4.12 不同集合上元素的迭代¶

问题¶

你想在多个对象执行相同的操作，但是这些对象在不同的容器中，你希望代码在不失可读性的情况下避免写重复的循环。

解决方案¶

`itertools.chain()` 方法可以用来简化这个任务。 它接受一个可迭代对象列表作为输入，并返回一个迭代器，有效的屏蔽掉在多个容器中迭代细节。 为了演示清楚，考虑下面这个例子：

```
>>> from itertools import chain
>>> a = [1, 2, 3, 4]
>>> b = ['x', 'y', 'z']
>>> for x in chain(a, b):
... print(x)
...
1
2
3
4
x
y
z
>>>
```

使用 `chain()` 的一个常见场景是当你想对不同的集合中所有元素执行某些操作的时候。比如：

```
# Various working sets of items
active_items = set()
inactive_items = set()

# Iterate over all items
for item in chain(active_items, inactive_items):
    # Process item
```

这种解决方案要比像下面这样使用两个单独的循环更加优雅，

```
for item in active_items:
    # Process item
    ...

for item in inactive_items:
    # Process item
    ...
```

讨论¶

`itertools.chain()` 接受一个或多个可迭代对象作为输入参数。 然后创建一个迭代器，依次连续的返回每个可迭代对象中的元素。 这种方式要比先将序列合并再迭代要高效的多。比如：

```
# Inefficent
for x in a + b:
    ...

# Better
for x in chain(a, b):
    ...
```

第一种方案中， `a + b` 操作会创建一个全新的序列并要求a和b的类型一致。 `chian()` 不会有这一步，所以如果输入序列非常大的时候会很省内存。 并且当可迭代对象类型不一样的时候 `chain()` 同样可以很好的工作。