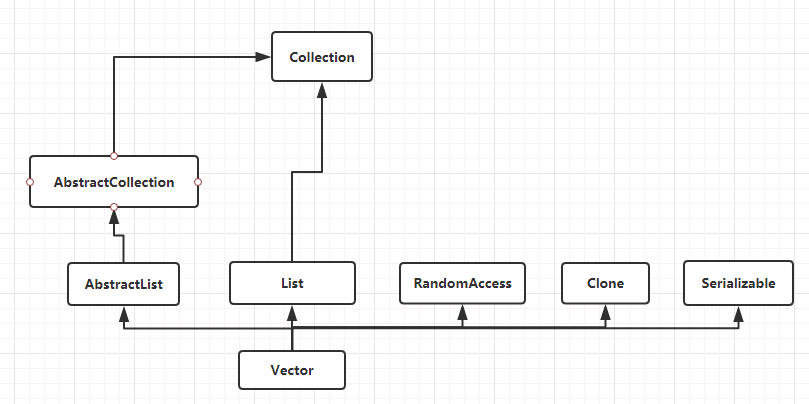
# Vector

## 简介

Vector是java中可以实现自动增长的对象数组，实现了动态数据的功能，他的继承和实现结果和ArrayList是相同的，ArrayList中的功能的实现方式基本相同，不过在涉及到线程安全的方法中使用了关键字synchronized来保证数据的安全，这也是和ArrayList的主要不同之处



## 成员变量

**protected** Object[] elementData;存放对象的数组

**protected** **int** elementCount;数组中存放对象的个数，相当于ArrayList中的size

**protected** **int** capacityIncrement;扩容大小

modCount 记录添加删除次数

## 构造函数

**public** Vector() {无参构造函数，内部调用参数为int的构造函数，参数表示数组的初始化长度，参数值为10，所以数组的初始长度为10

**this**(10);

}

**public** Vector(**int** initialCapacity) {内部还是调用自身的重载构造函数，这个构造函数多了一个扩容参数，在数组扩容的时候使用，默认为0

**this**(initialCapacity, 0);

}

**public** Vector(**int** initialCapacity, **int** capacityIncrement) {

**super**();

**if** (initialCapacity < 0)

**throw** **new** IllegalArgumentException("Illegal Capacity: "+

initialCapacity);

**this**.elementData = **new** Object[initialCapacity];创建数组

**this**.capacityIncrement = capacityIncrement;设置自动扩容大小

}

**public** Vector(Collection<? **extends** E> c) {

参数为集合是的构造函数，将集合转换为数组赋值给elementData即可，

elementData = c.toArray();

elementCount = elementData.length;

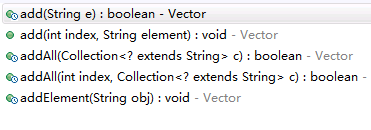
// c.toArray might (incorrectly) not return Object[] (see 6260652)

**if** (elementData.getClass() != Object[].**class**)

elementData = Arrays.*copyOf*(elementData, elementCount, Object[].**class**);

}

## 添加操作



### add(e)

add(e);在数组下标的size为添加元素

**public** **synchronized** **boolean** add(E e) {

modCount++;

ensureCapacityHelper(elementCount + 1);决定是否扩容

elementData[elementCount++] = e;

**return** **true**;

}

ensureCapacityHelper方法，参数表示数组的最小容量

private void ensureCapacityHelper(int minCapacity) {

// overflow-conscious code

if (minCapacity - elementData.length > 0)当最小容量大于数组长度时扩容

grow(minCapacity);

}

**private** **void** grow(**int** minCapacity) {

// overflow-conscious code

**int** oldCapacity = elementData.length;

**int** newCapacity = oldCapacity + ((capacityIncrement > 0) ?

capacityIncrement : oldCapacity);

**if** (newCapacity - minCapacity < 0)

newCapacity = minCapacity;

设置扩容多少，如果初始化的扩容值capacityIncrement不大于0，那么就新长度就是原长度的2被，

如果初始化扩容值大于0 ，那么新长度就是原长度加上初始化的扩容值

如果扩容后的数组容量小于当前添加对象后的对象总数了，那么扩容的大小就是添加对象后的对象总数

如果扩容值超过的*MAX\_ARRAY\_SIZE*，那么数组的最大长度就是Integer.MAX\_VALUE

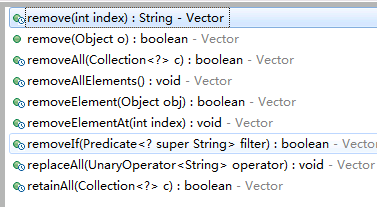
**if** (newCapacity - *MAX\_ARRAY\_SIZE* > 0)

newCapacity = *hugeCapacity*(minCapacity);

elementData = Arrays.*copyOf*(elementData, newCapacity);

}

Arrays.*copyOf就是根据原始数组类型创建一个新数组，数组长度就是扩容后的长度*



## 删除

删除和ArrayList的实现思路基本一致

删除指定对象，内部调用方法removeElement

**public** **boolean** remove(Object o) {

**return** removeElement(o);

}

**public** **synchronized** **boolean** removeElement(Object obj) {

modCount++;

**int** i = indexOf(obj);

**if** (i >= 0) {

removeElementAt(i);

**return** **true**;

}

**return** **false**;

}

通过index方法找到对象所在位置下标，如果没有找到返回-1，找到相等对象就返回下标

removeElementAt实现数组的删除

**public** **synchronized** **void** removeElementAt(**int** index) {

modCount++;

**if** (index >= elementCount) {

**throw** **new** ArrayIndexOutOfBoundsException(index + " >= " +elementCount);

}

**else** **if** (index < 0) {

**throw** **new** ArrayIndexOutOfBoundsException(index);

}

**int** j = elementCount - index - 1;计算出删除位置之后的对象的个数

**if** (j > 0) {当删除对象位置之后的对象个数大于0，那么删除不在数组末尾，将删除位置之后的对象全部向前移动一位

System.*arraycopy*(elementData, index + 1, elementData, index, j);

}

elementCount--;

elementData[elementCount] = **null**; /\* to let gc do its work \*/将最后一位设置为null

}

其他删除方法略过

## 修改

set方法，替换指定位置的对象，如果参数index不在当前数组对象所在的范围内抛出异常

**public** **synchronized** E set(**int** index, E element) {

**if** (index >= elementCount)

**throw** **new** ArrayIndexOutOfBoundsException(index);

E oldValue = elementData(index);

elementData[index] = element;

**return** oldValue;

}

Vector的setSize(index)方法,手动设置数组长度，数组实际容量小于设置的值，扩容；

数组的实际容量大于设置的值，那么就样从index位置之后的节点都设置为null

**public** **synchronized** **void** setSize(**int** newSize) {

modCount++;

**if** (newSize > elementCount) {

ensureCapacityHelper(newSize);

} **else** {

**for** (**int** i = newSize ; i < elementCount ; i++) {

elementData[i] = **null**;

}

}

elementCount = newSize;

}

## 迭代器

Vector内部迭代器类和ArrayList是一样的，只是在操作时增加了同步管理，

## ArrayList和Vector对比

扩容时，

ArrayList当一次（添加元素的个数+原数组的容量）不超过当前数组的1.5倍时，扩容1.5倍；如果

（添加元素的个数+原数组的容量）超过了当前数组的1.5倍，扩容为（添加元素的个数+原数组的容量）之和；

当扩容后的容量超过了最大容量值（*MAX\_ARRAY\_SIZE*）时，扩容容量就是*Integer.MAX\_VALUE* int的最大值；

Vector有扩容值capacityIncrement，如果初始化的扩容值capacityIncrement不大于0，那么就新长度就是原长度的2被，如果初始化扩容值大于0 ，那么新长度就是原长度加上初始化的扩容值

如果扩容后的数组容量小于当前添加对象后的数组总数容量，那么扩容的大小就是添加对象后的数组总容量

如果扩容值超过的*MAX\_ARRAY\_SIZE*，那么数组的最大长度就是Integer.MAX\_VALUE

相比于ArrayList,Vector对数组的操作是线程安全的