python复习4 迭代器生成器

迭代器

1. 概念

- 可迭代对象

每次能返回其一个成员的对象(An object capable of returning its members one at a time)，即实现__

iter__()或__getitem__()协议的对象。

Python提供了两个通用迭代器对象：

›序列对象：

list, str, tuple

›非序列对象：

dict, file objects

可迭代对象可用于for循环，及其它需要序列的地方（如zip()、map() ...）

使用内置函数iter()，或者__iter__()方法，可将可迭代对象转换为迭代器iterator。

示例：

```
my_list = [1, 2, 3, 4]
my_iterable = iter(my_list) # 迭代器

for item in my_iterable:
    print(item)
```

- 迭代器

迭代器是一种对象，它实现了迭代器协议，包含 `iter()` 和 `next()` 方法。 `iter()` 返回迭代器对象自身，而 `next()` 返回下一个元素。

迭代器可以通过next()逐个返回元素。

iter()方法可以返回迭代器本身。

示例：

```
my_list = [1, 2, 3, 4]
```

```
my_iterator = iter(my_list)  # 迭代器

print(next(my_iterator))  # 输出 1
print(next(my_iterator))  # 输出 2
```

2.

生成器

1. 概念

生成器是一种特殊的迭代器。它允许在迭代过程中生成值，而不需要一次性存储所有值在内存中。生成器使用了一种称为 "惰性计算"（Lazy Evaluation）的机制，只在需要时才生成值，因此非常适合处理大量数据或需要逐个产生值的情况。

2. 创建方式

- 使用生成器函数

生成器函数是包含 `yield` 关键字的函数。`yield` 语句用于产生一个值，并暂停函数的执行，保留函数的状态。当再次调用生成器函数时，它将从上次暂停的位置恢复执行。

```
def simple_generator():
    yield 1
    yield 2
    yield 3

gen = simple_generator()  # 创建一个生成器对象

print(next(gen))  # 输出生成器的第一个值，即 1
print(next(gen))  # 输出生成器的下一个值，即 2
print(next(gen))  # 输出生成器的下一个值，即 3

```

- 使用生成器表达式

生成器表达式类似于列表推导式，但使用圆括号而不是方括号，并且返回一个生成器对象。

```
gen = (x for x in range(5))  # 创建一个生成器对象，生成 0 到 4 的数字

for value in gen:
    print(value)

```

3. 案例

```
def add(n,i):
    return n+i
def test():
    for i in range(4):
        yield i
g = test()
for n in [1,10]:
    g = (add(n,i) for i in g)
print(list(g))
```

```
# 该案例输出
[20,21,22,23]
# 解析
# 函数求和
def add(n, i):
    return n + i
# 调用之前是函数 调用之后是生成器
def test():
    for i in range(4):
        yield i
g = test()  # 初始化生成器对象
for n in [1, 10]:
    g = (add(n, i) for i in g)
'''
        第一次for循环
                g = (add(n, i) for i in g)  此时in后面g为test生成器
        第二次for循环
                g = (add(n, i) for i in g) 此时的g为第一次for循环得到的生成器g
                所以此时的可以看成   g = (add(n, i) for i in (add(n, i) for i in
g))
'''
res = list(g)
'''
调用时g是一个生成器 n的值为10  in后面的g 生成器产生的值就是 range(4)
g 可以看成
 g = (add(10, i) for i in (add(10, i) for i in range(4)))
'''
print(res)  # [20,21,22,23]

```