8.24 让类支持比较操作¶

问题¶

你想让某个类的实例支持标准的比较运算(比如>=,!=,<=,<等)，但是又不想去实现那一大丢的特殊方法。

解决方案¶

Python类对每个比较操作都需要实现一个特殊方法来支持。 例如为了支持>=操作符，你需要定义一个 `__ge__()` 方法。尽管定义一个方法没什么问题，但如果要你实现所有可能的比较方法那就有点烦人了。

装饰器 `functools.total_ordering` 就是用来简化这个处理的。 使用它来装饰一个来，你只需定义一个 `__eq__()` 方法， 外加其他方法(__lt__, __le__, __gt__, or __ge__)中的一个即可。 然后装饰器会自动为你填充其它比较方法。

作为例子，我们构建一些房子，然后给它们增加一些房间，最后通过房子大小来比较它们：

```
from functools import total_ordering

class Room:
    def __init__(self, name, length, width):
        self.name = name
        self.length = length
        self.width = width
        self.square_feet = self.length * self.width

@total_ordering
class House:
    def __init__(self, name, style):
        self.name = name
        self.style = style
        self.rooms = list()

    @property
    def living_space_footage(self):
        return sum(r.square_feet for r in self.rooms)

    def add_room(self, room):
        self.rooms.append(room)

    def __str__(self):
        return '{}: {} square foot {}'.format(self.name,
                self.living_space_footage,
                self.style)

    def __eq__(self, other):
        return self.living_space_footage == other.living_space_footage

    def __lt__(self, other):
        return self.living_space_footage < other.living_space_footage
```

这里我们只是给House类定义了两个方法： `__eq__()` 和 `__lt__()` ，它就能支持所有的比较操作：

```
# Build a fewhouses, and add rooms to them
h1 = House('h1', 'Cape')
```

```
h1.add_room(Room('Master Bedroom', 14, 21))
h1.add_room(Room('Living Room', 18, 20))
h1.add_room(Room('Kitchen', 12, 16))
h1.add_room(Room('Office', 12, 12))
h2 = House('h2', 'Ranch')
h2.add_room(Room('Master Bedroom', 14, 21))
h2.add_room(Room('Living Room', 18, 20))
h2.add_room(Room('Kitchen', 12, 16))
h3 = House('h3', 'Split')
h3.add_room(Room('Master Bedroom', 14, 21))
h3.add_room(Room('Living Room', 18, 20))
h3.add_room(Room('Office', 12, 16))
h3.add_room(Room('Kitchen', 15, 17))
houses = [h1, h2, h3]
print('Is h1 bigger than h2?', h1 > h2) # prints True
print('Is h2 smaller than h3?', h2 < h3) # prints True
print('Is h2 greater than or equal to h1?', h2 >= h1) # Prints False
print('Which one is biggest?', max(houses)) # Prints 'h3: 1101-square-foot Split'
print('Which is smallest?', min(houses)) # Prints 'h2: 846-square-foot Ranch'
```

讨论¶

其实 `total_ordering` 装饰器也没那么神秘。 它就是定义了一个从每个比较支持方法到所有需要定义的其他方法的一个映射而已。 比如你定义了 `__le__()` 方法，那么它就被用来构建所有其他的需要定义的那些特殊方法。 实际上就是在类里面像下面这样定义了一些特殊方法：

```
class House:
    def __eq__(self, other):
        pass
    def __lt__(self, other):
        pass
    # Methods created by @total_ordering
    __le__ = lambda self, other: self < other or self == other
    __gt__ = lambda self, other: not (self < other or self == other)
    __ge__ = lambda self, other: not (self < other)
    __ne__ = lambda self, other: not self == other
```

当然，你自己去写也很容易，但是使用 `@total_ordering` 可以简化代码，何乐而不为呢。