Contents

Chapter 1

迭代和推导

1.1 迭代器：初次探索

出于明确性，这里倾向于使用术语可迭代对象来指代有一个支持 iter 调用的对象，使用术语迭代器来指代一个（iter 调用为传入的可迭代对象返回的）支持 next(I) 调用的对象。

1.1.1 迭代协议：文件迭代器

所有带有 __next__ 方法的对象会前进到下一个结果，而在一系列结果的末尾时，则会引发 StopIteration 异常，这种对象在 Python 中也被称为迭代器。任何这类对象也能以 for 循环或其他迭代工具遍历，因为所有迭代工具内部工作起来都是在每次迭代中调用 __next__，并且捕捉 StopIteration 异常来确定何时离开。

while 循环会比基于迭代器的 for 循环运行得更慢，因为迭代器在 Python 中是以 C 语言的速度运行的，而 while 循环版本则是通过Python 虚拟机运行 Python 字节码的。任何时候，我们把 Python 代码换成 C 程序代码，速度都应该会变快。然而，并非绝对如此。

1.1.2 手动迭代：iter 和 next

迭代协议还有一点值得注意。当 for 循环开始时，会通过它传给 iter 内置函数，以便从可迭代对象中获得一个迭代器，返回的对象含有需要的 next 方法。

完整得迭代协议

作为更正式的定义，Figure 1.1 描绘了这个完整的迭代协议，Python 中的每个迭代工具都使用它，并受到各种对象类型的支持。它实际上基于两个对象，由迭代工具在两个不同的步骤中使用：

- 您请求迭代的可迭代对象，其 __iter__ 由 iter 运行
- 由迭代器返回的迭代器对象，在迭代过程中实际产生值，其 __next__ 由 next 运行，并在完成产生结果时引发 StopIteration

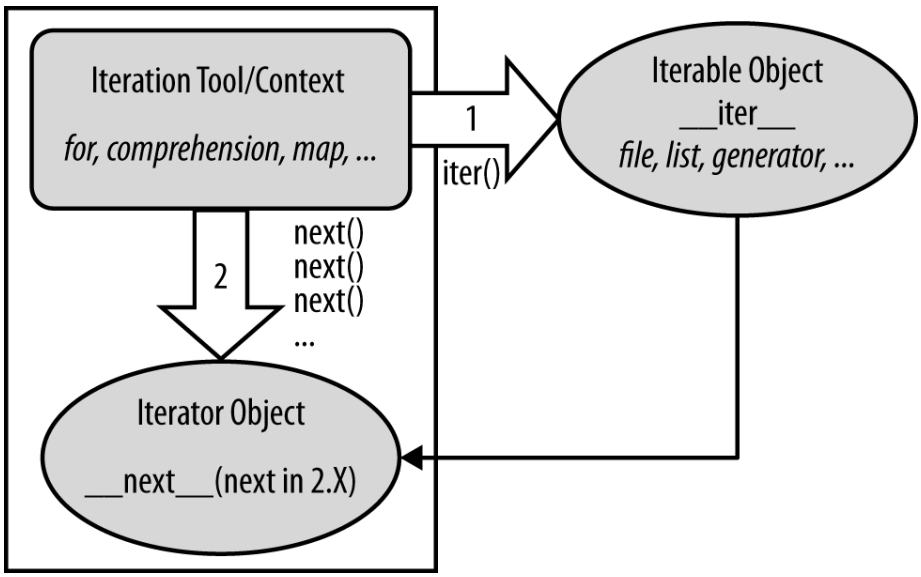

Figure 1.1: Python 迭代协议，由 for 循环、推导式、映射等使用，并受文件、列表、字典、Chapter 2 的生成器等支持。有些对象既是迭代上下文又是可迭代对象，例如生成器表达式和 3.X 风格的某些工具（例如 map 和 zip）。有些对象既是可迭代的又是迭代器，为 iter() 调用返回自身，这就是一个无操作。

Chapter 2

推导与生成