1.数组长度在初始化时指定，意味着只能保存定长的数据。而集合可以保存数量不确定的数据。同时可以保存具有映射关系的数据（即关联数组，键值对 key-value）。

2.数组元素即可以是基本类型的值，也可以是对象。集合里只能保存对象（实际上只是保存对象的引用变量），基本数据类型的变量要转换成对应的包装类才能放入集合类中。

ArrayList经典Demo：

```
package list;
import java.util.ArrayList;
import java.util.Iterator;

public class ArrayListDemo {

    public static void main(String[] srgs){
        ArrayList<Integer> arrayList = new ArrayList<Integer>();

        System.out.printf("Before add:arrayList.size() =
%d\n",arrayList.size());

        arrayList.add(1);
        arrayList.add(3);
        arrayList.add(5);
        arrayList.add(7);
        arrayList.add(9);
        System.out.printf("After add:arrayList.size() =
%d\n",arrayList.size());

        System.out.println("Printing elements of arrayList");
        // 三种遍历方式打印元素
        // 第一种：通过迭代器遍历
        System.out.print("通过迭代器遍历:");
        Iterator<Integer> it = arrayList.iterator();
        while(it.hasNext()){
```

```
            System.out.print(it.next() + " ");
        }
        System.out.println();

        // 第二种：通过索引值遍历
        System.out.print("通过索引值遍历:");
        for(int i = 0; i < arrayList.size(); i++){
            System.out.print(arrayList.get(i) + " ");
        }
        System.out.println();

        // 第三种：for循环遍历
        System.out.print("for循环遍历:");
        for(Integer number : arrayList){
            System.out.print(number + " ");
        }

        // toArray用法
        // 第一种方式(最常用)
        Integer[] integer = arrayList.toArray(new Integer[0]);

        // 第二种方式(容易理解)
        Integer[] integer1 = new Integer[arrayList.size()];
        arrayList.toArray(integer1);

        // 抛出异常，java不支持向下转型
        //Integer[] integer2 = new Integer[arrayList.size()];
        //integer2 = arrayList.toArray();
        System.out.println();

        // 在指定位置添加元素
        arrayList.add(2,2);
        // 删除指定位置上的元素
        arrayList.remove(2);
        // 删除指定元素
        arrayList.remove((Object)3);
        // 判断arrayList是否包含5
        System.out.println("ArrayList contains 5 is: " +
```

```
arrayList.contains(5));

        // 清空ArrayList
        arrayList.clear();
        // 判断ArrayList是否为空
        System.out.println("ArrayList is empty: " +
arrayList.isEmpty());
    }
}
```

LinkedList类经典demo：

```
package list;

import java.util.Iterator;
import java.util.LinkedList;

public class LinkedListDemo {
    public static void main(String[] srgs) {
        //创建存放int类型的linkedList
        LinkedList<Integer> linkedList = new LinkedList<>();
        /************************ linkedList的基本操作 ************************/
        linkedList.addFirst(0); // 添加元素到列表开头
        linkedList.add(1); // 在列表结尾添加元素
        linkedList.add(2, 2); // 在指定位置添加元素
        linkedList.addLast(3); // 添加元素到列表结尾

        System.out.println("LinkedList（直接输出的）: " + linkedList);

        System.out.println("getFirst()获得第一个元素: " + linkedList.getFirst()); //
返回此列表的第一个元素
        System.out.println("getLast()获得第最后一个元素: " + linkedList.getLast()); //
返回此列表的最后一个元素
        System.out.println("removeFirst()删除第一个元素并返回: " +
linkedList.removeFirst()); // 移除并返回此列表的第一个元素
        System.out.println("removeLast()删除最后一个元素并返回: " +
linkedList.removeLast()); // 移除并返回此列表的最后一个元素
        System.out.println("After remove:" + linkedList);
        System.out.println("contains()方法判断列表是否包含1这个元素:" +
linkedList.contains(1)); // 判断此列表包含指定元素，如果是，则返回true
        System.out.println("该linkedList的大小 : " + linkedList.size()); // 返回此列表
的元素个数

        /************************ 位置访问操作 ************************/
        System.out.println("--------------------------------------");
```

```
        linkedList.set(1, 3); // 将此列表中指定位置的元素替换为指定的元素
        System.out.println("After set(1, 3):" + linkedList);
        System.out.println("get(1)获得指定位置（这里为1）的元素: " +
linkedList.get(1)); // 返回此列表中指定位置处的元素

        /************************** Search操作 ************************/
        System.out.println("----------------------------------------");
        linkedList.add(3);
        System.out.println("indexOf(3): " + linkedList.indexOf(3)); // 返回此列表中首
次出现的指定元素的索引
        System.out.println("lastIndexOf(3): " + linkedList.lastIndexOf(3));// 返回此列
表中最后出现的指定元素的索引

        /************************** Queue操作 ************************/
        System.out.println("----------------------------------------");
        System.out.println("peek(): " + linkedList.peek()); // 获取但不移除此列表的头
        System.out.println("element(): " + linkedList.element()); // 获取但不移除此列
表的头
        linkedList.poll(); // 获取并移除此列表的头
        System.out.println("After poll():" + linkedList);
        linkedList.remove();
        System.out.println("After remove():" + linkedList); // 获取并移除此列表的头
        linkedList.offer(4);
        System.out.println("After offer(4):" + linkedList); // 将指定元素添加到此列表
的末尾

        /************************** Deque操作 ************************/
        System.out.println("----------------------------------------");
        linkedList.offerFirst(2); // 在此列表的开头插入指定的元素
        System.out.println("After offerFirst(2):" + linkedList);
        linkedList.offerLast(5); // 在此列表末尾插入指定的元素
        System.out.println("After offerLast(5):" + linkedList);
        System.out.println("peekFirst(): " + linkedList.peekFirst()); // 获取但不移除
此列表的第一个元素
        System.out.println("peekLast(): " + linkedList.peekLast()); // 获取但不移除此
列表的第一个元素
        linkedList.pollFirst(); // 获取并移除此列表的第一个元素
        System.out.println("After pollFirst():" + linkedList);
        linkedList.pollLast(); // 获取并移除此列表的最后一个元素
        System.out.println("After pollLast():" + linkedList);
        linkedList.push(2); // 将元素推入此列表所表示的堆栈（插入到列表的头）
        System.out.println("After push(2):" + linkedList);
        linkedList.pop(); // 从此列表所表示的堆栈处弹出一个元素（获取并移除列表第一个
元素）
        System.out.println("After pop():" + linkedList);
        linkedList.add(3);
        linkedList.removeFirstOccurrence(3); // 从此列表中移除第一次出现的指定元素（从
```

```
头部到尾部遍历列表）
        System.out.println("After removeFirstOccurrence(3):" + linkedList);
        linkedList.removeLastOccurrence(3); // 从此列表中移除最后一次出现的指定元素
（从尾部到头部遍历列表）
        System.out.println("After removeFirstOccurrence(3):" + linkedList);

        /************************ 遍历操作 ************************/
        System.out.println("----------------------------------------");
        linkedList.clear();
        for (int i = 0; i < 100000; i++) {
            linkedList.add(i);
        }
        // 迭代器遍历
        long start = System.currentTimeMillis();
        Iterator<Integer> iterator = linkedList.iterator();
        while (iterator.hasNext()) {
            iterator.next();
        }
        long end = System.currentTimeMillis();
        System.out.println("Iterator: " + (end - start) + " ms");

        // 顺序遍历(随机遍历)
        start = System.currentTimeMillis();
        for (int i = 0; i < linkedList.size(); i++) {
            linkedList.get(i);
        }
        end = System.currentTimeMillis();
        System.out.println("for: " + (end - start) + " ms");

        // 另一种for循环遍历
        start = System.currentTimeMillis();
        for (Integer i : linkedList)
            ;
        end = System.currentTimeMillis();
        System.out.println("for2: " + (end - start) + " ms");

        // 通过pollFirst()或pollLast()来遍历LinkedList
        LinkedList<Integer> temp1 = new LinkedList<>();
        temp1.addAll(linkedList);
        start = System.currentTimeMillis();
        while (temp1.size() != 0) {
            temp1.pollFirst();
        }
        end = System.currentTimeMillis();
        System.out.println("pollFirst()或pollLast(): " + (end - start) + " ms");

        // 通过removeFirst()或removeLast()来遍历LinkedList
```

```
        LinkedList<Integer> temp2 = new LinkedList<>();
        temp2.addAll(linkedList);
        start = System.currentTimeMillis();
        while (temp2.size() != 0) {
            temp2.removeFirst();
        }
        end = System.currentTimeMillis();
        System.out.println("removeFirst()或removeLast(): " + (end - start) + " ms");
    }
}
```