第 1 章

栈和队列

设计一个有 getMin 功能的栈

【题目】

实现一个特殊的栈，在实现栈的基本功能的基础上，再实现返回栈中最小元素的操作。

【要求】

1．pop、push、getMin 操作的时间复杂度都是 $O(1)$。

2．设计的栈类型可以使用现成的栈结构。

【难度】

士 ★☆☆☆

【解答】

在设计上我们使用两个栈，一个栈用来保存当前栈中的元素，其功能和一个正常的栈没有区别，这个栈记为 stackData；另一个栈用于保存每一步的最小值，这个栈记为 stackMin。具体的实现方式有两种。

第一种设计方案如下。

- **压入数据规则**

假设当前数据为 newNum，先将其压入 stackData。然后判断 stackMin 是否为空：

- 如果为空，则 newNum 也压入 stackMin。
- 如果不为空，则比较 newNum 和 stackMin 的栈顶元素中哪一个更小：
- 如果 newNum 更小或两者相等，则 newNum 也压入 stackMin；
- 如果 stackMin 中栈顶元素小，则 stackMin 不压入任何内容。

举例：依次压入 3、4、5、1、2、1 的过程中，stockData 和 stackMin 的变化如图 1-1 所示。

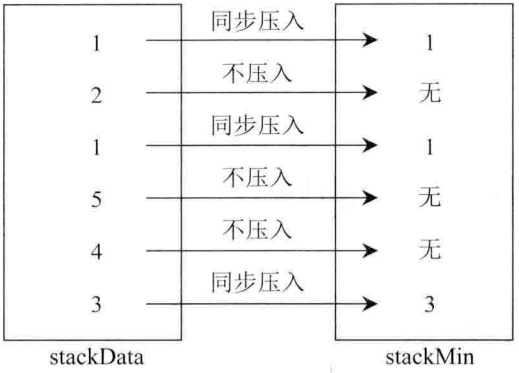

图 1-1

- 弹出数据规则

先在 stackData 中弹出栈顶元素，记为 value。然后比较当前 stackMin 的栈顶元素和 value 哪一个更小。

通过上文提到的压入规则可知，stackMin 中存在的元素是从栈底到栈顶逐渐变小的，stackMin 栈顶的元素既是 stackMin 栈的最小值，也是当前 stackData 栈的最小值。所以不会出现 value 比 stackMin 的栈顶元素更小的情况，value 只可能大于或等于 stackMin 的栈顶元素。

当 value 等于 stackMin 的栈顶元素时，stackMin 弹出栈顶元素；当 value 大于 stackMin 的栈顶元素时，stackMin 不弹出栈顶元素；返回 value。

很明显可以看出，压入与弹出规则是对应的。

- 查询当前栈中的最小值操作

由上文的压入数据规则和弹出数据规则可知，stackMin 始终记录着 stackData 中的最小值，所以，stackMin 的栈顶元素始终是当前 stackData 中的最小值。

方案一的代码实现如 MyStack1 类所示：

```
public class MyStack1 {
        private Stack<Integer> stackData;
        private Stack<Integer> stackMin;

        public MyStack1() {
                this.stackData = new Stack<Integer>();
                this.stackMin = new Stack<Integer>();
        }

        public void push(int newNum) {
                if (this.stackMin.isEmpty()) {
                        this.stackMin.push(newNum);
                } else if (newNum <= this.getmin()) {
                        this.stackMin.push(newNum);
                }
                this.stackData.push(newNum);
        }

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

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

第二种设计方案如下。

- **压入数据规则**

假设当前数据为 newNum，先将其压入 stackData。然后判断 stackMin 是否为空。

如果为空，则 newNum 也压入 stackMin；如果不为空，则比较 newNum 和 stackMin 的栈顶元素中哪一个更小：

如果 newNum 更小或两者相等，则 newNum 也压入 stackMin；如果 stackMin 中栈顶元素小，则把 stackMin 的栈顶元素重复压入 stackMin，即在栈顶元素上再压入一个栈顶

元素。

举例：依次压入 3、4、5、1、2、1 的过程中，stockData 和 stackMin 的变化如图 1-2 所示。

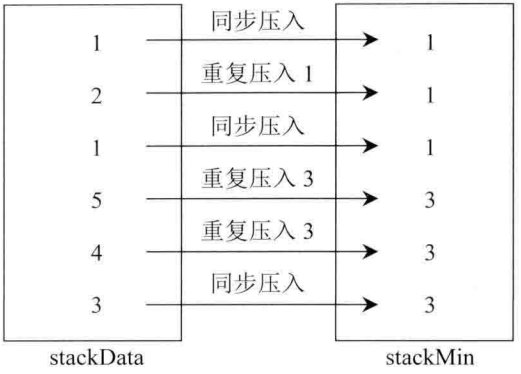

图 1-2

- **弹出数据规则**

在 stackData 中弹出数据，弹出的数据记为 value；弹出 stackMin 中的栈顶；返回 value。很明显可以看出，压入与弹出规则是对应的。

- **查询当前栈中的最小值操作**

由上文的压入数据规则和弹出数据规则可知，stackMin 始终记录着 stackData 中的最小值，所以 stackMin 的栈顶元素始终是当前 stackData 中的最小值。

方案二的代码实现如 MyStack2 类所示：

```
public class MyStack2 {
        private Stack<Integer> stackData;
        private Stack<Integer> stackMin;

        public MyStack2() {
                this.stackData = new Stack<Integer>();
                this.stackMin = new Stack<Integer>();
        }

        public void push(int newNum) {
                if (this.stackMin.isEmpty()) {
                        this.stackMin.push(newNum);
                } else if (newNum < this.getmin()) {
                        this.stackMin.push(newNum);
                } else {
```

```
                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。