   __index = function(mytable, key)

      if key == "key2" then
         return "metatablevalue"
      else
         return mytable[key]
      end
   end
})

print(mytable.key1,mytable.key2)
```

When we run the above program, we will get the following output.

```
value1 metatablevalue
```

Let us explain what happened in the above example in steps.

- The table mytable here is **{key1 = "value1"}**.
- Metatable is set for mytable that contains a function for __index, which we call as a metamethod.
- The metamethod does a simple job of looking up for an index "key2", if it's found, it returns "metatablevalue", otherwise returns mytable's value for corresponding index.

We can have a simplified version of the above program as shown below.

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

__newindex

When we add __newindex to metatable, if keys are not available in the table, the behavior of new keys will be defined by meta methods. A simple example where metatable's index is set when index is not available in the main table is given below.

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

print(mytable.key1)

mytable.newkey = "new value 2"
print(mytable.newkey,mymetatable.newkey)

mytable.key1 = "new  value 1"
print(mytable.key1,mymetatable.newkey1)
```

When you run the above program, you get the following output.

```
value1
nil new value 2
new  value 1 nil
```

You can see in the above program, if a key exists in the main table, it just updates it. When a key is not available in the maintable, it adds that key to the metatable.

Another example that updates the same table using rawset function is shown below.

```
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)
```

When we run the above program we will get the following output.

```
new value "4"
```

rawset sets value without using __newindex of metatable. Similarly there is rawget that gets value without using __index.

Adding Operator Behavior to Tables

A simple example to combine two tables using &plus; operator is shown below −

```
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
```

When we run the above program, we will get the following output.

```
1 1
2 2
3 3
4 4
5 5
6 6
```

The __add key is included in the metatable to add behavior of operator &plus;. The table of keys and corresponding operator is shown below.

Mode	Description
__add	Changes the behavior of operator '&plus;'.
__sub	Changes the behavior of operator '-'.
__mul	Changes the behavior of operator '*'.
__div	Changes the behavior of operator '/'.
__mod	Changes the behavior of operator '%'.
__unm	Changes the behavior of operator '-'.
__concat	Changes the behavior of operator '..'.
__eq	Changes the behavior of operator '=='.
__lt	Changes the behavior of operator '<'.
__le	Changes the behavior of operator '<='.

__call

Adding behavior of method call is done using __call statement. A simple example that returns the sum of values in main table with the passed table.

```
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))
```

When we run the above program, we will get the following output.

```
70
```

__tostring

To change the behavior of the print statement, we can use the __tostring metamethod. A simple example is shown below.

```
mytable = setmetatable({ 10, 20, 30 }, {
   __tostring = function(mytable)
   sum = 0

      for k, v in pairs(mytable) do
         sum = sum + v
      end

      return "The sum of values in the table is " .. sum
   end
})
print(mytable)
```

When we run the above program, we will get the following output.

```
The sum of values in the table is 60
```

If you know the capabilities of meta table fully, you can really perform a lot of operations that would be very complex without using it. So, try to work more on using metatables with different options available in meta tables as explained in the samples and also create your own samples.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js