

# 05 集合\_List

JAVAAEE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、在本质上ArrayList是对象引用的一个变长数组

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

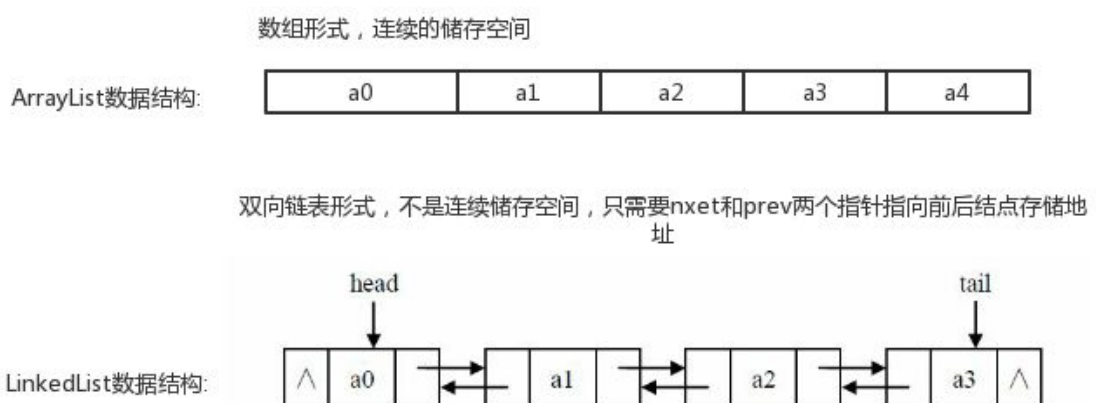
```
1.      @Test
2.      public void getTest(){
3.          //父类引用指向子类对象
4.          Collection c = new ArrayList();
5.          c.add("a");
6.          c.add("b");
7.          c.add("c");
8.          c.add("d");
9.          c.remove("b");          //删除指定元素
10.         c.clear();              //清空集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

```

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);

```

```
5.         System.out.println(list);
6.         Object[] array = list.toArray();
7.     }
```

## 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

```
1.     @Test
2.     public void getTest() {
3.         LinkedList linkedList = new LinkedList();
4.         linkedList.add("我-");
5.         linkedList.add("要");
6.         linkedList.add("学");
7.         for (Object object : linkedList) {
8.             System.out.println(object);
9.         }
10.    }
```

## 2、数据结构图ArrayList/LinkedList

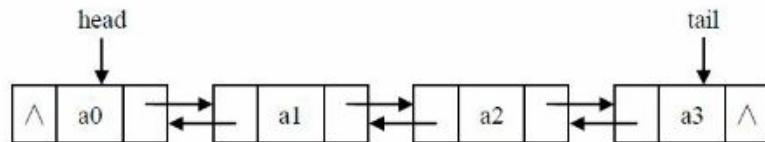
数组形式，连续的储存空间

ArrayList数据结构:



双向链表形式，不是连续储存空间，只需要next和prev两个指针指向前后结点存储地址

LinkedList数据结构:



## 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

```

1.  /**
2.   *遍历vector集合
3.   */
4.  @Test
5.      public void getTest() {
6.          Vector vector = new Vector();
7.          vector.add("我");
8.          vector.add("要");
9.          vector.add("学");
10.         for (int i = 0; i < vector.size(); i++) {
11.             System.out.println(vector.get(i));
12.         }
13.     }

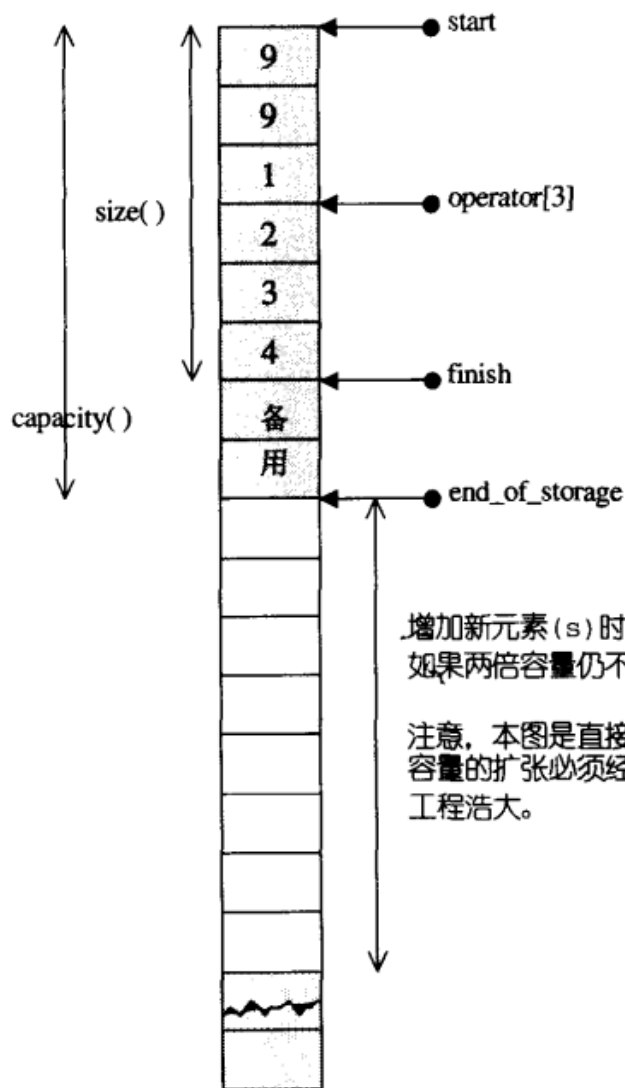
```

```

1.  @Test
2.      public void getTest() {
3.          Vector v = new Vector();
4.          v.addElement("a");
5.          v.addElement("d");
6.          Enumeration en = v.elements(); // 获取枚举
7.          while (en.hasMoreElements()) { // 判断集合中是否有元素
8.              System.out.println(en.nextElement()); // 获取集合中的元素
9.          }
10.     }

```

#### 4、数据结构图



经过以下操作：

```
vector<int> iv(2, 9);
iv.push_back(1);
iv.push_back(2);
iv.push_back(3);
iv.push_back(4);
```

vector 内存及各成员呈现左图状态

增加新元素(s)时，如果超过当时的容量，**则容量会扩充至两倍。**如果两倍容量仍不足，就扩张至足够大的容量。

注意，本图是直接在原空间之后画上新增空间，其实没那么单纯。容量的扩张必须经历 “重新配置、元素移动、释放原空间” 等过程，工程浩大。

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

```
1.  /**
2.   *Iterator
3.   */
4.   @Test
```

```

5.     public void getTest() {
6.         Collection c = new ArrayList();
7.         c.add(23);
8.         c.add(24);
9.         // 获取迭代器
10.        Iterator it = c.iterator();
11.        while (it.hasNext()) {
12.            Object next = it.next(); // 向下转型
13.            System.out.println(next);
14.        }
15.    }

```

### 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

```

1.
2.     /**
3.     *ListIterator
4.     */
5.     @Test
6.     public void getTest() {
7.         ArrayList c = new ArrayList();
8.         c.add(23);
9.         c.add(24);
10.        // 获取迭代器
11.        ListIterator it = c.listIterator();
12.        while (it.hasNext()) {
13.            Object next = it.next(); // 向下转型
14.            System.out.println(next);
15.        }
16.    }

```

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

```

1.     @Test
2.     public void getTest() {

```



```

3.      ArrayList c = new ArrayList();
4.      c.add(23);
5.      c.add(24);
6.      // 获取迭代器
7.      ListIterator it = c.listIterator();
8.      while (it.hasNext()) {
9.          System.out.println(it.next());
10.     }
11.     it.add("1234");
12.     System.out.println("。 。 。 。 。");
13.     while (it.hasPrevious()) {
14.         System.out.println(it.previous());
15.     }
16.
17. }

```

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

```

1.      @Test
2.      public void getTest() {
3.          List list = new ArrayList();
4.          list.add("one");
5.          list.add("two");
6.          // 用ListIterator 进行修改
7.          for (ListIterator it = list.listIterator(); it.hasNext();) {
8.              Object obj = it.next();
9.              if ("two".equals(obj)) {
10.                 /* it.add("1111"); */
11.                 it.set("3");
12.             }
13.         }
14.         System.out.println(list);
15.     }

```

## 四：案例

### 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)

```
1.  @Test
2.      public void getTest() {
3.          ArrayList list = new ArrayList();
4.          list.add("a");
5.          list.add("a");
6.          list.add("b");
7.          list.add("b");
8.          list.add("b");
9.          list.add("c");
10.         list.add("c");
11.         list.add("c");
12.         list.add("c");
13.         System.out.println(list);
14.         ArrayList newList = getSingle(list);
15.         System.out.println(newList);
16.
17.     }
18.
19.     public static ArrayList getSingle(ArrayList list) {
20.         ArrayList newList = new ArrayList(); // 创建一个新集合
21.         Iterator it = list.iterator(); // 获取迭代器
22.         while (it.hasNext()) { // 判断老集合中是否有元素
23.             String temp = (String) it.next(); // 将每一个元素临时记录住
24.             if (!newList.contains(temp)) { // 如果新集合中不包含该元素
25.                 newList.add(temp); // 将该元素添加到新集合中
26.             }
27.         }
28.         return newList; // 将新集合返回
29.     }
```

## 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

```
1.  public class Person {
2.      private String name;
3.      private int age;
```

```

4.      //构造方法 与 get;set方法
5.      //重写equals方法
6.      @Override
7.      public boolean equals(Object obj) {
8.          Person p = (Person)obj;
9.          return this.name.equals(p.name) && this.age == p.age;
10.     }
11.
12.     }

```

```

1.      @Test
2.      public void getTest() {
3.      ArrayList list = new ArrayList(); // 创建集合对象
4.          list.add(new Person("张三", 23));
5.          list.add(new Person("张三", 23));
6.          list.add(new Person("李四", 24));
7.          list.add(new Person("李四", 24));
8.          list.add(new Person("李四", 24));
9.          list.add(new Person("李四", 24));
10.         ArrayList newList = getSingle(list); // 调用方法去除重复
11.         System.out.println(newList);
12.     }
13.
14.     /*
15.     * 分析: 1,创建新集合
16.     *        2,根据传入的集合(老集合)获取迭代器
17.     *        3,遍历老集合
18.     *        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
19.     */
20.     public static ArrayList getSingle(ArrayList list) {
21.         ArrayList newList = new ArrayList<>(); // 1,创建新集合
22.         Iterator it = list.iterator(); // 2,根据传入的集合(老集合)获取迭代器
23.         while (it.hasNext()) { // 3,遍历老集合
24.             Object obj = it.next(); // 记录住每一个元素
25.             if (!newList.contains(obj)) { // 如果新集合中不包含老集合中的元
素
26.                 newList.add(obj); // 将该元素添加
27.             }
28.         }
29.         return newList;
30.     }

```

## 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");
```