对于可迭代数据结构中的所有元素，使用列表生成式代替for，一次性批量处理，避免单个处理。

＃1 列表生成式

//使用不同区域的数据

import numpy as np

[m\*n for m in np.array([[1,2],[3,4]]) for n in np.array([[1,2],[3,4]])]

输出：

[array([1, 4]), array([3, 8]), array([3, 8]), array([ 9, 16])]

//使用条件判断

[m\*n for m in range(1,10) if m in (2,5,6) for n in range(2,5) if n%2==0]

输出：

[4, 8, 10, 20, 12, 24]

//对列表里每个元素进行函数操作

L=['java','C++','python']

[s.upper() for s in L]

输出：

['JAVA', 'C++', 'PYTHON']

//回去找一下map()

＃2 生成器：

当只需要列表生成式前几个元素时，使用生成器，节省空间，加快速度。生成器的声明只需要将列表生成器的［］改为（）就可，需要元素的时候，调用next()产生下一个，但一般用for循环

>>> g=(x+x for x in range(1,3))

>>> g

<generator object <genexpr> at 0x101184fc0>

>>> next(g)

2

>>> next(g)

4

>>> next(g)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

StopIteration

>>>

>>>for n in g:

... print(n)

...

2

4

//未完成，下午接着做

＃3将函数处理成生成器

将函数处理成生成器，需要改变时在直接调用，不像列表那么死板。如果想要得到generator最终的返回值，需要捕捉错误

def fib(max):

n, a, b = 0, 0, 1

while n < max:

yield b

a, b = b, a + b

n = n + 1

return 'done'

>>> g=fib(2)

>>> while True:

... try:

... x=next(g)

... print('g:',x)

... except StopIteration as e:

... print('Generator return Value:',e.value)

... break

...

g: 1

g: 1

Generator return Value: done

＃4 使用map/reduce处理可迭代数据结构（应该可以迭代class，那么就可以对自定义类中数据批处理）

>>> def f(x):

... return x \* x

...

>>> r = map(f, [1, 2, 3, 4, 5, 6, 7, 8, 9])

>>> list(r)

[1, 4, 9, 16, 25, 36, 49, 64, 81]

reduce将结果与下一元素累计运算，相当于

reduce(f,[a1,a2,a3])=(f(a1,a2),a3)

from functools import reduce

def str2int(s):

def fn(x, y):

return x \* 10 + y

def char2num(s):

return {'0': 0, '1': 1, '2': 2, '3': 3, '4': 4, '5': 5, '6': 6, '7': 7, '8': 8, '9': 9}[s]

return reduce(fn, map(char2num, s))