# 0232. 用栈实现队列

ITCharge 大约 2 分钟

- 标签：栈、设计、队列
- 难度：简单

## 题目链接

- 0232. 用栈实现队列 - 力扣

## 题目大意

**要求**：仅使用两个栈实现先入先出队列。

要求实现 `MyQueue` 类：

- `void push(int x)` 将元素 `x` 推到队列的末尾。
- `int pop()` 从队列的开头移除并返回元素。
- `int peek()` 返回队列开头的元素
- `boolean empty()` 如果队列为空，返回 `True` ；否则，返回 `False` 。

**说明**：

- 只能使用标准的栈操作 —— 也就是只有 `push to top` , `peek / pop from top` , `size` , 和 `is empty` 操作是合法的。
- 可以使用 list 或者 deque（双端队列）来模拟一个栈，只要是标准的栈操作即可。
- $1 <= x <= 9$。
- 最多调用 $100$ 次 `push` 、 `pop` 、 `peek` 和 `empty` 。
- 假设所有操作都是有效的 （例如，一个空的队列不会调用 `pop` 或者 `peek` 操作）。
- 进阶：实现每个操作均摊时间复杂度为 `O(1)` 的队列。换句话说，执行 `n` 个操作的总时间复杂度为 `O(n)` ，即使其中一个操作可能花费较长时间。

**示例**：

- 示例 1：

```py
输入：
["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]
输出：
[null, null, null, 1, 1, false]

解释：
MyQueue myQueue = new MyQueue();
myQueue.push(1); // queue is: [1]
myQueue.push(2); // queue is: [1, 2] (leftmost is front of the queue)
myQueue.peek(); // return 1
myQueue.pop(); // return 1, queue is [2]
myQueue.empty(); // return false
```

# 解题思路

## 思路 1：双栈

使用两个栈，`inStack` 用于输入，`outStack` 用于输出。

- `push` 操作：将元素压入 `inStack` 中。
- `pop` 操作：如果 `outStack` 输出栈为空，将 `inStack` 输入栈元素依次取出，按顺序压入 `outStack` 栈。这样 `outStack` 栈的元素顺序和之前 `inStack` 元素顺序相反，`outStack` 顶层元素就是要取出的队头元素，将其移出，并返回该元素。如果 `outStack` 输出栈不为空，则直接取出顶层元素。
- `peek` 操作：和 `pop` 操作类似，只不过最后一步不需要取出顶层元素，直接将其返回即可。
- `empty` 操作：如果 `inStack` 和 `outStack` 都为空，则队列为空，否则队列不为空。

## 思路 1：代码

```py
class MyQueue:

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

```
    """


def push(self, x: int) -> None:
    self.inStack.append(x)
    """
    Push element x to the back of queue.
    """


def pop(self) -> int:
    if(len(self.outStack) == 0):
        while(len(self.inStack) != 0):
            self.outStack.append(self.inStack[-1])
            self.inStack.pop()
    top = self.outStack[-1]
    self.outStack.pop()
    return top
    """
    Removes the element from in front of queue and returns that element.
    """


def peek(self) -> int:
    if (len(self.outStack) == 0):
        while (len(self.inStack) != 0):
            self.outStack.append(self.inStack[-1])
            self.inStack.pop()
    top = self.outStack[-1]
    return top
    """
    Get the front element.
    """


def empty(self) -> bool:
    return len(self.outStack) == 0 and len(self.inStack) == 0
    """
    Returns whether the queue is empty.
    """
```

## 思路 1：复杂度分析

- **时间复杂度**：`push` 和 `empty` 为 $O(1)$，`pop` 和 `peek` 为均摊 $O(1)$。
- **空间复杂度**：$O(n)$。