0155. 最小栈

ITCharge 大约 2 分钟

- 标签：栈、设计
- 难度：中等

题目链接

- 0155. 最小栈 - 力扣

题目大意

要求：设计一个「栈」。实现 `push` ， `pop` ， `top` ， `getMin` 操作，其中 `getMin` 要求能在常数时间内实现。

说明：

- $-2^{31} \leq val \leq 2^{31} - 1$。
- `pop` 、 `top` 和 `getMin` 操作总是 　　空栈上调用
- `push` ， `pop` ， `top` 和 `getMin` 最多被调用 $3 * 10^4$ 次。

示例：

- 示例 1：

```py
输入：
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

输出：
[null,null,null,null,-3,null,0,-2]

解释：
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
```

```
minStack.push(-3);
minStack.getMin();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.getMin();   --> 返回 -2.
```

解题思路

题目要求在常数时间内获取最小值，所以我们不能在 `getMin` 操作时，再去计算栈中的最小值。而是应该在 `push` 、 `pop` 操作时就已经计算好了最小值。我们有两种思路来解决这道题。

思路 1：辅助栈

使用辅助栈保存当前栈中的最小值。在元素入栈出栈时，两个栈同步保持插入和删除。具体做法如下：

- `push` 操作：当一个元素入栈时，取辅助栈的栈顶存储的最小值，与当前元素进行比较得出最小值，将最小值插入到辅助栈中；该元素也插入到正常栈中。
- `pop` 操作：当一个元素要出栈时，将辅助栈的栈顶元素一起弹出。
- `top` 操作：返回正常栈的栈顶元素
- `getMin` 操作：返回辅助栈的栈顶元素值。

思路 1：代码

py

```py
class MinStack:

    def __init__(self):
        self.stack = []
        self.minstack = []

    def push(self, val: int) -> None:
        if not self.stack:
            self.stack.append(val)
            self.minstack.append(val)
        else:
            self.stack.append(val)
            self.minstack.append(min(val, self.minstack[-1]))
```

```
    def pop(self) -> None:
        self.stack.pop()
        self.minstack.pop()

    def top(self) -> int:
        return self.stack[-1]

    def getMin(self) -> int:
        return self.minstack[-1]
```

思路 1：复杂度分析

- **时间复杂度**：$O(1)$。栈的插入、删除、读取操作都是 $O(1)$。
- **空间复杂度**：$O(n)$。其中 n 为总操作数。

思路 2：单个栈

使用单个栈，保存元组：（当前元素值，当前栈内最小值）。具体操作如下：

- `push` 操作：如果栈不为空，则判断当前元素值与栈顶元素所保存的最小值，并更新当前最小值，然后将新元素和当前最小值组成的元组保存到栈中。
- `pop` 操作：正常出栈，即将栈顶元素出栈。
- `top` 操作：返回栈顶元素保存的值。
- `getMin` 操作：返回栈顶元素保存的最小值。

思路 2：代码

py

```
class MinStack:
    def __init__(self):
        """
        initialize your data structure here.
        """
        self.stack = []

    class Node:
        def __init__(self, x):
            self.val = x
            self.min = x

    def push(self, val: int) -> None:
```

```
        node = self.Node(val)
        if len(self.stack) == 0:
            self.stack.append(node)
        else:
            topNode = self.stack[-1]
            if node.min > topNode.min:
                node.min = topNode.min

            self.stack.append(node)

    def pop(self) -> None:
        self.stack.pop()

    def top(self) -> int:
        return self.stack[-1].val

    def getMin(self) -> int:
        return self.stack[-1].min
```

思路 2：复杂度分析

- **时间复杂度**：$O(1)$。栈的插入、删除、读取操作都是 $O(1)$。
- **空间复杂度**：$O(n)$。其中 n 为总操作数。