1.3 保留最后 N 个元素¶

问题¶

在迭代操作或者其他操作的时候，怎样只保留最后有限几个元素的历史记录？

解决方案¶

保留有限历史记录正是 `collections.deque` 大显身手的时候。比如，下面的代码在多行上面做简单的文本匹配， 并返回匹配所在行的最后N行：

```
from collections import deque


def search(lines, pattern, history=5):
    previous_lines = deque(maxlen=history)
    for line in lines:
        if pattern in line:
            yield line, previous_lines
        previous_lines.append(line)

# Example use on a file
if __name__ == '__main__':
    with open(r'../../cookbook/somefile.txt') as f:
        for line, prevlines in search(f, 'python', 5):
            for pline in prevlines:
                print(pline, end='')
            print(line, end='')
            print('-' * 20)
```

讨论¶

我们在写查询元素的代码时，通常会使用包含 `yield` 表达式的生成器函数，也就是我们上面示例代码中的那样。 这样可以将搜索过程代码和使用搜索结果代码解耦。如果你还不清楚什么是生成器，请参看 4.3 节。

使用 `deque(maxlen=N)` 构造函数会新建一个固定大小的队列。当新的元素加入并且这个队列已满的时候， 最老的元素会自动被移除掉。

代码示例：

```
>>> q = deque(maxlen=3)
>>> q.append(1)
>>> q.append(2)
>>> q.append(3)
>>> q
deque([1, 2, 3], maxlen=3)
>>> q.append(4)
>>> q
deque([2, 3, 4], maxlen=3)
>>> q.append(5)
>>> q
deque([3, 4, 5], maxlen=3)
```

尽管你也可以手动在一个列表上实现这一的操作（比如增加、删除等等）。但是这里的队列方案会更加优雅并且运行得更快些。

更一般的， `deque` 类可以被用在任何你只需要一个简单队列数据结构的场合。 如果你不设置最大队列大小，那么就会得到一个无限大小队列，你可以在队列的两端执行添加和弹出元素的操作。

代码示例：

```
>>> q = deque()
>>> q.append(1)
>>> q.append(2)
>>> q.append(3)
>>> q
deque([1, 2, 3])
>>> q.appendleft(4)
>>> q
deque([4, 1, 2, 3])
>>> q.pop()
3
>>> q
deque([4, 1, 2])
>>> q.popleft()
4
```

在队列两端插入或删除元素时间复杂度都是 `O(1)` ，区别于列表，在列表的开头插入或删除元素的时间复杂度为 `O(N)` 。