链滴

数据结构 - 线性表: 数组

作者：kyrie92
原文链接：https://ld246.com/article/1667983814508
来源网站：链滴

数据结构-线性表：数组

一、概述

线性表是由N个元素组成的有序排列，也是最常见的一种数据结构，主要包括：**数组** 和 **链表**

二、数组结构

1. 描述

数组是一种存储单元连续，用来存储固定大小元素的线性表。Java中对应的集合实现如：ArrayList

2. 特征

- 一致性：数组只能保存相同数据类型元素，元素的数据类型可以是任何相同的数据类型。
- 有序性：数组中的元素是有序的，通过下标访问。
- 不可变性：数组一旦初始化，则长度（数组中元素的个数）不可变。

3. 分析

<此处使用 集合 ArrayList 来讲解数组>

- 添加元素源码分析

```
// ArrayList 的add方法
public boolean add(E e) {
    ensureCapacityInternal(size + 1);  // Increments modCount!!
    elementData[size++] = e;
    return true;
}
private void ensureCapacityInternal(int minCapacity) {
    ensureExplicitCapacity(calculateCapacity(elementData, minCapacity));
}
// 计算容量
private static int calculateCapacity(Object[] elementData, int minCapacity) {
    if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
        return Math.max(DEFAULT_CAPACITY, minCapacity);    // 若集合为空，则判度add后的容
是否超过默认容量10
    }
    return minCapacity;
}
private void ensureExplicitCapacity(int minCapacity) {
    modCount++;    // modCount主要用来记录集合被修改的次数，详细作用请参考源码介绍
    // overflow-conscious code
    if (minCapacity - elementData.length > 0)    // 判断是否需要扩容
        grow(minCapacity);
}
// 扩容动作 minCapacity：至少需要的容量
private void grow(int minCapacity) {
```

```
    // overflow-conscious code
    int oldCapacity = elementData.length;    // 原数组长度
    // 进行扩容，扩大为原来的1.5倍，那为什么不直接*1.5呢，因为位操作速度更快
    int newCapacity = oldCapacity + (oldCapacity >> 1);
    if (newCapacity - minCapacity < 0)       // 一次扩容后，检查扩容后的容量是否够用
        newCapacity = minCapacity;           // 若还是不够用，则容量直接为 上面计算后的最小容量 m
ncapacity
    if (newCapacity - MAX_ARRAY_SIZE > 0)    // 若扩容后的容量超过了最大值，则计算最大值容量
        newCapacity = hugeCapacity(minCapacity);
    // minCapacity is usually close to size, so this is a win:
    // 扩容
    elementData = Arrays.copyOf(elementData, newCapacity);
}
//扩容
public static <T> T[] copyOf(T[] original, int newLength) {
    return (T[]) copyOf(original, newLength, original.getClass());
}
// 因为ArrayList底层数据结构为数组，所以下面申请内存空间时，若没有足够的连续内存空间，将会
出OOM异常，即使内存足够
public static <T,U> T[] copyOf(U[] original, int newLength, Class<? extends T[]> newType) {
    @SuppressWarnings("unchecked")
    T[] copy = ((Object)newType == (Object)Object[].class)
        ? (T[]) new Object[newLength]
        : (T[]) Array.newInstance(newType.getComponentType(), newLength); // 申请内存空间，
没有连续的内存空间，则会抛出OOM
    // 将原数组拷贝到新的空间
    System.arraycopy(original, 0, copy, 0,
                     Math.min(original.length, newLength));
    return copy;
}
// 申请新的数组
public static native void arraycopy(Object src,  int  srcPos,
                                     Object dest, int destPos,
                                     int length);
```

添加元素总结：添加元素伴随着创建新数组，再将老的数组拷贝到新的数组中

- 删除元素源码分析

```
public E remove(int index) {    // 假设共有5个元素，现在移除第二个元素，即:index=1、size=5
    rangeCheck(index);    // 检查索引是否越界
    modCount++;
    E oldValue = elementData(index);
    int numMoved = size - index - 1; // numMoved=3
    if (numMoved > 0)    // 拷贝到新数组中
        System.arraycopy(elementData, index+1, elementData, index,
                         numMoved);
    elementData[--size] = null; // clear to let GC do its work 让GC来回收该部分资源
    return oldValue;
}
```

删除元素总结：删除元素伴随着创建新数组，再将老的数组拷贝到新的数组中

三、总结

数组结构基于索引下标进行查找，因此查找操作块，增删会判随着新数组的创建和数组的拷贝，增删慢。