Research about iterable object or iterator in python and the application

黄家睿　　202283890036　　IoT major

Abstract

In the python class, we had learned about build-in functions, some build-in function involve iterable objects or iterator. This theory is about some aspects of iterable objects or iterator.

Part 1:　The definition of iterable object or iterablor

1. Iterable object

Many typies of data are iterable object, involve list, tuple, set and dictionary. They are all iterable.　And the definition of an iterable object is as follows: **If an object has implemented the __iter__ method, then that object is an iterable**.

It is easy to see the output:

```
1.  from collections.abc import Iterable, Iterator
2.
3.  print(isinstance([1, 2, 3], Iterable))      # True
```

```
print(isinstance((1, 2, 3), Iterable))      # True
print(isinstance(set([1, 2, 3]), Iterable))      # True
print(isinstance("python", Iterable))       # True
print(isinstance({'a': 1}, Iterable))       # True
print("分割线----------------")
print(isinstance([1, 2, 3], Iterator))      # False
print(isinstance((1, 2, 3), Iterator))      # False
print(isinstance(set([1, 2, 3]), Iterator))      # False
print(isinstance("python", Iterator))      # False
print(isinstance({'a': 1}, Iterator))      # False
```

2. Iterator

An iterator is an object that implements the iterator protocol, which consists of two basic methods: __iter__() and __next__(). The __iter__() method returns the iterator object itself, while the __next__() method returns the next element in the collection. When there are no more elements to iterate over, the __next__() method raises a StopIteration exception, indicating that the iteration process is over.

Here are some **feature** of iterator

(1) : **Lazy calculations**

Iterators employ a lazy evaluation strategy, meaning they only calculate the next element when it is needed. This feature makes iterators efficient when dealing with large amounts of data, as it does not require all the data to be loaded into memory at once.

(2) : Unified Access Method

Iterators provide a uniform way to access elements of a

collection, regardless of how the underlying implementation of the collection changes. The interface of the iterator remains consistent, which makes the code more maintainable and extensible.

(3) : Save memory

Because iterators use a lazy evaluation strategy, they can handle large amounts of data without consuming a lot of memory. This is particularly useful for scenarios such as processing large files or real-time data streams.

Part 2: How to implementation iterable objects or iterablor in python programming language

Whether it's an iterable or an iterator, the core is the _iter_() and _next_() methods.

If an object contains an embedded function _iter_(), this object will be recognized by Python as an iterable, regardless of whether the _iter_() function within the object can work properly or not.

The _iter_() method is like an identity card for the object; as long as this method exists, the object is considered an iterable at a macro level.

If an iterable object implements the __iter__ method, the built-in

function iter will call the object's __iter__ method to return an iterator. Since the Color class implements the __next__ method, instances of Color are also iterators. Therefore, returning self in the __iter__ method is sufficient.

```
class Color(object):

    def __init__(self):
        self.index = -1
        self.colors = ['red', 'white', 'black', 'green']

    def __iter__(self):
        self.index = -1
        return self
```

We use the built-in function next to iterate through iterators. In this process, we are calling the __next__ method of the iterator. The purpose of the built-in function is to return the next value of the iterator. To implement this functionality, we need to place it within the __next__ method.

```
def __next__(self):
        self.index += 1
        if self.index >= len(self.colors):
            raise StopIteration

        return self.colors[self.index]
```