4.14 展开嵌套的序列¶

问题¶

你想将一个多层嵌套的序列展开成一个单层列表

解决方案¶

可以写一个包含 `yield from` 语句的递归生成器来轻松解决这个问题。比如：

```
from collections import Iterable

def flatten(items, ignore_types=(str, bytes)):
    for x in items:
        if isinstance(x, Iterable) and not isinstance(x, ignore_types):
            yield from flatten(x)
        else:
            yield x

items = [1, 2, [3, 4, [5, 6], 7], 8]
# Produces 1 2 3 4 5 6 7 8
for x in flatten(items):
    print(x)
```

在上面代码中， `isinstance(x, Iterable)` 检查某个元素是否是可迭代的。 如果是的话， `yield from` 就会返回所有子例程的值。最终返回结果就是一个没有嵌套的简单序列了。

额外的参数 `ignore_types` 和检测语句 `isinstance(x, ignore_types)` 用来将字符串和字节排除在可迭代对象外，防止将它们再展开成单个的字符。 这样的话字符串数组就能最终返回我们所期望的结果了。比如：

```
>>> items = ['Dave', 'Paula', ['Thomas', 'Lewis']]
>>> for x in flatten(items):
...     print(x)
...
Dave
Paula
Thomas
Lewis
>>>
```

讨论¶

语句 `yield from` 在你想在生成器中调用其他生成器作为子例程的时候非常有用。 如果你不使用它的话，那么就必须写额外的 `for` 循环了。比如：

```
def flatten(items, ignore_types=(str, bytes)):
    for x in items:
        if isinstance(x, Iterable) and not isinstance(x, ignore_types):
            for i in flatten(x):
                yield i
        else:
            yield x
```

尽管只改了一点点，但是 `yield from` 语句看上去感觉更好，并且也使得代码更简洁清爽。

之前提到的对于字符串和字节的额外检查是为了防止将它们再展开成单个字符。 如果还有其他你不想展开的类型，修改参数 `ignore_types` 即可。

最后要注意的一点是， `yield from` 在涉及到基于协程和生成器的并发编程中扮演着更加重要的角色。 可以参考12.12小节查看另外一个例子。