ArrayList底层基于数组（Array）实现，默认数据大小 ***DEFAULT_CAPACITY*** = 10，真正存储元素 **elementData，** 一切的操作其根本就是对数据的操作。

一、插入(add)操作过程

1. 尾部添加 add(E element)

2.指定位置添加 add(int index, E element)

二、删除操作 remove(int index)

三、查询获取 get(int index)

关键属性介绍：

```
/**
 * 默认容量
 */
private static final int DEFAULT_CAPACITY = 10;

/**
 * 实际存储数据的数组
 */
transient Object[] elementData; // non-private to simplify nested class access

 /**
  * 当前list的长度大小
  *
  */
 private int size;
```

一、插入(add)操作过程

1. 尾部添加 add(E element)

```
/**
 * Appends the specified element to the end of this list.
 *
 * @param e element to be appended to this list
 * @return <tt>true</tt> (as specified by {@link Collection#add})
 */
public boolean add(E e) {
    ensureCapacityInternal(size + 1);  // Increments modCount!!
    elementData[size++] = e;
     return true;
 }
```

如上图可知插入流程如下：

1. 判断是否要进行扩容
2. 进行存值

扩容细节：

a.计算容器最小容量值

```
private static int calculateCapacity(Object[] elementData, int minCapacity) {
    if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
        return Math.max(DEFAULT_CAPACITY, minCapacity);
    }
    return minCapacity;
}

private void ensureCapacityInternal(int minCapacity) {
    ensureExplicitCapacity(calculateCapacity(elementData, minCapacity));
}

private void ensureExplicitCapacity(int minCapacity) {
    modCount++;

    // overflow-conscious code
    if (minCapacity - elementData.length > 0)
        grow(minCapacity);
}
```

b.根据旧的容量(oldCapacity)计算新的容量(newCapacity)

```
/**
 * Increases the capacity to ensure that it can hold at least the
 * number of elements specified by the minimum capacity argument.
 *
 * @param minCapacity the desired minimum capacity
 */
private void grow(int minCapacity) {
    // overflow-conscious code
    int oldCapacity = elementData.length;
    // 15 = 10 + 10/2 每次扩容50%
    int newCapacity = oldCapacity + (oldCapacity >> 1);
    if (newCapacity - minCapacity < 0)
        newCapacity = minCapacity;
    if (newCapacity - MAX_ARRAY_SIZE > 0)
        newCapacity = hugeCapacity(minCapacity);
    // minCapacity is usually close to size, so this is a win:
    elementData = Arrays.copyOf(elementData, newCapacity);
}
```

其中从 扩容代码 **int** newCapacity = oldCapacity + (oldCapacity >> 1) 可以看出，每次扩容倍数为0.5倍，
比如 原始容量为 int oldCapacity = 10;那么下次扩容后容量为 15 = 10 + 10/2， 当前为右移1位。
ps:
位移运算符中：右移缩小、左右扩大。

c. 进行系统级别的扩容拷贝 **elementData** = Arrays.*copyOf*(**elementData**, newCapacity);

2.指定位置添加 add(int index, E element)

```
/**
 * Inserts the specified element at the specified position in this
 * list. Shifts the element currently at that position (if any) and
 * any subsequent elements to the right (adds one to their indices).
 *
 * @param index index at which the specified element is to be inserted
 * @param element element to be inserted
 * @throws IndexOutOfBoundsException {@inheritDoc}
 */
```

```
public void add(int index, E element) {
    rangeCheckForAdd(index);

    ensureCapacityInternal(size + 1);  // Increments modCount!!
    System.arraycopy(elementData, index, elementData, index + 1,
                     size - index);
    elementData[index] = element;
    size++;
}
```

指定位置进行添加一般来说比较耗费资源，对比与add(E e)方法，指定位置进行添加，index后续元素要进行移动（复制）

二、删除操作 remove(int index)

```
/**
 * Removes the element at the specified position in this list.
 * Shifts any subsequent elements to the left (subtracts one from their
 * indices).
 *
 * @param index the index of the element to be removed
 * @return the element that was removed from the list
 * @throws IndexOutOfBoundsException {@inheritDoc}
 */
public E remove(int index) {
    rangeCheck(index);

    modCount++;
    E oldValue = elementData(index);

    int numMoved = size - index - 1;
    if (numMoved > 0)
        System.arraycopy(elementData, index+1, elementData, index,
                         numMoved);
    elementData[--size] = null; // clear to let GC do its work

    return oldValue;
}
```

删除操作比较消耗性能，因为会涉及到数据之间的排序、copy

三、查询获取 get(int index)

```
/**
 * Returns the element at the specified position in this list.
 *
 * @param  index index of the element to return
 * @return the element at the specified position in this list
 * @throws IndexOutOfBoundsException {@inheritDoc}
 */
public E get(int index) {
    rangeCheck(index);

    return elementData(index);
}
```

直接定位数据索引，效率较高。