232. 用栈实现队列

> 题目：Valid Anagram
> 语言: python3
> 英文版链接：https://leetcode.com/problems/implement-queue-using-stacks/description/
> 中文版链接：https://leetcode-cn.com/problems/implement-queue-using-stacks/

题目分析

本题比较简单，基本上是数据结构的复习，栈的顺序为后进先出，而队列的顺序为先进先出。使用两个栈实现队列，一个元素需要经过两个栈才能出队列，在经过第一个栈时元素顺序被反转，经过第二个栈时再次被反转，此时就是先进先出顺序。

答案

```python
class MyQueue:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        # 初始化两个列表，当作栈来使用
        self.stack_in = []
        self.stack_out = []

    def push(self, x: int) -> None:
        """
        Push element x to the back of queue.
        """
        # 队列的push操作
        self.stack_in.append(x)

    def pop(self) -> int:
        """
        Removes the element from in front of queue and returns that element.
        """
        # 队列的pop操作
        if self.stack_out:
            return self.stack_out.pop()
        else:
            # 将stack_in的数据全部push进stack_out中，然后stack_out的顺序和队列pop的顺序一致
            while self.stack_in:
                self.stack_out.append(self.stack_in.pop())
            return self.stack_out.pop()

    def peek(self) -> int:
        """
        Get the front element.
        """
        if self.stack_out:
            return self.stack_out[-1]
        else:
            while self.stack_in:
                self.stack_out.append(self.stack_in.pop())
            return self.stack_out[-1]

    def empty(self) -> bool:
        """
        Returns whether the queue is empty.
        """
        return not self.stack_in and not self.stack_out
```

扩展

上面的实现是一个非常简单的思路，但是对于python语言来讲，却没有那么麻烦，我们并不需要用两个栈来解决问题，因为python的list既可以当栈用也可以当队列用，非常强大。

我们更新下代码如下：

```
class MyQueue:

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.stack = []

    def push(self, x):
        """
        Push element x to the back of queue.
        :type x: int
        :rtype: void
        """
        self.stack.append(x)

    def pop(self):
        """
        Removes the element from in front of queue and returns that element.
        :rtype: int
        """
        return self.stack.pop(0)

    def peek(self):
        """
        Get the front element.
        :rtype: int
        """
        return self.stack[0]

    def empty(self):
        """
        Returns whether the queue is empty.
        :rtype: bool
        """
        return self.stack == []
```