以下是一些在Stackoverflow上经常被问起的与Java集合相关的问题。在你查阅这些问题之前，最好先去看看[【Simple Java】Java集合框架的接口和类层次关系结构图](http://www.cnblogs.com/chenpi/p/5488296.html)。

**什么时候优先选择LinkedList，而不是ArrayList**

ArrayList本质上是一个数组，它的元素可以直接通过索引访问。但是，当数组满的时候，需要申请新的更大的数组空间，并将所有元素复制到新数组中，这将花费O(n)的时间。另外，插入和删除元素需要移动数组中的其它元素，这也许是ArrayList最大的劣势。

LinkedList是一个双向链表，因此，当访问链表中间的元素的时候，它需要从链表的头结点处开始搜索。但好的一方面是，对于插入和删除操作，LinkedList相对更快，因为它只需要改变链表的局部位置。

总的来说，在最差情况下，两者的时间复杂度比较如下：

| Arraylist | LinkedList

------------------------------------------

get(index) | O(1) | O(n)

add(E) | O(n) | O(1)

add(E, index) | O(n) | O(n)

remove(index) | O(n) | O(n)

Iterator.remove() | O(n) | O(1)

Iterator.add(E) | O(n) | O(1)

除了运行时间外，内存空间的使用也应该被考虑，特别是对于大的列表。在LinkedList中，每个节点需要两个额外的指针指向前节点和后节点，然而ArrayList只需要一个存放元素的数组即可。

其它List之间的比较，可查阅：[【Simple Java】ArrayList vs LinkedList vs Vector](http://www.cnblogs.com/chenpi/p/5505375.html)。

**迭代遍历集合的时候，正确的删除元素**

在迭代集合的时候，唯一正确的方法修改集合的方法是通过Iterator.remove()方法，如下示例：

Iterator<Integer> itr = list.iterator();

while(itr.hasNext()) {

// do something

itr.remove();

}

另外，举一个常见的错误代码：

|  |  |
| --- | --- |
| 1  2  3 | for(Integer i: list) {      list.remove(i);  } |

 运行以上错误代码，你将会得到ConcurrentModificationException异常，这是因为以上代码产生了一个迭代器（由for语句产生）去遍历列表，但是同时，列表被修改了（通过Iterator.remove()）。在Java中，一个线程在迭代遍历集合的过程中，是不允许另外一个线程去修改集合的。

**怎样将List转成int[]**

最简单的方法是使用Apache Commons Lang库下的ArrayUtils工具类，如下：

int[] array = ArrayUtils.toPrimitive(list.toArray(new Integer[0]));

在JDK中，没有捷径去做以上转换。注意你不能使用List.toArray()方法，因为这将会把List转成Integer[]。正确的方法如下：

int[] array = new int[list.size()];

for(int i=0; i < list.size(); i++) {

array[i] = list.get(i);

}

**怎样将int[]转成List**

最简单的方法仍然是使用Apache Commons Lang的ArrayUtils工具类，如下：

List list = Arrays.asList(ArrayUtils.toObject(array));

在JDK中，仍然没有捷径，只能使用如下方法：

int[] array = {1,2,3,4,5};

List<Integer> list = new ArrayList<Integer>();

for(int i: array) {

list.add(i);

}

**什么是过滤集合最好的方法**

同样，你可以使用第三方库，如google的Guava库或Apache Commons Lang去实现这个功能，两者都提供了filter()方法（Guava的Collections2或Apache的CollectionUtils类）。filter()方法会返回匹配的元素。

在JDK中，这将会变得困难，好消息是，在java 8中，增加了Predicate接口，可以实现该功能。但是现在，我们只能使用迭代器去遍历整个集合：

[复制代码](javascript:void(0);)

Iterator<Integer> itr = list.iterator();

while(itr.hasNext()) {

int i = itr.next();

if (i > 5) { // filter all ints bigger than 5

itr.remove();

}

}

[复制代码](javascript:void(0);)

当然，你可以模仿Guava或Apache的操作方法，通过引入一个新的接口Predicate，这可能是高级开发人员才会做的事，如下：

[复制代码](javascript:void(0);)

public interface Predicate<T> {

boolean test(T o);

}

public static <T> void filter(Collection<T> collection, Predicate<T> predicate) {

if ((collection != null) && (predicate != null)) {

Iterator<T> itr = collection.iterator();

while(itr.hasNext()) {

T obj = itr.next();

if (!predicate.test(obj)) {

itr.remove();

}

}

}

}

[复制代码](javascript:void(0);)

然后，我们使用如下代码去过滤集合：

filter(list, new Predicate<Integer>() {

public boolean test(Integer i) {

return i <= 5;

}

});

**List转Set最简单的方法**

有两种方法来实现该功能，取决于你如何定义“相等”。第一种方法将list存入HashSet，元素的重复主要由hashCode()来区分，大多数情况下，这是可行的。但是，如果你需要自己定义相等的比较方式，最好使用第二种方法，定义自己的比较器。

Set<Integer> set = new HashSet<Integer>(list);

Set<Integer> set = new TreeSet<Integer>(aComparator);

set.addAll(list);

**ArrayList中删除重复元素**

这个问题和上一个很像，如果你不关心ArrayList中元素的顺序的话，一个聪明的方法是通过将list元素存入set集合中来去除重复元素，然后将set集合的元素移回到List中。如下代码：

ArrayList\*\* list = ... // initial a list with duplicate elements

Set<Integer> set = new HashSet<Integer>(list);

list.clear();

list.addAll(set);

如果你关心元素的顺序的话，可以使用标准JDK中的LinkedHashSet来实现该功能。

**对集合排序**

Java中有几种方式来维持集合中元素的顺序，它们提供了默认的排序顺序或者通过指定比较器来排序。不过即使是默认的排序，集合中的任何元素也需要实现Comparable接口。

* Collections.sort()方法能够对一个List集合进行排序，正如javadoc中描述的，这种排序方法是稳定且能保证排序性能为*n log(n)*。
* PriorityQueue为一个有序队列，它与Collections.sort()的区别是PriorityQueue队列一直是有序的，但是你只能访问队头和队尾，不能随即访问元素，如PriorityQueue.get(4)之类的操作。
* 如果集合中没有重复的元素，TreeSet是另外一种选择。跟PriorityQueue类似，它能一直维护元素的顺序，你能直接获取TreeSet中的第一个和最后一个元素，但是你仍然不能随即访问集合中的元素。

简单的说，Collections.sort()提供了对List的一次性排序，PriorityQueue和TreeSet能一直维持集合中元素的顺序，但是不能随即访问元素。

**Collections.emptyList()与直接new一个实例的区别**

该问题也适用于emptyMap()和emptySet()。

这两种方式都返回了一个空集合，但是Collections.emptyList()返回的是一个不可变集合，意味着你不能往这个空集合新增元素。事实上，每次调用Collections.emptyList()并不会创建一个空集合，而是复用已经存在的空集合实例。如果你熟悉单例模式的话，你应该知道我所说的，如果你频繁调用的话，这将会提供更好的性能。

**Collections.copy方法**

有两种方法讲一个List集合拷贝到另外一个List集合，其中一种是使用ArrayList的构造方法，如下：

ArrayList<Integer> dstList = new ArrayList<Integer>(srcList);

另一种是使用Collections.copy()方法（如下），注意第一行，我们分配了一个和源List集合长度相等的初始容量。

ArrayList<Integer> dstList = new ArrayList<Integer>(srcList.size());

Collections.copy(dstList, srcList);

这两种方法都使用浅拷贝，那么这两种方法的区别是什么呢？

* 首先，当目标集合没有足够的空间存放源集合中的元素时，Collections.copy()方法不会对目标集合扩容，它会抛出一个IndexOutOfBoundsException异常。
* Collections.copy()的参数类型只能是List接口的实现类，而ArrayList的构造方法可以接受Collection接口的实现类作为入参，因此更加普通。