迭代器

廖雪峰
资深软件开发工程师，业余马拉松选手。

我们已经知道，可以直接作用于 `for` 循环的数据类型有以下几种：

一类是集合数据类型，如 `list` 、 `tuple` 、 `dict` 、 `set` 、 `str` 等；

一类是 `generator` ，包括生成器和带 `yield` 的generator function。

这些可以直接作用于 `for` 循环的对象统称为可迭代对象： `Iterable` 。

可以使用 `isinstance()` 判断一个对象是否是 `Iterable` 对象：

```
>>> from collections.abc import Iterable
>>> isinstance([], Iterable)
True
>>> isinstance({}, Iterable)
True
>>> isinstance('abc', Iterable)
True
>>> isinstance((x for x in range(10)), Iterable)
True
>>> isinstance(100, Iterable)
False
```

而生成器不但可以作用于 `for` 循环，还可以被 `next()` 函数不断调用并返回下一个值，直到最后抛出 `StopIteration` 错误表示无法继续返回下一个值了。

可以被 `next()` 函数调用并不断返回下一个值的对象称为迭代器： `Iterator` 。

可以使用 `isinstance()` 判断一个对象是否是 `Iterator` 对象：

```
>>> from collections.abc import Iterator
>>> isinstance((x for x in range(10)), Iterator)
True
>>> isinstance([], Iterator)
False
>>> isinstance({}, Iterator)
False
>>> isinstance('abc', Iterator)
False
```

生成器都是 `Iterator` 对象，但 `list` 、 `dict` 、 `str` 虽然是 `Iterable` ，却不是 `Iterator` 。

把 `list` 、 `dict` 、 `str` 等 `Iterable` 变成 `Iterator` 可以使用 `iter()` 函数：

```
>>> isinstance(iter([]), Iterator)
True
>>> isinstance(iter('abc'), Iterator)
True
```

你可能会问，为什么 `list` 、 `dict` 、 `str` 等数据类型不是 `Iterator` ？

这是因为Python的 `Iterator` 对象表示的是一个数据流， `Iterator` 对象可以被 `next()` 函数调用并不断返回下一个数据，直到没有数据时抛出 `StopIteration` 错误。可以把这个数据流看做是一个有序序列，但我们却不能提前知道序列的长度，只能不断通过 `next()` 函数实现按需计算下一个数据，所以 `Iterator` 的计算是惰性的，只有在需要返回下一个数据时它才会计算。

`Iterator` 甚至可以表示一个无限大的数据流，例如全体自然数。而使用list是永远不可能存储全体自然数的。

参考源码

do_iterator.py

小结

凡是可作用于 `for` 循环的对象都是 `Iterable` 类型；

凡是可作用于 `next()` 函数的对象都是 `Iterator` 类型，它们表示一个惰性计算的序列；

集合数据类型如 `list` 、 `dict` 、 `str` 等是 `Iterable` 但不是 `Iterator` ，不过可以通过 `iter()` 函数获得一个 `Iterator` 对象。

Python的 `for` 循环本质上就是通过不断调用 `next()` 函数实现的，例如：

```
for x in [1, 2, 3, 4, 5]:
    pass
```

实际上完全等价于：

```
# 首先获得Iterator对象:
it = iter([1, 2, 3, 4, 5])
# 循环:
while True:
    try:
        # 获得下一个值:
        x = next(it)
    except StopIteration:
        # 遇到StopIteration就退出循环
        break
```

« 生成器　　函数式编程 »

Comments

Comments loaded. To post a comment, please Sign In

午间特困猿 @ 2025/12/6 03:28:25

如果想要充分了解，可以了解一下C++20的协程，充分认识懒汉模式（惰性）的思想，包括next或者for循环针对generator进行resume的原理

Floris @ 2025/11/27 01:55:08

特性	可迭代对象 (Iterable)	迭代器 (Iterator)
定义	可以被 迭代（遍历）的对象。	实现了迭代协议，知道如何进行下一次迭代 的对象。
包含	物 列表、元组、字符串、字典、集合、文件等。	通过对可迭代对象调用 iter() 函数返回的对象。
必须实现的方法	必须实现 **iter**() 方法。	必须实现 **iter**() 和 **next**() 方法。
作用	产生一个迭代器。	追踪迭代状态，并按需返回序列中的下一个元素。
自我迭代	不能原地进行迭代（需要先转换为迭代器）。	可以原地进行迭代（**iter** 通常返回 self）。

狼图腾-崛起 @ 2025/9/28 17:25:31

生活比喻总结

Iterable就像相册

- 可以被翻看（for循环）
- 但你不能直接问它"下一页是什么"
- 需要专门的翻页工具才能一页页看

Iterator就像翻页器

- 知道当前在哪一页
- 可以通过next()获取下一页内容
- 有记忆功能，按顺序翻页

Read More ▼

@ 2025/7/18 08:35:44

```
# 记得导入函数
from collections.abc import Iterator,Iterable
```

Salvatore @ 2025/9/24 03:40:55

so good

气过小张 @ 2025/9/9 21:39:49

问了下 gpt，解答了下为什么有可迭代对象还要有迭代器。 可迭代对象相当于是数据容器 迭代器是记录遍历过程 一个相同的可迭代对象可以有多个不同的迭代器 也就是相当于一本书可以有不同的书签记录当前的阅读位置 这样可以做到更高效的数据遍历能力

。。 @ 2025/9/22 05:14:08

thank you

西瓜 @ 2025/9/7 01:14:05

Iterable存储空间是所有迭代的元素，空间占用大 Iterator存储空间小，next()函数调用一次计算一次，惰性计算

ss @ 2025/8/20 22:38:52

```
Iterable（可迭代对象）：就像数学里的 有限集合，可以一次性“看完所有元素”。例如列表 [1, 2, 3]、字
Iterator（迭代器）：就像数学里的 自然数序列，理论上是 无限的，你可以不断“取下一个”，每次调用 nex
```


钟馗 @ 2025/8/14 23:09:24

看了两遍看懂了，结合C++ primer里的 迭代器 理解。能看懂并深入理解的必须对语言要有一定的功底才行，不是小白就可以的。

关于我在地球混日子那些事 @ 2025/8/13 05:24:02

云里雾里，不太明白实际的用处，还是要等待后续实际运用深入理解

幸运橙子 @ 2025/8/5 21:43:19

听不懂，用到了再细想下

零零星星 @ 2025/8/5 06:29:01

这东西实际开发中用的多么，学多了感觉还有点绕

早起的虫子 @ 2025/7/9 21:47:13

PYTHON万物皆对象

Can't touch this @ 2025/6/17 07:42:33

Iterator是一个长度未知的序列，可以不断的通过next调用获取下一个元素，而Iterable本身的数据流长度是已知的例如dict、set、list，我们可以通过iter将Iterable变成一个Iterator对象，通过next调用直到报错

😜 @ 2025/6/9 05:43:19

day8

静下来 @ 2025/6/4 04:16:04

2025-06-04，留下足迹~

森林 @ 2025/5/26 01:49:59

打卡

路 @ 2025/5/12 21:15:50

7.5. 迭代器 done

日向夏橘 @ 2025/5/5 00:23:35

打卡

Super-String @ 2023/5/28 22:52:26

GitHub Copilot: 迭代和递归都是解决问题的方法，但它们的实现方式不同。

迭代是通过循环来重复执行一段代码，每次迭代都会更新变量的值，直到达到预期的结果。迭代通常使用 `for` 循环或 `while` 循环来实现，可以有效地处理大量数据和复杂的算法。迭代的优点是效率高，易于理解和调试，缺点是代码量较大。

递归是通过函数调用自身来解决问题，每次调用都会将问题分解为更小的子问题，直到达到基本情况。递归通常使用递归函数来实现，可以有效地处理树形结构和分治算法。递归的优点是代码简洁，易于理解和实现，缺点是效率低，容易导致栈溢出。

迭代和递归都可以解决同样的问题，但在不同的情况下选择不同的方法可以提高代码的效率和可读性。通常情况下，迭代更适合处理大量数据和复杂的算法，而递归更适合处理树形结构和分治算法。

一耳万年. @ 2023/5/31 23:28:05

说得漂亮

小黄鱼 @ 2025/2/25 09:09:58

真想收藏这个评论

ᓚᓗE1/even @ 2025/5/4 02:49:12

寥寥星辰 @ 2025/4/16 07:21:57

day8