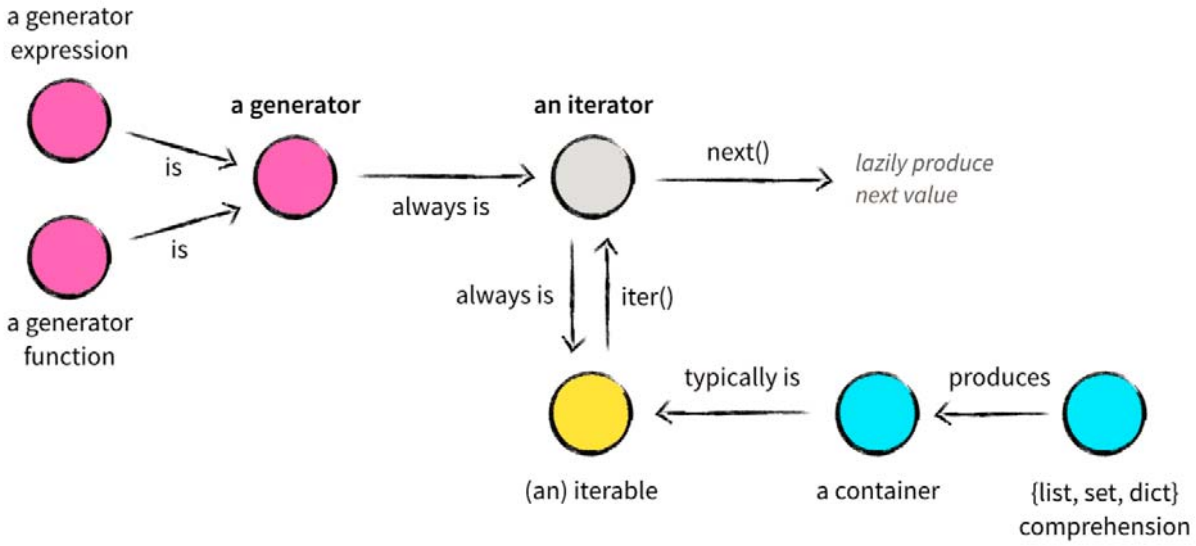
python中的生成器和迭代器

概念框图



生成器 <generator>

1. 第一种创建生成器的方式 (序列)

```
gen = (x for x in rage(1000000))
```

2.第二种创建生成器的方式 yield 关键字

```
def gen():
    print('ok1')
    yield 1
    print('ok2')
    yield 2

g = gen()
print(next(g)) # 打印ok1 和 1
print(next(g)) # 打印ok2 和 2
```

3.几种遍历生成器的方式

- python2.0 用 `g.next()` *or* `g.__next__()`
- python3.0 用 `g.__next__()` *or* `next(g)`
- 用for循环进行遍历

```
for i in generator_obj: # for循环内部调用next
    print(i)
```

- 用 send() 方法传值

```
def gen():
    print('ok1')
    msg = yield 1
    print('ok2')
    yield 2

g = gen()
g.send(None) # 第一次取值时必须传入None
g.send('first msg')
# 第一次send()时， 执行到 yield 1, 将 1 返回
# 第二次send()时，从yield 1开始执行，将'first msg'赋值给msg，执行到yield 2
```

4. 通过生成器实现伪并发

```
import time
def consumer(name):
    print("%s 准备吃包子了" % (name))
    while True:
        baozi = yield
        print("包子[%s]来了，被[%s]吃了!" % (baozi, name))

def producer(name):
    c = consumer('A')
    next(c)
    print('开始准备做包子了！')
    for i in range(10):
      time.sleep(1)
      print('做了1个包子！')
      c.send(i+1)

producer('BAOZI')
```

迭代器 <iterator>

生成器都是迭代器，迭代器不一定是生成器

通过 iter() 返回迭代器对象

```
# list, tuple, dict, string... -> Iterable （可迭代对象）
l = ['a', 'b', 'c', 'd']
iterator_obj = iter(l)
print(iterator_obj) # <list_iterator object>
```

满足两个条件就是迭代器

1. 具有 `__iter__()` 方法

2. 具有 `__next__()` 方法