youngyangyang04 Update History

3 contributors

477 lines (390 sloc) 12.8 KB

欢迎大家参与本项目，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

> 用队列实现栈还是有点别扭。

225. 用队列实现栈

https://leetcode-cn.com/problems/implement-stack-using-queues/

使用队列实现栈的下列操作：

- push(x) -- 元素 x 入栈
- pop() -- 移除栈顶元素
- top() -- 获取栈顶元素
- empty() -- 返回栈是否为空

注意:

- 你只能使用队列的基本操作-- 也就是 push to back, peek/pop from front, size, 和 is empty 这些操作是合法的。
- 你所使用的语言也许不支持队列。 你可以使用 list 或者 deque（双端队列）来模拟一个队列 , 只要是标准的队列操作即可。
- 你可以假设所有操作都是有效的（例如, 对一个空的栈不会调用 pop 或者 top 操作）。

思路

(这里要强调是单向队列)

有的同学可能疑惑这种题目有什么实际工程意义，**其实很多算法题目主要是对知识点的考察和教学意义远大于其工程实践的意义，所以面试题也是这样！**

刚刚做过栈与队列：我用栈来实现队列怎么样？的同学可能依然想着用一个输入队列，一个输出队列，就可以模拟栈的功能，仔细想一下还真不行！

队列模拟栈，其实一个队列就够了，那么我们先说一说两个队列来实现栈的思路。

队列是先进先出的规则，把一个队列中的数据导入另一个队列中，数据的顺序并没有变，并有变成先进后出的顺序。

所以用栈实现队列， 和用队列实现栈的思路还是不一样的，这取决于这两个数据结构的性质。

但是依然还是要用两个队列来模拟栈，只不过没有输入和输出的关系，而是另一个队列完全用又来备份的！

如下面动画所示，**用两个队列que1和que2实现队列的功能，que2其实完全就是一个备份的作用**，把que1最后面的元素以外的元素都备份到que2，然后弹出最后面的元素，再把其他元素从que2导回que1。

模拟的队列执行语句如下：

```
queue.push(1);
queue.push(2);
queue.pop();   // 注意弹出的操作
queue.push(3);
queue.push(4);
queue.pop();  // 注意弹出的操作
queue.pop();
queue.pop();
queue.empty();
```

详细如代码注释所示：

```
class MyStack {
public:
    queue<int> que1;
    queue<int> que2; // 辅助队列，用来备份
    /** Initialize your data structure here. */
    MyStack() {

    }

    /** Push element x onto stack. */
    void push(int x) {
        que1.push(x);
    }

    /** Removes the element on top of the stack and returns that element. */
    int pop() {
        int size = que1.size();
        size--;
```

```
        while (size--) { // 将que1 导入que2，但要留下最后一个元素
            que2.push(que1.front());
            que1.pop();
        }

        int result = que1.front(); // 留下的最后一个元素就是要返回的值
        que1.pop();
        que1 = que2;            // 再将que2赋值给que1
        while (!que2.empty()) { // 清空que2
            que2.pop();
        }
        return result;
    }

    /** Get the top element. */
    int top() {
        return que1.back();
    }

    /** Returns whether the stack is empty. */
    bool empty() {
        return que1.empty();
    }
};
```

优化

其实这道题目就是用一个队里就够了。

一个队列在模拟栈弹出元素的时候只要将队列头部的元素（除了最后一个元素外） 重新添加到队列尾部，此时在去弹出元素就是栈的顺序了。

C++优化代码

```
class MyStack {
public:
    queue<int> que;
    /** Initialize your data structure here. */
    MyStack() {

    }
    /** Push element x onto stack. */
    void push(int x) {
        que.push(x);
    }
    /** Removes the element on top of the stack and returns that element. */
    int pop() {
        int size = que.size();
        size--;
        while (size--) { // 将队列头部的元素（除了最后一个元素外） 重新添加到队列尾部
            que.push(que.front());
            que.pop();
        }
        int result = que.front(); // 此时弹出的元素顺序就是栈的顺序了
        que.pop();
```

```
        return result;
    }

    /** Get the top element. */
    int top() {
        return que.back();
    }

    /** Returns whether the stack is empty. */
    bool empty() {
        return que.empty();
    }
};
```

其他语言版本

Java：

使用两个 Queue 实现

```
class MyStack {

    Queue<Integer> queue1; // 和栈中保持一样元素的队列
    Queue<Integer> queue2; // 辅助队列

    /** Initialize your data structure here. */
    public MyStack() {
        queue1 = new LinkedList<>();
        queue2 = new LinkedList<>();
    }

    /** Push element x onto stack. */
    public void push(int x) {
        queue2.offer(x); // 先放在辅助队列中
        while (!queue1.isEmpty()){
            queue2.offer(queue1.poll());
        }
        Queue<Integer> queueTemp;
        queueTemp = queue1;
        queue1 = queue2;
        queue2 = queueTemp; // 最后交换queue1和queue2，将元素都放到queue1中
    }

    /** Removes the element on top of the stack and returns that element. */
    public int pop() {
        return queue1.poll(); // 因为queue1中的元素和栈中的保持一致，所以这个和下面两个的操作只
    }

    /** Get the top element. */
    public int top() {
        return queue1.peek();
    }

    /** Returns whether the stack is empty. */
    public boolean empty() {
```

```
        return queue1.isEmpty();
    }
}

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue obj = new MyQueue();
 * obj.push(x);
 * int param_2 = obj.pop();
 * int param_3 = obj.peek();
 * boolean param_4 = obj.empty();
 */
```

使用两个 Deque 实现

```
class MyStack {
    // Deque 接口继承了 Queue 接口
    // 所以 Queue 中的 add、poll、peek等效于 Deque 中的 addLast、pollFirst、peekFirst
    Deque<Integer> que1; // 和栈中保持一样元素的队列
    Deque<Integer> que2; // 辅助队列
    /** Initialize your data structure here. */
    public MyStack() {
        que1 = new ArrayDeque<>();
        que2 = new ArrayDeque<>();
    }

    /** Push element x onto stack. */
    public void push(int x) {
        que1.addLast(x);
    }

    /** Removes the element on top of the stack and returns that element. */
    public int pop() {
        int size = que1.size();
        size--;
        // 将 que1 导入 que2 ，但留下最后一个值
        while (size-- > 0) {
            que2.addLast(que1.peekFirst());
            que1.pollFirst();
        }

        int res = que1.pollFirst();
        // 将 que2 对象的引用赋给了 que1 ，此时 que1，que2 指向同一个队列
        que1 = que2;
        // 如果直接操作 que2，que1 也会受到影响，所以为 que2 分配一个新的空间
        que2 = new ArrayDeque<>();
        return res;
    }

    /** Get the top element. */
    public int top() {
        return que1.peekLast();
    }

    /** Returns whether the stack is empty. */
    public boolean empty() {
```

```
        return que1.isEmpty();
    }
}
```

优化，使用一个 Deque 实现

```
class MyStack {
    // Deque 接口继承了 Queue 接口
    // 所以 Queue 中的 add、poll、peek等效于 Deque 中的 addLast、pollFirst、peekFirst
    Deque<Integer> que1;
    /** Initialize your data structure here. */
    public MyStack() {
        que1 = new ArrayDeque<>();
    }

    /** Push element x onto stack. */
    public void push(int x) {
        que1.addLast(x);
    }

    /** Removes the element on top of the stack and returns that element. */
    public int pop() {
        int size = que1.size();
        size--;
        // 将 que1 导入 que2 ，但留下最后一个值
        while (size-- > 0) {
            que1.addLast(que1.peekFirst());
            que1.pollFirst();
        }

        int res = que1.pollFirst();
        return res;
    }

    /** Get the top element. */
    public int top() {
        return que1.peekLast();
    }

    /** Returns whether the stack is empty. */
    public boolean empty() {
        return que1.isEmpty();
    }
}
```

Python：

```
from collections import deque
class MyStack:
    def __init__(self):
        """
        Initialize your data structure here.
        """
        #使用两个队列来实现
        self.que1 = deque()
```

```
        self.que2 = deque()

    def push(self, x: int) -> None:
        """
        Push element x onto stack.
        """
        self.que1.append(x)

    def pop(self) -> int:
        """
        Removes the element on top of the stack and returns that element.
        """
        size = len(self.que1)
        size -= 1#这里先减一是为了保证最后面的元素
        while size > 0:
            size -= 1
            self.que2.append(self.que1.popleft())


        result = self.que1.popleft()
        self.que1, self.que2= self.que2, self.que1#将que2和que1交换 que1经过之前的操作应该是空
        #一定注意不能直接使用que1 = que2 这样que2的改变会影响que1 可以用浅拷贝
        return result

    def top(self) -> int:
        """
        Get the top element.
        """
        return self.que1[-1]

    def empty(self) -> bool:
        """
        Returns whether the stack is empty.
        """
        #print(self.que1)
        if len(self.que1) == 0:
            return True
        else:
            return False


# Your MyStack object will be instantiated and called as such:
# obj = MyStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.empty()
```

Go：

javaScript:

使用数组（push, shift）模拟队列

```
// 使用两个队列实现
/**
```

```
 * Initialize your data structure here.
 */
var MyStack = function() {
    this.queue1 = [];
    this.queue2 = [];
};

/**
 * Push element x onto stack.
 * @param {number} x
 * @return {void}
 */
MyStack.prototype.push = function(x) {
    this.queue1.push(x);
};

/**
 * Removes the element on top of the stack and returns that element.
 * @return {number}
 */
MyStack.prototype.pop = function() {
    // 减少两个队列交换的次数， 只有当queue1为空时，交换两个队列
    if(!this.queue1.length) {
        [this.queue1, this.queue2] = [this.queue2, this.queue1];
    }
    while(this.queue1.length > 1) {
        this.queue2.push(this.queue1.shift());
    }
    return this.queue1.shift();
};

/**
 * Get the top element.
 * @return {number}
 */
MyStack.prototype.top = function() {
    const x = this.pop();
    this.queue1.push(x);
    return x;
};

/**
 * Returns whether the stack is empty.
 * @return {boolean}
 */
MyStack.prototype.empty = function() {
    return !this.queue1.length && !this.queue2.length;
};
```

```
// 使用一个队列实现
/**
 * Initialize your data structure here.
 */
var MyStack = function() {
    this.queue = [];
};
```

```
/**
 * Push element x onto stack.
 * @param {number} x
 * @return {void}
 */
MyStack.prototype.push = function(x) {
    this.queue.push(x);
};

/**
 * Removes the element on top of the stack and returns that element.
 * @return {number}
 */
MyStack.prototype.pop = function() {
    let size = this.queue.length;
    while(size-- > 1) {
        this.queue.push(this.queue.shift());
    }
    return this.queue.shift();
};

/**
 * Get the top element.
 * @return {number}
 */
MyStack.prototype.top = function() {
    const x = this.pop();
    this.queue.push(x);
    return x;
};

/**
 * Returns whether the stack is empty.
 * @return {boolean}
 */
MyStack.prototype.empty = function() {
    return !this.queue.length;
};
```

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录