Lua 调试(Debug)

Lua 中 debug 库包含以下函数：

序号	方法 & 用途
1.	debug(): 进入一个用户交互模式，运行用户输入的每个字符串。 使用简单的命令以及其它调试设置，用户可以检阅全局变量和 变量， 改变变量的值，计算一些表达式，等等。 输入一行仅包含 cont 的字符串将结束这个函数， 这样调用者就可以继续向下运行。
2.	getfenv(object): 返回对象的环境变量。
3.	gethook(optional thread): 返回三个表示线程钩子设置的值： 当前钩子函数，当前钩子掩码，当前钩子计数
4.	getinfo ([thread,] f [, what]): 返回关于一个函数信息的表。 你可以直接提供该函数， 也可以用一个数字 f 表示该函数。 数字 f 表示运行在指定 的调用栈对应层次上的函数： 0 层表示当前函数（getinfo 自身）； 1 层表示调用 getinfo 的函数 （除非是尾调用 种情况不计入栈）；等等。 如果 f 是一个比活动函数数量还大的数字， getinfo 返回 nil。
5.	debug.getlocal ([thread,] f, local): 此函数返回在栈的 f 层处函数的索引为 local 的局部变量 的名字和值。 这个函数不仅用于访问显式定义的局部变量 包括形参、临时变量等。
6.	getmetatable(value): 把给定索引指向的值的元表压入堆栈。如果索引无效，或是这个值没有元表，函数将返回 0 并且不会向栈上压任何东
7.	getregistry(): 返回注册表表，这是一个预定义出来的表， 可以用来保存任何 C 代码想保存的 Lua 值。
8.	getupvalue (f, up) 此函数返回函数 f 的第 up 个上值的名字和值。 如果该函数没有那个上值，返回 nil 。 以 '(' （开括号）打头的变量名表示没有名字的变量 （去除了调试信息的代码块）。
10.	sethook ([thread,] hook, mask [, count]):

	将一个函数作为钩子函数设入。 字符串 mask 以及数字 count 决定了钩子将在何时调用。 掩码是由下列字符组合 符串，每个字符有其含义： 'c'：每当 Lua 调用一个函数时，调用钩子； 'r'：每当 Lua 从一个函数内返回时，调用钩子； 'l'：每当 Lua 进入新的一行时，调用钩子。
11.	**setlocal ([thread,] level, local, value):** 这个函数将 value 赋给 栈上第 level 层函数的第 local 个局部变量。 如果没有那个变量，函数返回 nil 。 如 el 越界，抛出一个错误。
12.	**setmetatable (value, table):** 将 value 的元表设为 table （可以是 nil）。 返回 value。
13.	**setupvalue (f, up, value):** 这个函数将 value 设为函数 f 的第 up 个上值。 如果函数没有那个上值，返回 nil 否则，返回该上值的名字。
14.	**traceback ([thread,] [message [, level]]):** 如果 message 有，且不是字符串或 nil， 函数不做任何处理直接返回 message。 否则，它返回调用栈的栈回溯信 字符串可选项 message 被添加在栈回溯信息的开头。 数字可选项 level 指明从栈的哪一层开始回溯 （默认为 1 ， 用 traceback 的那里）。

上表列出了我们常用的调试函数，接下来我们可以看些简单的例子：