## 数据结构与算法4-数组

**笔记本:** 我的笔记
**创建时间:** 2020/9/18 22:09 **更新时间:** 2020/10/3 10:05
**作者:** liuhouer
**标签:** 算法

---

**自己实现一个动态数组:**

```
public class Array<E> {
    private E[] data;
    private int size;
    // 构造函数，传入数组的容量capacity构造Array
    public Array(int capacity){
        data = (E[])new Object[capacity];
        size = 0;
    }
    // 无参数的构造函数，默认数组的容量capacity=10
    public Array(){
        this(10);
    }
    // 获取数组的容量
    public int getCapacity(){
        return data.length;
    }
    // 获取数组中的元素个数
    public int getSize(){
        return size;
    }
    // 返回数组是否为空
    public boolean isEmpty(){
        return size == 0;
    }
    // 在index索引的位置插入一个新元素e
    public void add(int index, E e){
        if(index < 0 || index > size)
            throw new IllegalArgumentException("Add failed. Require index >= 0
and index <= size.");
        if(size == data.length)
            resize(2 * data.length);
        for(int i = size - 1; i >= index ; i --)
            data[i + 1] = data[i];
        data[index] = e;
        size ++;
    }
    // 向所有元素后添加一个新元素
    public void addLast(E e){
        add(size, e);
    }
    // 在所有元素前添加一个新元素
    public void addFirst(E e){
        add(0, e);
    }
    // 获取index索引位置的元素
    public E get(int index){
        if(index < 0 || index >= size)
            throw new IllegalArgumentException("Get failed. Index is
illegal.");
        return data[index];
    }
    // 修改index索引位置的元素为e
    public void set(int index, E e){
        if(index < 0 || index >= size)
```

```
            throw new IllegalArgumentException("Set failed. Index is
illegal.");
        data[index] = e;
    }
    // 查找数组中是否有元素e
    public boolean contains(E e){
        for(int i = 0 ; i < size ; i ++){
            if(data[i].equals(e))
                return true;
        }
        return false;
    }
    // 查找数组中元素e所在的索引，如果不存在元素e，则返回-1
    public int find(E e){
        for(int i = 0 ; i < size ; i ++){
            if(data[i].equals(e))
                return i;
        }
        return -1;
    }
    // 从数组中删除index位置的元素，返回删除的元素
    public E remove(int index){
        if(index < 0 || index >= size)
            throw new IllegalArgumentException("Remove failed. Index is
illegal.");
        E ret = data[index];
        for(int i = index + 1 ; i < size ; i ++)
            data[i - 1] = data[i];
        size --;
        data[size] = null; // loitering objects != memory leak
        if(size == data.length / 4 && data.length / 2 != 0)
            resize(data.length / 2);
        return ret;
    }
    // 从数组中删除第一个元素，返回删除的元素
    public E removeFirst(){
        return remove(0);
    }
    // 从数组中删除最后一个元素，返回删除的元素
    public E removeLast(){
        return remove(size - 1);
    }
    // 从数组中删除元素e
    public void removeElement(E e){
        int index = find(e);
        if(index != -1)
            remove(index);
    }
    @Override
    public String toString(){
        StringBuilder res = new StringBuilder();
        res.append(String.format("Array: size = %d , capacity = %d\n", size,
data.length));
        res.append('[');
        for(int i = 0 ; i < size ; i ++){
            res.append(data[i]);
            if(i != size - 1)
                res.append(", ");
        }
        res.append(']');
        return res.toString();
    }
    // 将数组空间的容量变成newCapacity大小
    private void resize(int newCapacity){
        E[] newData = (E[])new Object[newCapacity];
        for(int i = 0 ; i < size ; i ++)
            newData[i] = data[i];
        data = newData;
    }
}
```

时间复杂度：

# 分析动态数组的时间复杂度

- 添加操作 O(n)

addLast(e) O(1)

addFirst(e) O(n)

add(index, e) O(n/2) = O(n)

} O(n) 最坏情况

resize O(n)

严格计算需要一些概率论知识

# 均摊复杂度 amortized time complexity

**resize O(n)**



addLast 的均摊复杂度为O(1)

同理，我们看removeLast操作，均摊复杂度也为O(1)

# 复杂度震荡

但是，当我们同时看addLast和removeLast操作：

capacity = n

addLast O(n)

removeLast O(n)

addLast O(n)

removeLast

# 复杂度震荡

出现问题的原因：removeLast 时 resize 过于着急（Eager）

解决方案：Lazy