List 源码 600行。

话术：List源码比较简单，里面就是定义了一些方法

读取(get)，插入(add)，删除(remove)，修改(set)

也可批量增加(addAll)，删除(removeAll,retainAll)

获取(subList)。  
还有一些判定操作：

包含(contains[All]),相等(equals),索引(indexOf,lastIndexOf),大小(size)。  
还有获取元素类型数组的操作：toArray()

ArrayList 源码 1172行。

**ArrayList**是基于数组实现的，是一个动态数组（长度可变的数据结构，允许null当元素，默认长度是10，一次增长0.5倍，其容量为原来的1.5倍。

）

**ArrayList提供了三种方式的构造器**：

1、可以构造一个默认初始容量为10的空列表

2、构造一个指定初始容量。

3、构造一个包含指定collection的元素的列表

// **ArrayList**带容量大小的构造函数。

public **ArrayList**(int initialCapacity) {

super();

if (initialCapacity < 0)

throw new IllegalArgumentException("Illegal Capacity: "+ initialCapacity);

// 新建一个指定容量的数组</span>

this.elementData = new Object[initialCapacity];

}

// **ArrayList**无参构造函数。默认容量是10。

public **ArrayList**() {

super();

this.elementData = EMPTY\_ELEMENTDATA;

}

//构造方法传入一个Collection， 则将Collection里面的值copy到**arrayList**

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

elementData = c.toArray();

size = elementData.length;

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

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

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

}

### 调整数组容量：

 ensureCapacity(int minCapacity) ： 从上面介绍的向**ArrayList**中 存储元素的代码中，我们看到， 每当向数组中添加元素时，都要去检查添加后元素的个数是否会超出当前数组的长度，如果超出，数组将会进行扩容，以满足添加数 据的需求。数组扩容通过一个公开的方法ensureCapacity(int minCapacity)来实现。在实际添加大量元素前，我也可以使用ensureCapacity来手动增加**ArrayList**实例的容量，以减少递增式再分配的数量。另外扩容一次其容量为原来的1.5倍

public void ensureCapacity(int minCapacity) {

if (minCapacity > 0)

ensureCapacityInternal(minCapacity);

}

问题：

数组进行扩容时，会将老数组中的元素重新拷贝一份到新的数组中，这种操作的代价是很高的，因此在实际使用时，我们应该尽量避免数组容量的扩张。所以在初始化**ArrayList**的时候尽量预算下大致的容量需求，降低频繁调整容量的开销。

void  trimToSize() ： 将底层数组的容量调整为当前列表保存的实际元素的大小的功能。

由于elementData的长度会被拓展，size标记的是其中包含的元素的个数。所以会出现size很小但 elementData.length很大的情况，将出现空间的浪费。trimToSize将返回一个新的数组给elementData，元素内容保持不 变，length**和**size相同，节省空间。

public void trimToSize() {

modCount++;

int oldCapacity = elementData.length;

if (size < oldCapacity) {

elementData = Arrays.copyOf(elementData, size);

}

}

补充方法：

**System.arraycopy**方法：如果是数组比较大，那么使用System.arraycopy会比较有优势，因为其使用的是内存复制，省去了大量的数组寻址访问等时间

1. 源码：public static native void arraycopy(Object src,  int  srcPos,
2. Object dest, int destPos,
3. int length);

复制指定源数组src到目标数组dest。复制从src的srcPos索引开始，复制的个数是length，复制到dest的索引从destPos开始。

Arrays.*copyOf*()

1. //基本数据类型（其他类似byte，short···）
2. public static int[] copyOf(int[] original, int newLength) {
3. int[] copy = new int[newLength];
4. System.arraycopy(original, 0, copy, 0,
5. Math.min(original.length, newLength));
6. return copy;
7. }

观察其源代码发现**copyOf()**，在其内部创建了一个新的数组，然后调用arrayCopy()向其复制内容，返回出去。  
总结：  
1.copyOf()的实现是用的是arrayCopy();  
2.arrayCopy()需要目标数组，对两个数组的内容进行可能不完全的合并操作。  
3.copyOf()在内部新建一个数组，调用arrayCopy()将original内容复制到copy中去，并且长度为newLength。返回copy;  
4.arraycopy 方法会因为新数组大小比旧数组大小 小而报IndexOutOfBoundsException;  
 copyOf 则不会因此报错，因为copyOf 的返回值是在内部new 好的copy 数组，而该copy 数组new 的大小就等于newLength ，  
故即使在客户端指定好了新数组newArray 的大小，接收到返回值后也是指向底层new 出来的数组copy 。换句话说( 也可以因此推出其他的区别) ，在客户端代码中即使不给新数组new 对象，如：String[] newStr = null;那么对于arraycopy 是会报NullPointerException 的错误的，而对于java.util.Arrays 中的copyOf 方法则由于jdk 底层已经new 出了对象而不会报该错误！不过需要特别注意的是：copyOf 方法最后也是调用System.arraycopy 的方法，不过由于前面的准备，异常情况就不会出现了。

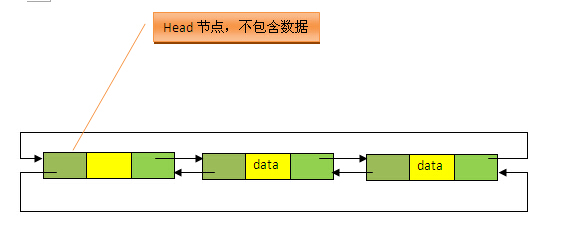
LinkedList源码有1138行。

LinkedList 是一个继承于AbstractSequentialList的双向链表。它也可以被当作堆栈、队列或双端队列进行操作。

链表主要包含单向链表，单向循环链表，双向链表，双向循环链表。具体的图我就不在这边画出了，不清楚的可以自行百度。LinkedList是属于双向链表，下图是包含头结点和尾节点的双向链表。

2、LinkedList数据结构原理

LinkedList底层的数据结构是基于双向链表的，且头结点中不存放数据,如下：



既然是双向链表，那么必定存在一种数据结构——我们可以称之为节点，节点实例保存业务数据，前一个节点的位置信息和后一个节点位置信息，如下图所示：

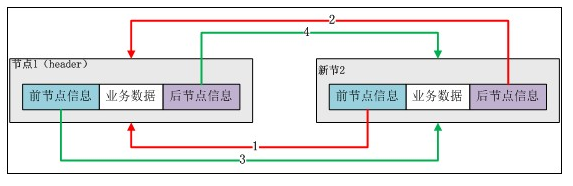


元素添加步骤：

第一步：调用无参构造方法，创建LinkedList实例。

第二步：初始化一个预添加的Entry实例（newEntry）。

第三步：调整新加入节点和头结点（header）的前后指针。



4、构造方法

LinkedList提供了两个构造方法。

1 public LinkedList() {

2 header.next = header.previous = header;

3 }

4 public LinkedList(Collection<? extends E> c) {

5 this();

6 addAll(c);

7 }

删除数据remove()

由于删除了某一节点因此调整相应节点的前后指针信息，如下：

**e.previous.next = e.next;//预删除节点的前一节点的后指针指向预删除节点的后一个节点。**

**e.next.previous = e.previous;//预删除节点的后一节点的前指针指向预删除节点的前一个节点。**

在LinkedList中查找元素：get()

为了提高效率，需要根据获取的位置判断是从头还是从尾开始遍历。

注 意细节：位运算与直接做除法的区别。先将index与长度size的一半比较，如果index<size/2，就只从位置0往后遍历到位置 index处，而如果index>size/2，就只从位置size往前遍历到位置index处。这样可以减少一部分不必要的遍历

在容量不够的时候可以扩容，然而LinkedList容量的说法，每次向其中加入元素时候，容量自动加1。

**Vector源码1212行。**

Vector:ArrayList的前身，数组结构，线程安全，速度慢

Vector构造函数。默认容量是10。一次增长原来的一倍。  
    public Vector() {  
        this(10);  
    }

其他分析，请参考ArrayList.