数组：

缺点：（1）长度固定，如果要扩容等需要程序自己维护，如果要删除和插入，程序员要移动元素等

（2）数组只支持“可重复，顺序存储”特点，比较单一

集合：很多种容器

实际开发中，数据的存储特点：（1）有序的（2）无序的（3）可以重复的（4）不能重复的（5）一对一的（6）一对多....

JDK在（1）数组（2）链式结构基础上，重新设计出了很多的容器类型。

主要是两大类：

1、Collection：一组对象，比喻"单身party"

2、Map：键值对，(key,value)，比喻“情侣party”，“家庭party"

容器有共同的行为特征，操作方式：

增、删、改、查....

把对容器的操作的行为标准化，用接口来声明。

一、java.util.Collection

（一）Collection概述

Collection 层次结构中的根接口。Collection 表示一组对象。

一些 collection 允许有重复的元素，而另一些则不允许。一些 collection 是有序的，而另一些则是无序的。

JDK 不提供此接口的任何直接实现：它提供更具体的子接口（如 Set 和 List）实现。

1、List

列表：可重复的，有序的（按顺序存储）

实现类：例如ArrayList（动态数组）

2、Set

集：不可重复的，无序的（和添加顺序无关）

（二）Collection的常用方法

1、添加

（1）add(Object obj)

（2）addAll(Collection other)

2、删除

3、修改

4、查询

```
*  5、获取有效元素的个数
*  int size()
```

```
1、添加
（1）add(Object obj)：一次添加一个
（2）addAll(Collection other)：一次添加多个，把other中的元素都添加到当前集合中
            this = this ∪(并) other
2、删除
（1）remove(Object o) ：一次删除一个
（2）removeAll(Collection other)：一次删除多个
             this = this - this ∩(交) other
（3）clear()

3、修改：Collection根接口中没有提供修改的方法

4、查询
（1）boolean contains(Object o) ：判断o是否在当前集合中
（2）boolean containsAll(Collection<?> c)：判断c中的元素都在当前集合中
          即判断c是否是this的子集
（3）boolean isEmpty()

5、获取有效元素的个数
int size()

6、遍历
（1）老方法
Object[] toArray()
```

```
* foreach：称为增强版for循环
* 语法结构：for(元素的类型 元素名 ： 可迭代的容器名){}
* 可迭代的容器名：例如：数组和集合
*
*
* 把元素名“看成”形参，每循环一次，把数组或集合的元素依次作为“实参”赋值给它。
* 即，如果元素是基本数据类型，那么把数组或集合的元素“数据值”赋值给它，那么对它怎么修改和实参无关。
*      如果元素是引用数据类型，那么把数组或集合的元素“地址值”赋值给它，那么对它的属性修改和实参有关。
*                                                          对它的地址修改和实参无关。
*
*
* 结论：如果你只是查看数组或集合的元素，用foreach比较简单，如果要涉及到删除，修改，就考虑其他的。
*/
```

```
* java.util.Iterator迭代器类型：用于遍历（迭代）Collection系列集合用的。
*
* 步骤：
* 1、先通过Collection系列集合.对象拿到迭代器对象
* 2、再通过Iterator的方法进行迭代
* boolean hasNext()：判断集合中是否有下一个元素需要迭代
* Object next()：取出下一个元素
* void remove()：删除刚刚迭代的元素，用于根据条件删除
```

```
/*
* java.util.List：接口
* （1）有序：可以对元素的索引index，进行控制。
* （2）可重复。
*
* 常用方法：
*  继承了Collection，因此，刚才学习的Collection的所有的方法和操作它都有。
*  List还增加了很多方法，这些方法都和index相关。
*
*  1、添加
* add(int index, E element) ：在index位置插入一个元素
* addAll(int index, Collection<? extends E> c)：在index位置插入多个元素
*
*/
```

```
* 2、删除
* remove(int index)
*
* 3、改
*  刚才Collection根接口没有提供修改的方法
* set(int index, E element)
*
* 4、查
* int indexOf(Object o)  ：从前往后
* int lastIndexOf(Object o )：从后往前
```

```
* 4、查
* int indexOf(Object o)  ：从前往后
* int lastIndexOf(Object o )：从后往前
* get(int index) ：返回index位置的元素
* subList(int fromIndex, int toIndex) ：截取[fromIndex,toIndex)
*
* 5、遍历
* （1）toArray
* （2）foreach
* （3）Iterator
* （4）ListIterator
*      ListIterator是Iterator的子接口，Iterator有的，ListIterator也有，还增加了：
*  A:Iterator只能从前往后遍历
*    ListIterator可以从任意位置开始，从前往后，或从后往前遍历
*
```

```
(1) toArray
(2) foreach
(3) Iterator
(4) ListIterator
     ListIterator是Iterator的子接口，Iterator有的，ListIterator也有，还增加了：
A:Iterator只能从前往后遍历
  ListIterator可以从任意位置开始，从前往后，或从后往前遍历

  ListIterator的使用步骤：
  第一步：先获取ListIterator的对象
        集合对象.listIterator()
  第二步：通过遍历方法
    hasNext()+next()
    hasPrevious() + previous()
B:不仅可以在遍历是删除了，还增加了set和add方法。
```

```
* List的常见的实现类：
* 1、Vector：动态数组
*       内部实现：数组，初始化大小为10
* 2、ArrayList：动态数组
*       内部实现：数组，初始化大小为10
*
*
*
* 面试题：Vector与ArrayList有什么区别？
* Vector：旧版，线程安全的，扩容为原来的2倍，
* ArrayList：新版，线程不安全，扩容为原来的1.5倍
*/
```

```
* List的常见的实现类：
* 1、Vector：动态数组
*       内部实现：数组，初始化大小为10
* 2、ArrayList：动态数组
*       内部实现：数组，初始化大小为10
* 3、LinkedList：双向链表、双端队列
* 4、Stack：栈，又是Vector的子类
*
* Stack：后进先出（LIFO）（Last in First out）、先进后出（FILO）（First in Last out）
*       压栈：push，弹栈：pop（移除栈顶元素），peek（返回栈顶元素，但不移除）
* 队列（Queue）：先进先出（FIFO）
*       添加到队列offer(e)，移出队列，poll()，返回队头不移除peek()
* 双端队列（Deque）（Double ended queue）：队头和队尾都可以添加元素和移除元素
*       offerFirst(e)、offerLast(e)
*       pollFirst()、pollLast()
*       peekFirst()、peekLast()
*
* 面试题：Vector与ArrayList有什么区别？
* Vector：旧版，线程安全的，扩容为原来的2倍，支持迭代的方式更多，支持旧版Enumeration迭代器
* ArrayList：新版，线程不安全，扩容为原来的1.5倍，不支持老版的Enumeration迭代器
*/
```

```
* 面试题：动态数组和LinkedList有什么区别？
*   （1）内部实现不同：
* 动态数组底层数组
* LinkedList是链表
* （2）动态数组：对索引的相关操作，效率很高
*       链表：对索引的相关操作
```

```
* 面试题：动态数组和LinkedList有什么区别？
*   （1）内部实现不同：
* 动态数组底层数组
* LinkedList是链表，元素的类型是节点类型，Node(prev,data,next)
* （2）动态数组：对索引的相关操作，效率很高
*       链表：对索引的相关操作，效率比较低
* （3）动态数组：插入、删除，涉及到移动元素
*      链表 ：插入，删除，只涉及到前后的元素的关系
*
* 结论：如果是后面的操作针对索引更多，那么选择动态数组，如果是添加和删除，插入等操作更多，选择链表
*/
```

```
* java.util.Set：接口，是Collection的子接口。
* （1）不支持重复
* （2）无序的（和添加顺序无关）
*
* Set：
* （1）HashSet：完全无序
* （2）TreeSet：大小顺序，和添加顺序无关
* （3）LinkedHashSet：遍历时可以保证添加顺序，存储和添加顺序无关
*/
```

```
* Set：
* （1）HashSet：完全无序
* （2）TreeSet：大小顺序，和添加顺序无关
* （3）LinkedHashSet：遍历时可以保证添加顺序，存储和添加顺序无关
*       LinkedHashSet是HashSet的子类，但是它的元素比HashSet的元素要多维护一个添加的顺序。
*       LinkedHashSet的效率就比HashSet低，每次添加，删除，要同时考虑顺序。
*
* 结论：如果既要元素不重复，又要按大小，选TreeSet
*       如果既要元素不重复，又要保证添加顺序，选LinkedHashSet
*       如果只是要元素不重复，选择HashSet
*/
```

```
* java.util.Set：接口，是Collection的子接口。
* （1）不支持重复
* （2）无序的（和添加顺序无关）
*
* Set没有增加方法，都是Collection接口的方法。
*
* Set：
* （1）HashSet：完全无序
*       如何保证两个元素不重复？依据元素的equals方法
* （2）TreeSet：大小顺序，和添加顺序无关
*       如何保证两个元素不重复？依据元素的“大小”顺序
* （3）LinkedHashSet：遍历时可以保证添加顺序，存储和添加顺序无关
*       LinkedHashSet是HashSet的子类，但是它的元素比HashSet的元素要多维护一个添加的顺序。
*       LinkedHashSet的效率就比HashSet低，每次添加，删除，要同时考虑顺序。
*       LinkedHashSet和HashSet一样依据equals，决定是否重复。
*
* 结论：如果既要元素不重复，又要按大小，选TreeSet
*      如果既要元素不重复，又要保证添加顺序，选LinkedHashSet
*      如果只是要元素不重复，选择HashSet
```

```
要重写Book类的hashCode和equals
要求：
（1）必须一起重写
（2）hashCode值相同，不一定相同
     hashCode值不相同，equlas一定不相同
     equals相同，hashCode值一定相同

  要求：参与hashCode值计算的属性，就要参数equals的比较
（3）equals方法的重写遵循几个原则
  对称性、自反性、传递性、一致性、非空与null比较返回false
  参考：Object的equlas的API
```

```
/*
 * 4、添加到一个集合中，要求不可重复，但是要求这次按照价格从低到高排序
     提示：TreeSet

     因为在上一题中，默认实现按照销量从高到低，我们就不方便再次修改Book类的compareTo方法
     说明，Book类的自然排序规则，无法满足本题的要求。

     要求：给TreeSet对象要多传一个“定制比较器对象”，即java.util.Comparator

     Arrays.sort(arr)：按照元素的自然排序
     Arrays.sort(arr,定制比较器对象)

     TreeSet set = new TreeSet();按照元素的自然排序
     TreeSet set = new TreeSet(定制比较器对象);按照定制比较器排序
 */
```

```
* java.util.Map：接口
*  和Collection最大的不同，就是它存储“键值对，映射关系”
*
* 常用方法：
* 1、添加
*  put(key,value)：一次添加一对映射关系
*  putAll(Map map)：一次添加多对映射关系
*      this = this ∪ map
* 2、删除
*
* 3、修改
* 4、查询
*
* 例如：存储男同学和他们的女朋友们
*/
```

```
* 常用方法：
* 1、添加
*  put(key,value)：一次添加一对映射关系
*  putAll(Map map)：一次添加多对映射关系
*      this = this ∪ map
* 2、删除
*  remove(Object key)：根据key删除一对
*  clear()：清空
*
* 3、修改
*  通过put可以替换value
*
* 4、查询
*
* 5、获取映射关系，键值对数：int size()
*
* 例如：存储男同学和他们的女朋友们
*/
```

```
* 1、添加
*  put(key,value)：一次添加一对映射关系
*  putAll(Map map)：一次添加多对映射关系
*      this = this ∪ map
* 2、删除
*  remove(Object key)：根据key删除一对
*  clear()：清空
*
* 3、修改
*  通过put可以替换value，只要key相同，就会替换
*
* 4、查询
* （1）containsKey(Object key)：判断某个key是否存在
* （2）containsValue(Object value)：判断某个value是否存在
* （3）V get(Object key)：根据key获取value
* （4）boolean isEmpty() ：是否为空
*
* 5、获取映射关系，键值对数：int size()
*
* 例如：存储男同学和他们的女朋友们
```

```
* Map的实现类们：
* 1、HashMap：哈希表
* 2、Hashtable：哈希表
* 3、LinkedHashMap：
*  它是HashMap的子类，比HashMap多维护了添加的顺序。
*
*
* 面试题：Hashtable与HashMap的区别
* Hashtable：最古老，线程安全的，它的key和value不允许为null
* HashMap：相对新，线程不安全，它的key和value都允许为null
*
* 类同：StringBuffer与StringBuilder
*      Vector与ArrayList
```

```
* Map的实现类们：
* 1、HashMap：哈希表
* 2、Hashtable：哈希表
* 3、LinkedHashMap：
*   它是HashMap的子类，比HashMap多维护了添加的顺序。
* 4、TreeMap：映射关系的顺序会按照key的“大小”顺序排列
*   要求：映射关系的key，必须支持排序，即实现java.lang.Comparable接口，或者单独为TreeMap指定定制比较器对象。
* 5、Properties：
*   （1）Properties是Hashtable的子类
*   （2）Properties的key和value的类型是String
*
* 面试题：Hashtable与HashMap的区别
* Hashtable：最古老，线程安全的，它的key和value不允许为null
* HashMap：相对新，线程不安全，它的key和value都允许为null
*
* 类同：StringBuffer与StringBuilder
*       Vector与ArrayList
```