题目 包含min函数的栈

考点 举例让抽象具体化 热点指数 59617 通过率 30.98%

具体题目

定义栈的数据结构，请在该类型中实现一个能够得到栈中所含最小元素的min函数（时间复杂度应为O（1））。

```
/*
   解题思路：
   我们可以设计两个栈：StackDate和StackMin，一个就是普通的栈，另外一个存储push进来的最小值。
首先是push操作：
   每次压入的数据newNum都push进StackDate中，然后判断StackMin是否为空，如果为空那也把newNum同步压入
   StackMin里；如果不为空，就先比较newNum和StackMin中栈顶元素的大小，如果newNum较大，那就不压入StackMin
里，否则
   就同步压入StackMin里。如：
接着是pop操作
   先将StackDate中取出的数据value与StackMin的栈顶元素比较，因为对应push操作，value不可能小于
   StackMin中的栈顶元素，最多是相等。如果相等，那么StackMin中也取出数据，同时返回value，否则只是返回value就
可以了。
最后是getMin操作
    由上就可知，StackMin中存储的数据就是当前最小的，所以只要返回StackMin中的栈顶元素就可以了。
*/

import java.util.Stack;
public class Solution {
    private Stack<Integer> stackDate;
    private Stack<Integer> stackMin;

    public Solution(){
        stackDate = new Stack<>();
        stackMin = new Stack<>();
    }
    public void push(int node) {
        stackDate.push(node);
        if(stackMin.isEmpty()){
            stackMin.push(node);
        }else if(node <= stackMin.peek()){
            stackMin.push(node);
        }
    }

    public void pop() {
         if(stackDate.isEmpty()){
            throw new RuntimeException("This stack is empty!");
         }
         if(stackDate.peek() == stackMin.peek()){
             stackMin.pop();
         }
         stackDate.pop();
    }

    public int top() {
        if(stackDate.isEmpty()){
            throw new RuntimeException("This stack is empty!");
         }
        int value = stackDate.pop();
         if(value == stackMin.peek()){
             stackMin.pop();
```

```
         }
        return value;
    }

    public int min() {
        if(stackMin.isEmpty()){
            throw new RuntimeException("This stack is empty!");
        }else{
            return stackMin.peek();
        }
    }
}
```

题目描述 定义栈的数据结构，请在该类型中实现一个能够得到栈中所含最小元素的min函数（时间复杂度应为O（1））。 解题思路 思路：利用一个辅助栈来存放最小值 栈 3，4，2，5，1 辅助栈 3，3，2，2，1 每入栈一次，就与辅助栈顶比较大小，如果小就入栈，如果大就入栈当前的辅助栈顶； 当出栈时，辅助栈也要出栈 这种做法可以保证辅助栈顶一定都当前栈的最小值 我的答案

```
import java.util.Stack;
public class Solution {
    //存放元素
    Stack<Integer> stack1 = new Stack<Integer>();
    //存放当前stack1中的最小元素
    Stack<Integer> stack2 = new Stack<Integer>();
    //stack1直接塞，stack2要塞比栈顶小的元素，要不然就重新塞一下栈顶元素
    public void push(int node) {
        stack1.push(node);
        if(stack2.isEmpty() || stack2.peek() > node){
            stack2.push(node);
        }else{
            stack2.push(stack2.peek());
        }
    }
    //都要pop一下
    public void pop() throws Exception{
        if(stack1.isEmpty()){
           throw new Exception("no element valid");
        }
        stack1.pop();
        stack2.pop();
    }
    public int top(){
        if(stack1.isEmpty()){
           return 0;
        }
        return stack1.peek();
    }
    public int min(){
        if(stack2.isEmpty()){
           return 0;
        }
        return stack2.peek();
    }
}
```