可迭代对象

```
str：字符串
list：列表
tuple：元组
set：集合
frozenset：不可变集合
dict：字典

1. 存储多个数据
2. 存储多种数据
3. 是可迭代对象
4. 部分支持索引操作
5. 部分支持切片操作
```

- 容器

```
container
1. 存储多个数据
2. 存储多种数据
3. 支持：成员关系：  in /not in
```

- 序列

```
sequence
1. 支持索引操作
2. 支持切片操作
3. 序列有顺序的概念
```

- 可迭代对象

```
iterable
1. 迭代：
        上一次运行的结果作为下一次运行的开始或条件
2. 拥有迭代器的对象  称之为可迭代对象
        实现了__iter__()
```

- 迭代器

```
iterator
1. 实现了__iter__()方法和__next__()方法的对象，是迭代器对象
        __iter__方法是魔法方法，用于返回一个迭代器
        __next__方法是魔法方法，用于获取下一个元素
2. 获取一个对象的迭代器：
        1. iter（可迭代对象）
            返回可迭代对象的迭代器
        2. 可迭代对象.__iter__()
            返回可迭代对象的迭代器
3. 迭代器的结构：
        1. 迭代器的数据是可迭代对象拷贝过来的
        2. 迭代器中有游标
```

游标一开始在第一个元素之前

每调用一次next()，游标向后挪动一个单位，并把游标划过的数据进行返回

如果游标之后没有任何数据，此时如果调用next()，则会抛出异常：StopIteration （停止迭代异常）

3. 游标可以记录执行位置

4. 获取迭代器中的一个元素：

1. next(可迭代对象)

返回迭代器的下一个元素

2. 迭代器.__next__()

返回迭代器的下一个元素

5. 迭代器也应该实现了__iter__()，迭代器的迭代器就是自己

迭代器拥有迭代器---可迭代对象

迭代器是一个可迭代对象

```
# l=[1,2,3]
#
# # l是列表  是可迭代对象（因为拥有迭代器）
# # 不是迭代器（只有iter 没有next）
#
# a=iter(l)
# # a 是迭代器（实现了next  ）
#
# print(a)
# print(iter(a))


# l=[1,2,3]
```

```
#
# print(iter(l))
# # print(next(l))
# a=iter(l)
# print(next(a))


# l=[1,2,3]
# for i in l: # next()
#     print(i)
#
# # for 循环的执行机制：
# # 1. 调用可迭代对象的迭代器
# # 2. 对迭代器调用next
# # 3. 捕获StopIteration，停止循环



# l=[1,2,3]
# a=iter(l)
#
# for i in a:
#     print(i)
#

# f=filter(None,[0,1,2,3])
# # 1,2,3
# print(id(f))
```

```
# print(id(iter(f)))
# for i in f:
#     print(i)
#
# print(list(f)) # list()---for循环


f=filter(None,[0,1,2,3])
# 1,2,3
print(id(f))
print(id(iter(f)))
for i in f:
    print(i)
    if i==2:
        break


print(list(f)) # list()---for循环
```

- 生成器

generator

1. 生成器表达式：

    语法：

    （i for i in range(10)）

2. 使用yield关键字创建

    如果一个函数中有yield关键字，函数（ ）返回的不再是函数的返回值，而是一个生成器对象

3. 生成器的特点：

    1. 是可迭代对象
    2. 可以存储多个数据
    3. 可以存储多种数据
    4. 生成器是一个特殊的迭代器
    5. 按需供应数据，节省大量空间