Vector 源码解析 | MrBird

> Vector 和 ArrayList 非常相似，它们都实现了相同的接口，继承相同的类，就连方法的实现也非常类似。

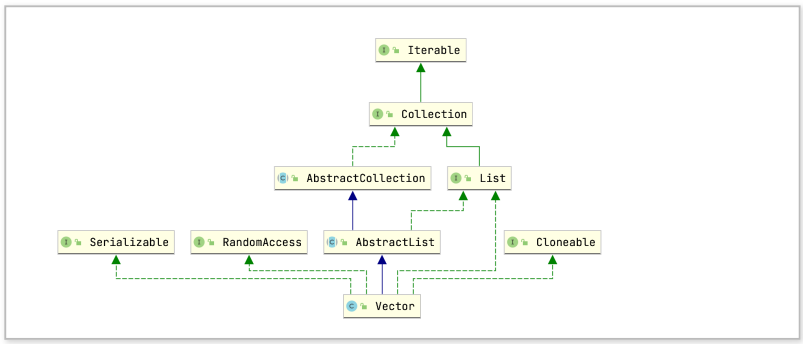
Vector 源码解析

2020-08-10 | Visit count 1058053

Vector 和 ArrayList 非常相似，它们都实现了相同的接口，继承相同的类，就连方法的实现也非常类似。和 ArrayList 不同的是，Vector 是线程安全的，关键方法上都加了 synchronized 同步锁，由于 Vector 效率不高，所以使用的较少，要使用线程安全的 ArrayList，推荐 CopyOnWriteArrayList，后续再做分析，这里仅记录下 Vector 源码，基于 JDK1.8。

类结构

Vector 的类关系图和 ArrayList 一致：



Vector 可以存放任意类型元素（包括 null），允许重复，和 ArrayList 一致，内部采用 Object 类型数组存放数据，包含以下三个成员变量：

```
// Object数组，存放数据
protected Object[] elementData;

// 元素个数
protected int elementCount;

// 当数组容量不足时，容量增加capacityIncrement，如果capac
protected int capacityIncrement;
```

方法解析

构造函数

```
public Vector(int initialCapacity, int capacityIncreme
    super();
    if (initialCapacity < 0)
        throw new IllegalArgumentException("Illegal Ca
                                           initialCapa
    this.elementData = new Object[initialCapacity];
    this.capacityIncrement = capacityIncrement;
}

public Vector(int initialCapacity) {
    this(initialCapacity, 0);
}

public Vector() {
    this(10);
}
```

可以看到，当我们调用 *new Vector()* 创建 Vector 集合时，直接创建了一个容量为 10 的 Object 数组（和 ArrayList 不同，ArrayList 内部数组初始容量为 0，只有在添加第一个元素的时候才扩容为 10），并且 capacityIncrement 为 0，意味着容量不足时，新数组容量为旧数组容量的 2 倍。

add(E e)

```
public synchronized boolean add(E e) {
    modCount++;
    ensureCapacityHelper(elementCount + 1);
    elementData[elementCount++] = e;
    return true;
}

private void ensureCapacityHelper(int minCapacity) {
    // overflow-conscious code
```

```
    if (minCapacity - elementData.length > 0)
        grow(minCapacity);
}

private void grow(int minCapacity) {
    // overflow-conscious code
    int oldCapacity = elementData.length;
    // capacityIncrement为0的话，新容量为旧容量的2倍，不为
    int newCapacity = oldCapacity + ((capacityIncrement
                                      capacityIncrement
    if (newCapacity - minCapacity < 0)
        newCapacity = minCapacity;
    if (newCapacity - MAX_ARRAY_SIZE > 0)
        newCapacity = hugeCapacity(minCapacity);
    elementData = Arrays.copyOf(elementData, newCapacit
}

private static int hugeCapacity(int minCapacity) {
    if (minCapacity < 0) // overflow
        throw new OutOfMemoryError();
    return (minCapacity > MAX_ARRAY_SIZE) ?
        Integer.MAX_VALUE :
        MAX_ARRAY_SIZE;
}
```

添加逻辑和 ArrayList 的 add 方法大体一致，区别在于扩容策略有些不同，并且方法使用 synchronized 关键字修饰。

set(int index, E element)

```
public synchronized E set(int index, E element) {
    if (index >= elementCount)
        throw new ArrayIndexOutOfBoundsException(index

    E oldValue = elementData(index);
    elementData[index] = element;
    return oldValue;
}
```

逻辑和 ArrayList 的 set 方法一致，方法使用 synchronized 关键字修饰。

get(int index)

```
public synchronized E get(int index) {
    if (index >= elementCount)
        throw new ArrayIndexOutOfBoundsException(index

    return elementData(index);
}
E elementData(int index) {
```

```
        return (E) elementData[index];
    }
```

逻辑和 ArrayList 的 get 方法一致，方法使用 synchronized 关键字修饰。

remove(int index)

```
    public synchronized E remove(int index) {
        modCount++;
        if (index >= elementCount)
            throw new ArrayIndexOutOfBoundsException(index)
        E oldValue = elementData(index);

        int numMoved = elementCount - index - 1;
        if (numMoved > 0)
            System.arraycopy(elementData, index+1, elementD
                             numMoved);
        elementData[--elementCount] = null; // Let gc do it

        return oldValue;
    }
```

逻辑和 ArrayList 的 remove 方法一致，方法使用 synchronized 关键字修饰。

trimToSize()

```
    public synchronized void trimToSize() {
        modCount++;
        int oldCapacity = elementData.length;
        if (elementCount < oldCapacity) {
            elementData = Arrays.copyOf(elementData, elemen
        }
    }
```

逻辑和 ArrayList 的 trimToSize 方法一致，方法使用 synchronized 关键字修饰。

> 剩下的方法源码自己查看，大体和 ArrayList 没有什么区别。Vector 的方法都用 synchronized 关键字来确保线程安全，每次只有一个线程能访问此对象，在线程竞争激烈的情况下，这种方法效率非常低，所以实际并不推荐使用 Vector。

全文完