4.1 手动遍历迭代器¶

问题¶

你想遍历一个可迭代对象中的所有元素，但是却不想使用for循环。

解决方案¶

为了手动的遍历可迭代对象，使用 `next()` 函数并在代码中捕获 `StopIteration` 异常。 比如，下面的例子手动读取一个文件中的所有行：

```
def manual_iter():
    with open('/etc/passwd') as f:
        try:
            while True:
                line = next(f)
                print(line, end='')
        except StopIteration:
            pass
```

通常来讲， `StopIteration` 用来指示迭代的结尾。 然而，如果你手动使用上面演示的 `next()` 函数的话，你还可以通过返回一个指定值来标记结尾，比如 `None` 。 下面是示例：

```
with open('/etc/passwd') as f:
    while True:
        line = next(f, None)
        if line is None:
            break
        print(line, end='')
```

讨论¶

大多数情况下，我们会使用 `for` 循环语句用来遍历一个可迭代对象。 但是，偶尔也需要对迭代做更加精确的控制，这时候了解底层迭代机制就显得尤为重要了。

下面的交互示例向我们演示了迭代期间所发生的基本细节：

```
>>> items = [1, 2, 3]
>>> # Get the iterator
>>> it = iter(items) # Invokes items.__iter__()
>>> # Run the iterator
>>> next(it) # Invokes it.__next__()
1
>>> next(it)
2
>>> next(it)
3
>>> next(it)
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
StopIteration
>>>
```

本章接下来几小节会更深入的讲解迭代相关技术，前提是你先要理解基本的迭代协议机制。 所以确保你已经把这章的内容牢牢记在心中。