

非徒教，**0元**入学，**不1万**就业不给1分钱学费，我们已**千四年**了！

笔记总链接：<http://bbs.itheima.com/thread-200600-1-1.html>

6. 集合

6.2 集合类

6.2.5 Map、HashMap、TreeMap

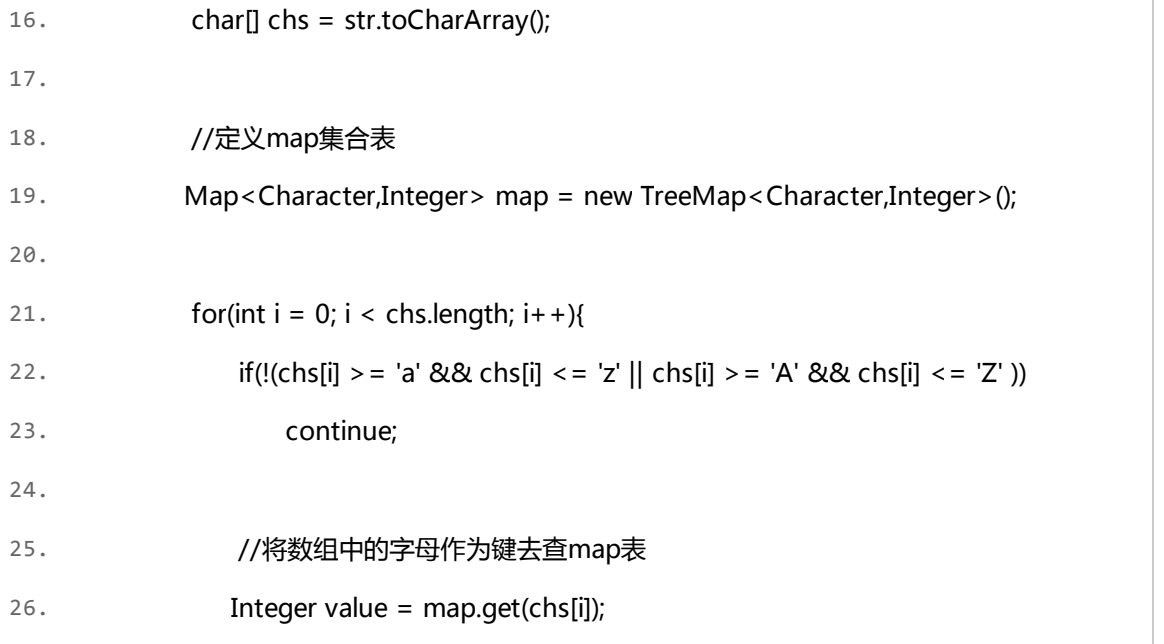
使用LinkedHashMap则是跟原来存入的顺序是一致的。

示例7：

```
01. import java.util.HashMap;
02. import java.util.Iterator;
03. import java.util.LinkedHashMap;
04. import java.util.Map;
05.
06. public class LinkedHashMapDemo{
07.     public static void main(String[] args){
08.         HashMap<Integer,String> hm = new LinkedHashMap<Integer,String>();
09.
10.         hm.put(7, "zhouqi");
11.         hm.put(3, "zhangsang");
12.         hm.put(1, "qianyi");
13.         hm.put(5, "wangwu");
14.
15.         Iterator<Map.Entry<Integer,String>> it =
16.             hm.entrySet().iterator();
17.
18.         while(it.hasNext()){
19.             Map.Entry<Integer,String> me = it.next();
20.
21.             Integer key = me.getKey();
22.             String value = me.getValue();
23.
24.             System.out.println(key + ":" + value);
25.         }
26.     }
27. }
```

复制代码

运行结果：



练习：

“fdqavcsacdfs” 获取该字符串中，每一个字母出现的次数。

要求打印结果是：a(2)b(1)...

思路：

对于结果的分析发现，字母和次数之间存在着映射的关系，而且这种关系很多。

很多就需要存储，能存储映射关系的容器有数组和Map结合。

关系中没有编号！那就使用Map结合。

又发现可以保证唯一的一方具备着顺序，如a、b、c...

所以可以使用TreeMap集合。

这个集合最终应该存储的是字母和次数的对应关系。

1. 因为操作的是字符串中的字母，所以先将字符串变成字符数组。

2. 遍历字符数组，用每一个字母作为键去查Map集合这个值。

如果该字符数组不存在，就将该字母作为键，1作为值存储到map集合中。

如果该字母存在，就将该字母键对应值取出并+1，再将该字母和+1后的值存储到map集合中。键相值会覆盖，这样就记录了该字母的次数。

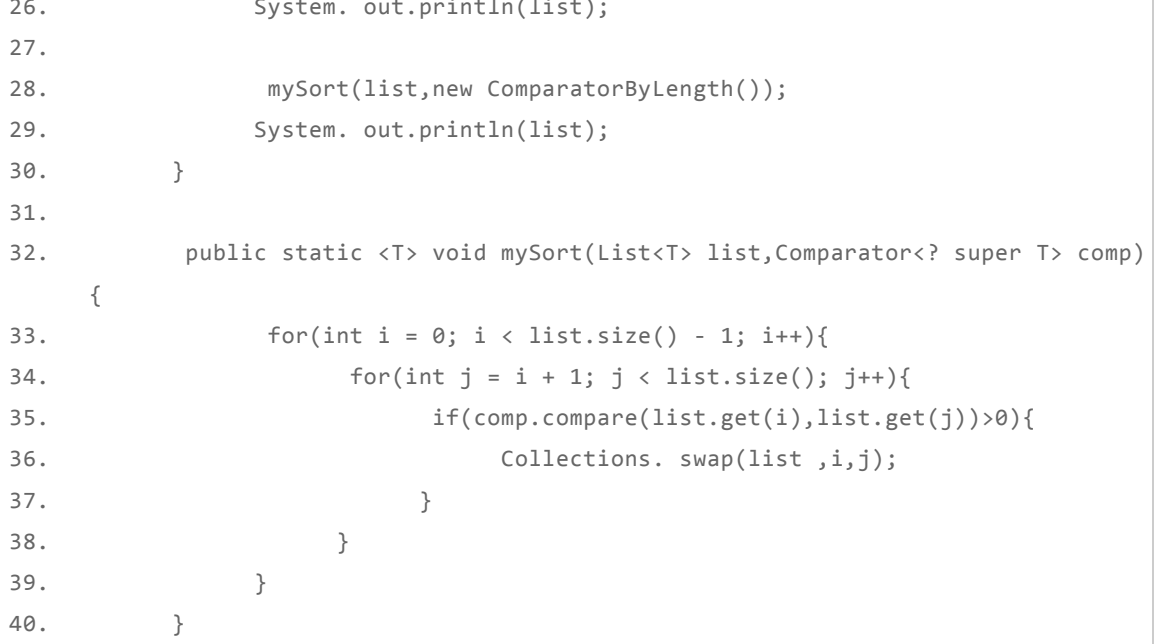
3. 遍历结果，map集合就记录所有字母的出现的次数。

代码：

```
01. import java.util.Iterator;
02. import java.util.Map;
03. import java.util.TreeMap;
04.
05. public class MapTest{
06.     public static void main(String[] args){
07.         String str = "fdqavcsacdfs";
08.
09.         String s = getCharCount(str);
10.
11.         System.out.println(s);
12.     }
13.
14.     public static String getCharCount(String str){
15.         //将字符串变为字符数组
16.         char[] chs = str.toCharArray();
17.
18.         //定义map集合表
19.         Map<Character,Integer> map = new TreeMap<Character,Integer>();
20.
21.         for(int i = 0; i < chs.length; i++){
22.             if((chs[i] >= 'a' && chs[i] <= 'z' || chs[i] >= 'A' && chs[i] <= 'Z' ))
23.                 continue;
24.
25.             //将数组中的字母作为键去查map表
26.             Integer value = map.get(chs[i]);
27.
28.             int count = 0;
29.
30.             //判断值是否为null
31.             if(value==null){
32.                 count = value;
33.             }
34.             count++;
35.
36.             map.put(chs[i],count);
37.         }
38.
39.         return map.toString();
40.     }
41.
42.     private static String mapToString(Map<Character,Integer> map){
43.         StringBuilder sb = new StringBuilder();
44.
45.         Iterator<Character> it = map.keySet().iterator();
46.
47.         while(it.hasNext()){
48.             Character key = it.next();
49.             Integer value = map.get(key);
50.
51.             sb.append(key + "(" + value + ")");
52.         }
53.         return sb.toString();
54.     }
55. }
```

复制代码

运行结果：



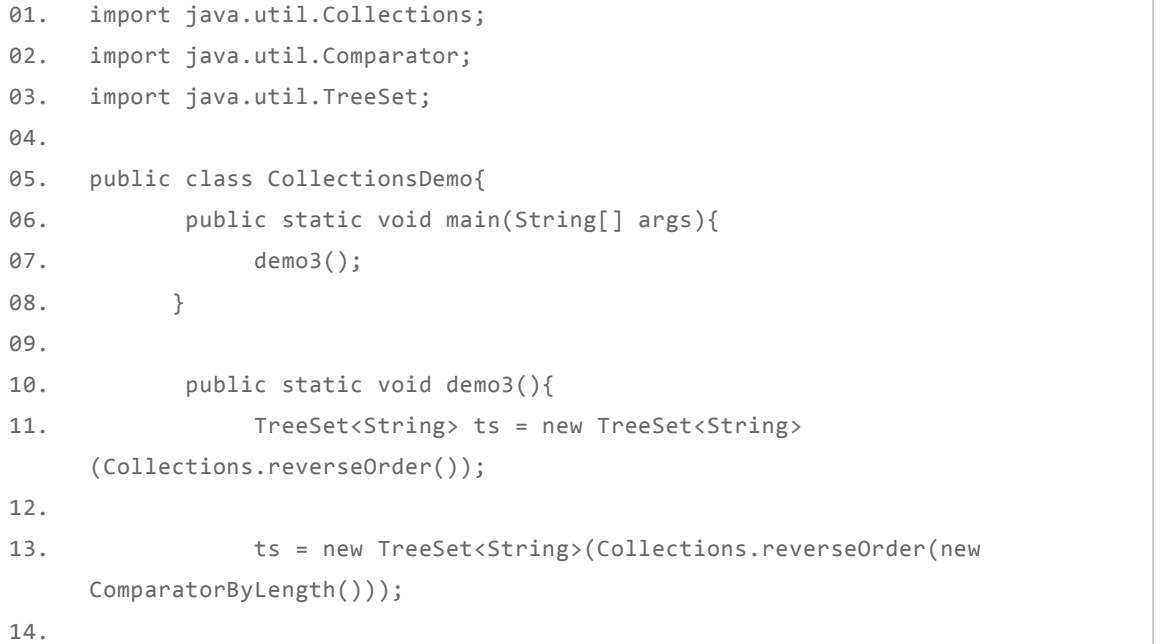
Map在有映射关系时，可以优先考虑在查表法中的应用较为多见。

示例：

```
01. import java.util.HashMap;
02. import java.util.Map;
03.
04. public class MapTest{
05.     public static void main(String[] args){
06.         String week = getWeek(1);
07.         System.out.println(week);
08.
09.         System.out.println(getWeekByMap(week));
10.     }
11.
12.     public static String getWeek(String week){
13.         Map<String,String> map = new HashMap<String,String>();
14.
15.         map.put( "星期一","Mon" );
16.         map.put( "星期二","Tue" );
17.         map.put( "星期三","Wes" );
18.         map.put( "星期四","Sun" );
19.         map.put( "星期五","Sun" );
20.
21.         return map.get(week);
22.     }
23.
24.     public static String getWeek(int week){
25.         if(week<1 || week>7)
26.             throw new RuntimeException("没有对应的星期，请您重新输入" );
27.
28.         String[] weeks = { "","星期一","星期二" };
29.
30.         return weeks[week];
31.     }
32. }
```

复制代码

运行结果：



6.2.6 Collections工具类

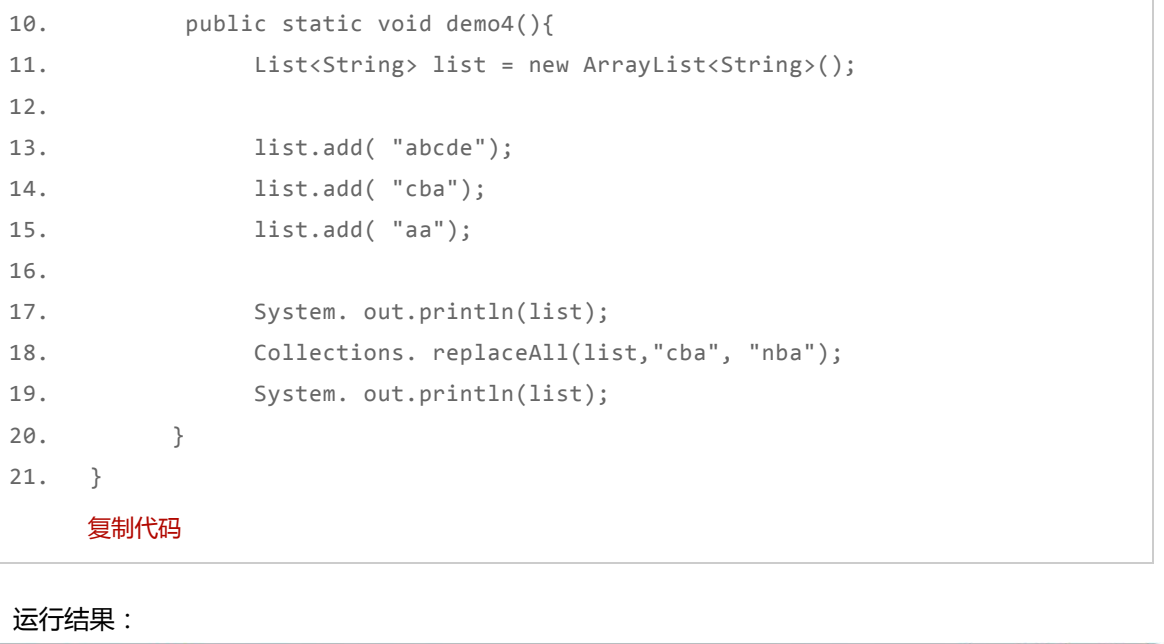
Collections是集合框架的工具类，里面的方法都是静态的。

示例1：

```
01. import java.util.ArrayList;
02. import java.util.Collections;
03. import java.util.Comparator;
04. import java.util.List;
05.
06. public class CollectionsDemo{
07.     public static void main(String[] args){
08.         demo1();
09.     }
10.
11.     public static void demo1(){
12.         List<String> list = new ArrayList<String>();
13.
14.         list.add( "abcde");
15.         list.add( "cba");
16.         list.add( "aaa");
17.         list.add( "zzz");
18.         list.add( "cba");
19.         list.add( "nbaa");
20.
21.         //对list集合进行指定顺序的排序
22.         Collections.sort(list);
23.         System.out.println(list);
24.
25.         Collections.sort(list,new Comparator<String>());
26.         System.out.println(list);
27.
28.         mySort(list,new Comparator<String>());
29.         System.out.println(list);
30.     }
31.
32.     public static <T> void mySort(List<T> list,Comparator<? super T> comp)
33.     {
34.         for(int i = 0; i < list.size() - 1; i++){
35.             for(int j = i + 1; j < list.size(); j++){
36.                 if(comp.compare(list.get(i),list.get(j))>0){
37.                     Collections.swap(list,i,j);
38.                 }
39.             }
40.         }
41.     }
42.
43.     class ComparatorByLength implements Comparator<String>{
44.         public int compare(String o1,String o2){
45.             int temp = o1.length() - o2.length();
46.             return temp == 0?0:1.compareTo(o2);temp;
47.         }
48.     }
```

复制代码

运行结果：

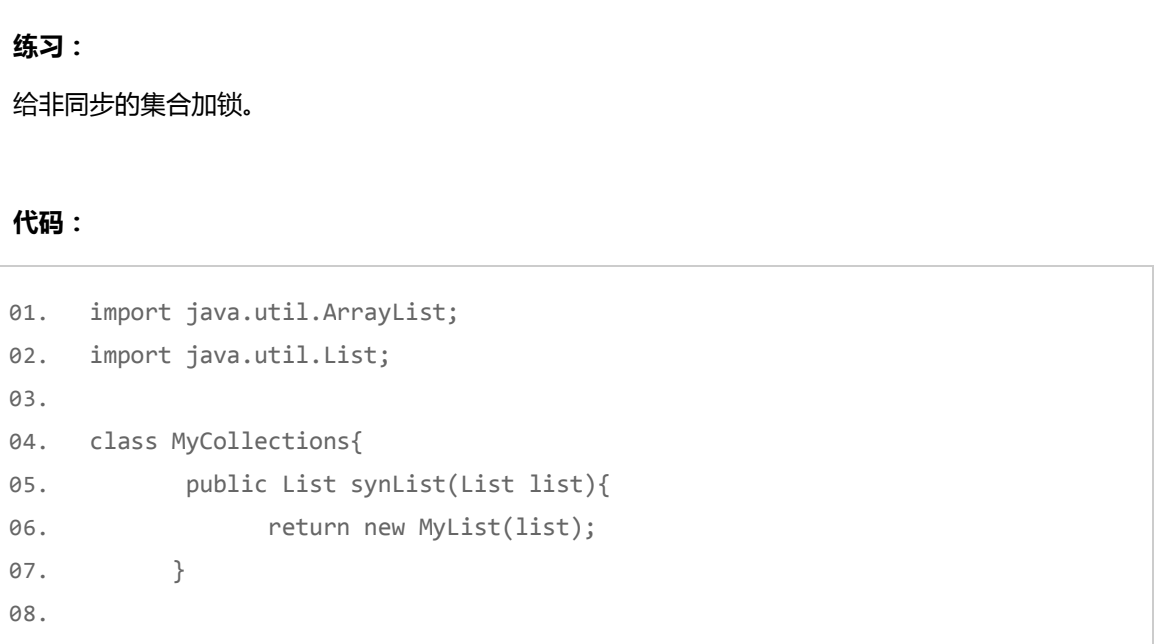


示例2：

```
01. import java.util.ArrayList;
02. import java.util.Collections;
03. import java.util.Comparator;
04. import java.util.List;
05.
06. public class CollectionsDemo{
07.     public static void main(String[] args){
08.         demo2();
09.     }
10.
11.     public static void demo2(){
12.         List<String> list = new ArrayList<String>();
13.
14.         list.add( "abcde");
15.         list.add( "cba");
16.         list.add( "aaa");
17.         list.add( "zzz");
18.         list.add( "cba");
19.         list.add( "nbaa");
20.
21.         Collections.sort(list);
22.         System.out.println(list);
23.
24.         int index = Collections.binarySearch(list,"aaa");
25.         System.out.println( "index = " + index);// -2 -index-1
26.
27.         //获取最大值
28.         String max = Collections.max(list, new Comparator<String>());
29.         System.out.println( "max = " + max);
30.     }
31.
32.     class ComparatorByLength implements Comparator<String>{
33.         public int compare(String o1,String o2){
34.             int temp = o1.length() - o2.length();
35.             return temp == 0?0:1.compareTo(o2);temp;
36.         }
37.     }
```

复制代码

运行结果：



原因分析：

binarySearch

使用二分法搜索指定列表，以返回指定对象，在进行此调用之前，必须根据列表元素的自然顺序对列表进行排序（通过 `sort()` 方法）。如果没有对列表进行排序，则结果是不确定的。如果列表包含多个等于指定搜索的元素，则无法保证找到的是哪一个。

此方法在“随机访问”列表（如 `ArrayList`）或“随机访问”列表（如 `LinkedList`）上使用。如果列表没有实现 `RandomAccess` 接口并且是一个大型列表，则此方法将执行最多于指定索引的二分搜索，执行 `O(log N)` 次循环搜索和 `O(log N)` 次元素比较。

参数：

- list - 要搜索的列表。
- key - 要搜索的键。

返回值：

如果键存在于列表中，则返回键的索引；否则，返回 `-(insertion point) - 1`。插入点被定义为将键插入列表的第一点，即第一个大于此键的元素索引。如果列表中的元素都与键小于指定键，则返回 `list.size() - 1`。这确保了当在有序列表中搜索时，返回的索引 `>= 0`。

注意：

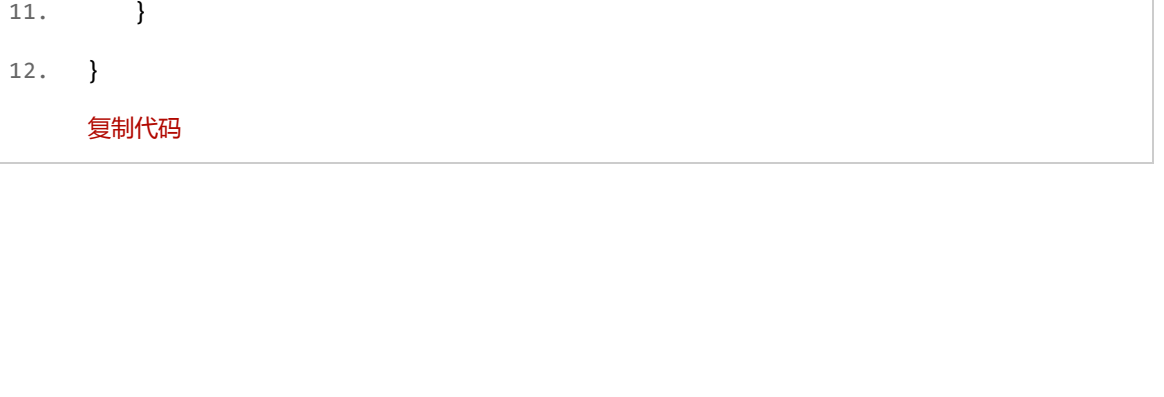
`Comparable` - 如果列表包含不可比较的元素（例如，字符串和整数），或者搜索键无法与列表的元素进行相互比较。

示例3：

```
01. import java.util.Collections;
02. import java.util.Comparator;
03. import java.util.TreeSet;
04.
05. public class CollectionsDemo{
06.     public static void main(String[] args){
07.         demo3();
08.     }
09.
10.     public static void demo3(){
11.         TreeSet<String> ts = new TreeSet<String>
12.             (Collections.reverseOrder());
13.
14.         ts = new TreeSet<String>(Collections.reverseOrder(new
15.             ComparatorByLength()));
16.
17.         ts.add( "abc");
18.         ts.add( "hahaha");
19.         ts.add( "zzz");
20.         ts.add( "aa");
21.         ts.add( "cba");
22.
23.         System.out.println(ts);
24.     }
25.
26.     class ComparatorByLength implements Comparator<String>{
27.         public int compare(String o1,String o2){
28.             int temp = o1.length() - o2.length();
29.             return temp == 0?0:1.compareTo(o2);temp;
30.         }
31.     }
```

复制代码

运行结果：



示例4：

```
01. import java.util.ArrayList;
02. import java.util.Collections;
03. import java.util.List;
04.
05. public class CollectionsDemo{
06.     public static void main(String[] args){
07.         demo4();
08.     }
09.
10.     public static void demo4(){
11.         List<String> list = new ArrayList<String>();
12.
13.         list.add( "abcde");
14.         list.add( "cba");
15.         list.add( "aaa");
16.
17.         System.out.println(list);
18.         Collections.replaceAll(list,"cba", "nba");
19.         System.out.println(list);
20.     }
21. }
```

复制代码

运行结果：

示例5：

```
01. import java.util.ArrayList;
02. import java.util.Collections;
03. import java.util.List;
04.
05. public class CollectionsDemo{
06.     public static void main(String[] args){
07.         demo5();
08.     }
09.
10.     public static void demo5(){
11.         List<String> list = new ArrayList<String>();
12.
13.         list.add( "abcde");
14.         list.add( "zhangsang");
15.         list.add( "zhaoqiang");
16.         list.add( "xiaoliu");
17.
18.         System.out.println(list);
19.         Collections.shuffle(list);
20.         System.out.println(list);
21.     }
22. }
```

复制代码

运行结果：

原因分析：

shuffle

public static void **shuffle**([List](#) <?> list)

使用默认随机源对指定列表进行置换。所有置换发生的可能性都是大致相等的。

练习：

给非同步的集合加锁。

代码：

```
01. import java.util.ArrayList;
02. import java.util.List;
03.
04. class MyCollections{
05.     public List synList(List list){
06.         return new MyList(list);
07.     }
08.
09.     private class MyList extends ArrayList{
10.         private List list;
11.
12.         private final Object lock = new Object();
13.
14.         MyList(List list){
15.             this.list = list;
16.         }
17.
18.         public boolean add(Object obj){
19.             synchronized(lock){
20.                 return list.add(obj);
21.             }
22.         }
23.
24.         public boolean remove(Object obj){
25.             synchronized(lock){
26.                 return list.remove(obj);
27.             }
28.         }
29.     }
30.
31.     class Test{
32.         public static void main(String[] args) {
33.             List list = new ArrayList();
34.             list = new MyCollections().synList(list);
35.             list = new MyCollections().synList(list);
36.         }
37.     }
```

复制代码

6.2.7 Arrays工具类

Arrays是集合框架的工具类，里面的方法都是静态的。

示例1：

```
01. import java.util.Arrays;
02.
03. class ArraysDemo{
04.     public static void main(String[] args){
05.         int[] arr = {3,1,5,6,4,7};
06.         System.out.println(Arrays.toString(arr));
07.     }
08. }
```

复制代码

运行结果：

重点：List asList(数组)将数组转成集合。

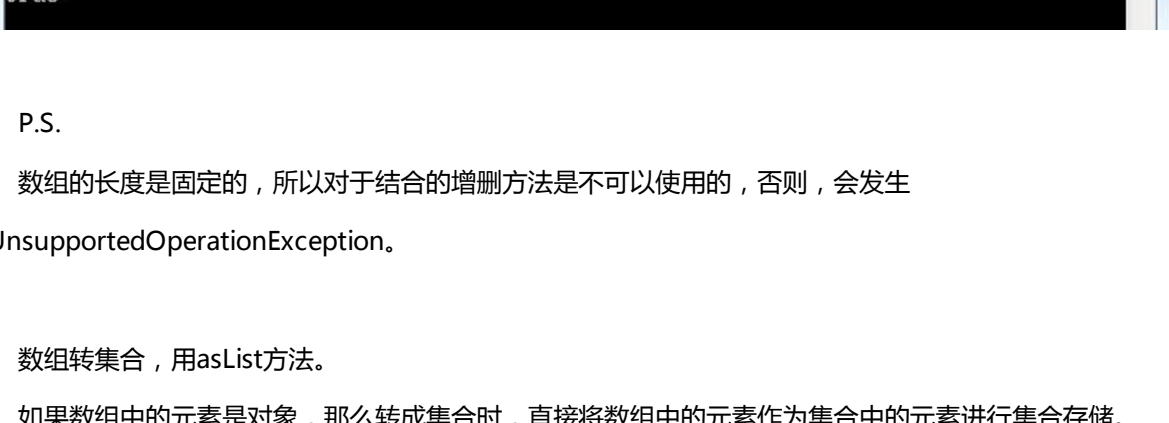
好处：可以使用集合的方法操作数组。

示例2：

```
01. import java.util.Arrays;
02. import java.util.List;
03.
04. class ArraysDemo{
05.     public static void main(String[] args){
06.         String[] arr = { "abc","haha", "xixi" };
07.
08.         List<String> list = Arrays.asList(arr);
09.
10.         boolean b = list.contains("xixi");
11.
12.         System.out.println(b);
13.     }
14. }
```

复制代码

运行结果：



P.S.

数组的长度是固定的，所以对于结合的增删方法是不可以使用的，否则，会发生
UnsupportedOperationException.

数组转集合，用asList方法。

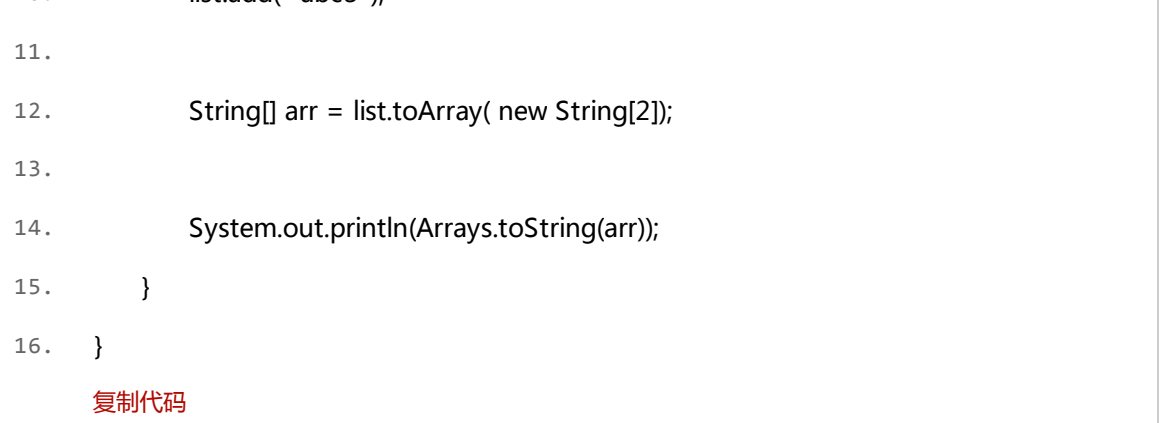
如果数组中的元素是对象，那么转成集合时，直接将数组中的元素作为集合中的元素进行集合存储。
如果数组中的元素是基本类型数值，那么会将该数组作为集合中的元素进行存储。

示例3：

```
01. import java.util.Arrays;
02. import java.util.List;
03.
04. class ArraysDemo{
05.     public static void main(String[] args){
06.         int[] arr1 = {31,11,51,61};
07.         List< int[]> list1 = Arrays.asList(arr1);
08.         System.out.println(list1);
09.
10.         Integer[] arr2 = {31,11,51,61};
11.         List list2 = Arrays.asList(arr2);
12.         System.out.println(list2);
13.     }
14. }
```

复制代码

运行结果：



集合转数组

使用的就是Collection接口中的toArray方法。

集合转成数组，可以对集合中的元素操作的方法进行限定，不允许对其进行增删。

toArray方法需要传入一个指定类型的数组。

长度该如何定义呢？

如果长度小于集合的size，那么该方法会创建一个同类型并和集合相同的size的数组。

如果长度大于集合的size，那么该方法就会使用指定的数组，存储集合中的元素，其他位置默认为null。

所以建议，最后长度就指定为，集合的size。

示例4：

```
01. import java.util.ArrayList;
02. import java.util.Arrays;
03. import java.util.List;
04.
05. public class ToArray{
06.     public static void main(String[] args){
07.         List<String> list = new ArrayList<String>();
08.         list.add("abc1");
09.         list.add("abc2");
10.         list.add("abc3");
11.
12.         String[] arr = list.toArray( new String[2]);
13.
14.         System.out.println(Arrays.toString(arr));
15.     }
16. }
```

复制代码

运行结果：



~END~



~爱上海，爱黑马~

