后端开发常问面试题集锦

——算法问题

实现一个 能在 O(1)时间复杂度 完成 Push、Pop、Min 操作的 栈

一，问题描述

实现一个栈（元素遵守先入后出顺序），能够通过 min 方法在 O(1)时间内获取栈中的最小元素。同时，栈的基本操作：入栈(Push)、出栈(Pop)，也是在 O(1)时间内完成的。

二，问题分析

之所以认为这个问题有趣，是因为在实现 min 方法的过程 牵涉到了 “缓存一致性”问题。是不是听起来很高大上？哈哈，我臆想的。

思路 1：添加一个成员变量总是保存当前栈中最小的元素。该思路的实现代码大致是这样的：

```java
public class MinStack {
    private LinkedList<Integer> stack;
    private int min;
    // save minimum ele of stack
    public int pop(){
        //check pop minimum ele?
    }
    public void push(int ele){
        //check push minimum ele?
    }
    public int getMin(){
        return min;
    }
}
```

这里就会存在一个问题：保存最小元素的 min 属性 与 栈中的最小元素不一致。

比如：当从栈中 pop 最小元素时，那 min 属性就要 保存 次最小元素了。那如何 找到次最小元素，然后赋值给 min 呢？

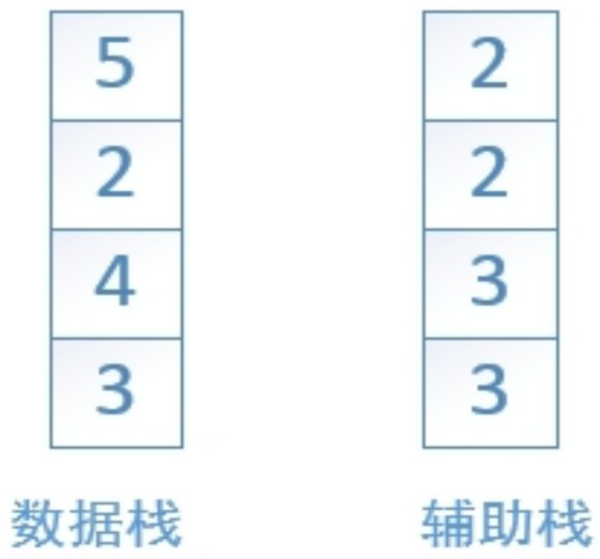
因此，问题的关键就是：当只使用一个 min 属性时，如何保证 min 属性 总是保存的是当前栈中最小的元素？---即： min 代表的最小元素 要与 栈中的最小元素保存一致。一种方式是当 pop 出最小元素之后，再遍历栈找出次最小的元素，并将之赋值给 min 。但是，由于遍历使得时间复杂度不再是 O(1)

思路 2：

使用一个辅助栈。此方法能够实现在 O(1)时间内获取栈中最小的元素，但是缺点是空间复杂度为 O(N)

现在有两个栈：一个是保存元素的数据栈，另一个是辅助栈，辅助栈的栈顶总是 当前数据栈中最小的元素。当 Push 元素时，首先将该元素 Push 到数据栈，然后再将该元素与辅助栈的栈顶元素比较：如果该元素比辅助栈的栈顶元素小，则将该元素 Push 到辅助栈中；否则将辅助栈的栈顶元素再 Push 到辅助栈中。

比如，现在要 Push 的元素顺序如下：3，4，2，5....　　在数据栈 和 辅助栈中保存的元素如下：



三，代码实现

代码中使用了 java.util.LinkedList 类作为 栈的实现。

```java
import java.util.LinkedList;
public class MinStack {
    private LinkedList<Integer> dataStack;
    private LinkedList<Integer> minStack;
    public MinStack() {
        dataStack = new LinkedList<Integer>();
        minStack = new LinkedList<Integer>();
    }
    //base operation
    public void push(int ele)
        {
        dataStack.push(ele);
        if(minStack.size() == 0 || ele < minStack.peek())
                    minStack.push(ele); else
                    minStack.push(minStack.peek());
    }
    public Integer pop(){
        if(dataStack.isEmpty())
                    return null;
        assert dataStack.isEmpty() == false && minStack.isEmpty() == false;
        int ele = dataStack.pop();
        minStack.pop();
        return ele;
    }
    public Integer min(){
        if(minStack.isEmpty())
                    return null;
        return minStack.peek();
    }
    //hapjin test
    public static void main(String[] args) {
        MinStack stack = new MinStack();
        int[] eles = {3,4,2,5};
        for (int i : eles) {
            stack.push(i);
        }
        System.out.println(stack.min());
        //2
        System.out.println(stack.pop());
        //5
        System.out.println(stack.pop());
        //2
        System.out.println(stack.min());
        //3
        stack.push(1);
        System.out.println(stack.min());
    }
}
```