**Python用zip函数并行迭代两个或更多可迭代对象**

Python的内置zip函数能轻松地并行迭代两个或者更多可迭代对象，它返回的元组可以拆包成变量，分别对应各个并行输入中的一个元素。注意，Python2中zip函数返回的是列表，而在Python3中返回的是生成器。

**zip内置函数的使用示例**

zip函数返回一个生成器，按需生成元组。

>>>zip(range(3),'ABC')

复制代码

为了输出，构建一个列表，通常，我们会迭代生成器。

>>>list(zip(range(3),"ABC"))

[(0, 'A'), (1, 'B'), (2, 'C')]

复制代码

返回值的长度，以参数中最短的可迭代对象为准。

**>>>** list(zip(range(3), 'ABC', [0.0, 1.1, 2.2, 3.3]))

[(0, 'A', 0.0), (1, 'B', 1.1), (2, 'C', 2.2)]

复制代码

itertools.zip\_longest函数的作用是以参数中最长的可迭代对象为准，使用可选的fillvalue（默认值为None）填充缺失的值。

>>> from itertools import zip\_longest

>>> list(zip\_longest(range(3), 'ABC', [0.0, 1.1, 2.2, 3.3], fillvalue=-1))

[(0, 'A', 0.0), (1, 'B', 1.1), (2, 'C', 2.2), (-1, -1, 3.3)]

复制代码

zip函数与\*运算符结合，实现将结果拆分成元组的功能。

>>> x = [1, 2, 3]

>>> y = ['A', 'B', 'C']

>>> zipped = zip(x, y)

>>> zipped

<zip object at 0x7fe6063763c8>

>>> list(zipped)

[(1, 'A'), (2, 'B'), (3, 'C')]

>>> x2, y2 = zip(\*zip(x, y))

>>> x == list(x2) and y == list(y2)

True

>>> x2

(1, 2, 3)

>>> y2

('A', 'B', 'C')

复制代码

**zip函数的实际应用场景**

**zip函数由两个列表分别作为键和值生成字典**

>>> keys = ['A', 'B', 'C']

>>> values = [1, 2, 3]

>>> d = dict(zip(keys, values))

>>> d

{'C': 3, 'B': 2, 'A': 1}

复制代码

**zip函数返回值是生成器的相关陷阱**

>>> d = {'Book1': 6.66, 'Book2': 3.33, 'Book3': 9.99}

>>> prices\_and\_names = zip(d.values(), d.keys())

>>> print(min(prices\_and\_names))

(3.33, 'Book2')

>>> print(max(prices\_and\_names))

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

ValueError: max() arg is an empty sequence

复制代码

由于zip函数返回的是生成器，因此它的内容只能被消费一次。如需解决这个问题，可以使用list()将生成器转成列表。示例如下：

>>> d = {'Book1': 6.66, 'Book2': 3.33, 'Book3': 9.99}

>>> prices\_and\_names = list(zip(d.values(), d.keys()))

>>> print(min(prices\_and\_names))

(3.33, 'Book2')

>>> print(max(prices\_and\_names))

(9.99, 'Book3')

复制代码

**zip函数的等价代码实现**

def zip(\*iterables):

sentinel = object()

iterators = [iter(it) **for** it **in** iterables]

**while** iterators:

result = []

**for** it **in** iterators:

elem = next(it, sentinel)

**if** elem is sentinel:

return

result.append(elem)

yield tuple(result)