什么是Metatable?

Metatable 对应的中文翻译是元表。那么，究竟什么是元表?

我们首先来看下`setmetatable`这个函数

- `setmetatable (table, metatable)`

  Sets the metatable for the given table. (You cannot change the metatable of other types from Lua, only from C.) If **metatable** is **nil**, removes the metatable of the given table. If the original metatable has a "**__metatable**" field, raises an error.
  This function returns **table**.

通过setmetatable的描述，我们大概可以得知，`setmetatable (table, metatable)`对指定table设置元表(metatable),如果metatable中存在__metatable字段，那么setmetatable会失败。

Talk is cheap. Show me the code.

```lua
local myTable = {} -- 普通表
local myMetatable = {} -- 元表
setmetatable(myTable, myMetatable) -- 设置myTable的元表为myMetatable
```

元表似乎就这么设置完了。
但是，为什么要设置元表，设置完元表后，我们又会获得什么新的特性，我们仍然一无所知。

这个时候，我们又引申出一个新的概念**Metamethod**

什么是Metamethod?

> Lua has a powerful extension mechanism which allows you to overload certain operations on Lua objects. Each overloaded object has a metatable of function metamethods associated with it; these are called when appropriate, similar to the concept of operator overloading from many other languages.
> A metatable is a regular Lua table containing a set of metamethods, which are associated with events in Lua. Events occur when Lua executes certain operations, like addition, string concatenation, comparisons etc.
> Metamethods are regular Lua functions which are called when a specific event occurs. The events have names like "add" and "concat" (see manual section 2.8) which correspond with string keys in the metatable like "__add" and "__concat". In this case to add (+) or concatenate (..) two Lua objects.
> From

上面这段引用比较抽象哈，我们先不管，我们先看下，有哪些metamethods：

- __add
- __sub
- __mul
- __div
- __mod
- __pow
- __unm
- __concat
- __len
- __eq
- __lt
- __le
- __index
- __newindex
- __call

看着似乎很多的样子哈，那么到底是不是这些或者有没有遗漏的呢？我们去lua源码那找一下定义。

```
// lua-5.1.5/src/ltm.c 30行

void luaT_init (lua_State *L) {
  static const char *const luaT_eventname[] = {  /* ORDER TM */
    "__index", "__newindex",
    "__gc", "__mode", "__eq",
    "__add", "__sub", "__mul", "__div", "__mod",
    "__pow", "__unm", "__len", "__lt", "__le",
    "__concat", "__call"
  };
  int i;
  for (i=0; i<TM_N; i++) {
    G(L)->tmname[i] = luaS_new(L, luaT_eventname[i]);
    luaS_fix(G(L)->tmname[i]);  /* never collect these names */
  }
}
```

可以看到，基本都在上面了。5.2和5.3版本中有新增，这里我们先不做讨论。

Talk is cheap. Show me the code.

```
local x = { num = 5 }
local y = x + x
-- output: attempt to perform arithmetic on local 'x' (a table
value)
```

上面对table x相加，直接报错。 那么我们定义metatable后，再相加看看。

```
local x = { num = 5 }
local mt = {}

setmetatable(x, mt)
local y = x + x
-- output: attempt to perform arithmetic on local 'x' (a table
value)
```

一样报错哈~

那么，我们再设置一下metamethod看看

```lua
local x = { num = 5 }
local mt = {
    __add = function (lhs, rhs)
        return { num = lhs.num + rhs.num}
    end
}

setmetatable(x, mt)
local y = x + x
print(y.num)
-- output: 10
```

这回可以了。那么这里，我们先对metamethod做下总结。
所谓的metamethod，对应的中文翻译是元方法，指的是对table的操作方法，根据不同的操作，定义不同的元方法。
两个table相加，我们可以使用`__add`，同样的，两个table相减，我们可以使用`__sub`。
然后这个时候，我们就明白了，**所谓的元表，只是一个用来存放元方法的table**。

那么，我们不定义元表，直接把元方法写在table里，然后两个table相加可以么？

```lua
local x = {
    num = 5,
    __add = function (lhs, rhs)
        return { num = lhs.num + rhs.num}
    end
}

local y = x + x
print(y.num)
-- output: attempt to perform arithmetic on local 'x' (a table
value)
```

报错了哈~

看到这，大家应该对Metatables 和 Metamethods有了一个初步的概念了吧~