简单集合：

ArrayList、HashMap、LinkedList

实现的类

List<E>：支持增删，遍历

RandomAccess：支持随机访问

Cloneable：可克隆

Serializable：可序列化

```
implements List<E>, RandomAccess, Cloneable, java.io.Serializable
```

属性定义

- new ArrayList()创建数组时的默认容量为10

```
private static final int DEFAULT_CAPACITY = 10;
```

- 通过new ArrayList(0)创建的空数组 当有数据时会初始化为10的大小

```
private static final Object[] EMPTY_ELEMENTDATA = {};
```

- 通过new ArrayList()创建的空数组 默认容量为10

```
private static final Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA = {};
```

- 存放元素
- transient 关键字：关闭序列化

```
transient Object[] elementData;
```

- 数组中真实的元素个数

```
private int size;
```

构造方法

- 给定数组容量，大于0时会变成初始化为传入的大小，等于0使用EMPTY_ELEMENTDATA

```
public ArrayList(int initialCapacity) {
 if (initialCapacity > 0) {
 this.elementData = new Object[initialCapacity];
 } else if (initialCapacity == 0) {
 this.elementData = EMPTY_ELEMENTDATA;
 } else {
 throw new IllegalArgumentException("Illegal Capacity: "+
 initialCapacity);
 }
}
```

- 不给定数组容量，使用默认值大小的空数组

```
public ArrayList() {
 this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA;
}
```

- 将传入的元素拷贝到elementData
- 如果传入个数0，则生成EMPTY_ELEMENTDAT空数组

```
public ArrayList(Collection<? extends E> c) {
 elementData = c.toArray();
 if ((size = elementData.length) != 0) {
 // c.toArray might (incorrectly) not return Object[] (see 62606
52)
 if (elementData.getClass() != Object[].class)
 elementData = Arrays.copyOf(elementData, size, Object[].class);
 } else {
 // replace with empty array.
 this.elementData = EMPTY_ELEMENTDATA;
  }
}
```

核心方法

- **新增元素 add 向后插入**

把一个元素新增到elementData，modCount++ 声明我新增元素了

```
public boolean add(E e) {
```

```
    ensureCapacityInternal(size + 1); // Increments modCount!!
    elementData[size++] = e;
    return true;
}
```

- **add 指定位置插入**

```
public void add(int index, E element) {
    //下标检查，是否越界了
    rangeCheckForAdd(index);
    //扩增容量，同时改变modcount
    ensureCapacityInternal(size + 1);
    //index后面的元素后移
    System.arraycopy(elementData, index, elementData, index + 1, si
ze - index);
    //指定位置放置元素
    elementData[index] = element;
     //元素数量大小自增
     size++;
     }
```

- **删除元素**
- 由上面的插入方法可以看到List底层的数组处理使用到了System.arraycopy()方法,下面综合删除和数组复制方法讲下删除原理，下面是删除代码:

```
public E remove(int index) {
    rangeCheck(index);

    modCount++;
    E oldValue = elementData(index);
    // 计算长度
    int numMoved = size - index - 1;
    if (numMoved > 0)
    // param1: 源数组
     // param2: 源数组要复制的起始位置
     // param3: 目标数组
     // param4: 目标数组放置的起始位置
     // param5: 复制的长度
```

```
14   System.arraycopy(elementData, index+1, elementData, index,
15   numMoved);
16   elementData[--size] = null; // clear to let GC do its work
17
18   return oldValue;
19 }
```