05 集合_List

JAVAEE高级

一：List接口

> 1、Java中数组用来存储数据的局限性
>
> 2、List集合类中元素有序、且可重复，集合中的每个元素都有其对应的顺序索引
>
> 3、List容器中的元素都对应一个整数型的序号记载其在容器中的位置，可以根据序号存取容器中的元素
>
> 4、List接口的实现类常用的有：ArrayList、LinkedList和Vector

1、 ：实现类ArrayList

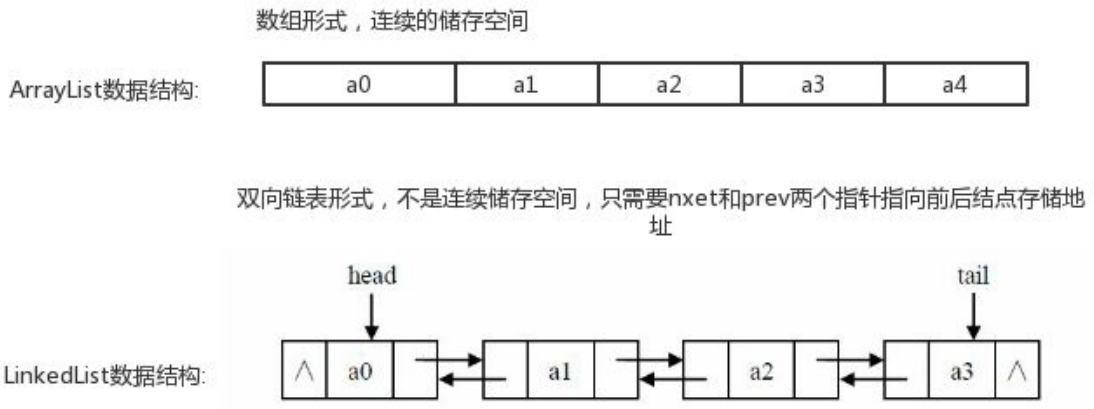
> 1、ArrayList 是 List 接口的典型实现类
>
> 2、在本职上ArrayList是对象引用的一个变长数组

```
    //创建一个对象数组
        Student[] arr = new Student[5];
        arr[0] = new Student("张三", 23);
```

```
@Test
    public void getTest(){
    //父类引用指向子类对象
        Collection c = new ArrayList();
        c.add("a");
        c.add("b");
        c.add("c");
        c.add("d");
        c.remove("b");       //删除指定元素
        //c.clear();         //清空集合
```

```
11.         System.out.println(c.contains("b"));     //判断是否包含
12.         System.out.println(c.isEmpty());     //如果此 collection 不包含元素，则返回 true。
13.         System.out.println(c.size());        //获取元素的个数
14.         System.out.println(c);
15.         Object[] array = c.toArray();
16.     }
```

3、数据结构图ArrayList/LinkedList



4、ArrayList 是线程不安全的(方法上没synchronized)，而 Vector 是线程安全的方法上有(synchronized)

```
1.  public synchronized void trimToSize() {
2.         modCount++;
3.         int oldCapacity = elementData.length;
4.         if (elementCount < oldCapacity) {
5.             elementData = Arrays.copyOf(elementData, elementCount);
6.         }
7.     }
```

5、集合与数组之间的转换

```
1.     @Test
2.     public void getTest() {
3.         Integer[] arr = {11,22,33,44,55};
4.         List<Integer> list = Arrays.asList(arr);
```

```
            System.out.println(list);
            Object[] array = list.toArray();
        }
```

2、数组(Array)和列表(ArrayList)有什么区别？

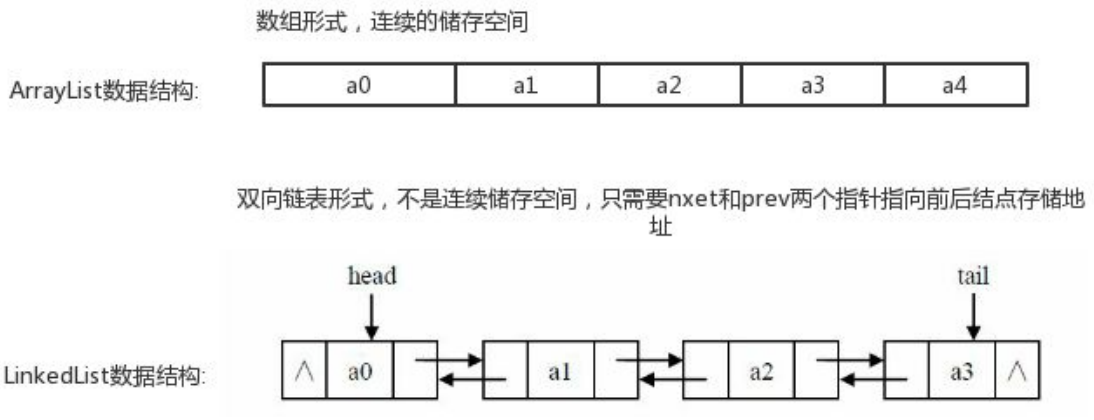
> 1、Array可以包含基本类型和对象类型，ArrayList只能包含对象类型。
> 2、Array大小是固定的，ArrayList的大小是动态变化的
> 3、ArrayList提供了更多的方法和特性，比如：addAll()，removeAll()，iterator()等等。

2：实现类LinkedList

> 1、对于频繁的插入或删除元素的操作，建议使用LinkedList类，效率较高

```
@Test
    public void getTest() {
         LinkedList linkedList = new  LinkedList();
         linkedList.add("我-");
         linkedList.add("要");
         linkedList.add("学");
        for (Object object : linkedList) {
            System.out.println(object);
        }
    }
```

> 2、数据结构图ArrayList/LinkedList



1、LinkedList模拟栈结构

> 1、LinkedList模拟栈结构
>
> 2、先进后出

```
1.  public class TestInt {
2.      @Test
3.      public void getTest() {
4.          Stack s = new Stack();
5.          s.in("a"); // 进栈
6.          s.in("b");
7.          s.in("c");
8.          s.in("d");
9.          while (!s.isEmpty()) { // 判断栈结构是否为空
10.             System.out.println(s.out()); // 弹栈
11.         }
12.     }
13. }
14.
15. class Stack {
16.     private LinkedList list = new LinkedList();
17.
18.     /*
19.      * 模拟进栈方法
20.      */
21.     public void in(Object obj) {
22.         list.addLast(obj);
23.     }
```

```
24.
25.     /*
26.      * 模拟出栈
27.      */
28.     public Object out() {
29.         return list.removeLast();
30.     }
31.
32.     /*
33.      * 模拟栈结构是否为空
34.      */
35.
36.     public boolean isEmpty() {
37.         return list.isEmpty();
38.     }
39. }
```

> 2、LinkedList模拟栈结构

```
1.  @Test
2.      public void getTest() {
3.          LinkedList list = new LinkedList();                    //创建集合对
    象
4.          list.addLast("a");
5.          list.addLast("b");
6.          list.addLast("c");
7.          list.addLast("d");
8.          while(!list.isEmpty()) {
9.              System.out.println(list.removeLast());
10.         }
11.     }
```

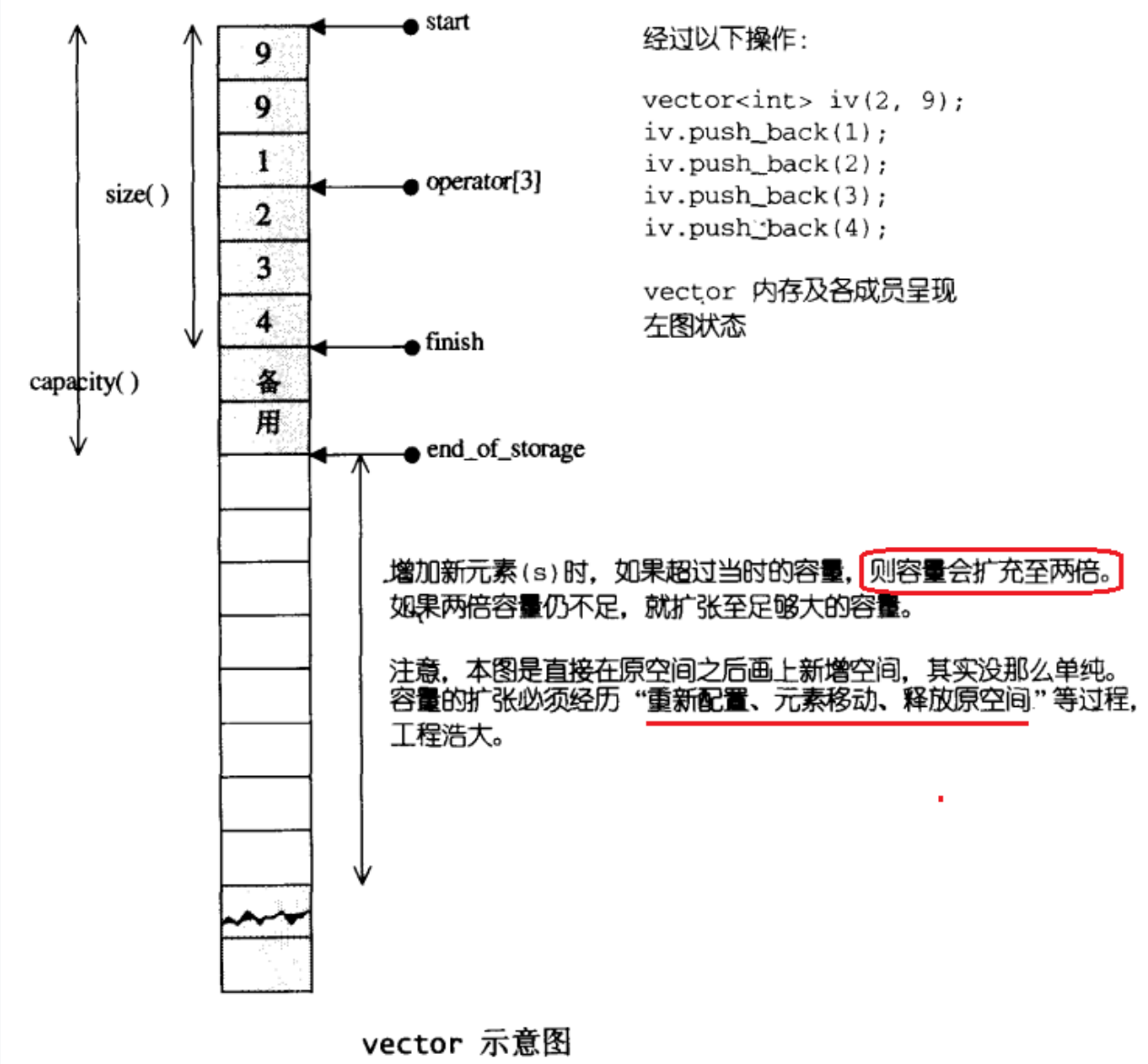
3：实现类Vector

> 1、Vector 是一个老的集合，JDK1.0就存在
>
> 2、大多数操作与ArrayList相同，区别之处在于Vector是线程安全的
>
> 3、当插入、删除频繁时，使用LinkedList；Vector总是比ArrayList慢，所以尽量避免使用

```java
/**
*遍历vecter集合
*/
@Test
    public void getTest() {
        Vector vector = new Vector();
        vector.add("我");
        vector.add("要");
        vector.add("学");
        for (int i = 0; i < vector.size(); i++) {
            System.out.println(vector.get(i));
        }
    }
```

```java
@Test
    public void getTest() {
        Vector v = new Vector();
        v.addElement("a");
        v.addElement("d");
        Enumeration en = v.elements(); // 获取枚举
        while (en.hasMoreElements()) { // 判断集合中是否有元素
            System.out.println(en.nextElement()); // 获取集合中的元素
        }
    }
```

4、数据结构图



vector 示意图

二：ListIterator接口

1、List 额外提供了一个 listIterator() 方法

2、该方法返回一个 ListIterator 对象， ListIterator 接口继承了 Iterator 接口，提供了专门操作 List 的方法

```
/**
*Iterator
*/
    @Test
```

```
    public void getTest() {
        Collection c = new ArrayList();
        c.add(23);
        c.add(24);
        // 获取迭代器
        Iterator it = c.iterator();
        while (it.hasNext()) {
            Object next = it.next();// 向下转型
            System.out.println(next);
        }
    }
```

三：Iterator和ListIterator主要区别

> 1、ListIterator和Iterator都有hasNext()和next()方法，可以实现顺序向后遍历

```

/**
*ListIterator
*/
@Test
    public void getTest() {
        ArrayList c = new ArrayList();
        c.add(23);
        c.add(24);
        // 获取迭代器
         ListIterator it = c.listIterator();
        while (it.hasNext()) {
            Object next = it.next();// 向下转型
            System.out.println(next);
        }
    }
```

> 2、但是ListIterator有hasPrevious()和previous()方法，可以实现逆向（逆序遍历）；Iterator就不可以
> 3、ListIterator有add()方法，可以向List中插入对象，而Iterator不能

```
@Test
    public void getTest() {
```

```
        ArrayList c = new ArrayList();
        c.add(23);
        c.add(24);
        // 获取迭代器
        ListIterator it = c.listIterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }
        it.add("1234");
        System.out.println("。。。。。。");
        while (it.hasPrevious()) {
            System.out.println(it.previous());

        }
    }
```

> 4、都可实现删除对象，但是ListIterator可以实现对象的修改，set()方法可以实现。Iterator仅能遍历，不能修改

```
@Test
    public void getTest() {
        List list = new ArrayList();
        list.add("one");
        list.add("two");
        // 用ListIterator 进行修改
        for (ListIterator it = list.listIterator(); it.hasNext();) {
            Object obj = it.next();
            if ("two".equals(obj)) {
                /* it.add("1111"); */
                it.set("3");
            }
        }
        System.out.println(list);
    }
```

四：案列

1、案例：<去除集合中字符串的重复值>

> 1、需求：ArrayList去除集合中字符串的重复值(字符串的内容相同)

```
@Test
    public void getTest() {
        ArrayList list = new ArrayList();
        list.add("a");
        list.add("a");
        list.add("b");
        list.add("b");
        list.add("b");
        list.add("c");
        list.add("c");
        list.add("c");
        list.add("c");
        System.out.println(list);
        ArrayList newList = getSingle(list);
        System.out.println(newList);

    }

    public static ArrayList getSingle(ArrayList list) {
        ArrayList newList = new ArrayList(); // 创建一个新集合
        Iterator it = list.iterator(); // 获取迭代器
        while (it.hasNext()) { // 判断老集合中是否有元素
            String temp = (String) it.next(); // 将每一个元素临时记录住
            if (!newList.contains(temp)) { // 如果新集合中不包含该元素
                newList.add(temp); // 将该元素添加到新集合中
            }
        }
        return newList; // 将新集合返回
    }
```

2、案例：<去除集合中对象的成员变量值相同>

> 1、需求：ArrayList去除集合中对象的成员变量值相同
> 2、实体类

```
public class Person {
    private String name;
    private int age;
```

```
    //构造方法 与 get;set方法
    //重写equals方法
    @Override
    public boolean equals(Object obj) {
        Person p = (Person)obj;
        return this.name.equals(p.name) && this.age == p.age;
    }

    }
```

```
@Test
    public void getTest() {
ArrayList list = new ArrayList(); // 创建集合对象
        list.add(new Person("张三", 23));
        list.add(new Person("张三", 23));
        list.add(new Person("李四", 24));
        list.add(new Person("李四", 24));
        list.add(new Person("李四", 24));
        list.add(new Person("李四", 24));
        ArrayList newList = getSingle(list); // 调用方法去除重复
        System.out.println(newList);
    }

    /*
     * 分析：1,创建新集合
     *      2,根据传入的集合(老集合)获取迭代器
     *      3,遍历老集合
     *      4,通过新集合判断是否包含老集合中的元素,如果包含就不添加,如果不包含就添加
     */
    public static ArrayList getSingle(ArrayList list) {
        ArrayList newList = new ArrayList<>(); // 1,创建新集合
        Iterator it = list.iterator(); // 2,根据传入的集合(老集合)获取迭代器
        while (it.hasNext()) { // 3,遍历老集合
            Object obj = it.next(); // 记录住每一个元素
            if (!newList.contains(obj)) { // 如果新集合中不包含老集合中的元素
                newList.add(obj); // 将该元素添加
            }
        }
        return newList;
    }
```

五：集合使用泛型

1、泛型好处

> 1、提高安全性(将运行期的错误转换到编译期)
> 2、省去强转的麻烦

2、泛型基本使用

> 1、<>中放的必须是引用数据类型

3、泛型使用注意事项

> 1、前后的泛型必须一致,或者后面的泛型可以省略不写(1.7的新特性菱形泛型)

```
1.      //泛型最好不要定义成Object,没有意义
2.       ArrayList<String> list = new ArrayList<>();
3.           list.add("aaa");
```