## 思考题

tcp状态机是怎样的流程（画图解释）？

* tcp连接有三个阶段：连接建立、数据传输、连接释放
* 连接建立（三次握手）

1. 第一次握手：客户端初始为CLOSED状态，客户端主动发起连接请求，发送SYN报文，此报文SYN = 1，seq = x，然后客户端处于SYN\_SEND（同步发送状态）。
2. 第二次握手：服务端初始为LISTEN状态，收到客户端的SYN报文之后，要回复给客户端一个报文，表示同意建立连接，此报文SYN = 1，ACK = 1，seq = y，ack = x + 1，然后服务端处于SYN\_RCVD（同步接收状态）。
3. 第三次握手：客户端收到了服务端同意建立连接的报文，然后发送一个ACK报文，表示确认已经收到，此报文ACK = 1，seq = x + 1，ack = y + 1，发送后客户端进入ESTABLISHED状态，服务端收到报文后也进入ESTABLISHED状态，之后就能进行数据传输了。

图示

描述已自动生成

* 连接释放（四次挥手）

1. 第一次挥手：客户端初始为ESTABLISHED状态，然后发起连接释放请求，发送一个FIN报文，此报文FIN = 1，seq = u，发送后客户端进入FIN\_WAIT\_1状态，并且客户端不再传输数据。
2. 第二次挥手：服务端初始为ESTABLISHED状态，收到客户端发来的连接释放请求，发送一个ACK报文，表示收到了客户端的连接释放请求，此报文ACK = 1，seq = u，ack = u + 1，然后服务端进入CLOSE\_WAIT（关闭等待状态），做连接关闭的准备工作，但是仍然可以向客户端传输数据。客户端收到报文后进入FIN\_WAIT\_2状态。
3. 第三次挥手：服务端做完连接关闭的准备工作后，发送一个报文给客户端，表示服务端现在发起连接释放请求，此报文FIN = 1，ACK = 1，seq = w，ack = u + 1，然后服务端进入LAST\_ACK（最后确认状态），等待客户端确认。
4. 第四次挥手：客户端收到报文后，发送一个ACK报文给服务端，表示确认关闭，服务端在收到报文后立即CLOSED，但客户端处于TIME-WAIT状态，需要等待2MSL（2 Maximum Segment Lifetime）后，才进入CLOSED（这么做主要是为了防止服务端没有收到客户端第四次挥手的ACK报文，在等待2MSL后，如果没有收到服务端的ACK报文，说明服务端确实已经关闭了，这样客户端才放心关闭）。

图表, 图示

描述已自动生成

## 算法题

设计一个栈，实现push、pop、top、getMin方法

class MyStack {

/\*\*

\* 数据栈，用于存储数据，JDK已实现push、pop、top方法

\*/

private Stack<Integer> dataStack;

/\*\*

\* 辅助栈，用于实现getMin方法，要在O(1)时间复杂度获取最小元素，所以要保证最小元素永远在栈顶

\*/

private Stack<Integer> minStack;

public MyStack() {

this.dataStack = new Stack<>();

this.minStack = new Stack<>();

}

public MyStack(Stack<Integer> dataStack, Stack<Integer> minStack) {

this.dataStack = dataStack;

this.minStack = minStack;

}

/\*\*

\* 入栈：

\* 1.对于数据栈，直接push e即可

\* 2.对于辅助栈，因为要保证栈顶元素永远是最小元素，所以：

\* \* 若辅助栈当前无元素，直接push e

\* \* 若辅助栈有元素，则比较e与辅助栈栈顶元素min：

\* \* 若e <= min，则push e

\* \* 否则，新建一个值等于min的元素push进去

\* @param e

\*/

public void push(Integer e) {

dataStack.push(e);

if (minStack.isEmpty()) {

minStack.push(e);

} else {

// 获取最小元素

Integer min = minStack.peek();

if (e <= min) {

minStack.push(e);

} else {

Integer min1 = new Integer(min);

minStack.push(min1);

}

}

}

/\*\*

\* 出栈：数据栈和辅助栈都弹出栈顶元素即可

\* @return

\*/

public Integer pop() {

if (!dataStack.isEmpty() && !minStack.isEmpty()) {

Integer result = dataStack.pop();

minStack.pop();

return result;

}

return null;

}

/\*\*

\* 获取栈顶元素：直接返回数据栈栈顶元素

\* @return

\*/

public Integer peek() {

return dataStack.peek();

}

/\*\*

\* 获取最小元素：直接返回辅助栈栈顶元素

\* @return

\*/

public Integer getMin() {

if (minStack.isEmpty()) {

return null;

}

return minStack.peek();

}

}