从尾到头打印链表

思考这个问题，链表是从头到尾遍历的，也就是最先遍历的到的数据最后输出，所以也就是“先进后出”，也就是栈的结构，我们可以使用栈来实现。

- 从头到尾遍历链表，压入栈中
- 将栈中数据逐一输出

实现代码如下：

```
/**
 * 逆向打印单链表，通过栈来实现
 */
public void printListReversingly() {
    Stack<E> stack = new Stack<>();
    Entry<E> node = header.next;
    while(node!= null) {
        stack.push(node.value);
        node = node.next;
    }
    while (!stack.isEmpty()) {
        System.out.println(stack.pop() + " ");
    }
}
```

递归本质上就是一个栈结构，所以我们可以通过递归来实现，访问一个节点的时候，先递归输出它后面的节点，再输出该节点本身。

```
/**
 * 使用递归的方式打印
 * @param node
 */
public void printListRecursively(Entry<E> node) {
    if (node != null) {
        if (node.next != null) {
            printListRecursively(node.next);
        }
        System.out.println(node.value);
    }
}
```

> 上面的代码看似简洁，但是当链表过长时，就会导致函数调用的层级很深，从而有可能导致函数调用栈溢出，所以第一种方法鲁棒性更好点。

完整的代码：

```java
public class PrintListInReversedOrder_06 {
    public static void main(String args[]) {
        SingleLinkedList<Integer> list = new SingleLinkedList();
        int[] values = {23, 4, 52, 13, 55};
        for(int i = 0; i < values.length; i++) {
            list.add(values[i]);
        }
//        list.printListReversingly();
        list.printListRecursively(list.header.next);
    }
}


// 定义单链表的数据结构
class Entry<T> {
    T value;
    Entry<T> next;

    public Entry(T value) {
        this.value = value;
    }
}

class SingleLinkedList<E> {
    // 创建头节点
    public Entry<E> header = new Entry<>(null);
    // 单链表长度
    public int size = 0;
    /**
     * 添加元素
     * @param e
     */
    public void add(E e) {
        Entry temp = header;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = new Entry<E>(e);
        size++;
    }

    /**
     * 逆向打印单链表，通过栈来实现
     */
    public void printListReversingly() {
        Stack<E> stack = new Stack<>();
        Entry<E> node = header.next;
        while(node!= null) {
            stack.push(node.value);
            node = node.next;
        }
        while (!stack.isEmpty()) {
            System.out.println(stack.pop() + " ");
        }
    }
```

```
    /**
     * 使用递归的方式打印
     * @param node
     */
    public void printListRecursively(Entry<E> node) {
        if (node != null) {
            if (node.next != null) {
                printListRecursively(node.next);
            }
            System.out.println(node.value);
        }
    }
}
```