正确重载运算符

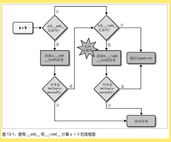
- 运算符重载的作用是让用户定义的对象使用中缀运算符（如 + 和 |）或一元运算符（如 - 和 ~）。
- Python 施加了一些限制，做好了灵活性、可用性和安全性方面的平衡
  - 不能重载内置类型的运算符
  - 不能新建运算符，只能重载现有的
  - 某些运算符不能重载——is、and、or 和 not（不过位运算符 &、| 和 ~ 可以）
- 一元运算符
  - 三个运算符和对应的特殊方法
    - -（__neg__）一元取负算术运算符。如果 x 是 -2，那么 -x == 2。
    - +（__pos__）一元取正算术运算符。通常，x == +x，但也有一些例外。如果好奇，请阅读“x 和 +x 何时不相等”附注栏。
      - 多数时候，+ 最好返回 self 的副本。
    - ~（__invert__）对整数按位取反，定义为 ~x == -(x+1)。如果 x 是 2，那么 ~x == -3。
  - 遵守运算符的一个基本规则：始终返回一个新对象。也就是说，不能修改 self，要创建并返回合适类型的新实例。
- 重载向量加法运算符+
  - 序列应该支持 + 运算符（用于拼接），以及 * 运算符（用于重复复制）。
  - 实现一元运算符和中缀运算符的特殊方法一定不能修改操作数。
    - 使用这些运算符的表达式期待结果是新对象。
    - 只有增量赋值表达式可能会修改第一个操作数（self），参见 13.6 节。
  - 为了支持涉及不同类型的运算，Python 为中缀运算符特殊方法提供了特殊的分派机制。
    - 对表达式 a + b 来说，解释器会执行以下几步操作
      - (1) 如果 a有 __add__方法，而且返回值不是 NotImplemented，调用 a.__add__(b)，然后返回结果。
      - (2) 如果 a 没有 __add__ 方法，或者调用 __add__ 方法返回 NotImplemented，检查 b 有没有__radd__ 方法，如果有，而且没有返回 NotImplemented，调用 b.__radd__(a)，然后返回结果。
      - (3) 如果 b 没有 __radd__ 方法，或者调用 __radd__ 方法返回 NotImplemented，抛出 TypeError，并在错误消息中指明操作数类型不支持。
      - 

        图 13-1：使用 __add__ 和 __radd__ 计算 a + b 的流程图
    - __radd__ 是 __add__ 的“反射”（reflected）版本或“反向”（reversed）版本。
    - 如果中缀运算符方法抛出异常，就终止了运算符分派机制。对 TypeError 来说，通常最好将其捕获，然后返回 NotImplemented。
      - NotImplemented 值不是异常
      - 这样，解释器会尝试调用反向运算符方法，如果操作数是不同的类型，对调之后，反向运算符方法可能会正确计算。
- 重载标量乘法运算符*
  - __mul__ 和 __rmul__ 方法
  - 采用鸭子类型技术，在 __mul__ 方法中捕获 TypeError。
  - 白鹅类型
    - 使用isinstance() 检查参数的类型，但是不硬编码具体的类型。
    - 参考数字塔，例如检查 numbers.Real 抽象基类
      - 这个抽象基类涵盖了我们所需的全部类型，而且还支持以后声明为 numbers.Real 抽象基类的真实子类或虚拟子类的数值类型。
  - 表 13-1. 中缀运算符方法的名称（如果运算符用于操作数类型，比较运算符在表 13-2 中）

运算符	正向方法	反向方法	就地方法	说明
+	__add__	__radd__	__iadd__	加法或连接
-	__sub__	__rsub__	__isub__	减法
*	__mul__	__rmul__	__imul__	乘法或序列重复
/	__truediv__	__rtruediv__	__itruediv__	除法
//	__floordiv__	__rfloordiv__	__ifloordiv__	向下舍入整除
%	__mod__	__rmod__	__imod__	取模
divmod()	__divmod__	__rdivmod__	__idivmod__	返回由商和模数组成的元组
** 或 pow()	__pow__	__rpow__	__ipow__	幂
@	__matmul__	__rmatmul__	__imatmul__	矩阵乘法
&	__and__	__rand__	__iand__	位与
|	__or__	__ror__	__ior__	位或
^	__xor__	__rxor__	__ixor__	异或
<<	__lshift__	__rlshift__	__ilshift__	按位左移
>>	__rshift__	__rrshift__	__irshift__	按位右移

    * pow()可选一个参数 modulo 也可调用。pow(a, b, modulo)，但是用内置函数时这种形式会引发类型错误。——pow.__doc__ 的输出。
    * Python 3.5 新引入的。
  - 表 13-2. 丰富比较运算符，正向方法调用rich(a)==rich(b)的话，调用反向方法

分组	中缀运算符	正向方法调用	反向方法调用	回退机制
相等性	a == b	a.__eq__(b)	b.__eq__(a)	返回 id(a) == id(b)
	a != b	a.__ne__(b)	b.__ne__(a)	返回 not (a == b)
排序	a > b	a.__gt__(b)	b.__lt__(a)	抛出 TypeError
	a < b	a.__lt__(b)	b.__gt__(a)	抛出 TypeError
	a >= b	a.__ge__(b)	b.__le__(a)	抛出 TypeError
	a <= b	a.__le__(b)	b.__ge__(a)	抛出 TypeError

  - 众多比较运算符
    - Python 解释器对众多比较运算符（==、!=、>、<、>=、<=）的处理与前文类似，不过在两个方面有重大区别。
      - 正向和反向调用使用的是同一系列方法。
      - 例如， 对 ==来说，正向和反向调用都是 __eq__ 方法，只是把参数对调了；而正向的 __gt__ 方法调用的是反向的 __lt__ 方法，并把参数对调。
    - 对 == 和 != 来说，如果反向调用失败，Python 会比较对象的 ID，而不抛出 TypeError。
    - 定义了 __eq__ 方法，而且它不返回 NotImplemented，__ne__ 会对 __eq__ 返回的结果取反。
  - 增量赋值运算符
    - 如果一个类没有实现表 13–1 列出的就地运算符，增量赋值运算符只是语法糖：a += b 的作用与 a = a + b 完全一样。
      - 对不可变类型来说，这是预期的行为，而且，如果定义了 __add__ 方法的话，不用编写额外的代码，+= 就能使用。
    - 然而，如果实现了就地运算符方法，例如 __iadd__，计算 a += b 的结果时会调用就地运算符方法。
      - 这种运算符的名称表明，它们会就地修改左操作数，而不会创建新对象作为结果。
      - 不可变类型，如 Vector 类，一定不能实现就地特殊方法。
    - 增量赋值特殊方法必须返回 self。
    - 一般来说，如果中缀运算符的正向方法（如 __mul__）只处理与 self 属于同一类型的操作数，那就无需实现对应的反向方法（如 __rmul__），因为按照定义，反向方法是为了处理类型不同的操作数。