****模式匹配函数****   
在string库中功能最强大的函数是：string.find (字符串查找), string.gsub (全局字符串替换), and string.gfind (全局字符串查找). 这些函数都是基于模式匹配的。   
与其他脚本语言不同的是，Lua并不使用POSIX规范的正则表达式(也写作regexp)来进行模式匹配(译者：POSIX是unix的工业标准,regexp最初来源于unix,POSIX对regexp也作了规范)。主要的原因出于程序大小方面的考虑：实现一个典型的符合POSIX标准的regexp大概需要4000行代码，这比整个Lua标准库加在一起都大。权衡之下，Lua中的模式匹配的实现只用了500行代码，当然这意味着不可能实现POSIX所规范的所有更能。然而，Lua中的模式匹配功能是很强大的，并且包含了一些使用标准POSIX模式匹配不容易实现的功能。   
string.find的基本应用就是用来在目标串(subject string)内搜索匹配指定的模式的串。函数如果找到匹配的串返回他的位置，否则返回nil.最简单的模式就是一个单词，仅仅匹配单词本身。比如，模式'hello'仅仅匹配目标串中的"hello"。当查找到模式的时候，函数返回两个值：匹配串开始索引和结束索引。   
  
s = "hello world"  
i, j = string.find(s, "hello")  
print(i, j)                   --> 1 5  
print(string.sub(s, i, j))    --> hello  
print(string.find(s, "world")) --> 7 11  
i, j = string.find(s, "l")  
print(i, j)                   --> 3 3  
print(string.find(s, "lll"))     --> nil  
例子中，匹配成功的时候，string.sub利用string.find返回的值截取匹配的子串。 (对简单模式而言，匹配的就是其本身。)   
string.find函数第三个参数是可选的：标示目标串中搜索的起始位置。当我们想查找目标串中所有匹配的子串的时候，这个选项非常有用。我们可以不断的循环搜索，每一次从前一次匹配的结束位置开始。下面看一个例子，下面的代码用一个字符串中所有的新行构造一个表：   
local t = {}                -- table to store the indices  
local i = 0  
while true do  
    i = string.find(s, "/n", i+1) -- find 'next' newline  
    if i == nil then break end  
    table.insert(t, i)  
end  
后面我们还会看到可以使用string.gfind迭代子来简化上面这个循环。   
string.gsub函数有三个参数：目标串，模式串，替换串。他基本作用是用来查找匹配模式的串，并将使用替换串其替换掉:   
s = string.gsub("Lua is cute", "cute", "great")  
print(s)       --> Lua is great  
s = string.gsub("all lii", "l", "x")  
print(s)       --> axx xii  
s = string.gsub("Lua is great", "perl", "tcl")  
print(s)       --> Lua is great  
第四个参数是可选的，用来限制替换的范围：   
s = string.gsub("all lii", "l", "x", 1)  
print(s)       --> axl lii  
s = string.gsub("all lii", "l", "x", 2)  
print(s)       --> axx lii  
string.gsub的第二个返回值表示他进行替换操作的次数。例如，下面代码涌来计算一个字符串中空格出现的次数：   
\_, count = string.gsub(str, " ", " ")  
  
(注意, \_ 只是一个哑元变量.)   
20.2模式   
你还可以在模式串中使用字符类。字符类指可以匹配一个特定字符集合内任何字符的模式项。比如，字符类 %d 匹配任意数字. 所以你可以使用模式串'%d%d/%d%d/%d%d%d%d'搜索dd/mm/yyyy 格式的日期 :   
s = "Deadline is 30/05/1999, firm"  
date = "%d%d/%d%d/%d%d%d%d"  
print(string.sub(s, string.find(s, date))) --> 30/05/1999  
下面的表列出了Lua支持的所有字符类:   
   .        任意字符  
%a        字母  
%c        控制字符  
%d        数字  
%l        小写字母  
%p        标点字符  
%s        空白符  
%u        大写字母  
%w        字母和数字  
%x        十六进制数字  
%z        代表0的字符  
上面字符类的大写形式表示小写所代表的集合的补集。例如, '%A'非字母的字符:   
print(string.gsub("hello, up-down!", "%A", "."))  
    --> hello..up.down. 4  
(数字4不是字符串结果的一部分，他是gsub返回的第二个结果，代表发生替换的次数。下面其他的关于打印gsub结果的例子中将会忽略这个数值。 ) 在模式匹配中有一些特殊字符，他们有特殊的意义，Lua中的特殊字符如下：   
( ) . % + - \* ? [ ^ $  
`%′用作特殊字符的转义字符，因此 '%.' 匹配点; '%%'匹配字符 `%′ .转义字符`%′不仅可以用来转义特殊字符，还可以用于所有的非字母的字符。当对一个字符有疑问的时候，为安全起见请使用转义字符转义他。   
对Lua而言，模式串就是普通的字符串。他们和其他的字符串没有区别，也不会受到特殊对待。只有他们被用作模式串用于函数的时候，`%′才作为转义字符。所以，如果你需要在一个模式串内放置引号的话，你必须使用在其他的字符串中放置引号的方法来处理，使用`/′转义引号，`/′是Lua的转义符。你可以使用方括号将字符类或者字符括起来创建自己的字符类(译者：Lua称之为char-set，就是指传统正则表达式概念中的括号表达式)。比如，'[%w\_]'将匹配字母数字和下划线，'[01]'匹配二进制数字，'[%[%]]'匹配一对方括号。下面的例子统计文本中元音字母出现的次数：   
\_, nvow = string.gsub(text, "[AEIOUaeiou]", "")  
在char-set中可以使用范围表示字符的集合，第一个字符和最后一个字符之间用连字符连接表示这两个字符之间范围内的字符集合。大部分的常用字符范围都已经预定义好了，所以一般你不需要自己定义字符的集合。比如，'%d'表示 '[0-9]'；'%x'表示'[0-9a-fA-F]'。然而，如果你想查找八进制数，你可能更喜欢使用'[0-7]'而不是'[01234567]'。你可以在字符集(char-set)的开始处使用 `^′ 表示其补集: '[^0-7]' 匹配任何不是八进制数字的字符； '[^/n]' 匹配任何非换行符户的字符。记住，可以使用大写的字符类表示其补集： '%S'比'[^%s]'要简短些.   
Lua的字符类依赖于本地环境，所以'[a-z]'可能与'%l'表示的字符集不同。在一般情况下，后者包括`ç´ 和 `ã´，而前者没有。应该尽可能的使用后者来表示字母，除非出于某些特殊考虑，因为后者更简单、方便、更高效。   
可以使用修饰符来修饰模式增强模式的表达能力，Lua中的模式修饰符有四个：   
+        匹配前一字符1次或多次  
\*       匹配前一字符0次或多次  
-        匹配前一字符0次或多次  
?        匹配前一字符0次或1次  
`+′ 匹配一个或多个字符，她总是进行最长的匹配. 比如，模式串 '%a+'匹配一个或多个字母或者一个单词 :   
print(string.gsub("one, and two; and three", "%a+", "word"))  
    --> word, word word; word word  
'%d+'匹配一个或多个数字 (整数):   
i, j = string.find("the number 1298 is even", "%d+")  
                        print(i,j) --> 12   15  
`\*′ 与 `+′类似, 但是他匹配一个字符0次或多次出现.一个典型的应用是匹配空白。比如，为了匹配一对圆括号()或者( )之间的空白，可以使用'%(%s\*%)'. ( '%s\*'用来匹配0个或多个空白. 由于圆括号在模式中有特殊的含义，所以我们必须使用`%′转义他.) 再看一个例子，'[\_%a][\_%w]\*'匹配Lua程序中的标示符：字母或者下划线开头的字母下划线数字序列。   
`-′与`\*′一样，都匹配一个字符的0次或多次出现，但是他进行的是最短匹配。某些时候这两个用起来没有区别，但有些时候结果将截然不同。比如，如果你使用模式'[\_%a][\_%w]-'来查找标示符，你将只能找到第一个字母，因为'[\_%w]-'永远匹配空。另一方面，假定你想查找C程序中的注释，很多人可能使用 '/%\*.\*%\*/' (也就是说 "/\*" 后面跟着任意多个字符，然后跟着 "\*/"). 然而，由于 '.\*'进行的是最长匹配，这个模式将匹配程序中第一个"/\*" 和最后一个"\*/"之间所有部分:   
test = "int x; /\* x \*/   int y; /\* y \*/"  
print(string.gsub(test, "/%\*.\*%\*/", "<COMMENT>"))  
    --> int x; <COMMENT>  
然而模式 '.-'进行的是最短匹配，她会匹配"/\*"开始到第一个"\*/"之前的部分:   
test = "int x; /\* x \*/   int y; /\* y \*/"  
print(string.gsub(test, "/%\*.-%\*/", "<COMMENT>"))  
   --> int x; <COMMENT>   int y; <COMMENT>  
`?′匹配一个字符0次或1次.举个例子，假定我们想在一段文本内查找一个整数，整数可能带有正负号。 模式 '[+-]?%d+'符合我们的要求，她可以匹配 像 "-12", "23" 和 "+1009"等数字. '[+-]' 是一个匹配`+′或者 `-′的字符类；接下来的 `?′意思是匹配前面的字符类0次或者1次.   
与其他系统的模式不同的是，Lua中的修饰符不能用字符类；不能将模式分组然后使用修饰符作用这个分组。比如，没有一个模式可以匹配一个可选的单词(除非这个单词只有一个字母)。下面我将看到，通常你可以使用一些高级技术绕开这个限制。   
以`^′开头的模式只匹配目标串的开始部分，相似的，以`$′结尾的模式只匹配目标串的结尾部分。这不仅可以用来限制你要查找的模式，还可以定位(anchor)模式。比如：   
if string.find(s, "^%d") then ...  
检查字符串s是否以数字开头，而   
if string.find(s, "^[+-]?%d+$") then ...  
检查字符串s是否是一个整数。   
'%b'用来匹配对称的字符.常写为 '%bxy',x和y是任意两个不同的字符；x作为匹配的开始,y作为匹配的结束。比如， '%b()'匹配以`(′开始， 以 `)′结束的字符串:   
print(string.gsub("a (enclosed (in) parentheses) line",  
                  "%b()", ""))  
    --> a   line  
常用的这种模式有： '%b()', '%b[]', '%b%{%}',和 '%b<>'。你也可以使用任何字符作为分隔符。

* x (这里 x 是指其不是这些转义字符 ^$()%.[]\*+-? 之一) --- 其代表了这个字符本身.
* . --- 代表任何字符
* %a --- 代表任何字母. 即[a-zA-Z]
* %c --- 代表任何的控制字符.
* %d ---代表任何的数字字符. 即[0-9]
* %l --- 代表所有的小写字母. 即[a-z]
* %p --- 代表所有的标点符号字符.
* %s --- 代表所有空格,tab 字符.
* %u --- 代表所有的大写字母. 即[A-Z]
* %w --- 代表所有的字母数字. 即[a-zA-Z0-9]
* %x --- 代表16进制数字.
* %z --- 代表字符值是 0 的字符. 注意:值为0 的字符是无法正常表达的在表达式中,如果你要使用他,请使用 %z .
* %x (x是任何非字母和数字的字符) --- 代表字符 x. 这是一种标准的方式来代表应用转义字符. 任何标点符号字符(即使不是转义字符) 在其前面添加一个 % 可以用来表示其自己 例如 %% 表示 % , %$ 表示 $.
* [set] --- 代表一个字符集合. 如果要表达一个范围集合,在范围开始的字符和结尾的字符之间使用 - , 例如要表达 3,4,5,6 这个集合 可以用 [3456] ,也可以用 [3-6] .上面提到的 %x 也可以用到集合中. 例如, [%w\_] 表达所有的字母和数字加一个下划线.
* [^set] --- 表达所有不出现在集合内的.

总的来说，在Lua中的正则表达式与C#中的是相似的，只是转义字符由/变成了%。

Lua中的正则表达式语言限定符：

* \* -- matches 0 or more repetitions of x. Will always match the longest possible chain.
* + -- matches 1 or more repetitions of x. Will always match the longest possible chain.
* - -- matches 0 or more repetitions of x. Will always match the shortest possible chain.
* ? -- matches 0 or 1 occurence of x.

另外一些用法：

* %n -- n must be a number between 1 and 9. Matches the nth captured substring (see below)
* %bxy -- matches a substring starting with x and ending with y. The substring must also have the same number of x and y.
* ^ -- When at the beginning of a pattern, it forces the pattern to match the beginning of a string
* $ -- When at the end of a pattern, it forces the pattern to match the end of a string
* When ^ or $ is anywhere else in a pattern, it has no special meaning.