浅谈ArrayList

几个重点

- 底层是数组，初始大小为10
- 插入时会判断数组容量是否足够，不够的话会进行扩容
- 所谓扩容就是新建一个新的数组，然后将老的数据里面的元素复制到新的数组里面
- 移除元素的时候也涉及到数组中元素的移动，删除指定index位置的元素，然后将index+1至数组最后一个元素往前移动一个格

CopyOnWriteArrayList

基本字段

- 内部持有一个ReentrantLock lock = new ReentrantLock();
- 底层是用volatile transient声明的数组 array

```
final transient ReentrantLock lock = new ReentrantLock();

private transient volatile Object[] array;
```

增删改查

增

```
public boolean add(E e) {
    final ReentrantLock lock = this.lock;
    //获得锁
    lock.lock();
    try {
        Object[] elements = getArray();
        int len = elements.length;
        //复制一个新的数组
        Object[] newElements = Arrays.copyOf(elements, len + 1);
```

```
            //插入新值
            newElements[len] = e;
            //将新的数组指向原来的引用
            setArray(newElements);
            return true;
        } finally {
            //释放锁
            lock.unlock();
        }
    }
```

删

```
public E remove(int index) {
    final ReentrantLock lock = this.lock;
    //获得锁
    lock.lock();
    try {
        Object[] elements = getArray();
        int len = elements.length;
        E oldValue = get(elements, index);
        int numMoved = len - index - 1;
        if (numMoved == 0)
            //如果删除的元素是最后一个，直接复制该元素前的所有元素到新的数组
            setArray(Arrays.copyOf(elements, len - 1));
        else {
            //创建新的数组
            Object[] newElements = new Object[len - 1];
            //将index+1至最后一个元素向前移动一格
            System.arraycopy(elements, 0, newElements, 0, index);
            System.arraycopy(elements, index + 1, newElements, index,
                             numMoved);
            setArray(newElements);
        }
        return oldValue;
    } finally {
        lock.unlock();
    }
}
```

改

```
public E set(int index, E element) {
    final ReentrantLock lock = this.lock;
    //获得锁
    lock.lock();
    try {
        Object[] elements = getArray();
        E oldValue = get(elements, index);

        if (oldValue != element) {
            int len = elements.length;
            //创建新数组
            Object[] newElements = Arrays.copyOf(elements, len);
            //替换元素
```

```
            newElements[index] = element;
            //将新数组指向原来的引用
            setArray(newElements);
        } else {
            // Not quite a no-op; ensures volatile write semantics
            setArray(elements);
        }
        return oldValue;
    } finally {
        //释放锁
        lock.unlock();
    }
}
```

查

```
//直接获取index对应的元素
public E get(int index) {return get(getArray(), index);}
private E get(Object[] a, int index) {return (E) a[index];}
```

总结

增删改都需要获得锁，并且锁只有一把，而读操作不需要获得锁，支持并发。为什么增删改中都需要创建一个新的数组，操作完成之后再赋给
能获取到元素，如果在增删改过程直接修改原来的数组，可能会造成执行读操作获取不到数据

扩展：为什么效率比同样是数组的Vector性能要好

Vector是增删改查方法都加了synchronized，保证同步，但是每个方法执行的时候都要去获得锁，性能就会大大下降，而CopyOnWriteArra
加锁，在读方面的性能就好于Vector，CopyOnWriteArrayList支持读多写少的并发情况