# CollectionDemo

```
package collection;

import java.util.ArrayList;
import java.util.Collection;

/**
 * JAVA 集合框架:
 * java.util.Collection接口，所有的集合都实现自该接口，
 * 该接口中定义了所有集合都必须具备的相关功能。
 * <p>
 * 集合与数组一样，可以保存一组元素，但是元素的相关操作都封装成了方法。
 * 集合有多种不同的数据结构，可根据实际应用场景选择最优的结构。
 * <p>
 * Collection下面有两个非常常见的子类集合:
 * java.util.List:线性表，是一个可以存放重复元素且有序的集合。
 * java.util.Set:是一个存放不可重复的元素且无序的集合。
 * 元素是否重复取决于相互的equals比较结果。
 * 元素是否有序指add添加的顺序是否与实际存储在集合中的顺序相同。
 */
public class CollectionDemo {
    //集合只能存放引用类型，8种基本类型不行
    //实际上能放8种基本类型是因为，自动装箱了
    public static void main(String[] args) {
```

```
Collection c = new ArrayList(); //向上造型
c.add("one");
c.add("two");
c.add("three");
c.add("four");
c.add("five");

//boolean add(E e)：添加一个元素，成功后返回true，返回值可以不接收
System.out.println(c.add("five")); //List元素可以重复
System.out.println(c);

/*
E是泛型
初步理解泛型：
假定我们有这样一个需求：写一个排序方法，
能够对整型数组、字符串数组甚至其他任何类型的数组进行排序，该如何实现？
答案是可以使用 Java 泛型。
使用 Java 泛型的概念，我们可以写一个泛型方法来对一个对象数组排序。
然后，调用该泛型方法来对整型数组、浮点数数组、字符串数组等进行排序。
 */

//int size()：返回集合的元素个数
//若集合元素个数超过Integer.MAX_VALUE，则返回Integer.MAX_VALUE
//数组的length是数组的长度，不是存了几个元素
//集合的size是存了几个元素，集合长度是不确定的
int size = c.size();
System.out.println("size:" + size);
```

```
        //boolean isEmpty()：判断集合是否为空集（空集：集合存在，但里面没元素）
        //size为0时isEmpty为true
        //集合=null意味着连集合都不存在
        boolean isEmpty = c.isEmpty();
        System.out.println("是否为空集:" + isEmpty);

        //void clear()：清空集合
        c.clear();
        System.out.println(c);
        System.out.println("size:" + c.size());
        System.out.println("是否为空集:" + c.isEmpty());
    }
}
```

# Point

```java
package collection;

import java.util.Objects;

//直角坐标系的任意一个点
public class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    @Override
    public String toString() {
        return "(" + x + "," + y + ")";
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Point point = (Point) o;
```

```
        return x == point.x && y == point.y;
    }

    @Override
    public int hashCode() {
        return Objects.hash(x, y);
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
}
```

# CollectionDemo2

```
package collection;

import java.util.ArrayList;
import java.util.Collection;

//集合受元素equals方法影响的操作
public class CollectionDemo2 {
    public static void main(String[] args) {
        Collection c = new ArrayList();
        c.add(new Point(1, 2));
        c.add(new Point(3, 4));
        c.add(new Point(5, 6));
        c.add(new Point(7, 8));
        c.add(new Point(9, 10));
        c.add(new Point(1, 2));
        //集合重写了toString方法，格式为：[元素1.toString()...]，输出的是地址
        //如果要输出元素的值就要重写元素的toString方法
        System.out.println(c);

        //boolean contains(Object o)：判断集合是否包含元素o
        //判断逻辑：该元素与集合现有元素进行equals比较，若为true则认为包含。
        //集合重写的equals方法默认比较的是地址，所以要比较值就要重写元素的equals方法。
        Point p = new Point(1, 2);
        boolean contains = c.contains(p);
```

```
        System.out.println("集合是否包含元素p:" + contains);

        //remove方法也是删除与给定元素equals比较为true的集合元素
        //但是！对于List集合这种可以存放重复元素的而言，仅会删除第一个equals比较为true的元素
        c.remove(p);
        System.out.println(c);
    }
}
```

# CollectionDemo3

```java
package collection;

import java.util.ArrayList;
import java.util.Collection;

//集合只能存放引用类型元素
//集合都把元素视为Object，不关心其具体类型
//集合存放的是元素的引用（地址）
public class CollectionDemo3 {
    public static void main(String[] args) {
        Collection c = new ArrayList();
        Point p = new Point(1, 2);
        c.add(p);

        System.out.println(p);
        System.out.println(c);

        p.setX(2);
        System.out.println(p);
        System.out.println(c);
    }
}
```

# CollectionDemo4

```java
package collection;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
//集合之间的操作
public class CollectionDemo4 {
    public static void main(String[] args) {
        //ArrayList：重复，有序，元素类型可以不同
        Collection c1 = new ArrayList();
        c1.add("java");
        c1.add("c++");
        c1.add("python");
        System.out.println(c1);

        //HashSet：不重复，无序
        Collection c2 = new HashSet();
        c2.add("android");
        c2.add("ios");
        c2.add("java"); //c1中重复的元素
        System.out.println(c2);

        //将一个集合的元素添加到另外一个集合
        //c1.addAll(c2); //addAll后元素重复了，有序
```

```
        //System.out.println(c1);
        c2.addAll(c1);
        System.out.println(c2);

        Collection c3 = new ArrayList();
        c3.add("java");
        c3.add("c++");

        //判断当前集合是否包含给定集合中的所有元素（真包含，子集）
        boolean containsAll = c1.containsAll(c3);
        System.out.println(containsAll);
        c3.add("php");
        containsAll = c1.containsAll(c3);
        System.out.println(containsAll);

        //contains与containsAll不同，contains是把c3整体当成c1的一个元素看待
        containsAll = c1.contains(c3);
        System.out.println(containsAll);

        System.out.println(c1);
        System.out.println(c3);
        //取交集，c1保留两者共有的元素
        //c1.retainAll(c3);
        //删交集，c1删除两者共有的元素
        c1.removeAll(c3);
        System.out.println(c1);
        System.out.println(c3);
    }
```

```
}
```

# IteratorDemo

```java
package collection;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

/**
 * 集合的遍历：
 * Collection提供了同一的遍历集合的方式：迭代器模式
 * 对应的方法：
 * Iterator iterator()：
 * 该方法会返回一个用于遍历当前集合的迭代器
 * <p>
 * java.util.Iterator迭代器接口：
 * 所有的集合都提供了一个用于遍历自身元素的迭代器实现类，
 * 我们无需记住这些类的名字，只需要把它们当作Iterator去操作即可。
 * 迭代器遍历集合遵循的步骤为：问->取->删，
 * 其中删除元素不是遍历过程中的必要操作。
 */
public class IteratorDemo {
    public static void main(String[] args) {
        Collection c = new ArrayList();
        c.add("one");
        c.add("#");
```

```
c.add("two");
c.add("#");
c.add("three");
c.add("#");
c.add("four");
c.add("#");
c.add("five");
System.out.println(c);

/*
    迭代器中两个重要的方法：
    boolean hasNext()：
    判断当前集合是否还有下一个元素可以遍历。
    注：迭代器默认开始位置为集合第一个元素之前，
    所以第一次调用是判断集合是否有第一个元素可以遍历。

    E next()：
    使迭代器向后移动到下一个元素的位置并获取该元素。
 */
Iterator it = c.iterator();
while (it.hasNext()) {
    //Object obj = it.next();
    //默认是Object类型，适用于任何类型
    //若很明确是某种类型比如String，则可以强转成String
    String str = (String) it.next();
    System.out.println(str);

    //迭代器在遍历的过程中不能通过集合的方式增删元素，否则会抛出异常!
```

```
            //迭代器提供的remove方法可以删除迭代器当前位置表示的集合元素
            if ("#".equals(str)) {
                //c.remove(str); //禁止的操作!
                it.remove(); //删掉后后面会自动补前
            }

            //ConcurrentModificationException：并发修改异常
            //迭代器：快速失败原则
            /*
            1：ConcurrentModificationException异常与modCount这个变量有关。
            2：modCount的作用：
            modCount就是修改次数，在具体的实现类中的Iterator中才会使用。在List集合中，ArrayList是List接口的实现类。
            modCount：表示list集合结构上被修改的次数。（在ArrayList所有涉及结构变化的方法中，都增加了modCount的值）
            list结构上被修改是指：改变了list的长度的大小或者是遍历结果中产生了不正确的结果的方式。
            add()和remove()方法会是modCount进行+1操作。
            modCount被修改后会产生ConcurrentModificationException异常，这是jdk的快速失败原则。
            3：modCount的变量从何而来。
            modCount被定义在ArrayList的父类AbstractList中，初值为0，protected transient int modCount = 0。
             */
        }
        System.out.println(c);
    }
}
```

# NewForDemo

```java
package collection;

import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;

/**
 * JDK5推出了一个新特性：增强for循环，使得我们可以用相同的语法遍历集合或数组。
 * 语法：
 * for(元素类型 变量名 : 遍历的集合或数组){
 * ...
 * }
 * <p>
 * 注意：新循环并不能取代传统for循环的工作，因为他仅用于遍历集合或数组，不能增删元素。
 */
public class NewForDemo {
    public static void main(String[] args) {
        String[] array = {"one", "two", "three", "four", "five"};

        for (int i = 0; i < array.length; i++) {
            String str = array[i];
            System.out.println(str);
        }
```

```java
//编译器会把新循环改为传统的for循环来遍历数组
for (String str : array) {
    System.out.println(str);
}

/*
    泛型：JDK5推出的另一个特性，又称为参数化类型（把类型作为参数传进去）
    泛型在集合中被广泛应用，用来指定该集合的元素类型（相当于约束了集合的元素类型）。
    注：泛型不指定时，使用原型Object。
 */
Collection<String> c = new ArrayList<>();
c.add("one");
c.add("two");
c.add("three");
c.add("four");
c.add("five");
System.out.println(c);

/*
    增强for循环遍历集合会被编译器改回成迭代器遍历，只是为了写法更优雅。
    因此注意：遍历的过程中不能通过集合的方法增删元素，否则会抛出异常。
 */
for (String str : c) {
    System.out.println(str);
}

//迭代器需要指定泛型，具体类型应与其遍历的集合指定的泛型一致
//迭代器指定泛型后，获取元素则无需手动向上造型
```

```
        Iterator<String> it = c.iterator();
        while (it.hasNext()) {
            String str = it.next();
            System.out.println(str);
        }
    }
}
```

# ListDemo

```java
package collection;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * List集合：
 * java.util.List接口：线性表，是可以存放重复元素且有序的集合。
 * 常见实现类：
 * java.util.ArrayList：内部使用数组实现，查询性能更好。
 * java.util.LinkedList：内部使用链表实现，增删性能更好，首尾增删性能最佳。
 * 实际开发中对性能没有特别苛刻的情况下，通常使用ArrayList。
 */
public class ListDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>(); //LinkedList也可
        list.add("one");
        list.add("two");
        list.add("three");
        list.add("four");
        list.add("five");
        System.out.println(list);
```

```java
/*
    List集合提供了一套通过下标操作元素的方法：
    E get(int index)：获取指定下标处的元素
    E set(int index,E e)：将元素e存入下标index处，返回值为该位置原有的元素。（相当于替换元素）
 */
String str = list.get(2);
System.out.println(str);

for (int i = 0; i < list.size(); i++) {
    str = list.get(i);
    System.out.println(str);
}

String old = list.set(2, "six");
System.out.println(list);
System.out.println(old);

//小算法题：
//在不创建新集合的前提下，将集合list的元素反转
//思路：正数位置的元素和倒数位置的元素交换

List<String> list1 = new ArrayList<>();
list1.add("one");
list1.add("two");
list1.add("three");
list1.add("four");
list1.add("five");
```

```
        //作弊做法：
        //Collections.reverse(list1);

        //自写：
        /*
        System.out.println(list1);
        int len = list1.size();
        int i = 0;
        while (i < len - 1 - i) {
            String str1 = list1.get(i);
            String str2 = list1.get(len - 1 - i);
            list1.set(len - 1 - i, str1);
            list1.set(i, str2);
            i++;
        }
        System.out.println(list1);
         */

        System.out.println(list1);
        for (int j = 0; j < list1.size() / 2; j++) {
            String s = list1.get(j);
            s = list1.set(list1.size() - 1 - j, s);
            list1.set(j, s);
        }
        System.out.println(list1);
    }
}
```

# ListDemo2

```java
package collection;

import java.util.ArrayList;
import java.util.List;

//List集合提供了一对重载的add、remove方法，也可以通过下标进行操作
public class ListDemo2 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("one");
        list.add("two");
        list.add("three");
        list.add("four");
        list.add("five");
        System.out.println(list);

        //void add(int index, E e)：将给定元素e插入到指定位置
        list.add(2, "six"); //将six插到下标2的位置，后面元素往后移
        System.out.println(list);

        //E remove(int index)：删除并返回指定位置上的元素
        String old = list.remove(3); //删除下标为3的元素，有返回值
        System.out.println(list);
        System.out.println(old);
```

```
    }
}
```

# ListDemo3

```java
package collection;

import java.util.ArrayList;
import java.util.List;

/**
 * List subList(int start, int end)
 * 获取当前List集合中指定范围的子集。注意，含头不含尾!
 */
public class ListDemo3 {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        for (int i = 0; i < 10; i++) {
            list.add(i);
        }
        System.out.println(list);
        List<Integer> subList = list.subList(3, 8); //含头不含尾!
        System.out.println(subList);

        for (int i = 0; i < subList.size(); i++) {
            subList.set(i, subList.get(i) * 10);
        }
        System.out.println(subList);
        //对子集subList的操作就是对原集合list对应元素的操作!
```

```
        System.out.println(list);

        //利用子集的上述特点删除原集合的元素（用clear()）
        list.subList(2, 9).clear(); //含头不含尾!
        System.out.println(list);
    }
}
```

# CollectionToArrayDemo

```java
package collection;

import java.util.ArrayList;
import java.util.Collection;

/**
 * 集合转换为数组
 * Collection提供了方法：toArray()可以将一个集合转换为数组
 */
public class CollectionToArrayDemo {
    public static void main(String[] args) {
        Collection<String> c = new ArrayList<>();
        c.add("one");
        c.add("two");
        c.add("three");
        c.add("four");
        c.add("five");
        System.out.println(c);

        //Object[] array = c.toArray();
        String[] array = c.toArray(new String[c.size()]);
        //c.size()可替换为比他大的数，比他小的数，0，都可以
        //传过来的数组能用就用，不能用就用自己的，传参是为了知道要创建什么类型的数组
        //若是比他大的数，则把集合转为数组后，后面多余的元素是默认值null
```

```
        //对ArrayList来说，给0会快一点
    }
}
```

# ArrayToListDemo

```java
package collection;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

/**
 * 数组转换为List集合
 * 数组的工具类Arrays提供了一个静态方法asList，可以将一个数组转换为List集合
 * 数组只能转为List，不能转为Set，因为数组是可重复的
 */
public class ArrayToListDemo {
    public static void main(String[] args) {
        //Arrays.asList()
        String[] array = {"one", "two", "three", "four", "five"};
        System.out.println(array); //数组默认输出的是地址
        System.out.println(Arrays.toString(array));
        List<String> list = Arrays.asList(array);
        System.out.println(list);

        //数组转为集合后，对该集合的操作就是对原数组对应的操作
        list.set(1, "six");
        System.out.println(list);
        System.out.println(Arrays.toString(array)); //数组元素也相应改变
```

```
        //因为对该集合的操作就是对原数组对应的操作，而且数组定长，所以集合不支持增删元素操作，会抛出异常
        //java.lang.UnsupportedOperationException（不支持的操作异常）
        //list.add("seven");
        //System.out.println(list);
        //System.out.println(Arrays.toString(array));

        //若要增删集合元素，则要创建一个新集合并包含原集合所有元素
        //可以在创建集合时传一个集合类型的参数过去
        List list2 = new ArrayList(list);
        System.out.println(list2);
        list2.add("seven");
        System.out.println(list2);

        String[] array3 = {"one", "two", "three", "four", "five", "six", "seven"};
        ArrayList<String> list3 = new ArrayList<>(Arrays.asList(array3)); //一步到位的写法
        System.out.println(Arrays.toString(array3));
        System.out.println(list3);
        list3.add("eight");
        System.out.println(Arrays.toString(array3));
        System.out.println(list3);
    }
}
```

# SortListDemo

```java
package collection;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

/**
 * 对List集合的排序：
 * java.util.Collections是集合的工具类，里面提供了很多静态方法便于我们操作集合
 * 其中提供了一个静态方法sort()可以对List集合进行自然排序
 * 注：JDK8时List集合自身也提供了sort()方法，具有同样的效果
 */
public class SortListDemo {
    public static void main(String[] args) {
        List<Integer> list = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            list.add(random.nextInt(100));
        }
        System.out.println(list);
        Collections.sort(list); //注意是Collections!
        System.out.println(list);
    }
```

```
}
```

# SortListDemo2

```
package collection;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

//排序那些元素是自定义类型的集合
public class SortListDemo2 {
    public static void main(String[] args) {
        List<Point> list = new ArrayList<>();
        list.add(new Point(13, 14));
        list.add(new Point(11, 12));
        list.add(new Point(9, 10));
        list.add(new Point(7, 8));
        list.add(new Point(5, 6));
        list.add(new Point(3, 4));
        list.add(new Point(1, 2));
        System.out.println(list);

        /*
            使用该sort方法排序List集合时有一个前提条件：集合元素必须实现了Comparable接口
            sort方法会利用该接口定义的抽象方法对元素两两比较得知大小后进行排序。
            若元素没有实现该接口则编译不通过。
```

```
    该操作具有侵入性：当我们使用一个API时，如果该API反过来要求我们的程序为其修改其它额外地方的代码，
    那么就具有了侵入性，这不利于我们程序的后期维护。
 */

//回调模式
/*
当我们要实现一个通用的排序函数时，事先并不知道其他开发者会用该函数来对哪些类型的元素进行排序，
也就不知道以何种标准来判断这些元素的偏序（大小）关系。
因此，可以要求其他开发者在使用排序函数时，必须提供一个比较函数 compare，
这样我们就可以用 compare 比较待排序元素的大小，
而无需事先知道元素是什么类型，也无需知道 compare 的具体实现。
这里 compare 函数对于排序函数来说，就是回调函数。
 */
Comparator<Point> c = new Comparator<Point>() {
    @Override
    public int compare(Point o1, Point o2) { //以点到原点的距离进行比较
        int len1 = o1.getX() * o1.getX() + o1.getY() * o1.getY();
        int len2 = o2.getX() * o2.getX() + o2.getY() * o2.getY();
        //^运算符不是平方！是异或（相同为0，不同为1）
        //Math中有平方的运算：Math.pow(x,y)：求x的y次方
        return len1 - len2; //从小到大
        //return len2 - len1; //从大到小
        //规则：
        //返回负数，意味着第一个参数比第二个参数小
        //返回0，意味着第一个参数与第二个参数相等
        //返回正数，意味着第一个参数比第二个参数大
    }
```

```
        };
        Collections.sort(list, c);

        //lambda表达式写法
        Collections.sort(list, (o1, o2) -> o1.getX() * o1.getX() + o1.getY() * o1.getY() -
o2.getX() * o2.getX() - o2.getY() * o2.getY());
        System.out.println(list);
    }
}
```

# SortListDemo3

```java
package collection;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class SortListDemo3 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("Java");
        list.add("C++");
        list.add("Python");
        list.add("Php");
        list.add("javascript");
        list.add("c#");
        list.add("kotlin");
        list.add("scala");
        list.add("苍老师");
        list.add("葵司老师");
        list.add("相泽南老师");
        list.add("河北彩花老师");
        list.add("波多野结衣老师");

        //Collections.sort(list); //按首字母的Unicode排序，所以大写字母在前面
```

```
        //Collections.sort(list, (o1, o2) -> o1.length() - o2.length()); //按字符串长度排序
        list.sort((o1, o2) -> o1.length() - o2.length());
        System.out.println(list);
    }
}
```