集合框架

1、数组的特点与弊端

1）一方面，面向对象语言对事物的体现都是以对象的形式，为了方便对多个对象的操作，就要对对象进行存储。

2）另一方面，使用数组存储对象方面具有一些弊端，而 Java 集合就像一种容器，可以动态地把多个对象的引用放入容器中。

3）**数组**在内存存储方面的**特点**：

1>数组**初始化以后**，**长度**就是**确定**的；

2>数组中的添加的元素是**依次紧密排列**的，**有序的，可以重复的**；

3>数组**声明的类型**，就决定了进行元素初始化时的类型，**不是此类型的变量，就不能添加**；

4>可以存储**基本数据类型值**，也可以存储**引用数据类型**的变量；

4）**数组**在存储数据方面的**弊端**：

1>数组一旦**初始化以后，长度就不可变**了，不便于扩展；

2>数组中提供的属性和方法少，**不便于进行添加、删除、插入、获取元素个数等操作**，且效率不高；

3>数组存储数据的特点单一，只能存储**有序的、可以重复的**数据；对于**无序、不可重复的**场景的多个数据就无能为力了；

4>针对数组中元素的**删除、插入**操作，**性能差**。

5）Java 集合框架中的类可以用于存储多个对象，还可用于保存具有映射关系的关联数组。

2、Java 集合框架体系（java.util 包内）

Java 集合分为 Collection 和 Map 两大体系：

1）Collection 接口：用于存储**一个一个的数据**，也称**单列**数据集合。

1>List 子接口：用来存储**有序的、可以重复的**数据（主要用来替换数组，**"动态"数组**），集合中的每个元素都有其对应的顺序索引；实现类：ArrayList(主要实现类)、LinkedList、Vector；

2>Set 子接口：用来存储**无序的、不可重复的**数据（类似于高中讲的"集合"）；实现类：HashSet(主要实现类)、LinkedHashSet、TreeSet；

2）Map 接口：用于存储**具有映射关系"key-value 对"的集合**，即**一对一对的数据**，也称**双列**数据集合。(类似于高中的函数、映射。(x1,y1),(x2,y2) ---> y = f(x))；HashMap(主要实现类)、LinkedHashMap、TreeMap、Hashtable、Properties。

3、Collection 接口及方法

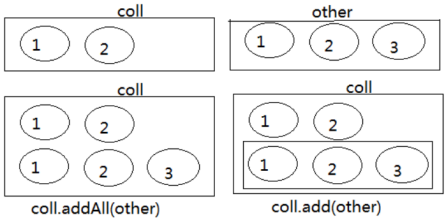
1）JDK **不提供此接口的任何直接实现**，而是提供**更具体的子接口（如 Set 和 List）去实现**。

2）Collection 接口是 List 和 Set 接口的父接口，该接口里定义的方法既可用于操作 Set 集合，也可用于操作 List 集合。

4、Collection 方法

1）**add(E obj)**：添加元素对象到当前集合中；

2）**addAll(Collection other)**：添加 other 集合中所有元素对象到当前集合，即 this = this ∪ other



3）**int size()**：获取当前集合中实际存储的元素个数；

4）**boolean isEmpty()**：判断当前集合是否为空集合；

5）**boolean contains(Object obj)**：判断当前集合中是否存在一个与 obj 对象 equals 返回 true 的元素；（对于**自定义的对象要重写 equals**）

6）**boolean containsAll(Collection coll)**：判断 coll 集合中的元素是否在当前集合中都存在，即 coll 集合是否是当前集合的"子集"；

7）**boolean equals(Object obj)**：判断当前集合与 obj 是否相等；

8）**void clear()**：清空集合元素；（**size()为 0，一个一个删除所有元素**）

9）**boolean remove(Object obj)** ：从当前集合中**删除第一个**找到的**与 obj 对象 equals** 返回 true 的元素；

10）**boolean removeAll(Collection coll)**：从当前集合中**删除所有**与 coll 集合中**相同的元素**。即 this = this - this ∩ coll；

11）**boolean retainAll(Collection coll)**：从当前集合中**删除两个集合中不同的元素**，使得当前集合仅保留与 coll 集合中的元素相同的元素，即当前集合中**仅保留两个集合的交集**，即 this = this ∩ coll；

12）**Object[] toArray()**：返回包含当前集合中所有元素的数组；（**转数组**）

13）**hashCode()**：获取**集合对象**的哈希值；

14）**iterator()**：返回**迭代器对象**，用于集合遍历。

【注意】**向 Collection 中添加元素时**，要求元素所属的类**一定要重写 equals 方法**；因为 Collection 中相关方法在使用时，要调用元素所在类的 equals 方法。

5、集合与数组的相互转换

集合→数组：toArray()

数组→集合：Arrays.asList(Object…objs)

```
Integer[] arr = new Integer[]{1,2,3};
List list = Arrays.asList(arr);// 3 个[1,2,3]
int [] arr = new int[]{1,2,3};
List list = Arrays.asList(arr);// 1 个[地址值]
```

6、Iterator(迭代器)接口

1）在程序开发中，经常需要遍历集合中的所有元素。针对这种需求，JDK 专门提供了一个接口 **java.util.Iterator**。**Iterator 接口**也是**Java 集合中**的一员，但它与 Collection、Map 接口有所不同：

1>Collection 接口与 Map 接口主要用于**存储元素**；

2>Iterator，被称为迭代器接口，本身并**不提供存储对象的能力**，主要用于**遍历 Collection 中的元素**；

2）Collection 接口**继承了 java.lang.Iterable 接口**，该接口有一个 iterator()方法，那么**所有实现了 Collection 接口的集合类**都有一个 iterator()方法，用以返回一个实现了 Iterator 接口的对象。

1>public Iterator iterator(): 获取**集合对应的迭代器**，用来遍历集合中的元素的。

2>集合对象**每次调用** iterator()方法都得到**一个全新的**迭代器对象，**默认游标**都在集合的**第一个元素之前**。

```
//方式2：错误的遍历
//每次调用coll.iterator()，都会返回一个新的迭代器对象.
while(coll.iterator().hasNext()){
    System.out.println(coll.iterator().next());
}
```

3）Iterator 接口的常用方法如下：

1>public E **next()**：**返回迭代的下一个**元素。

2>public boolean **hasNext()**：如果**仍有元素可以迭代**，则返回 true。

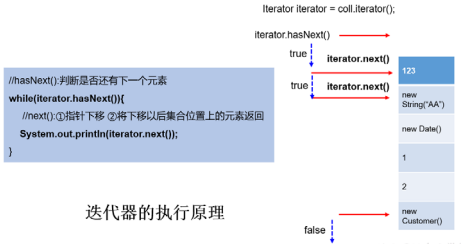
【注意】在调用 **it.next()方法之前**必须要调用 **it.hasNext()进行检测**。若不调用，且下一条记录无效，直接调用 it.next() 会抛出 **NoSuchElementException** 异常。

```
Iterator iterator = coll.iterator();//获取迭代器对象↓
while(iterator.hasNext()) {//判断是否还有元素可迭代↓
    System.out.println(iterator.next());//取出下一个元素↓
}↓
```

7、迭代器的执行原理

1）Iterator 迭代器对象在遍历集合时，内部采用**指针**的方式来跟踪集合中的元素。

2）使用 Iterator 迭代器**删除元素**：java.util.Iterator 迭代器中 **void remove()方法**。



【注意】

1>Iterator 可以删除集合的元素，但是遍历过程中**通过迭代器对象的 remove 方法**，**不是**集合对象的 remove 方法。

2>如果**还未调用 next()**或在**上一次调用 next()方法之后已经调用了 remove()方法，再调用 remove()**都会报 IllegalStateException。

3>Collection 已经有 remove(xx)方法了，为什么 Iterator 迭代器还要提供删除方法呢？因为**迭代器的 remove()**可以**按指定的条件**删除。

4>在**JDK8.0 时**，Collection 接口有了**removeIf 方法**，即可以根据**条件删除**。

8、foreach 循环（增强 for 循环）

1）JDK5.0 中定义的一个高级 for 循环，专门用来**遍历数组和集合**的。

2）通常**只进行遍历**元素，**不要在遍历过程中**对集合元素**进行增删操作**。

3）内部原理其实是个 Iterator 迭代器。

foreach 循环的语法格式：

```
for(元素的数据类型 局部变量 : Collection集合或数组){
      //操作局部变量的输出操作
}
```

9、Collection 子接口 1：List 接口方法

List 除了从 Collection 集合继承的方法外，List 集合里添加了一些根据索引来操作集合元素的方法。

1）插入元素

1>void **add(int index, Object ele)**:在**index 位置**插入 ele 元素；

2>boolean **addAll(int index, Collection eles)**:从**index 位置开始**将 **eles 中的所有元素**添加进来；

2）获取元素

1>Object **get(int index)**:获取**指定 index 位置**的元素；

2>List **subList(int fromIndex, int toIndex)**:返回从 **fromIndex 到 toIndex 位置**的子集合；

3）获取元素索引

1>**int indexOf(Object obj)**:返回 obj 在集合中**首次出现**的位置；

2>**int lastIndexOf(Object obj)**:返回 obj 在当前集合中**末次出现**的位置；

4）删除和替换元素

1>**Object remove(int index)**:移除**指定 index 位置**的元素，并返回此元素；

2>**Object set(int index, Object ele)**:设置**指定 index 位置**的元素为 ele。

【注意】list.remove(2)指删除**索引为 2** 的元素；list.remove(**Integer.valueOf(2)**)指删除**数据 2**。

10、List 不同实现类的对比

1）List 接口主要实现类：ArrayList

1>**线程不安全的**、**效率高**；底层是**用 Object[]数组存储**；本质上，ArrayList 是**对象引用的一个"变长"数组**。

2>在**添加**数据、**查找**数据时，**效率较高**；在**插入、删除**数据时，**效率较低**。

【注意】**Arrays.asList(…)方法返回的 List 集合**，既不是 ArrayList 实例，也不是 Vector 实例。Arrays.asList(…) 返回值是**一个固定长度的 List 集合**。

2）List 的实现类之二：LinkedList

1>底层采用**链表（双向链表）**结构存储数据；

2>在**插入、删除**数据时，**效率较高**；在**添加**数据、**查找**数据时，**效率较低**。对集合频繁的插入或删除元素的操作，建议使用 LinkedList 类，效率较高。
3>特有方法：
- void addFirst(Object obj)添加第一个
- void addLast(Object obj) 添加最后一个
- Object getFirst()获取第一个
- Object getLast()获取最后一个
- Object removeFirst()删除第一个
- Object removeLast()删除最后一个

3）List 的实现类之三：Vector
1>Vector 是一个古老的实现类，JDK1.0 就有了。大多数操作与 ArrayList 相同，区别之处在于 Vector 是**线程安全的、效率低**；底层是用 **Object[]数组存储**。
2>在各种 List 中，最好把 ArrayList 作为默认选择。当插入、删除频繁时，使用 LinkedList；Vector 总是比 ArrayList 慢，故尽量避免使用。
3>特有方法：
- void addElement(Object obj)
- void insertElementAt(Object obj,int index)
- void setElementAt(Object obj,int index)
- void removeElement(Object obj)
- void removeAllElements()

11、Collection 子接口 2：Set 接口概述
1）Set 接口是 Collection 的子接口，Set 接口相较于 Collection 接口没有提供额外的方法；
2）Set 集合**不允许包含相同的元素**，如果试把两个相同的元素加入同一个 Set 集合中，则添加操作失败；
3）Set 集合支持的遍历方式和 Collection 集合一样：for、foreach 和 Iterator。
4）开发中 Set 较 List 和 Map 来说使用频率较少；**可用来过滤重复数据**；
5）Set 的常用实现类有：HashSet、TreeSet、LinkedHashSet。

12、Set 主要实现类：HashSet
1）HashSet 概述
1>底层使用的是 HashMap，即使用**数组+单向链表+红黑树结构**进行存储（jdk8 中）；
2> HashSet 按 Hash 算法来存储集合中的元素，因此具有很好的存储、查找、删除性能。
3>HashSet 具有以下特点：
- 不能保证元素的排列顺序（**无序**）；
- HashSet **不是线程安全的**；
- 集合元素**可以是 null**。

4>HashSet 集合**判断两个元素相等**的标准：两个对象通过 **hashCode()方法**得到的**哈希值相等**，且两个对象 **equals()方法返回值为 true**。
5>**对于存放在 Set 容器中的对象**，对应的类**一定要重写** hashCode()和 equals(Object obj)方法，以实现对象相等规则。即："**相等的对象必须具有相等的散列码**"。
6>HashSet 集合中元素的**无序性**，不等同于随机性。这里的无序性与元素的添加位置有关。具体来说：我们在添加每一个元素到数组中时，具体的**存储位置**是**由元素的 hashCode()调用后返回的 hash 值决定**的。导致在数组中每个元素**不是依次紧密存放的**，表现出一定的无序性。

【注意】HashSet 如何检查重复?
当你把对象加入 HashSet 时，HashSet 会**先计算对象的 hashcode 值**来**判断**对象加入的**位置**，同时也会**与其他加入的对象**的 hashcode 值作**比较**，如果**没有相符的 hashcode**，HashSet 会假设对象**没有重复出现**。但是如果发现**有相同 hashcode 值的对象**，这时会**调用 equals()方法**来检查 hashcode 相等的**对象是否真的相同**。如果两者相同，HashSet 就不会让加入操作成功。

2）重写 hashCode()方法的基本原则
1>在程序运行时，同一个对象**多次**调用 hashCode()方法**应该返回相同**的值；
2>当两个对象的 **equals()方法**比较**返回 true 时**，这两个对象的 **hashCode()方法**的返回值**也应相等**；
3>对象中用作 equals()方法比较的 Field，都应该用来计算 hashCode 值；
【注意】如果两个元素的 equals()方法返回 true，但它们的 **hashCode()返回值不相等**，hashSet 将会把它们**存储在不同的位置**，但依然**可以添加成功**；

3）重写 equals()方法的基本原则
1>重写 equals 方法的时候一般都需要同时复写 hashCode 方法。通常参与计算 hashCode 的对象的属性也应该参与到 equals()中进行计算。
2>推荐：开发中直接调用 Eclipse/IDEA 里的快捷键自动重写 equals()和 hashCode()方法即可。

3>为什么用 Eclipse/IDEA 复写 hashCode 方法，有 31 这个数字？
- 首先，选择系数的时候要选择尽量大的系数。因为如果计算出来的 **hash 地址越大**，所谓的**"冲突"就越少**，查找起来**效率也会提高**。（**减少冲突**）
- 其次，**31 只占用 5bits**,相乘造成**数据溢出**的**概率较小**。
- 再次，31 可以由 i*31==**(i<<5)-1** 来表示,现在很多虚拟机里面都有做相关优化。（左移 5 个减一，**提高算法效率**）
- 最后，**31 是一个素数**，素数作用就是如果我用一个数字来乘以这个素数，那么最终出来的结果只能被素数本身和被乘数还有 1 来整除！**(减少冲突)**

13、Set 实现类之二：LinkedHashSet
1）LinkedHashSet 是 HashSet 的子类，**不允许集合元素重复**；
2）LinkedHashSet 在现有的**数组+单向链表+红黑树结构**的基础上，又增加一组双向链表，用于记录添加元素的先后顺序。即：根据元素的 **hashCode 值来决定元素的存储位置**，但它同时使用**双向链表维护元素的次序**，这使得**元素看起来是以添加顺序保存**的，便于频繁的查询操作。
3)LinkedHashSet 插入性能略低于 HashSet，但在迭代访问 Set 里的全部元素时有很好的性能。

14、Set 实现类之三：TreeSet 概述
1)TreeSet 是 SortedSet 接口的实现类，TreeSet 可以按照添加的元素的**指定的属性的大小顺序**进行遍历；
2）TreeSet 底层使用**红黑树结构**存储数据；
3）新增的方法如下： (了解)
- Comparator comparator()
- Object first()
- Object last()
- Object lower(Object e)
- Object higher(Object e)
- SortedSet subSet(fromElement, toElement)
- SortedSet headSet(toElement)
- SortedSet tailSet(fromElement)

4）TreeSet **特点**：不允许重复、实现排序（自然排序或定制排序）；
5）TreeSet 两种排序方法：自然排序和定制排序。**默认情况下，TreeSet 采用自然排序**。
1>自然排序：TreeSet 会调用集合元素的 compareTo(Object obj) 方法来比较元素之间的大小关系，然后将集合元素按升序(默认情况)排列。
- 如果试图把一个对象**添加到 TreeSet** 时，则**该对象的类必须实现 Comparable 接口**；
- 实现 Comparable 的类**必须实现 compareTo(Object obj) 方法**，两个对象即通过 compareTo(Object obj) 方法的返回值来比较大小。

2>定制排序：如果元素所属的类没有实现 Comparable 接口，或不希望按照升序(默认情况)的方式排列元素或希望按照其它属性大小进行排序，则考虑使用定制排序。定制排序，**通过 Comparator 接口来实现**。**需要重写 compare(T o1,T o2)方法。**
- 利用 int compare(T o1,T o2)方法，比较 o1 和 o2 的大小：如果方法**返回正整数**，则表示 o1 大于 o2；如果**返回 0**，表示相等；**返回负整数**，表示 o1 小于 o2。
- 要实现定制排序，需要将实现 Comparator 接口的实例**作为形参传给 TreeSet 的构造器**。

6）因为只有相同类的两个实例才会比较大小，所以向 TreeSet 中添加的应该是**同一个类的对象**。
7）对于 TreeSet 集合而言，它**判断两个对象是否相等的唯一标准**是：两个对象通过 **compareTo(Object obj)或 compare(Object o1,Object o2)方法比较返回值**。返回值为 0，则认为两个对象相等。（不再考虑 hashCode()和 equals()方法，意味着**不需要重写这俩方法**）

15、Map 接口概述
1）Map 与 Collection **并列**存在。用于保存具有**映射关系的数据**：key-value；
- Collection 集合称为单列集合，元素是孤立存在的（理解为单身）。
- Map 集合称为双列集合，元素是成对存在的(理解为夫妻)。

2）Map 中的 key 和 value 都可以是**任何引用类型**的数据。但常用 String 类作为 Map 的"键"；
3）Map 接口的常用实现类：HashMap、LinkedHashMap、TreeMap 和 Properties。其中，HashMap 是 Map 接口使用频率最高的实现类。

16、HashMap 中 key-value 特点
1）HashMap 中所有的 **key** 彼此之间是不可重复的、无序的；所有的 key 就构成一个 Set 集合；故 key 所在类要重写 hashCode()和 equals()。
2）HashMap 中所有的 **value** 彼此之间是可重复的、无序的；所有的 value 就构成一个 Collection 集合；value 所在类要重写 equals()。
3)HashMap 中的一个 key-value 构成一个 entry。
4）HashMap 中所有的 **entry** 彼此之间是不可重复的、无序的；所有的 entry 构成一个 Set 集合。

17、Map 接口的常用方法
1）添加、修改操作：
- Object **put**(Object key,Object value)：将指定 key-value **添加（或修改）**到当前 map 对象中；
- void **putAll**(Map m):将 m 中的所有 key-value 对存放到当前 map 中；

2）删除操作：
- Object **remove**(Object key)：移除指定 key 的 key-value 对，并返回 value；
- void **clear()**：清空当前 map 中的所有数据；

3）元素查询的操作：
- Object **get**(Object key)：获取指定 key 对应的 value；
- boolean **containsKey**(Object key)：是否包含指定的 key；
- boolean **containsValue**(Object value)：是否包含指定的 value；
- int **size()**：返回 map 中 key-value 对的个数；
- boolean isEmpty()：判断当前 map 是否为空；
- boolean **equals**(Object obj)：判断当前 map 和参数对象 obj 是否相等；

4）元视图操作的方法：
- Set keySet()：返回所有 key 构成的 Set 集合；
- Collection **values()**：返回所有 value 构成的 Collection 集合；
- Set **entrySet()**：返回所有 key-value 对构成的 Set 集合。

18、Map 不同实现类的对比
1）Map 的主要实现类：HashMap

1>HashMap 是 Map 接口**使用频率最高**实现类。
2>HashMap 是**线程不安全的**；允许添加 **null 键和 null 值**。
3>存储数据采用的**哈希表结构**，底层使用**一维数组+单向链表+红黑树**进行 key-value 数据的存储；与 HashSet 一样，元素的**存取顺序不能保证一致**。
4>HashMap 判断**两个 key 相等**的标准是：两个 key 的 **hashCode 值**相等，通过 **equals()方法**返回 true。
5>HashMap 判断**两个 value 相等**的标准是：两个 value 通过 **equals()方法**返回 true。

2）Map 实现类之二：LinkedHashMap

1>LinkedHashMap 是 **HashMap 的子类**；
2>存储数据采用的**哈希表结构+链表**结构，在 HashMap 存储结构的基础上，使用了一对**双向链表**来**记录**添加元素的**先后顺序**，可以保证遍历元素时，与添加的顺序一致。
3>通过**哈希表结构**可以保证**键的唯一、不重复**，需要**键所在类**重写 hashCode()方法、equals()方法。

3）Map 实现类之三：TreeMap

1>TreeMap 存储 **key-value 对**时，需要根据 **key-value 对进行排序**；TreeMap 可以保证所有的 **key-value 对**处于**有序状态**；
2>TreeSet 底层使用**红黑树结构**存储数据；
3>TreeMap 的 Key 的排序：
- **自然排序**：TreeMap 的所有的 Key 必须实现 Comparable 接口，而且**所有的 Key** 应该是**同一个类的对象**，否则将会抛出 ClasssCastException；
- **定制排序**：创建 TreeMap 时，**构造器传入一个 Comparator 对象**，该对象负责对 TreeMap 中的**所有 key 进行排序**。此时不需要 Map 的 Key 实现 Comparable 接口；

4>TreeMap 判断两个 **key 相等的标准**：两个 key 通过 **compareTo()方法**或者 **compare()方法**返回 0。

4）Map 实现类之四：Hashtable

1>Hashtable 是 Map 接口的**古老实现类**，JDK1.0 就提供了；不同于 HashMap，Hashtable 是**线程安全的**。
2>Hashtable 实现原理和 HashMap 相同，功能相同。底层都使用**哈希表结构**（**数组+单向链表**），查询速度快。
3>与 HashMap 一样，Hashtable 也**不能保证其中 Key-Value 对的顺序**；
4>Hashtable **判断**两个 **key 相等**、两个 **value 相等**的**标准**，与 HashMap 一致。
5>与 HashMap 不同，**Hashtable 不允许使用 null 作为 key 或 value。**

【注意】Hashtable 和 HashMap 的区别

1>**HashMap**：底层是一个哈希表（jdk7:数组+链表;jdk8:数组+链表+红黑树），是一个线程不安全的集合,执行效率高（jdk8:当**链表长度大于阈值（默认为 8）**时，将链表转化为红黑树（将链表**转换**成红黑树**前会判断**，如果当前**数组的长度小于 64**，那么会选择**先进行数组扩容**，而**不是转换**为红黑树），以减少搜索时间）。**Hashtable**：底层也是一个哈希表（数组+链表），是一个线程安全的集合,执行效率低。
2>**HashMap 集合**：可以存储 null 的键、null 的值。**Hashtable 集合**：不能存储 null 的键、null 的值
3>**初始容量大小每次扩充容量大小的不同**：
①创建时**如果不指定容量初始值**，**Hashtable** 默认的初始大小为 **11**，之后**每次扩充**，容量**变为原来的 2n+1**。**HashMap** 默认的初始化大小为 **16**。之后**每次扩充**，容量**变为原来的 2 倍**。
②创建时**如果给定了容量初始值**，那么 **Hashtable** 会**直接使用你给定的大小**，而 **HashMap** 会**将其扩充为 2 的幂次方大小**。
4>Hashtable 的子类 Properties（配置文件）依然活跃在历史舞台；**Properties 集合**是一个**唯一和 IO 流相结合**的集合。

5）Map 实现类之五：Properties

1>Properties 类是 **Hashtable 的子类**，该对象用于**处理属性文件**；
2>由于属性文件里的 key、value 都是字符串类型，所以 Properties 中要求 **key** 和 **value** 都是**字符串类型**；
3>**存取数据时**，建议使用 **setProperty**(String key,String value)方法和 **getProperty**(String key)方法。

19、Collections 工具类

Collections 是一个操作 Set、List 和 Map 等集合的工具类。常用方法：

1）排序：

1>reverse(List)：反转 List 中元素的顺序；
2>shuffle(List)：对 List 集合元素随机排序；
3>sort(List)：根据元素的自然顺序对指定 List 集合元素按升序排序；
4>sort(List，Comparator)：根据指定的 Comparator 产生的顺序对 List 集合元素进行排序；
5>swap(List，int，int)：将指定 list 集合中的 i 处元素和 j 处元素进行交换。

2）查找

1>Object max(Collection)：根据元素的自然顺序，返回给定集合中的最大元素；
2>Object max(Collection，Comparator)：根据 Comparator 指定的顺序，返回给定集合中的最大元素；
3>Object min(Collection)：根据元素的自然顺序，返回给定集合中的最小元素；
4>Object min(Collection，Comparator)：根据 Comparator 指定的顺序，返回给定集合中的最小元素；
【注意】max 是取最右边；min 是取最左边的。即如果是降序排列，max 取的是最小的那个。
5>**int binarySearch(List list,T key)**在 List 集合中**查找某个元素的下标**，但是 List 的元素必须是 T 或 T 的子类对象，而且必须是可比较大小的，即**支持自然排序**的。而且**集合也事先必须是有序的**，否则结果不确定。
6>int binarySearch(List list,T key,Comparator c)在 List 集合中查找某个元素的下标，但是 List 的元素必须是 T 或 T 的子类对象，而且集合也事先**必须是按照 c 比较器规则进行排序过的**，否则结果不确定。
【注意】二分法查找必须传有序的 list。
7>**int frequency(Collection c，Object o)**：返回指定集合中指定元素的出现次数。

3）复制、替换

1>void copy(List dest,List src)：将 src 中的内容复制到 dest 中；

```
List src = Arrays.asList(1,2,3,4,5);
//错误写法 IndexOutOfBoundsException
List dest = new ArrayList();
Collections.copy(dest,src);
//正确写法
List dest= new ArrayList(new Object[src.size()]);
Collections.copy(dest,src);
```

2>boolean replaceAll(List list，Object oldVal，Object newVal)：使用新值替换 List 对象的所有旧值；
3>提供了多个 unmodifiableXxx()方法，该方法返回**指定 Xxx 的不可修改的视图**。

4）添加

boolean addAll(Collection c,T... elements)将所有指定元素添加到指定 collection 中。

5）同步（不推荐，用 JUC 包下的并发集合）

Collections 类中提供了多个 **synchronizedXxx()方法**，该方法可使将指定集合包装成线程同步的集合，从而可以**解决多线程并发访问集合时的线程安全问题**。

【面试题】

1、说说什么是 fail-fast？

fail-fast 机制是 Java 集合（Collection）中的**一种错误机制**。当**多个线程**对**同一个集合**的内容进行操作时，就**可能会产生 fail-fast** 事件。
例如：当**线程 A** 通过 iterator 去**遍历某集合的过程中**，若该集合的内容**被其他线程所改变**了（**对集合元素个数进行修改的** add、remove 和 clear 操作），那么**线程 A 访问集合时**，就会抛出 ConcurrentModificationException 异常，产生 fail-fast 事件。
解决办法：建议使用“**java.util.concurrent 包下的类**”去取代“java.util 包下的类”。
modCount 是 ArrayList 中的一个成员变量，它**表示该集合实际被修改的次数**。expectedModCount 表示这个迭代器预期该集合被修改的次数。在遍历之前，把 modCount 记下来 expectModCount，后面 expectModCount 去和 modCount 进行比较，如果不相等了，证明已并发被修改了，于是抛出 ConcurrentModificationException 异常。

2、HashMap 的长度为什么是 2 的 N 次方呢？

为了能让 HashMap **存取数据的效率高**，尽可能地**减少 hash 值的碰撞**，也就是说尽量把数据能**均匀的分配**，每个链表或者红黑树长度尽量相等。【重点】**取余（%）操作**中如果**除数是 2 的幂次**，则等价于**与其除数减一的与（&）操作**。采用二进制位操作&，相对于%能够提高运算效率。

3、HashMap 与 ConcurrentHashMap 的异同

4、深拷贝和浅拷贝？

浅拷贝只是**增加了一个指针**指向已存在的内存地址；**深拷贝**是**增加了一个指针**并且**申请了一个新的内存**，使这个增加的指针指向这个新内存，使用深拷贝的情况下，释放内存的时候**不会**因为出现浅拷贝时**释放同一个内存的错误**。