[**python 列表排序方法sort、sorted技巧篇**](http://www.cnblogs.com/whaben/p/6495702.html)

Python list内置sort()方法用来排序，也可以用python内置的全局sorted()方法来对可迭代的序列排序生成新的序列。

**1）排序基础**

简单的升序排序是非常容易的。只需要调用sorted()方法。它返回一个新的list，新的list的元素基于小于运算符(\_\_lt\_\_)来排序。

>>> sorted([5, 2, 3, 1, 4])

[1, 2, 3, 4, 5]

你也可以使用list.sort()方法来排序，此时list本身将被修改。通常此方法不如sorted()方便，但是如果你不需要保留原来的list，此方法将更有效。

>>> a = [5, 2, 3, 1, 4]

>>> a.sort()

>>> a

[1, 2, 3, 4, 5]

另一个不同就是list.sort()方法仅被定义在list中，相反地sorted()方法对所有的可迭代序列都有效。

>>> sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})

[1, 2, 3, 4, 5]

**2）key参数/函数**

从python2.4开始，list.sort()和sorted()函数增加了key参数来指定一个函数，此函数将在每个元素比较前被调用。 例如通过key指定的函数来忽略字符串的大小写：

>>> sorted("This is a test string from Andrew".split(), key=str.lower)

['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']

key参数的值为一个函数，此函数只有一个参数且返回一个值用来进行比较。这个技术是快速的因为key指定的函数将准确地对每个元素调用。

更广泛的使用情况是用复杂对象的某些值来对复杂对象的序列排序，例如：

[复制代码](javascript:void(0);)

>>> student\_tuples = [

        ('john', 'A', 15),

        ('jane', 'B', 12),

        ('dave', 'B', 10),

]

>>> sorted(student\_tuples, key=lambda student: student[2])   # sort by age

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

[复制代码](javascript:void(0);)

同样的技术对拥有命名属性的复杂对象也适用，例如：

>>> class Student:  
        def \_\_init\_\_(self, name, grade, age):  
                self.name = name  
                self.grade = grade  
                self.age = age  
        def \_\_repr\_\_(self):  
                return repr((self.name, self.grade, self.age))  
>>> student\_objects = [  
        Student('john', 'A', 15),  
        Student('jane', 'B', 12),  
        Student('dave', 'B', 10),  
]  
>>> sorted(student\_objects, key=lambda student: student.age)   # sort by age  
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

**3）Operator 模块函数**

上面的key参数的使用非常广泛，因此python提供了一些方便的函数来使得访问方法更加容易和快速。operator模块有itemgetter，attrgetter，从2.6开始还增加了methodcaller方法。使用这些方法，上面的操作将变得更加简洁和快速：

>>> from operator import itemgetter, attrgetter

>>> sorted(student\_tuples, key=itemgetter(2))

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

>>> sorted(student\_objects, key=attrgetter('age'))

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

operator模块还允许多级的排序，例如，先以grade，然后再以age来排序：

>>> sorted(student\_tuples, key=itemgetter(1,2))

[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]

>>> sorted(student\_objects, key=attrgetter('grade', 'age'))

[('john', 'A', 15), ('dave', 'B', 10), ('jane', 'B', 12)]

**4）升序和降序**

list.sort()和sorted()都接受一个参数reverse（True or False）来表示升序或降序排序。例如对上面的student降序排序如下：

>>> sorted(student\_tuples, key=itemgetter(2), reverse=True)

[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]

>>> sorted(student\_objects, key=attrgetter('age'), reverse=True)

[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]

**5）排序的稳定性和复杂排序**

从python2.2开始，排序被保证为稳定的。意思是说多个元素如果有相同的key，则排序前后他们的先后顺序不变。

>>> data = [('red', 1), ('blue', 1), ('red', 2), ('blue', 2)]

>>> sorted(data, key=itemgetter(0))

[('blue', 1), ('blue', 2), ('red', 1), ('red', 2)]

注意在排序后'blue'的顺序被保持了，即'blue', 1在'blue', 2的前面。  
更复杂地你可以构建多个步骤来进行更复杂的排序，例如对student数据先以grade降序排列，然后再以age升序排列。

>>> s = sorted(student\_objects, key=attrgetter('age'))     # sort on secondary key

>>> sorted(s, key=attrgetter('grade'), reverse=True)       # now sort on primary key, descending

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

**6）最老土的排序方法-DSU**

我们称其为DSU（Decorate-Sort-Undecorate）,原因为排序的过程需要下列三步：  
第一：对原始的list进行装饰，使得新list的值可以用来控制排序；  
第二：对装饰后的list排序；  
第三：将装饰删除，将排序后的装饰list重新构建为原来类型的list；

例如，使用DSU方法来对student数据根据grade排序：  
>>> decorated = [(student.grade, i, student) for i, student in enumerate(student\_objects)]  
>>> decorated.sort()  
>>> [student for grade, i, student in decorated]               # undecorate  
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]  
上面的比较能够工作，原因是tuples是可以用来比较，tuples间的比较首先比较tuples的第一个元素，如果第一个相同再比较第二个元素，以此类推。

并不是所有的情况下都需要在以上的tuples中包含索引，但是包含索引可以有以下好处：  
第一：排序是稳定的，如果两个元素有相同的key，则他们的原始先后顺序保持不变；  
第二：原始的元素不必用来做比较，因为tuples的第一和第二元素用来比较已经是足够了。

此方法被RandalL.在perl中广泛推广后，他的另一个名字为也被称为Schwartzian transform。

对大的list或list的元素计算起来太过复杂的情况下，在python2.4前，DSU很可能是最快的排序方法。但是在2.4之后，上面解释的key函数提供了类似的功能。

**7）其他语言普遍使用的排序方法-cmp函数**

在python2.4前，sorted()和list.sort()函数没有提供key参数，但是提供了cmp参数来让用户指定比较函数。此方法在其他语言中也普遍存在。

在python3.0中，cmp参数被彻底的移除了，从而简化和统一语言，减少了高级比较和\_\_cmp\_\_方法的冲突。

在python2.x中cmp参数指定的函数用来进行元素间的比较。此函数需要2个参数，然后返回负数表示小于，0表示等于，正数表示大于。例如：

>>> def numeric\_compare(x, y):

        return x - y

>>> sorted([5, 2, 4, 1, 3], cmp=numeric\_compare)

[1, 2, 3, 4, 5]

或者你可以反序排序：

>>> def reverse\_numeric(x, y):

        return y - x

>>> sorted([5, 2, 4, 1, 3], cmp=reverse\_numeric)

[5, 4, 3, 2, 1]

当我们将现有的2.x的代码移植到3.x时，需要将cmp函数转化为key函数，以下的wrapper很有帮助：

[复制代码](javascript:void(0);)

def cmp\_to\_key(mycmp):

    'Convert a cmp= function into a key= function'

    class K(object):

        def \_\_init\_\_(self, obj, \*args):

            self.obj = obj

        def \_\_lt\_\_(self, other):

            return mycmp(self.obj, other.obj) < 0

        def \_\_gt\_\_(self, other):

            return mycmp(self.obj, other.obj) > 0

        def \_\_eq\_\_(self, other):

            return mycmp(self.obj, other.obj) == 0

        def \_\_le\_\_(self, other):

            return mycmp(self.obj, other.obj) <= 0

        def \_\_ge\_\_(self, other):

            return mycmp(self.obj, other.obj) >= 0

        def \_\_ne\_\_(self, other):

            return mycmp(self.obj, other.obj) != 0

    return K

[复制代码](javascript:void(0);)

当需要将cmp转化为key时，只需要：

>>> sorted([5, 2, 4, 1, 3], key=cmp\_to\_key(reverse\_numeric))

[5, 4, 3, 2, 1]

从python2.7，cmp\_to\_key()函数被增加到了functools模块中。

**8)其他注意事项**

\* 对需要进行区域相关的排序时，可以使用locale.strxfrm()作为key函数，或者使用local.strcoll()作为cmp函数。

\* reverse参数任然保持了排序的稳定性，有趣的时，同样的效果可以使用reversed()函数两次来实现：

>>> data = [('red', 1), ('blue', 1), ('red', 2), ('blue', 2)]

>>> assert sorted(data, reverse=True) == list(reversed(sorted(reversed(data))))

\* 其实排序在内部是调用元素的\_\_cmp\_\_来进行的，所以我们可以为元素类型增加\_\_cmp\_\_方法使得元素可比较，例如：

>>> Student.\_\_lt\_\_ = lambda self, other: self.age < other.age

>>> sorted(student\_objects)

[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]

\* key函数不仅可以访问需要排序元素的内部数据，还可以访问外部的资源，例如，如果学生的成绩是存储在dictionary中的，则可以使用此dictionary来对学生名字的list排序，如下：

>>> students = ['dave', 'john', 'jane']

>>> newgrades = {'john': 'F', 'jane':'A', 'dave': 'C'}

>>> sorted(students, key=newgrades.\_\_getitem\_\_)

['jane', 'dave', 'john']

\*当你需要在处理数据的同时进行排序的话，sort(),sorted()或bisect.insort()不是最好的方法。在这种情况下，可以使用heap，red-black tree或treap。