Python可迭代对象、迭代器和生成器

序列可以迭代的原因：iter 函数。解释器需要迭代对象 x 时，会自动调用 iter(x)。

内置的 iter 函数有以下作用：

（1）检查对象是否实现了 iter 方法，如果实现了就调用它，获取一个迭代器。

（2）如果没有实现 iter 方法，但是实现了 getitem 方法，而且其参数是从零开始的索引，Python 会创建一个迭代器，尝试按顺序（从索引 0 开始）获取元素。

（3）如果前面两步都失败，Python 抛出 TypeError 异常，通常会提示“C objectis not iterable”（C 对象不可迭代），其中 C 是目标对象所属的类。

由此我们可以明确知道什么是 可迭代的对象： 使用 iter 内置函数可以获取迭代器的对象。即要么对象实现了能返回迭代器的 iter 方法，要么对象实现了 getitem 方法，而且其参数是从零开始的索引。

下面看一个实现了getitem方法的例子：

```
class Eg1:
    def __init__(self, text):
        self.text = text
        self.sub_text = text.split(' ')

    def __getitem__(self, index):
        return self.sub_text[index]

o1 = Eg1('Hello, the wonderful new world!')
for i in o1:
    print(i)
```

输出结果：

```
Hello,
the
wonderful
new
world!
```

我们创建了一个类Eg1，并且为这个类实现了 getitem 方法， 它的实例化对象o1 就是可迭代对象。

下面我们看一个实现 iter 方法的例子，因为用到了迭代器，所以在此我们必须在明确一下迭代器的用法。 标准的迭代器接口有两个方法：

```
__next__
```

返回下一个可用的元素，如果没有元素了，抛出 StopIteration异常。

1. `__iter__`

返回 self，以便在应该使用可迭代对象的地方使用迭代器，例如在 for 循环中。

```
class Eg2:
    def __init__(self, text):
        self.text = text
        self.sub_text = text.split(' ')

    def __iter__(self):
        return Eg2Iterator(self.sub_text)


class Eg2Iterator:
    def __init__(self, sub_text):
        self.sub_text = sub_text
        self.index = 0

    def __next__(self):
        try:
            subtext = self.sub_text[self.index]
        except IndexError:
            raise StopIteration()
        self.index += 1
        return subtext

    def __iter__(self):
        return self
```

我们创建了Eg2类，并为它实现了 iter 方法，此方法返回一个迭代器Eg2Iterator。Eg2Iterator 实现了我们之前所说的next和iter方法。 实例化对象，并循环输出：

```
o2 = Eg2('Hello, the wonderful new world!')
for i in o2:
    print(i)
Hello,
the
wonderful
new
world!
```

可见，和o1是一样的。

我们通过两种方法实现了一个自己的可迭代对象，再此过程中我们要明确可迭代的对象和迭代器之间的关系：

Python 从可迭代的对象中获取迭代器。

iter方法从我们自己创建的迭代器类中获取迭代器，而getitem方法是python内部自动创建迭代器。

至此，我们明白了如何正确地实现可迭代对象，并且引出了怎样实现迭代器，但是使用迭代器方法（即上面的例子2）的代码量有点大，下面我们来了解一下如何使用更符合 Python 习惯

的方式实现 Eg2类。

```
class Eg3:
    def __init__(self, text):
        self.text = text
        self.sub_text = text.split(' ')

    def __iter__(self):
        for item in self.sub_text:
            yield item
```

哦了！就这么简单优雅！不用再单独定义一个迭代器类！

这里我们使用了yield 关键字， 只要 Python 函数的定义体中有 yield 关键字，该函数就是生成器函数。调用生成器函数时，会返回一个生成器对象。也就是说，生成器函数是生成器工厂。 当然，例子3的代码还可以使用yield from进一步简化：

```
class Eg4:
    def __init__(self, text):
        self.text = text
        self.sub_text = text.split(' ')

    def __iter__(self):
        yield from self.sub_text
o4 = Eg4('Hello, the wonderful new world!')
for i in o4:
    print(i)
Hello,
the
wonderful
new
world!
```

到这里我们明白了 可迭代对象 和 迭代器，还引申出了生成器，但还有一点没有提，那就是生成器表达式。

使用生成器表达式例子4的代码可以修改为：

```
class Eg5:
    def __init__(self, text):
        self.text = text
        self.sub_text = text.split(' ')

    def __iter__(self):
        return (item for item in self.sub_text)
```

在python中，所有生成器都是迭代器。

最后，总结一下：

（1）什么是可迭代对象？ 可迭代对象要么实现了能返回迭代器的 `iter` 方法，要么实现了 `getitem` 方法而且其参数是从零开始的索引。

（2）什么是迭代器？ 迭代器是这样的对象：实现了无参数的 `next` 方法，返回下一个元素，

如果没有元素了，那么抛出 StopIteration 异常；并且实现`iter` 方法，返回迭代器本身。

（3）什么是生成器？ 生成器是带有 yield 关键字的函数。调用生成器函数时，会返回一个生成器对象。

（4）什么是生成器表达式？ 生成器表达式是创建生成器的简洁句法，这样无需先定义函数再调用。