LUA 中的 TABLE 函数库

Posted on 2009-08-10 20:43 白耘 阅读(1982) 评论(0) 编辑 收藏 所属分类: Lua 学习笔记

一部分的 table 函数只对其数组部分产生影响，而另一部分则对整个 table 均产生影响. 下面会分开说明.

table.concat(table, sep, start, end)

concat 是 concatenate(连锁，连接)的缩写. table.concat()函数列出参数中指定 table 的数组部分从 start 位置到 end 位置的所有元素，元素间以指定的分隔符(sep)隔开。除了 table 外，其他的参数都不是必须的，分隔符的默认值是空字符，start 的默认值是 1，end 的默认值是数组部分的总长.

sep，start，end 这三个参数是顺序读入的，所以虽然它们都不是必须参数，但如果要指定靠后的参数，必须同时指定前面的参数.

```
> tbl = {"alpha", "beta", "gamma"}
> print(table.concat(tbl, ":"))
alpha:beta:gamma
> print(table.concat(tbl, nil, 1, 2))
alphabeta
> print(table.concat(tbl, "\n", 2, 3))
beta
gamma
```

table.insert(table, pos, value)

table.insert()函数在 table 的数组部分指定位置(pos)插入值为 value 的一个元素. pos 参数可选, 默认为数组部分末尾.

```
> tbl = {"alpha", "beta", "gamma"}
> table.insert(tbl, "delta")
> table.insert(tbl, "epsilon")
> print(table.concat(tbl, ", ")
alpha, beta, gamma, delta, epsilon
> table.insert(tbl, 3, "zeta")
> print(table.concat(tbl, ", ")
alpha, beta, zeta, gamma, delta, epsilon
```

table.maxn(table)

table.maxn()函数返回指定 table 中所有正数 key 值中最大的 key 值. 如果不存在 key 值为正数的元素, 则返回 0. 此函数不限于 table 的数组部分.

```
> tbl = {[1] = "a", [2] = "b", [3] = "c", [26] = "z"}
> print(#tbl)
3                -- 因为 26 和之前的数字不连续, 所以不算在数组部分内
> print(table.maxn(tbl))
26
> tbl[91.32] = true
> print(table.maxn(tbl))
91.32
```

table.remove(table, pos)

table.remove()函数删除并返回 table 数组部分位于 pos 位置的元素. 其后的元素会被前移. pos 参数可选, 默认为 table 长度, 即从最后一个元素删起.

table.sort(table, comp)

table.sort()函数对给定的 table 进行升序排序.

```
> tbl = {"alpha", "beta", "gamma", "delta"}
> table.sort(tbl)
> print(table.concat(tbl, ", "))
alpha, beta, delta, gamma
```

comp 是一个可选的参数, 此参数是一个外部函数, 可以用来自定义 sort 函数的排序标准.

此函数应满足以下条件: 接受两个参数(依次为 a, b), 并返回一个布尔型的值, 当 a 应该排在 b 前面时, 返回 true, 反之返回 false.

例如, 当我们需要降序排序时, 可以这样写:

```
> sortFunc = function(a, b) return b < a end
> table.sort(tbl, sortFunc)
> print(table.concat(tbl, ", "))
gamma, delta, beta, alpha
```

用类似的原理还可以写出更加复杂的排序函数. 例如, 有一个 table 存有工会三名成员的姓名及等级信息:

```
guild = {}

table.insert(guild, {
```

```
  name = "Cladhaire",
  class = "Rogue",
  level = 70,
})

table.insert(guild, {
  name = "Sagart",
  class = "Priest",
  level = 70,
})

table.insert(guild, {
  name = "Mallaithe",
  class = "Warlock",
  level = 40,
})
```

对这个 table 进行排序时，应用以下的规则：按等级升序排序，在等级相同时，按姓名升序排序.

可以写出这样的排序函数:

```
function sortLevelNameAsc(a, b)
  if a.level == b.level then
    return a.name < b.name
  else
```

```
    return a.level < b.level
  end
end
```

测试功能如下:

```
> table.sort(guild, sortLevelNameAsc)
> for idx, value in ipairs(guild) do print(idx, value.name) end
1, Mallaithe
2, Cladhaire
3, Sagart
```

table.foreachi(table, function(i, v))

会期望一个从 1（数字 1）开始的连续整数范围，遍历 table 中的 key 和 value 逐对进行 function(i, v)操作

```
t1 = {2, 4, 6, language="Lua", version="5", 8, 10, 12, web="hello lua"};
table.foreachi(t1, function(i, v) print (i, v) end) ; --等价于 foreachi(t1, print)
```

输出结果:

```
1 2
2 4
3 6
4 8
5 10
```

6 12

table.foreach(table, function(i, v))

与 foreachi 不同的是，foreach 会对整个表进行迭代

```
t1 = {2, 4, 6, language="Lua", version="5", 8, 10, 12, web="hello
lua"};
table.foreach(t1, function(i, v) print (i, v) end) ;
```

输出结果：

1 2

2 4

3 6

4 8

5 10

6 12

web hello lua

language Lua

version 5

table.getn(table)

返回 table 中元素的个数

```
t1 = {1, 2, 3, 5};
print(getn(t1))
```

->4

table.setn(table, nSize)

设置 table 中的元素个数