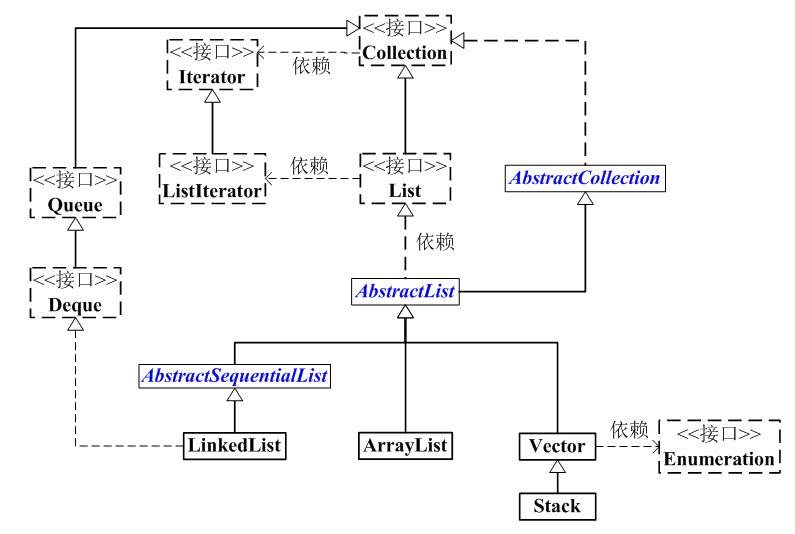
第1部分 List概括

先回顾一下List的框架图

[](https://images0.cnblogs.com/blog/497634/201309/08214705-fbbcdd2d785f40f5b773041801bd74f1.jpg)

(01) List 是一个接口，它继承于Collection的接口。它代表着有序的队列。  
(02) AbstractList 是一个抽象类，它继承于AbstractCollection。AbstractList实现List接口中除size()、get(int location)之外的函数。  
(03) AbstractSequentialList 是一个抽象类，它继承于AbstractList。AbstractSequentialList 实现了“链表中，根据index索引值操作链表的全部函数”。

(04) ArrayList, LinkedList, Vector, Stack是List的4个实现类。  
　　ArrayList 是一个**数组队列，相当于动态数组**。它由数组实现，随机访问效率高，随机插入、随机删除效率低。  
　　LinkedList 是一个**双向链表**。它也可以被当作堆栈、队列或双端队列进行操作。LinkedList随机访问效率低，但随机插入、随机删除效率低。  
　　Vector 是**矢量队列，和ArrayList一样，它也是一个动态数组，由数组实现**。但是ArrayList是非线程安全的，而Vector是线程安全的。  
　　Stack 是**栈，它继承于Vector**。它的特性是：先进后出(FILO, First In Last Out)。

第2部分 List使用场景

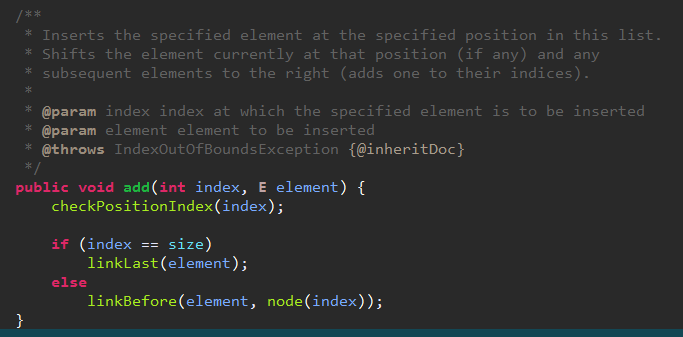
学东西的最终目的是为了***能够理解、使用它***。下面**先概括的说明一下各个List的使用场景**，**后面再分析原因**。

**如果涉及到“栈”、“队列”、“链表”等操作，应该考虑用List，具体的选择哪个List，根据下面的标准来取舍。**  
(01) 对于需要快速插入，删除元素，应该使用LinkedList。  
(02) 对于需要快速随机访问元素，应该使用ArrayList。  
(03) 对于“**单线程环境**” 或者 “**多线程环境，但List仅仅只会被单个线程操作**”，此时应该使用非同步的类(如ArrayList)。  
       对于“**多线程环境**，且List可能同时被多个线程操作”，此时，应该使用同步的类(如Vector)。

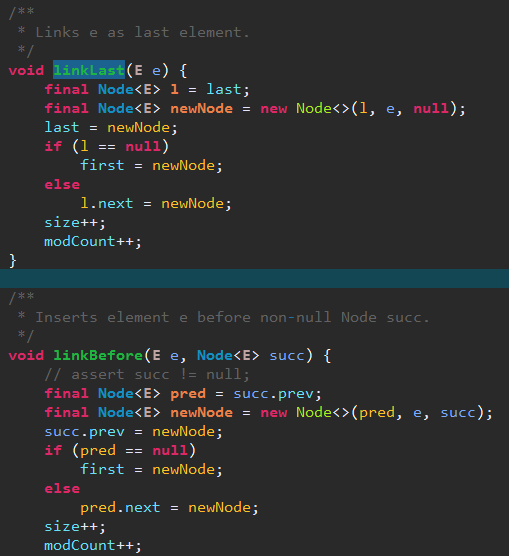
考虑到Vector是支持同步的，而Stack又是继承于Vector的；因此，得出结论：  
**(01) 对于需要快速插入，删除元素，应该使用LinkedList。**  
**(02) 对于需要快速随机访问元素，应该使用ArrayList。**  
**(03) 对于“单线程环境” 或者 “多线程环境，但List仅仅只会被单个线程操作”，此时应该使用非同步的类。**

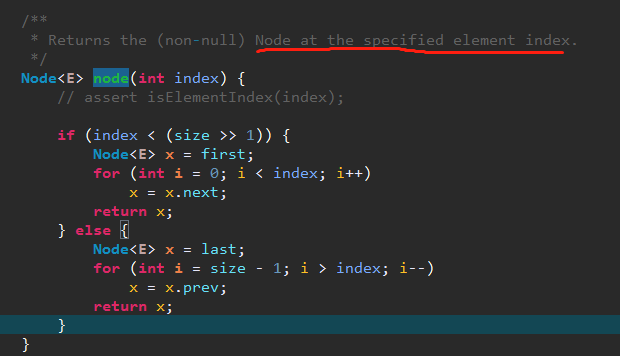
第3部分 LinkedList和ArrayList性能差异分析

下面我们看看**为什么LinkedList中插入元素很快，而ArrayList中插入元素很慢**！

**LinkedList.java中向指定位置插入元素的代码如下**：

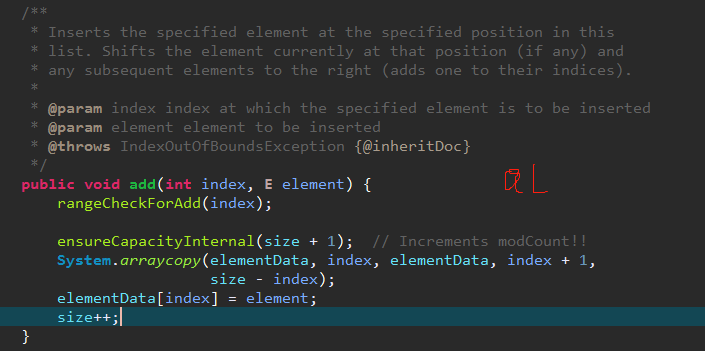
从中，我们可以看出：通过add(int index, E element)向LinkedList插入元素时。先是**在双向链表中找到要插入节点的位置index**；找到之后，**再插入一个新节点**。  
双向链表查找index位置的节点时，有一个**加速动作**：*若index < 双向链表长度的1/2，则从前向后查找; 否则，从后向前查找*。





**接着，我们看看ArrayList.java中向指定位置插入元素的代码。**如下：

// 将e添加到ArrayList的指定位置



ensureCapacityinternal(size+1) 的作用是“**确认ArrayList的容量，若容量不够，则增加容量。**”  
真正耗时的操作是 System.arraycopy(elementData, index, elementData, index + 1, size - index);

Sun JDK包的java/lang/System.java中的arraycopy()声明如下：

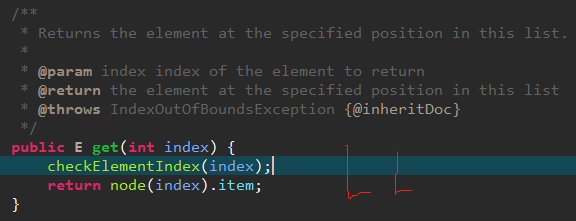
public static native void arraycopy(Object src, int srcPos, Object dest, int destPos, int length);

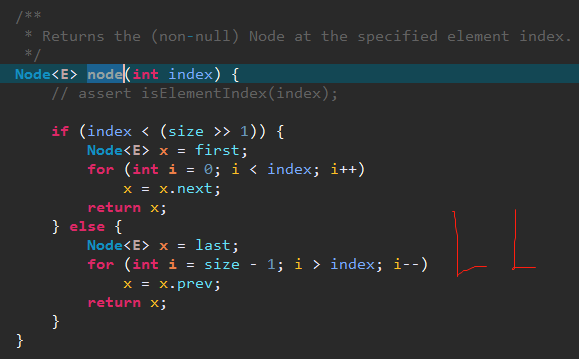
arraycopy()是个JNI函数，它是在JVM中实现的。sunJDK中看不到源码，不过可以在OpenJDK包中看到的源码。  
实际上，我们只需要了解：*System.arraycopy(elementData, index, elementData, index + 1, size - index);****会移动index之后所有元素即可****。*这就意味着，ArrayList的add(int index, E element)函数，会引起index之后所有元素的改变！

**通过上面的分析，我们就能理解为什么LinkedList中插入元素很快，而ArrayList中插入元素很慢。**  
**“删除元素”与“插入元素”的原理类似，这里就不再过多说明。**

接下来，我们看看**“为什么LinkedList中随机访问很慢，而ArrayList中随机访问很快”**。

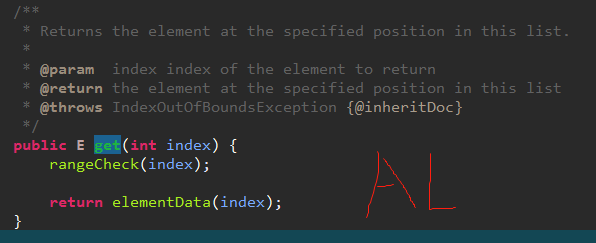
先看看**LinkedList随机访问的代码** ：





从中，我们可以看出：通过get(int index)**获取LinkedList第index个元素时**。先是在双向链表中找到要index位置的元素；找到之后再返回。  
双向链表查找index位置的节点时，有一个**加速动作**：若index < 双向链表长度的1/2，则从前向后查找; 否则，从后向前查找。

下面看看**ArrayList随机访问的代码** ：



从中，我们可以看出：通过get(int index)获取ArrayList第index个元素时。直接返回数组中index位置的元素，而不需要像LinkedList一样进行查找。

第4部分 Vector和ArrayList比较

相同之处

**1 它们都是List**

它们都继承于AbstractList，并且实现List接口。  
ArrayList和Vector的类定义如下：

[复制代码](javascript:void(0);)

// ArrayList的定义

public class ArrayList<E> extends AbstractList<E>

implements List<E>, RandomAccess, Cloneable, java.io.Serializable

// Vector的定义

public class Vector<E> extends AbstractList<E>

implements List<E>, RandomAccess, Cloneable, java.io.Serializable {}

[复制代码](javascript:void(0);)

**2 它们都实现了RandomAccess和Cloneable接口**

   实现RandomAccess接口，意味着它们都支持快速随机访问；  
   实现Cloneable接口，意味着它们能克隆自己。

**3 它们都是通过数组实现的，本质上都是动态数组**

ArrayList.java中定义数组elementData用于保存元素

// 保存ArrayList中数据的数组

private transient Object[] elementData;

Vector.java中也定义了数组elementData用于保存元素

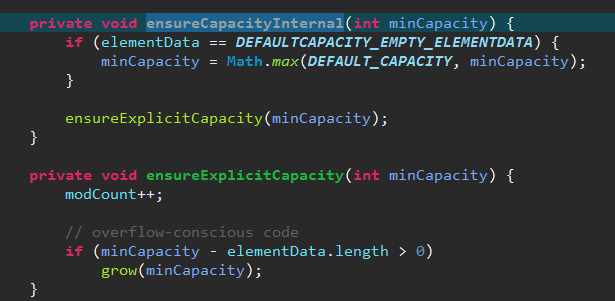
// 保存Vector中数据的数组

protected Object[] elementData;

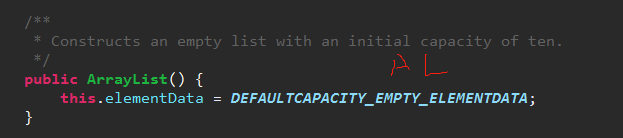
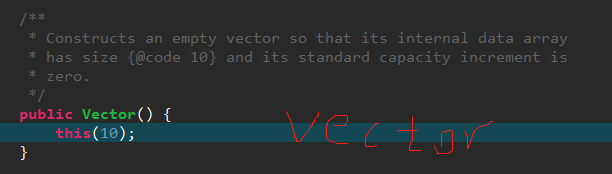
**4 它们的默认数组容量是10**

   若创建ArrayList或Vector时，没指定容量大小；则使用默认容量大小10。

附注：ArrayList的10 是在首次对数组进行操作时才会产生的；代码如下：



ArrayList的默认构造函数如下：

Vector的默认构造函数如下： 

**5 它们都支持Iterator和listIterator遍历**

   它们都继承于AbstractList，而AbstractList中分别实现了 “iterator()接口返回Iterator迭代器” 和 “listIterator()返回ListIterator迭代器”。

不同之处

**1 线程安全性不一样**

   ArrayList是非线程安全；  
   而Vector是线程安全的，它的函数都是synchronized的，即都是支持同步的。  
   ArrayList适用于单线程，Vector适用于多线程。

**3 构造函数个数不同**  
   ArrayList有3个构造函数，而Vector有4个构造函数。Vector除了包括和ArrayList类似的3个构造函数之外，另外的一个构造函数可以指定容量增加系数。

**ArrayList的构造函数如下**：

[复制代码](javascript:void(0);)

// 默认构造函数

ArrayList()

// capacity是ArrayList的默认容量大小。当由于增加数据导致容量不足时，容量会添加上一次容量大小的一半。

ArrayList(int capacity)

// 创建一个包含collection的ArrayList

ArrayList(Collection<? extends E> collection)

[复制代码](javascript:void(0);)

**Vector的构造函数如下**：

[复制代码](javascript:void(0);)

// 默认构造函数

Vector()

// capacity是Vector的默认容量大小。当由于增加数据导致容量增加时，每次容量会增加一倍。

Vector(int capacity)

// 创建一个包含collection的Vector

Vector(Collection<? extends E> collection)

// capacity是Vector的默认容量大小，capacityIncrement是每次Vector容量增加时的增量值。

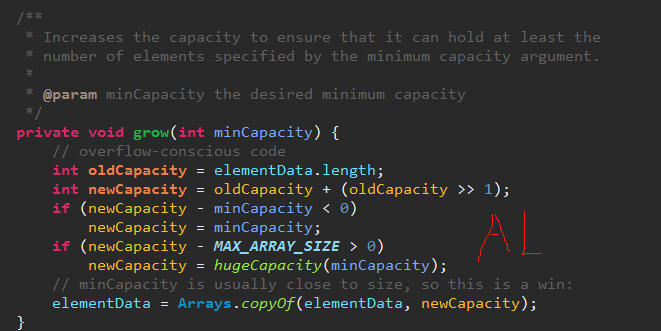
Vector(int capacity, int capacityIncrement)

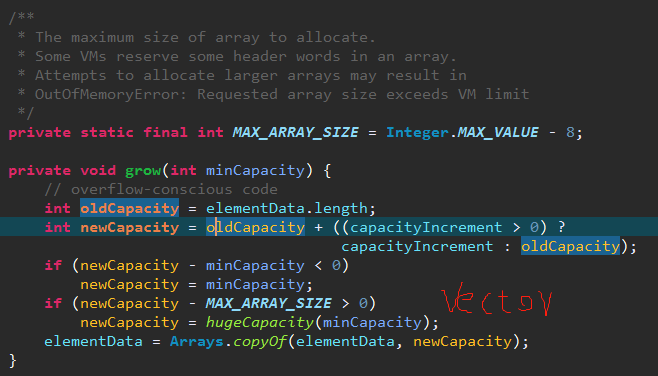
[复制代码](javascript:void(0);)

***4 容量增加方式不同***

   逐个添加元素时，若ArrayList容量不足时，“新的容量”=“(原始容量x3)/2”。  
   而Vector的容量增长与“增长系数有关”，若指定了“增长系数”，且“增长系数有效(即，大于0)”；那么，每次容量不足时，“新的容量”=“原始容量+增长系数”。若增长系数无效(即，小于/等于0)，则“新的容量”=“原始容量 x 2”。

ArrayList中容量增长的主要函数如下：

Vector中容量增长的主要函数如下：



**5 对Enumeration的支持不同。Vector支持通过Enumeration去遍历，而List不支持**

Vector中实现Enumeration的代码如下：

[复制代码](javascript:void(0);)

public Enumeration<E> elements() {

// 通过匿名类实现Enumeration

return new Enumeration<E>() {

int count = 0;

// 是否存在下一个元素

public boolean hasMoreElements() {

return count < elementCount;

}

// 获取下一个元素

public E nextElement() {

synchronized (Vector.this) {

if (count < elementCount) {

return (E)elementData[count++];

}

}

throw new NoSuchElementException("Vector Enumeration");

}

};

}