List 接口的实现类

ArrayList：基于数组的实现，非线程安全，效率高，所有的方法都没有 synchronized 修饰。

Vector：线程安全，效率低，实现线程安全直接通过 synchronized 修饰方法来完成。

Stack：Vector 的子类，实现了栈的数据结构，（后进先出）

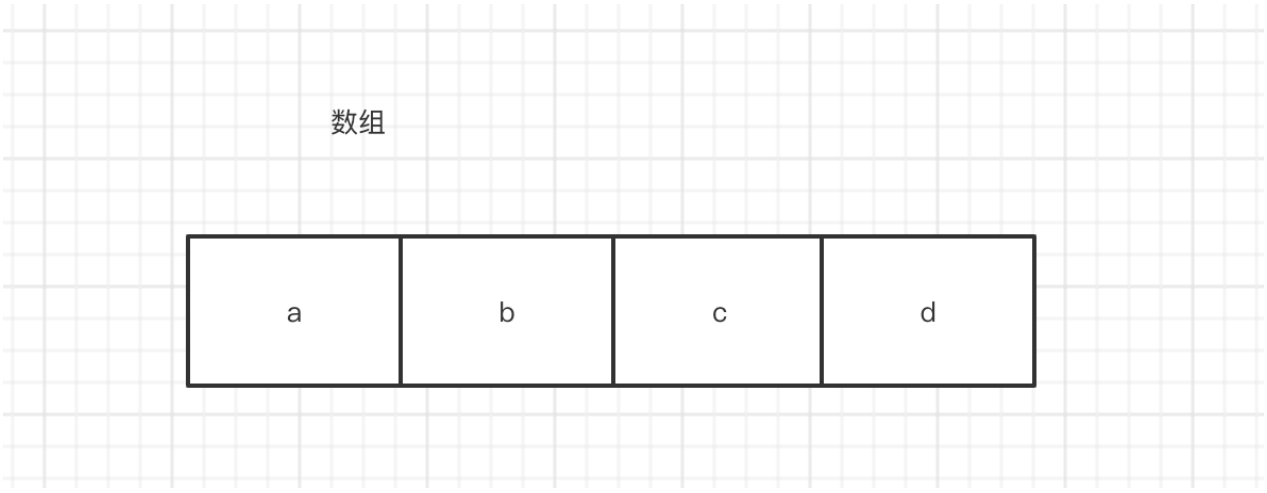
- push：入栈方法
- peek：取出栈顶元素，将栈顶复制一份取出，取完之后栈内的数据不变。
- pop：取出栈顶元素，直接取出栈顶元素，取完之后栈内的数据减一。

LikedList：实现了先进先出的队列，采用链表的形式存储。

ArrayList 和 LikedList 的区别：内存中存储的形式不同，ArrayList 采用的数组的方式，LinkedList 采用的是链表的形式。

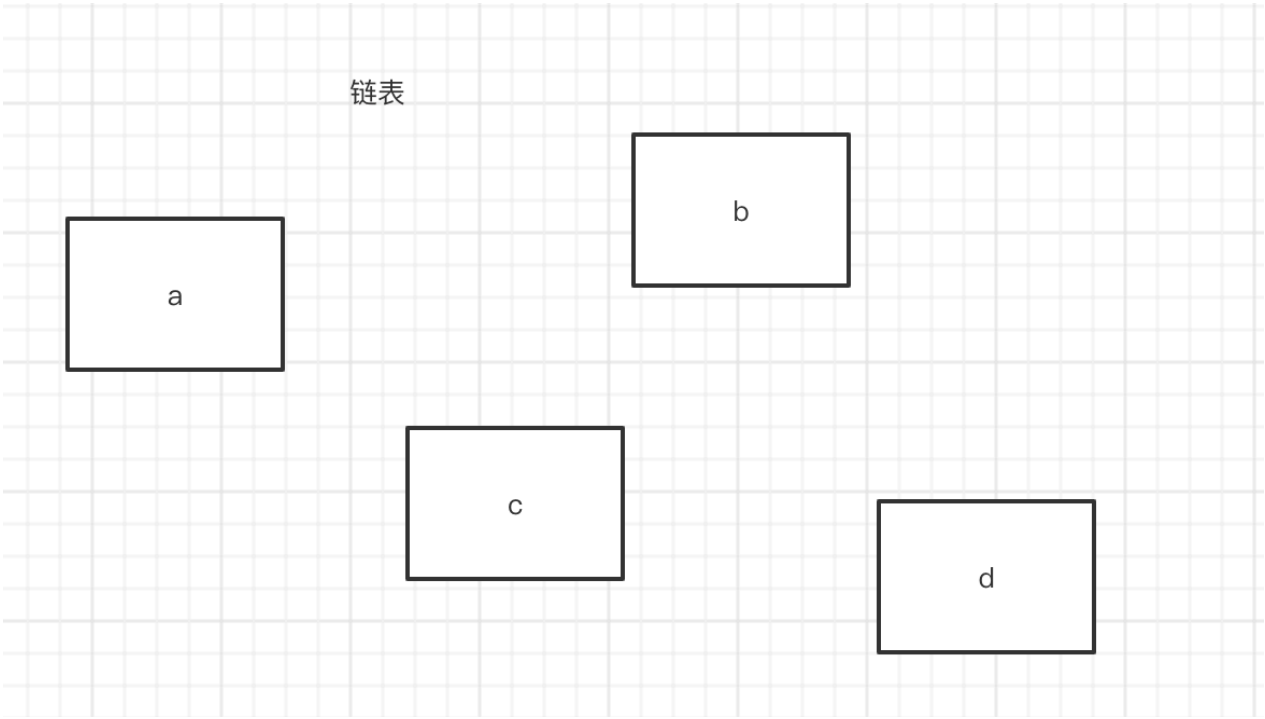
数组在内存中存储空间是连续的，读取快，增删慢。

因为数组在内存中是连续的，所以取数据可以通过寻址公式很快求出目标元素的内存地址，因为内存是连续的，所以新增或者删除元素，必然需要移动数据，而且数组长度越长，需要移动的元素越多，操作就越慢。



链表在内存中存储空间是不连续的，读取慢，增删快。链表在内存中是不连续的，没有固定的公式可以使用，要读取只能从第一位开始一直遍历到目标元素，数据规模越大，操作越慢。

增删快，因为只需要重新设置目标元素前后两个节点的后置指针即可，与数据规模无关。




```
package com.southwind.demo4;

import java.util.LinkedList;

public class Test {
    public static void main(String[] args) {
        LinkedList linkedList = new LinkedList();
        linkedList.add("Hello");
        linkedList.add("World");
        linkedList.add("Java");
        System.out.println(linkedList);
        linkedList.offer("JavaSE");
        System.out.println(linkedList);
        linkedList.push("JavaME");
        System.out.println(linkedList);
        linkedList.addFirst("First");
        System.out.println(linkedList);
        linkedList.addLast("Last");
        System.out.println(linkedList);
        System.out.println(linkedList.peek());
        System.out.println(linkedList.peekFirst());
        System.out.println(linkedList.peekLast());
        System.out.println(linkedList.pop());
        System.out.println(linkedList);
    }
}
```

LinkedList 和 Stack 都有 pop 方法，有什么区别和相同点?

pop 方法都是取出集合中的第一个元素，但是两者的顺序是相反的，Stack 是“后进先出”，所以 pop 取出的是最后一个元素，LinkedList 是“先进先出”，所以 pop 取出的是第一个元素。

LinkedList 实现了 Deque 接口，而 Deque 接口是 Queue 的子接口，Queue 就是队列，底层实现了队列的数据结构。

实际开发中，不能直接实例化 Queue 对象。

Queue 的实现类是 AbstractQueue，它是一个抽象类，不能直接实例化，开发中需要实现它的子类 PriorityQueue。

Queue 中添加的数据必须是有顺序的。

```
package com.southwind.demo5;

import java.util.PriorityQueue;

public class Test {
    public static void main(String[] args) {
        PriorityQueue queue = new PriorityQueue();
//        queue.add(1);
//        queue.add(2);
//        queue.add(3);
//        queue.add("a");
//        queue.add("b");
//        queue.add("c");

        queue.add(new A(1));
        queue.add(new A(2));
        System.out.println(queue);
    }
}

class A implements Comparable{

    private int num;

    public A(int num) {
        this.num = num;
    }

    @Override
    public int compareTo(Object o) {
        A a = (A)o;
        if(this.num > a.num){
            return 1;
        }else if(this.num == a.num){
            return 0;
        }else{
            return -1;
```

```
        }
    }

    @Override
    public String toString() {
        return "A{" +
                "num=" + num +
                '}';
    }
}
```

Queue 默认给元素进行升序排列，即自然排序。