PYTHON BASICS
生成器和迭代器

2020-05-07

STEVEN WANG

上节总结

类型	含义
列表解析式	[expr_item for item in collection if cond]
字典解析式	{key:expr_val for key,val in collection if cond}
集合解析式	{ expr_item for item in collection if cond }
元组解析式	(expr_item for item in collection if cond)

生成器

生成器 迭代器

- 生成器
- 迭代器
- 专用迭代器

创建生成器

生成函数

```
def make_list(collection):
    result = []
    for x in collection:
        result.append(x**x)
    return result
```

```
def make_generator(collection):
    for x in collection:
        yield x**x
```

生成表达式

```
[ x**x for x in collection ]
```

```
( x**x for x in collection )
```

生成器的特点

列表	生成器
有长度、可以索引	无长度、不能索引
一次性出值	一个个出值
无记忆，但能循环多次	有记忆，只能循环一次
速度慢、占内存	速度快、省内存

神秘

懒惰

神奇

高效

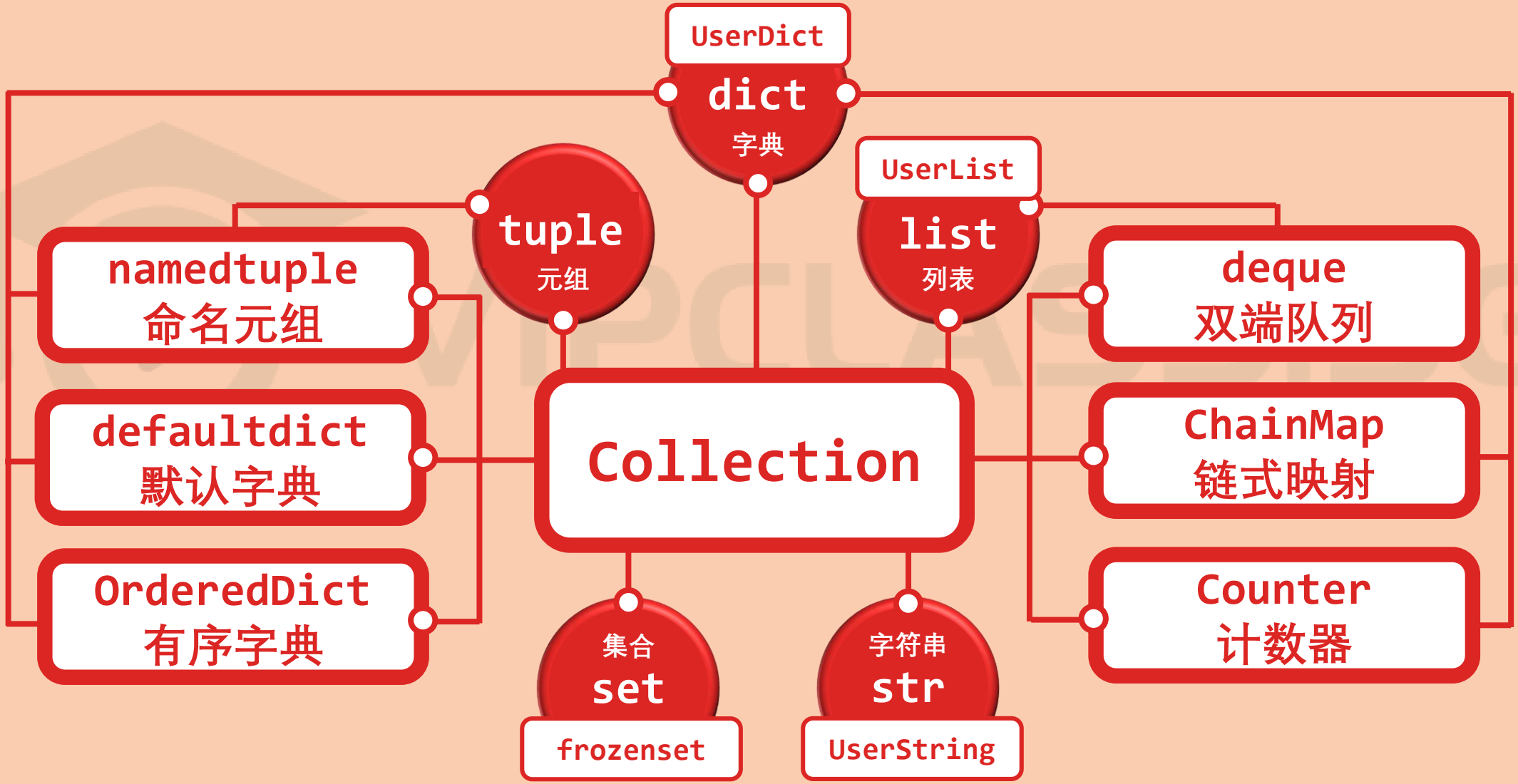
当只需进行一次循环操作，用生成器

生成器 迭代器

- 生成器
- 迭代器
- 专用迭代器

可迭代对象

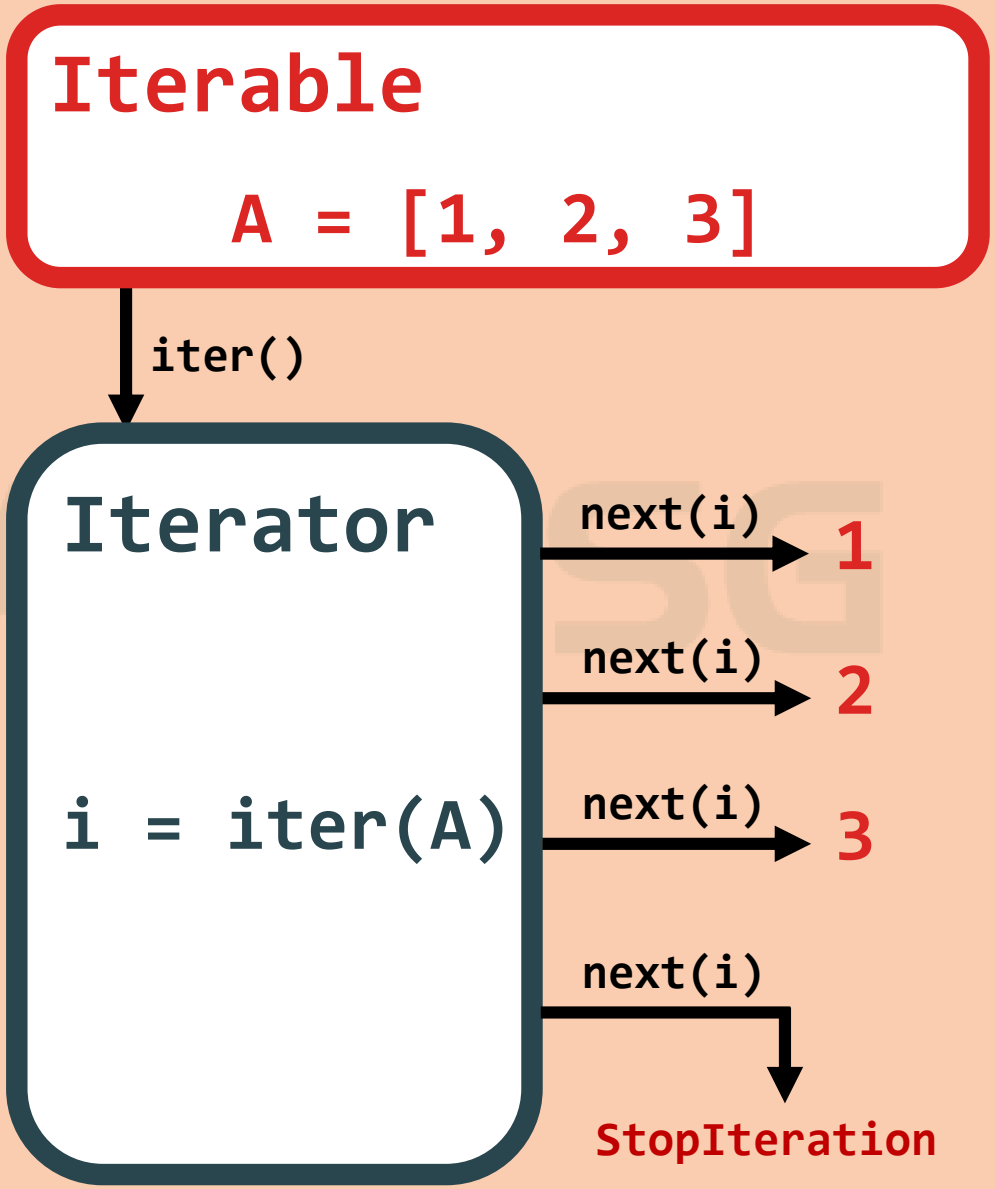
任何可循环的东西就是可迭代对象 (iterable)，类定义中有 `__iter__()` 方法。



可迭代对象 Vs 迭代器


```
for item in iterable:
    statement(s)
```

迭代器的类定义中有 `__iter__()` 和 `__next__()` 方法，因此迭代器一定就是可迭代对象，但可迭代对象不一定是迭代器。



创建迭代器

用类

```
class MyRange:

    def __init__(self, start, end):
        self.value = start
        self.end = end

    def __iter__(self):
        return self

    def __next__(self):
        if self.value >= self.end:
            raise StopIteration
        current = self.value
        self.value += 1
        return current
```

用生成器

```
def range_generator(start, end):
    current = start
    while current < end:
        yield current
        current += 1
```

内置迭代器


```
from collections import abc
l = [1,2,3]
r = [4,5,6]
```

enumerate
zip
map
filter

```
print( isinstance(enumerate(l), abc.Iterator) )
print( isinstance(zip(l,r), abc.Iterator) )
print( isinstance(map(lambda x:x*x,l), abc.Iterator) )
print( isinstance(filter(lambda x:x>1,l), abc.Iterator) )
```

```
True
True
True
True
```

range

```
print( isinstance(range(10), abc.Iterator) )
print( isinstance(range(10), abc.Iterable) )
```

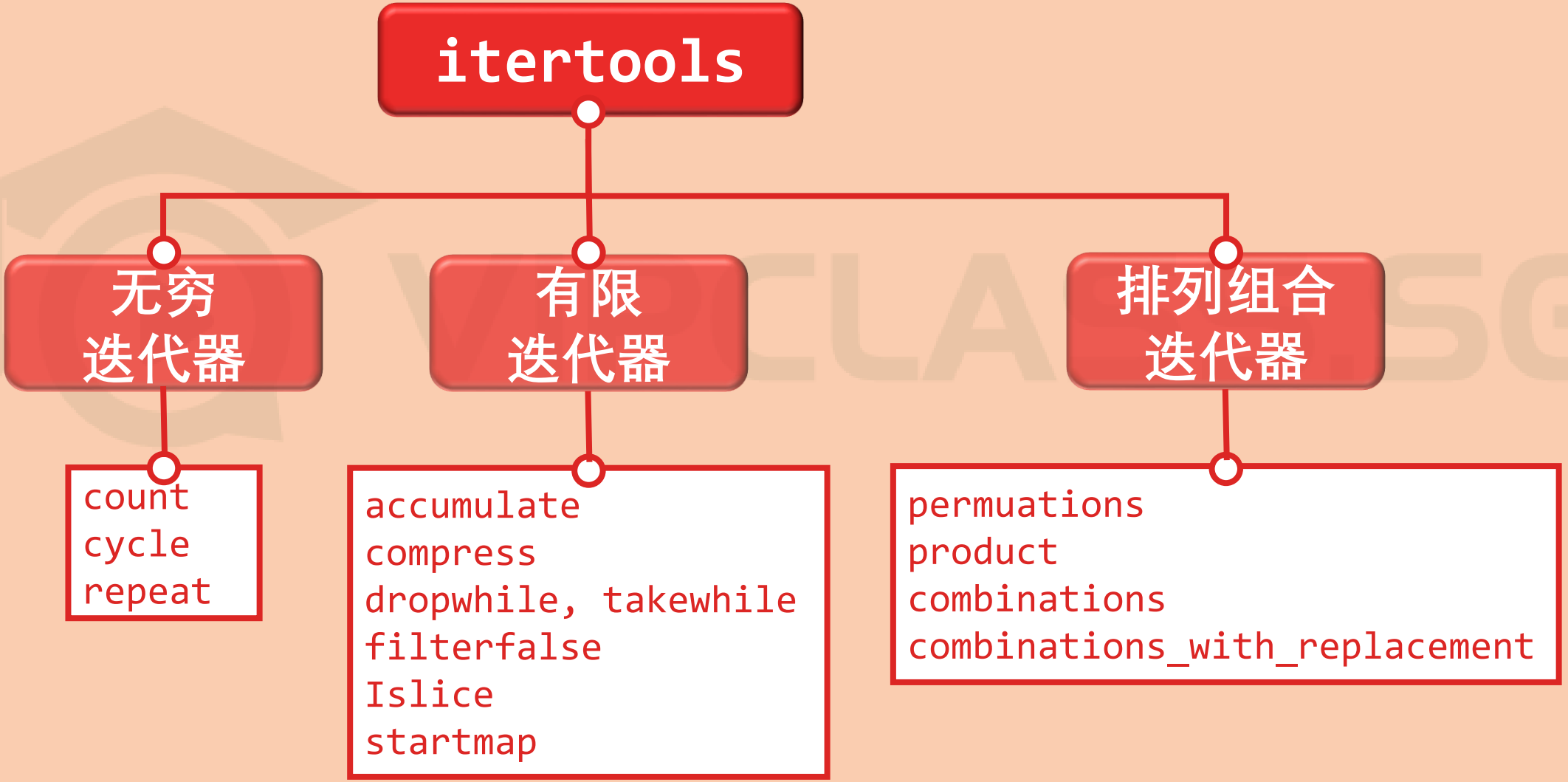
```
False
True
```

迭代器 生成器

- 生成器
- 可迭代对象
- 专用迭代器

itertools

itertools 包里实现不少专用迭代器，专门为高效循环而创建迭代器。



无穷迭代器

无限计数　无限循环　无限重复

count(start=0, step=1)　　1, 2, 3, 4, 5, 6, ……

cycle(🔴⚫🟣🟢)　　🔴⚫🟣🟢🔴⚫🟣🟢 ……

repeat(🔴, times=None)　　🔴🔴🔴🔴🔴🔴🔴🔴 ……

有限迭代器

`accumulate( iterable, func )`	用 func 累积元素
`compress( iterable, selectors )`	用 selectors 的真假来筛选元素
`dropwhile( pred, iterable )`	当 pred 第一次为真，丢弃之前元素
`takewhile( pred, iterable )`	当 pred 第一次为真，收集之前元素
`filterfalse( pred, iterable )`	丢弃 pred 为假的元素
`islice( iterable, stop )`	根据索引切片元素
`starmap( func, iterable )`	用 func 映射元素
`chain( *iterable )`	将所有 iterable 里的元素串联起来

排列组合迭代器

从 4 个球中选 2 个

$n = 4, r = 2$

排列 $\Leftrightarrow \mathrm{P}(n,r) = n \times (n-1) \times \cdots \times (n-r-1) = \frac{n!}{(n-r)!}$ $\mathrm{P}(4,2) = \frac{4!}{2!} = 12$

`permutations(`  `, r)`

穷举 $\Leftrightarrow \mathrm{P}^R(n,r) = \underbrace{n \times n \times \cdots \times n}_{r} = n^r$ $\mathrm{P}^R(4,2) = 4^2 = 16$

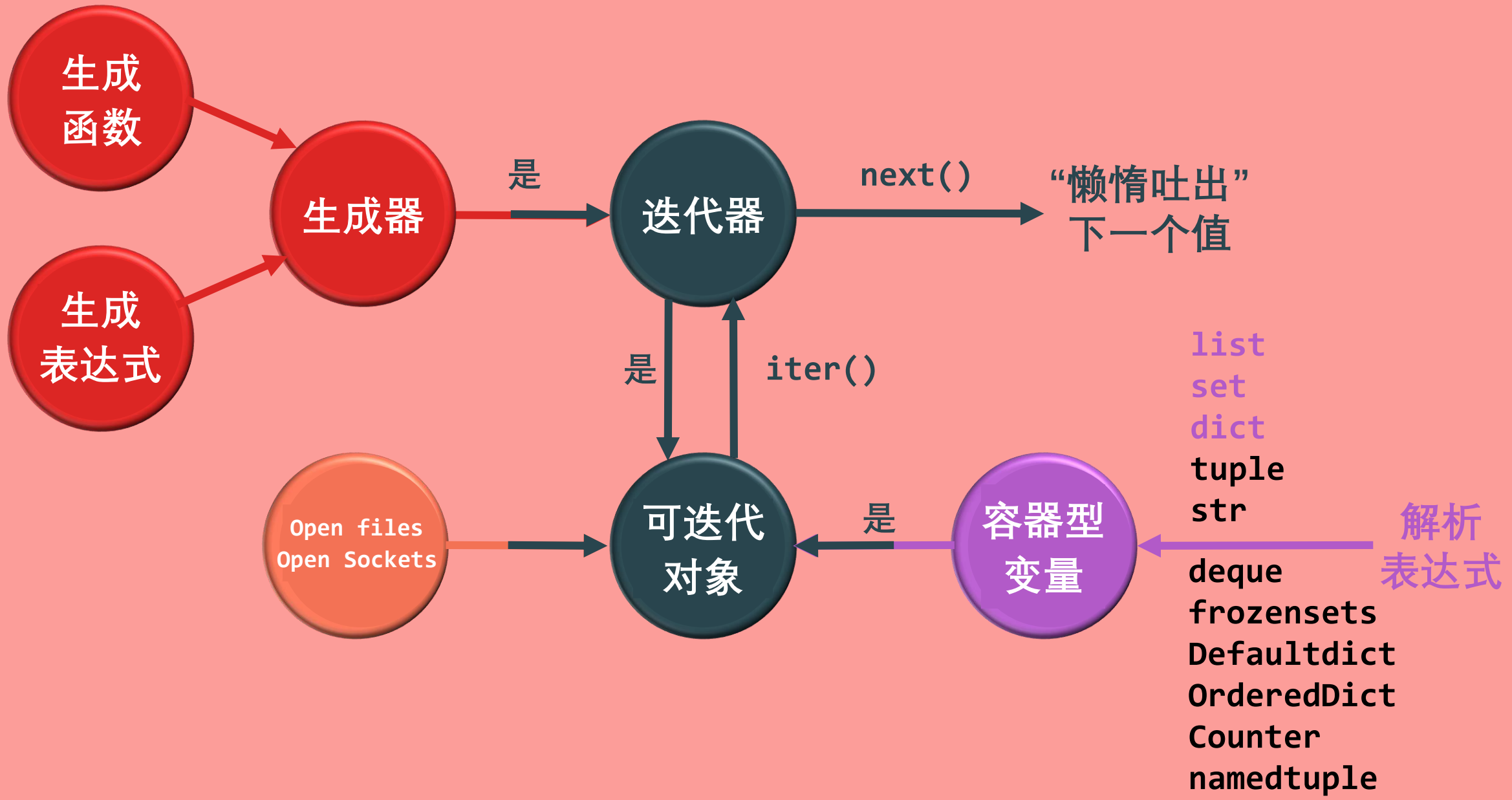
`product(`  `, repeat=r)`

组合 $\Leftrightarrow \mathrm{C}(n,r) = \frac{\mathrm{P}(n,r)}{r!} = \frac{n!}{r!(n-r)!}$ $\mathrm{C}(4,2) = \frac{4!}{2!\,2!} = 6$

`combinations(`  `, r)`

置换组合 $\Leftrightarrow \mathrm{C}^R(n,r) = \frac{(n+r-1)!}{r!(n-1)!}$ $\mathrm{C}^R(4,2) = \frac{5!}{2!\,3!} = 10$

`combinations_with_replacement(`  `, r)`


生成
函数
生成
表达式
生成器
是
迭代器
next()
“懒惰吐出”
下一个值
是
iter()
Open files
Open Sockets
可迭代
对象
是
容器型
变量
list
set
dict
tuple
str
deque
frozensets
Defaultdict
OrderedDict
Counter
namedtuple
解析
表达式

总结

类型	创建方法	用处
生成器	用生成函数 + `yield` 用生成表达式 + `()`	一次循环
可迭代对象	用解析表达式	多次循环
迭代器	用类实现 `iter()` 和 `next()` 用生成器	一次循环
专用迭代器	使用 `itertools` 包	花式循环

下节预告：装饰器


VIPCLASS.SG
small class, world class
终身学习
快乐学习
王圣元 Steven Wang
微信公众号：王的机器