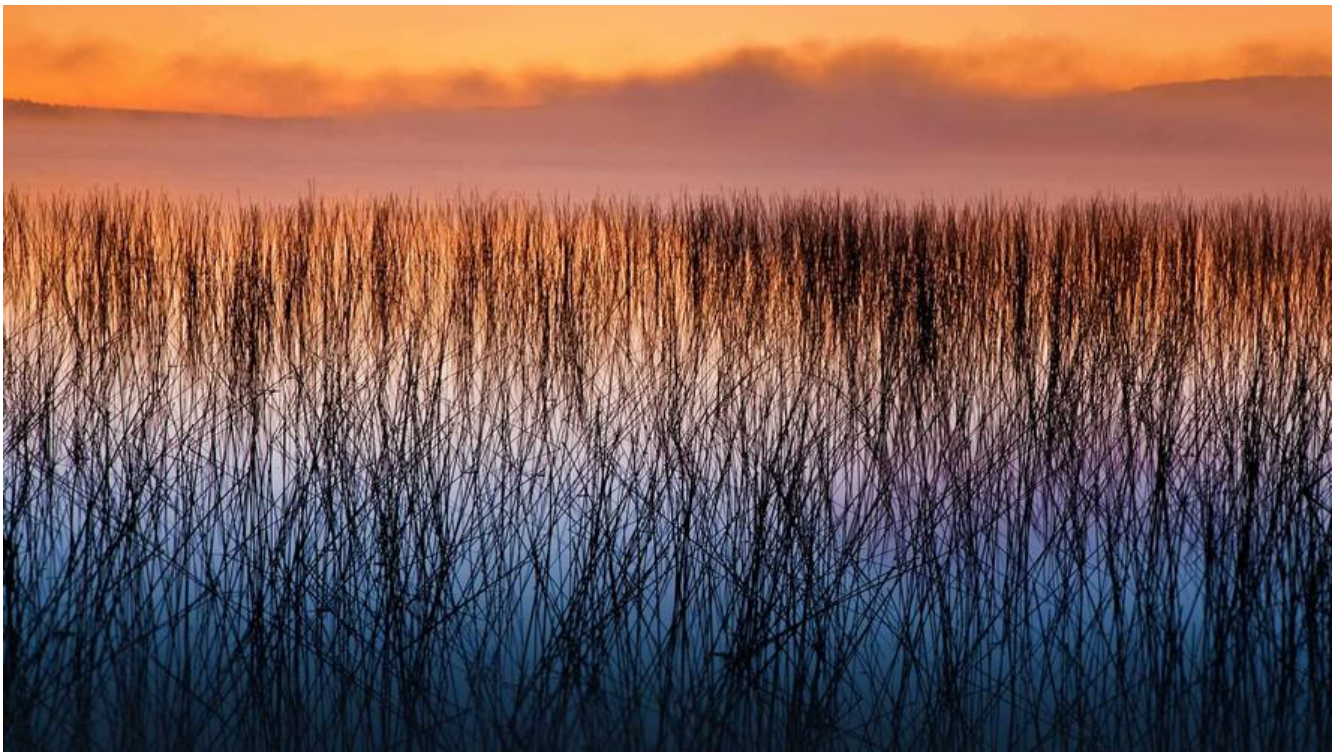
链滴

# ArrayList 源码分析

作者：Pleuvoir

原文链接：https://ld246.com/article/1522497288923

来源网站：链滴



# 一、概览

容器主要包括 Collection 和 Map 两种，Collection 又包含了 List、Set 以及 Queue。

# Collection

## 1. Set

- HashSet：基于哈希实现，支持快速查找，但不支持有序性操作，例如根据一个范围查找元素的操。并且失去了元素的插入顺序信息，也就是说使用 Iterator 遍历 HashSet 得到的结果是不确定的。

- TreeSet：基于红黑树实现，支持有序性操作，但是查找效率不如 HashSet，HashSet 查找时间复度为 O(1)，TreeSet 则为 O(logn)；

- LinkedHashSet：具有 HashSet 的查找效率，且内部使用链表维护元素的插入顺序。

## 2. List

- ArrayList：基于动态数组实现，支持随机访问；

- Vector：和 ArrayList 类似，但它是线程安全的；

- LinkedList：基于双向循环链表实现，只能顺序访问，但是可以快速地在链表中间插入和删除元素不仅如此，LinkedList 还可以用作栈、队列和双端队列。

## 3. Queue

- LinkedList：可以用它来支持双向队列；

- PriorityQueue 是基于堆结构实现，可以用它来实现优先级队列。

# Map

- HashMap：基于哈希实现；

- HashTable：和 HashMap 类似，但它是线程安全的，这意味着同一时刻多个线程可以同时写入 HashTable 并且不会导致数据不一致。它是遗留类，不应该去使用它。现在可以使用 ConcurrentHashMap 来支持线程安全，并且 ConcurrentHashMap 的效率会更高，因为 ConcurrentHashMap 引入了段锁。

- LinkedHashMap：使用链表来维护元素的顺序，顺序为插入顺序或者最近最少使用（LRU）顺序。

- TreeMap：基于红黑树实现。

# 二、源码分析

# ArrayList

## 1. 概览

实现了 RandomAccess 接口，因此支持随机访问，这是理所当然的，因为 ArrayList 是基于数组实的。

```
public class ArrayList extends AbstractList
        implements List, RandomAccess, Cloneable, java.io.Serializable
```

基于数组实现，保存元素的数组使用 transient 修饰，该关键字声明数组默认不会被序列化。这是 ArrayList 具有动态扩容特性，因此保存元素的数组不一定都会被使用，那么就没必要全部进行序列化。ArrayList 重写了 writeObject() 和 readObject() 来控制只序列化数组中有元素填充那部分内容。

```
transient Object\[\] elementData; // non-private to simplify nested class access
```

数组的默认大小为 10。

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

删除元素时需要调用 System.arraycopy() 对元素进行复制，因此删除操作成本很高。

```
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;
}
```

添加元素时使用 ensureCapacity() 方法来保证容量足够，如果不够时，需要使用 grow() 方法进行扩，使得新容量为旧容量的 1.5 倍（oldCapacity + (oldCapacity >> 1))。扩容操作需要把原数组整个制到新数组中，因此最好在创建 ArrayList 对象时就指定大概的容量大小，减少扩容操作的次数。

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

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

private void grow(int minCapacity) {
    // overflow-conscious code
    int oldCapacity = elementData.length;
    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);
}
```

## 2. Fail-Fast

modCount 用来记录 ArrayList 结构发生变化的次数。结构发生变化是指添加或者删除至少一个元素所有操作，或者是调整内部数组的大小，仅仅只是设置元素的值不算结构发生变化。

在进行序列化或者迭代等操作时，需要比较操作前后 modCount 是否改变，如果改变了需要抛出 CocurrentModificationException。

```
private void writeObject(java.io.ObjectOutputStream s)
    throws java.io.IOException{
    // Write out element count, and any hidden stuff
    int expectedModCount = modCount;
    s.defaultWriteObject();

    // Write out size as capacity for behavioural compatibility with clone()
    s.writeInt(size);

    // Write out all elements in the proper order.
    for (int i=0; i()); 返回一个线程安全的 ArrayList，也可以使用 concurrent 并发包下的 CopyOnWrieArrayList 类;
```

## 4. 和 LinkedList 的区别

- ArrayList 基于动态数组实现，LinkedList 基于双向循环链表实现；
- ArrayList 支持随机访问，LinkedList 不支持；
- LinkedList 在任意位置添加删除元素更快。