```
                int newMin = this.stackMin.peek();
                this.stackMin.push(newMin);
            }
            this.stackData.push(newNum);
        }

        public int pop() {
            if (this.stackData.isEmpty()) {
                throw new RuntimeException("Your stack is empty.");
            }
            this.stackMin.pop();
            return this.stackData.pop();
        }

        public int getmin() {
            if (this.stackMin.isEmpty()) {
                throw new RuntimeException("Your stack is empty.");
            }
            return this.stackMin.peek();
        }
    }
```

【点评】

方案一和方案二其实都是用 stackMin 栈保存着 stackData 每一步的最小值。共同点是所有操作的时间复杂度都为 $O(1)$、空间复杂度都为 $O(n)$。区别是：方案一中 stackMin 压入时稍省空间，但是弹出操作稍费时间；方案二中 stackMin 压入时稍费空间，但是弹出操作稍省时间。

由两个栈组成的队列

【题目】

编写一个类，用两个栈实现队列，支持队列的基本操作（add、poll、peek）。

【难度】

尉 ★★☆☆

【解答】

栈的特点是先进后出，而队列的特点是先进先出。我们用两个栈正好能把顺序反过来实现类似队列的操作。

具体实现上是一个栈作为压入栈，在压入数据时只往这个栈中压入，记为 stackPush；另一个栈只作为弹出栈，在弹出数据时只从这个栈弹出，记为 stackPop。

因为数据压入栈的时候，顺序是先进后出的。那么只要把 stackPush 的数据再压入 stackPop 中，顺序就变回来了。例如，将 1~5 依次压入 stackPush，那么从 stackPush 的栈顶到栈底为 5~1，此时依次再将 5~1 倒入 stackPop，那么从 stackPop 的栈顶到栈底就变成了 1~5。再从 stackPop 弹出时，顺序就像队列一样，如图 1-3 所示。

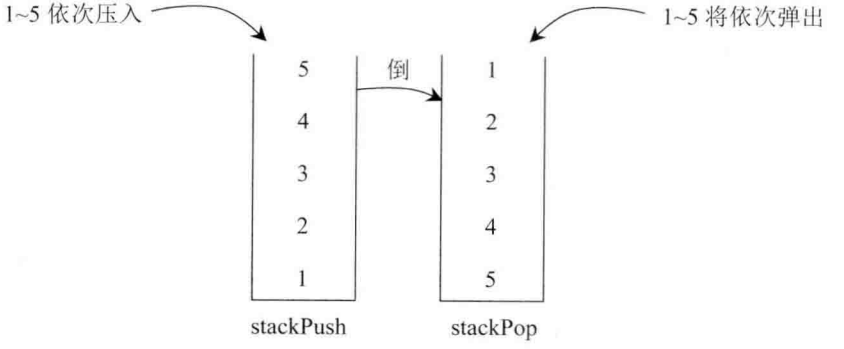

图 1-3

听起来虽然简单，实际上必须做到以下两点。

1．如果 stackPush 要往 stackPop 中压入数据，那么必须一次性把 stackPush 中的数据全部压入。

2．如果 stackPop 不为空，stackPush 绝对不能向 stackPop 中压入数据。

违反了以上两点都会发生错误。

违反 1 的情况举例：1~5 依次压入 stackPush，stackPush 的栈顶到栈底为 5~1，从 stackPush 压入 stackPop 时，只将 5 和 4 压入了 stackPop，stackPush 还剩下 1、2、3 没有压入。此时如果用户想进行弹出操作，那么 4 将最先弹出，与预想的队列顺序就不一致。

违反 2 的情况举例：1~5 依次压入 stackPush，stackPush 将所有的数据压入了 stackPop，此时从 stackPop 的栈顶到栈底就变成了 1~5。此时又有 6~10 依次压入 stackPush，stackPop 不为空，stackPush 不能向其中压入数据。如果违反 2 压入了 stackPop，从 stackPop 的栈顶到栈底就变成了 6~10、1~5。那么此时如果用户想进行弹出操作，6 将最先弹出，与预想的队列顺序就不一致。

上面介绍了压入数据的注意事项。那么这个压入数据的操作在何时发生呢？

这个选择的时机可以有很多，调用 add、poll 和 peek 三种方法中的任何一种时发生“压”入数据的行为都是可以的。只要满足如上提到的两点，就不会出错。

本书的实现是在调用 poll 和 peek 方法时进行压入数据的过程。

具体实现请参看如下的 TwoStacksQueue 类：

```
public class TwoStacksQueue {
        public Stack<Integer> stackPush;
        public Stack<Integer> stackPop;

        public TwoStacksQueue() {
                stackPush = new Stack<Integer>();
                stackPop = new Stack<Integer>();
        }

        public void add(int pushInt) {
                stackPush.push(pushInt);
        }

        public int poll() {
                if (stackPop.empty() && stackPush.empty()) {
                        throw new RuntimeException("Queue is empty!");
                } else if (stackPop.empty()) {
                        while (!stackPush.empty()) {
                                stackPop.push(stackPush.pop());
                        }
                }
                return stackPop.pop();
        }

        public int peek() {
                if (stackPop.empty() && stackPush.empty()) {
                        throw new RuntimeException("Queue is empty!");
                } else if (stackPop.empty()) {
                        while (!stackPush.empty()) {
                                stackPop.push(stackPush.pop());
                        }
                }
                return stackPop.peek();
        }
}
```