思考题

1. 两个堆栈实现队列

分析

堆栈是先进后出，队列是先进先出，用两个堆栈倒腾两次正好把顺序反过来就能实现队列的操作。

一个堆栈用来出队（可以叫stackPop），一个堆栈用来入队(可以叫stackPush)。

要实现先进先出，要把数据从stachPush中倒腾到stackPop中，倒腾的时候，必须保证两点：

(1). stackPush往stackPop倒腾的时候，必须一次把stackPush的时候全部弹出去。

(2).stackPop中有数据的时候，不能倒腾。

入队很简单，直接push到stackPop中。 出队时stackPop里没有元素的话，就把stackPush里面的元素都倒腾过来，然后出去一个就好了。出队时如果stackPop里有元素，就直接出去一个就可以了。

代码

public class StackIntoQueue {

Stack<Integer> stack1 = new Stack<Integer>();

Stack<Integer> stack2 = new Stack<Integer>();

public void push(int node) {

stack1.push(node);

}

public int pop() {

if(stack2.empty()){

while(!stack1.empty()){

stack2.push(stack1.pop());

}

}

int ans = stack2.pop();

return ans;

}

}

2.参考答案

两个队列实现堆栈

分析

用一个变量来表示目前正在操作的队列。入栈时就把元素加入该队列。出栈时，把该队列中的元素（除了最后一个）都转移到另一个队列中，最后出栈最后这个元素，并把那个变量指示为含有元素的那个队列。

（Java中其实LinkedList就封装了堆栈和队列（包括双端队列）的所有操作，LinkedList底层是一个就是一个双向链表。所以下面的代码有些牵强，就仅仅是用LinkedList中队列的操作来模拟栈的操作）

思路：

1、 一个队列加入元素，弹出元素时，需要把队列中的 元素放到另外一个队列中，删除最后一个元素

2、 两个队列始终保持只有一个队列是有数据的

代码如下

package com.datastructure.stackqueue;

import java.util.LinkedList;

import java.util.Queue;

/\*\*

\* 两个队列实现一个栈

\*

\* 一个队列加入元素，弹出元素时，需要把队列中的 元素放到另外一个队列中，删除最后一个元素

\* 两个队列始终保持只有一个队列是有数据的

\*

\*/

public class StackByQueue<T> {

private Queue<T> queue1 = new LinkedList<>();

private Queue<T> queue2 = new LinkedList<>();

/\*\*

\* 压栈

\*

\* 入栈非空的队列

\*/

public boolean push(T t) {

if (!queue1.isEmpty()) {

return queue1.offer(t);

} else {

return queue2.offer(t);

}

}

/\*\*

\* 弹出并删除元素

\*/

public T pop() {

if (queue1.isEmpty() && queue2.isEmpty()) {

throw new RuntimeException("queue is empty");

}

if (!queue1.isEmpty() && queue2.isEmpty()) {

while (queue1.size() > 1) {

queue2.offer(queue1.poll());

}

return queue1.poll();

}

if (queue1.isEmpty() && !queue2.isEmpty()) {

while (queue2.size() > 1) {

queue1.offer(queue2.poll());

}

return queue2.poll();

}

return null;

}

@Override

public String toString() {

return this.queue1.toString() + ", " +this.queue2.toString();

}

public static void main(String[] args) {

StackByQueue<Integer> s = new StackByQueue<>();

s.push(1);

s.push(2);

s.push(3);

s.pop();

System.out.println(s);

s.push(4);

s.push(5);

s.pop();

System.out.println(s);

}

}