集合

1 Collection接口

1.1 概述

- 数组和集合都是Java中提供的容器
- 集合: 英文译为 Collection，用来存放对象的容器，集合中可以存放不同类型的对象，并且集合的长度可变。在编程时，常常需要集中存放多个数据，可以使用数组来保存多个对象，但数组长度不可变化，一旦在初始化数组时指定了数组长度，这个数组长度就是不可变的，如果需要保存数量变化的数据，数组就有点无能为力了；为了保存数量不确定的数据，以及保存具有映射关系的数据，Java提供了集合类。集合类主要负责保存、盛装其他数据，因此集合类也被称为容器类。
- 集合和数组的对比:
    - 数组中的元素可以基本类型的值，也可以是对象; 而集合中只能保存对象
    - 数组一旦指定了长度，长度就不能再改变; 而集合的长度是可以随时改变的
    - 往数组中插入元素非常麻烦,需要将插入位置后面的元素往后移动; 或者删除数组中间位置的某一个元素, 需要将删除位置后的元素往前移动; 而如果往集合中插入元素或者删除集合中的某一个元素,直接使用现成的方法操作即可

1.2 集合的继承结构

- 由于需求不同，Java就提供了不同的集合类。这多个集合类的数据结构不同，但是它们都是要提供存储和遍历功能的，我们把它们的共性不断的向上提取，最终就形成了集合的继承体系结构图。
- Collection接口
    - List接口
        - ArrayList类
        - LinkedList类
    - Set接口
        - HashSet类
        - TreeSet类
- 解释说明:
    - Collection集合是所有单值集合的顶层接口, 其中定义了常用的用于操作集合以及集合中元素的方法例如: 添加元素、删除元素、获取元素、获取集合长度、判断功能、将集合转成数组、迭代器遍历元素等功能
    - List是Collection的子接口，特点是其中的元素是有序的(即:元素存入集合时的顺序和取出的顺序一致)可以通过下标访问List中的元素，另,List集合中的元素是可以重复的(包括null)
    - Set也是Collection的子接口，特点是其中的元素是无序(即:元素存入集合时的顺序和取出的顺序不一定一致)无法通过下标访问Set中的元素，另外,Set集合中的元素是不可以重复的
- 学习集合的建议:
    - 学习接口中提供的共性方法
    - 通过实现类创建对象, 调用这些共性方法

1.3 常用方法

```
boolean add(E e) //往集合中添加指定元素e
boolean addAll(Collection c) //将小集合添加到大集合中
boolean isEmpty() //如果集合中没有任何元素(空集合), 返回true
boolean contains(Object o) //如果此集合中包含指定元素o, 则返回true
boolean containsAll(Collection c) //如果此集合中包含指定 集合c 中的所有元素, 则返回true。
int size() 返回集合的大小(元素个数)
boolean remove(Object o) //从集合中删除指定的元素o, 删除成功则返回true
boolean removeAll(Collection c) //删除此集合中那些也包含在指定集合c中的所有元素
boolean retainAll(Collection c) //仅保留此集合中那些也包含在指定集合c中的元素
c1.retainAll(c2) //只保留c1中两个共同的元素 "a","b", 对c2没有影响
void clear() //删除此集合中的所有元素
Object[] toArray() //将此集合转成对象数组
boolean equals(Object o) //比较此 collection 与指定对象是否相等。
Iterator<E> iterator() //返回此集合中所有元素组成的迭代器。
```

1.4 CollectionDemo1

```
package cn.tedu.collection;

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

/**
 * JAVA集合类
 * 集合和数组一样,都是用于保存一组数据,但是集合将元素的操作都封装成了方法,操作更加简便
 * 并且集合提供了很多不同的实现类供我们使用
 * <p>
 * java.util.Collection是所有集合的顶级接口,里面定义了所有集合都必须具备的功能方法
 * 集合的两类常用子类:
 * ①java.util.List: 可以重复的集合,且有序,通常我们称为叫做线性表
 * ②java.util.Set: 不可以重复的集合
 * 上述两个集合都是接口,而元素是否重复取决于equals方法
 */
public class CollectionDemo01 {
    public static void main(String[] args) {
        Collection c = new ArrayList();
        /*
         * boolean add(E e);
         * 添加指定元素到集合中,添加成功返回true,否则返回false
         * 此处的参数E其实是泛型,先理解为是Object类型
         * 集合存储的元素必须要是引用类型
         * 并且一般使用时,集合中存储的元素的类型都是一致的
         */
        c.add("one");
        c.add("two");
        c.add("three");
        c.add("four");
        c.add("five");
        System.out.println(c);
        c.add("six");
        c.add("seven");
        //1是基本类型,此处触发自动装箱特性 int→Integer
        c.add(1);
        c.add(true);
        c.add(new Object());
        System.out.println(c);
        /*
         * int size();
```

```
42            * 返回当前集合中的元素个数
43            * size要区别与数组的length
44            * length表示的数组能存储多少个元素(如果数组中的元素不够长度那么长,但是长度是不会变化的)
45            * size表示集合中现在有几个元素
46            */
47           System.out.println(c.size());
48           /*
49            * boolean isEmpty();
50            * 判断集合是否为一个空集合,空集合等价于集合的size方法返回0
51            */
52           System.out.println("集合是否为空集:" + c.isEmpty());
53           /*
54            * 清空集合,将集合中存储的所有元素删除
55            */
56           System.out.println("开始清空集合!");
57           c.clear();
58           System.out.println("集合是否为空集:" + c.isEmpty());
59           System.out.println(c.size());
60       }
61   }
```

1.5 CollectionDemo2

```
1   package cn.tedu.collection;
2
3   import java.util.ArrayList;
4   import java.util.Collection;
5   import java.util.HashSet;
6
7   public class CollectionDemo02 {
8       public static void main(String[] args) {
9           //Collection c = new ArrayList();
10          Collection c = new HashSet();//不可重复集合,底层也是通过equals方法判断元素是否重复
11          c.add(new Point(1, 2));
12          c.add(new Point(3, 4));
13          //ArrayList是可以存储重复元素的
14          c.add(new Point(3, 4));
15          c.add(new Point(5, 6));
16          c.add(new Point(7, 8));
17          c.add(new Point(9, 0));
18          /*
19           * 集合重写了toString方法
20           * 格式:
21           * [元素1.toString(),元素2.toString(),...]
22           */
23          System.out.println(c);
24          Point p = new Point(3, 4);
25          /*
26           * boolean contains(Object o);
27           * 判断当前集合中是否包含给定的元素
28           * 集合是否包含给定的元素取决于给定的元素和集合中的元素通过equals比较的结果是否为true
29           * Object中提供的equals方法,该方法作用是比较内存地址
30           * 如果希望equals比较的是内容是否相同时,需要重写equals方法
31           * alt+insert→equals and hashcode→Next→Next→Finish
32           */
33          boolean contains = c.contains(p);
34          System.out.println("集合c中是否包含给定的(3,4)点:" + contains);
35          /*
36           * boolean remove(Object o);
```

```
            * 如果集合中存在给定的元素,则删除
            * 底层也是通过equals的比较结果来判断是否存在
            * 会删除最早出现的那一个
            */
            c.remove(p);
            System.out.println(c);
        }
    }
```

1.6 Point

```
package cn.tedu.collection;

import java.util.Objects;

public class Point {
    private int x;
    private int y;

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

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

    //alt+insert→equals and hashcode→Next→Next→Finish
    @Override
    public boolean equals(Object o) {
        /*
         * p2.equals(p);
         * this: 指的调用当前方法的实例 就是p2
         * o: 指的是传递的参数 就是p
         */
        if (this == o) return true;
        /*
         * ①如果传入的对象为空,则没有可比性,直接返回false
         * ②如果p2和p不是同一个类的实例,则直接返回false
         */
        if (o == null || getClass() != o.getClass()) return false;
        //由于需要比较熟悉,而参数是Object,需要向下转换为原类型
        Point point = (Point) o;
        //开始进行两个对象的属性值的比较 p2的x和p的x是否相同,p2的y和p的y是否相同
```

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

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

1.7 CollectionDemo3

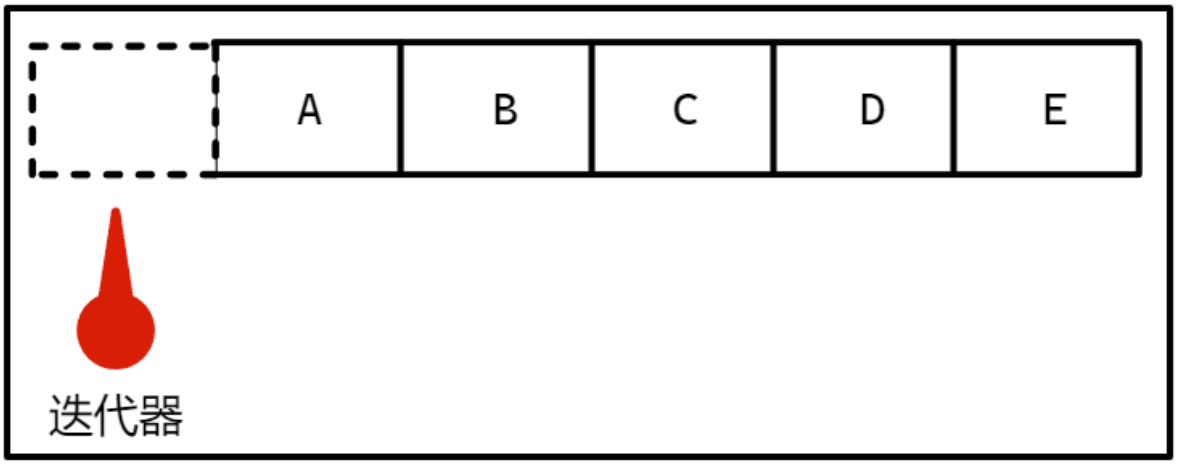
```
package cn.tedu.collection;

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

/**
 * 集合间的操作
 */
public class CollectionDemo03 {
    public static void main(String[] args) {
        Collection c1 = new ArrayList();
        c1.add("大娃");
        c1.add("二娃");
        c1.add("三娃");
        System.out.println("c1 = " + c1);
        Collection c2 = new ArrayList();
        c2.add("四娃");
        c2.add("五娃");
        //c2.add("大娃");
        System.out.println("c2 = " + c2);
        /*
         * boolean addAll(Collection c);
         * 将给定集合中的所有元素都添加到当前集合中(取并集)
         * 操作后,如果当前集合发生了改变,则返回true
         */
        c1.addAll(c2);
        System.out.println("c1 = " + c1);
        System.out.println("c2 = " + c2);
        Collection c3 = new ArrayList();
        c3.add("二娃");
        c3.add("大娃");
        c3.add("七娃");
        System.out.println("c3 = " + c3);
        /*
         * boolean containsAll(Collection c);
         * 判断当前集合中是否包含给定集合中的所有元素
         */
        boolean b = c1.containsAll(c3);
        System.out.println("c1集合是否包含c3集合:" + b);
        /*
         * boolean retainAll(Collection c);
         * 将当前集合中的元素保留和给定集合中的元素相同的部分
         */
        c1.retainAll(c3);
        System.out.println("c1 = " + c1);
        System.out.println("c3 = " + c3);
        c1.add("小明");
        /*
         * boolean removeAll(Collection c);
         * 将当前集合中和给定集合中共有的元素删除
         */
```

```
        c1.removeAll(c3);
        System.out.println("c1 = " + c1);
        System.out.println("c3 = " + c3);
    }
}
```

2 集合的遍历



①获取该集合的迭代器
②迭代器在创建初始,默认位置在要遍历的集合的第一个元素之前
③调用hasNext(),判断当前迭代器所处位置是否有下一个元素
④如果有下一个元素,则调用next()来获取当前迭代器所处位置的下一个元素,并且会将迭代器向后移动一个位置

```
package cn.tedu.collection;

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

/**
 * 集合支持随机访问(根据指定的位置获取对应的元素)
 * 但是Collection层面不支持随机访问 原因是Collection层面没有下标概念
 * 所以Collection选择采取其他的方式,就是迭代器遍历
 * 而迭代器遍历是父类中定义的,所以所有集合本身都是支持迭代器的
 */
public class IteratorDemo {
    public static void main(String[] args) {
        Collection c = new ArrayList();
        c.add("A");
        c.add("B");
        c.add("C");
        c.add("D");
        c.add("E");
        c.add("F");
        System.out.println("c = " + c);
        //①获取迭代器
        Iterator it = c.iterator();
        //②判断当前迭代器所处位置是否有下一个元素
        while (it.hasNext()) {
            //③如果有,则取出下一个元素,并且将迭代器的位置向后移动一个
            Object e = it.next();
            System.out.println(e);
            //④判断遍历的元素是否是D,如果是,则删除
            if ("D".equals(e)) {
                //⑤迭代器中提供的删除方法 remove()
```

```
                    it.remove();
                }
            }
            System.out.println("c = " + c);
        }
    }
```

3 增强型for循环

```
package cn.tedu.collection;

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

/**
 * JDK5推出时,推出了一个新的特性: 增强型for循环
 * 通常也称为新循环
 * 新循环不能取代传统for循环的工作,它的出现仅为了使用相同的格式遍历集合和数组
 * 语法:
 * for(元素类型 e : 集合或数组){
 * <p>
 * }
 */
public class NewForDemo {
    public static void main(String[] args) {
        String[] arr = {"A", "B", "C", "D", "E"};
        System.out.println("==========传统for循环遍历数组==========");
        //arr.fori
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + "\t");
        }
        System.out.println("\r\n==========增强for循环遍历数组==========");
        //arr.for
        for (String s : arr) {
            System.out.print(s + "\t");
        }
        Collection<String> c = new ArrayList();
        c.add("1");
        c.add("2");
        c.add("3");
        c.add("4");
        c.add("5");
        System.out.println("\r\n==========增强for循环遍历集合==========");
        //c.for
        for (String s : c) {
            System.out.println(s + "\t");
        }
    }
}
```

3.1 泛型

```
package cn.tedu.collection;

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

/**
 * 泛型是JDK1.5增加的一个新特性,泛型本质是参数化类型,也就是说,操作的数据类型可以被指定为一个参数,
 * 增加泛型这个概念,主要是为了让集合能记住其元素的数据类型
```

```java
 */
public class GenericsDemo {
    public static void main(String[] args) {
        //声明一个集合,但是没有指定泛型
        Collection c = new ArrayList();
        //集合在没有泛型的约束下,可以装任意的类型
        c.add("任意类型的元素");
        //声明了一个集合,泛型是String类型
        Collection<String> c1 = new ArrayList<>();
        c1.add("hello");
        // c1.add(1);//编译错误,add方法不能添加int类型
        Collection<Integer> c2 = new ArrayList();
        c2.add(1);
        // c2.add("hello"); //编译错误,add方法不能添加String类型
        //在不指定泛型时,泛型其实就是Object类型
        Test test = new Test();
        test.setObj("你好");
        Object obj = test.getObj();
        //声明泛型时,Test类中,所有T的部门都是String类型
        Test<String> test1 = new Test<>();
        test1.setObj("我不好");
        String obj1 = test1.getObj();
    }
}
//①在类名后使用泛型约束 T → Type E → Element K → key V → Value
class Test<T> {
    //②将类中需要参数控制的地方改成泛型
    private T obj;

    public T getObj() {
        return obj;
    }

    public void setObj(T obj) {
        this.obj = obj;
    }
}
```

4 List集合

4.1 概述

- List是一个有序的Collection(List是Collection的子接口)，使用此接口能够精确的控制每个元素插入的位置，能够通过索引(类似于数组的下标)来访问List中的元素，第一个元素的索引为 0，而且允许有相同的元素。
- List 接口存储一组可重复、有序（插入顺序）的对象。

4.2 特点

- 元素有下标,可以通过下标访问元素
- 元素是有序的(存入集合的顺序和取出的顺序一定相同)
- 元素可以重复(包括null)

4.3 List方法测试

4.3.1 ListDemo

```java
package cn.tedu.collection;

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

/**
 * 学习List中定义的方法
 */
public class ListDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("A");
        list.add("B");
        list.add("C");
        list.add("D");
        list.add("E");
        System.out.println("list = " + list);
        //E get(int index) 取出指定下标位置的元素(下标起始是0)
        String str = list.get(2);
        System.out.println("str = " + str);
        //通过下标遍历集合
        for (int i = 0; i < list.size(); i++) {
            str = list.get(i);
            System.out.println("集合中的第" + (i + 1) + "个元素: " + str);
        }
        /*
         * E set(int index, E element);
         * 将给定的元素element设置到指定的index位置上,并将替换的元素返回
         */
        str = list.set(2, "★");
        System.out.println("被替换的元素 = " + str);
        System.out.println("list = " + list);
        //翻转集合 java.util.Collections 集合的工具类
        Collections.reverse(list);
        System.out.println("翻转后的集合: " + list);
        ArrayList<String> list1 = new ArrayList<>();
        Collections.addAll(list1, "1","2","3","4","5");
        System.out.println("list1 = " + list1);
    }
}
```

4.3.2 ListDemo2

```java
package cn.tedu.collection;

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

/**
 * 给指定的下标位置添加元素或者删除元素
 */
public class ListDemo02 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
        list.add("A");
```

```
        list.add("B");
        list.add("C");
        list.add("D");
        list.add("E");
        System.out.println("list = " + list);
        //将元素添加到集合的最后面
        list.add("F");
        //将元素添加到指定的位置
        /*
         * void add(int index, E element);
         * 将给定元素element添加到指定的index位置,然后将原位置的元素整体后移
         */
        list.add(3, "★");
        System.out.println("list = " + list);
        String str = list.remove(3);
        System.out.println("删除了: " + str);
        System.out.println("list = " + list);
    }
}
```

4.4 集合和数组的转换

4.4.1 集合转换数组

```
package cn.tedu.collection;

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

/**
 * 集合转换为数组
 */
public class CollectionToArrayDemo {
    public static void main(String[] args) {
        Collection<String> c = new ArrayList<>();
        c.add("A");
        c.add("B");
        c.add("C");
        c.add("D");
        c.add("E");
        System.out.println("集合c = " + c);
        /*
         * Object[] toArray();
         * 将当前集合转换为一个Object类型的数组
         * 由于返回的是一个Object类型的数组,所以用的很少
         */
        Object[] array = c.toArray();
        /*
         * T[] toArray(T[] a);
         * 将当前集合转换为给定数组类型的数组
         * 其中参数声明的数组有一些要注意的事项:
         * ①定义的数组类型,要和待转换的集合的泛型最好一致
         * ②定义的数组长度,最好和待转换的集合的存储的元素个数一致
         */
        String[] array2 = c.toArray(new String[c.size()]);
        System.out.println("数组的长度:" + array2.length);
        System.out.println("数组array2 = " + Arrays.toString(array2));
    }
}
```

4.4.2 数组转换集合

```java
package cn.tedu.collection;

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

/**
 * 数组转换为集合
 */
public class ArrayToListDemo {
    public static void main(String[] args) {
        String[] array = {"A", "B", "C", "D", "E"};
        System.out.println("数组array = " + Arrays.toString(array));
        List<String> list = Arrays.asList(array);
        System.out.println("集合list = " + list);
    }
}
```

4.5 集合的排序

4.5.1 SortListDemo

```java
package cn.tedu.collection;

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

/**
 * 集合的排序
 * 集合的工具类Collections中提供了一个静态的sort方法,可以对集合中的元素进行自然排序
 */
public class SortDemo01 {
    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("乱序list = " + list);
        Collections.sort(list);//自然排序,从小到大
        System.out.println("自然排序list = " + list);
    }
}
```

4.5.2 SortListDemo2

```java
package cn.tedu.collection;

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

/**
 * 集合的自定义排序
 */
```

```
public class SortDemo02 {
    public static void main(String[] args) {
        List<Point> list = new ArrayList<>();
        list.add(new Point(1, 4));
        list.add(new Point(3, 12));
        list.add(new Point(2, 5));
        list.add(new Point(0, 8));
        list.add(new Point(3, 3));
        list.add(new Point(9, 2));
        System.out.println("乱序: " + list);
        /*
         * Collections提供的sort方法,如果对集合进行排序,必要要求集合中的元素要实现Comparable接口,
         * 该接口中的compareTo方法定义排序规则
         * sort方法会自动将集合中的两个元素进行比较,而比较时,会通过重写的compareTo方法比较,
         * 格式
         * A.compareTo(B) A就是调用该方法的集合的参数 B就是和A比较的集合的参数
         * A 大于 B 返回正数
         * A 等于 B 返回0
         * A 小于 B 返回负数
         */
        Collections.sort(list);
        System.out.println("正序:" + list);
    }
}
```

4.5.3 SortListDemo3

```
package cn.tedu.collection;

import java.util.*;

/**
 * 集合的自定义排序
 */
public class SortDemo02 {
    public static void main(String[] args) {
        List<Point> list = new ArrayList<>();
        list.add(new Point(1, 4));
        list.add(new Point(3, 12));
        list.add(new Point(2, 5));
        list.add(new Point(0, 8));
        list.add(new Point(3, 3));
        list.add(new Point(9, 2));
        System.out.println("乱序: " + list);
        /*
         * 内部比较器
         * Collections提供的sort方法,如果对集合进行排序,必要要求集合中的元素要实现Comparable接口,
         * 该接口中的compareTo方法定义排序规则
         * sort方法会自动将集合中的两个元素进行比较,而比较时,会通过重写的compareTo方法比较,
         * 格式
         * A.compareTo(B) A就是调用该方法的集合的参数 B就是和A比较的集合的参数
         * A 大于 B 返回正数
         * A 等于 B 返回0
         * A 小于 B 返回负数
         * 但是上述的方式具有侵入性,为了调用该API,反而要求我们去修改其他的代码,导致代码结构出现混乱,
         * 就造成了侵入性,侵入性不利于后期维护,尽可能的避免
         */
```

```
        //Collections.sort(list);
        /*
         * 外部比较器
         * 创建Comparator比较器实例,然后重写compare方法,定义比较规则
         * 然后将比较器实例传入到sort方法的第二个参数中,sort方法就会自动根据该比较器定义的规则,进行排序
         * 格式
         * int compare(Point o1, Point o2)
         * o1 大于 o2 返回正数
         * o1 等于 o2 返回0
         * o1 小于 o2 返回负数
         */
        Comparator<Point> com = new Comparator<Point>() {
            /**
             * 定义比较规则
             */
            @Override
            public int compare(Point o1, Point o2) {
                //比较y坐标的大小,y坐标越大,该点就越大
                int y1 = o1.getY();
                int y2 = o2.getY();
                return y1 - y2;
            }
        };
        Collections.sort(list, com);
        System.out.println("正序:" + list);
    }
}
```

4.5.4 Point

```
package cn.tedu.collection;

import java.util.Objects;

public class Point implements Comparable<Point> {
    private int x;
    private int y;

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

    @Override
    public String toString() {
```

```
        return "(" + x + ", " + y + ")";
    }

    //alt+insert→equals and hashcode→Next→Next→Finish
    @Override
    public boolean equals(Object o) {
        /*
         * p2.equals(p);
         * this: 指的调用当前方法的实例 就是p2
         * o: 指的是传递的参数 就是p
         */
        if (this == o) return true;
        /*
         * ①如果传入的对象为空,则没有可比性,直接返回false
         * ②如果p2和p不是同一个类的实例,则直接返回false
         */
        if (o == null || getClass() != o.getClass()) return false;
        //由于需要比较熟悉,而参数是Object,需要向下转换为原类型
        Point point = (Point) o;
        //开始进行两个对象的属性值的比较 p2的x和p的x是否相同,p2的y和p的y是否相同
        return x == point.x && y == point.y;
    }

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

    /**
     * 定义排序规则
     *
     * @param o
     * @return
     */
    @Override
    public int compareTo(Point o) {
        //定义比教两个点的x坐标
        int x1 = this.getX();
        int x2 = o.getX();
        // //判断x1大于x2说明x1大
        // if (x1 - x2 > 0) {
        //     return 1;
        // } else if (x1 - x2 < 0) {//说明x2大
        //     return -1;
        // } else {//x1和x2相等
        //     return 0;
        // }
        return x1 - x2;
    }
}
```

5 Map接口

5.1 概述

- Map用于保存具有映射关系的数据，因此Map集合里保存着两组值，一组值用于保存Map里的键(key)另外一组值用于保存Map里的值(value)，键和值是一一对应的关系，称为映射。根据键就能找到对应的值，类似于生活中一张身份证对应一个人一样。
- Map的key和value可以是任何引用类型的数据，其中key不允许重复，同一个Map对象的任何两个key通过equals方法比较总是返回false。

5.2 特点

- Map集合中每个元素都有两个值, 分别是key(键) 和 value(值)
- Map集合中的key(键)不允许重复, 在第二次添加已有的key时, value会被会覆盖
- Map集合中的元素是无序的(即元素存入集合的顺序和取出时的顺序很可能不一致)
- Map集合中的key和value具有映射关系, 可以通过key(键)来获取对应的value(值)
    - key和value之间存在单向一对一关系，即通过指定的key，总能找到唯一的、确定的value。从Map中取出数据时，只要给出指定的key，就可以取出对应的value。
    - 如果把Map里的所有key放在一起来看，它们就组成了一个Set集合（所有的key没有顺序，key与key之间不能重复），实际上Map确实包含了一个keySet()方法，用于返回Map里所有key组成的Set集合。
    - 如果把Map里的所有value放在一起来看，它们又非常类似于一个List：元素与元素之间可以重复，每个元素可以根据索引来查找，只是Map中的索引(也就是key)不是从0开始的整数值，而是任意类型的对象；
    - 如果需要从List集合中取出元素，则需要提供该元素的数字索引；如果需要从Map中取出元素，则需要提供该元素的key。因此，Map有时也被称为字典，或关联数组。

5.3 继承结构

- Map接口
    - HashMap类
    - TreeMap类
    - Hashtable类
- 解释说明:
    - Map集合是采用键-值对(key-value)的存储方式, 键(key)、值(value)可以是引用类型的数据, key不允许重复, vaue可以重复, key和value是一对一的关系, 通过指定的key总能找到唯一的、确定的value值
    - HashMap 和 Hashtable 都是Map接口的实现类，它们之间的关系完全类似于ArrayList和Vector的关系
        - HashMap是线程不安全的, 所以HashMap的性能要比HashTable高一些
        - HashMap可以使用null作为key或value, Hashtable不允许使用null作为key和value;
        - Hashtable是一个古老的Map实现类，JDK 1.0出现，出现时，Java还没有提供Map接口，命名没有遵守Java的命名规范，与Vector类似的是，尽量少用Hashtable实现类，即使需要创建线程安全的Map实现类，也无须使用Hashtable实现类，可以通过别的方式来解决线程安全问题。
    - TreeMap是Map的子接口SortedMap的的实现类, 是可以支持对内部元素进行排序的类, 也正因为如此, TreeMap的执行效率通常要比HashMap和HashTable慢。

5.4 MapDemo

```
1  package cn.tedu.map;
2
3  import java.util.HashMap;
4  import java.util.LinkedHashMap;
5  import java.util.Map;
6
7  /**
```

```java
 * java.util.Map接口 查找表
 * Map体现的结构就是多行两列的表格,左列称为"key",右列称为"value"
 * Map总是以key-value的形式保存一组数据,并且可以根据key获取对应的value
 */
public class MapDemo01 {
    public static void main(String[] args) {
        //LinkedHashMap 记录存储的顺序,更占用内存
        //HashMap不会记录存储的顺序
        Map<String, Integer> map = new HashMap<>();
        /*
         * V put(K key, V value);
         * 向当前map存储一组键值对,
         * 如果存储的键值对在map不存在,则返回null,
         * 如果存储的键值对在map中已存在,则会覆盖原先的键值对,并且将被覆盖的value作为返回值返回
         */
        map.put("语文", 90);
        map.put("数学", 99);
        map.put("物理", 80);
        map.put("化学", 85);
        System.out.println(map);
        //Map中的value是允许重复的
        Integer score = map.put("英语", 80);
        System.out.println(map);
        System.out.println(score);
        //Map中的key是不允许重复的
        score = map.put("数学", 100);
        System.out.println(score);
        System.out.println(map);
        /*
         * V get(Object key);
         * 根据给定的key,获取对应的value
         * 如果给定的key不存在,则返回null
         */
        score = map.get("数学");//获取数学成绩,将数学的value返回
        System.out.println(score);
        score = map.get("体育");//获取体育成绩,体育不存在,返回null
        System.out.println(score);
        int size = map.size();
        System.out.println("map中包含" + size + "个元素!");
        /*
         * V remove(Object key);
         * 删除给定的key的这组键值对,如果删除成功,会将删除的value返回,
         * 如果删除的key不存在,则直接返回null
         */
        score = map.remove("数学");
        System.out.println(score);
        score = map.remove("体育");
        System.out.println(score);
        boolean k = map.containsKey("物理");
        if (k) {
            System.out.println("包含'物理'这个key");
        } else {
            System.out.println("不包含'物理'这个key");
        }
        boolean v = map.containsValue(105);
        if (v) {
            System.out.println("包含'105'这个value");
        } else {
            System.out.println("不包含'105'这个value");
        }
    }
}
```

5.5 MapDemo2

```java
package cn.tedu.map;

import java.util.*;
import java.util.function.BiConsumer;
import java.util.function.Consumer;

/**
 * Map的遍历
 * Map的三种遍历方式
 * ①单独遍历key
 * ②单独遍历value(基本不用)
 * ③遍历每一组键值对
 * <p>
 * JDK8之后,集合和Map都支持了基于Lambda表达式的遍历
 */
public class MapDemo02 {
    public static void main(String[] args) {
        Map<String, Integer> map = new HashMap<>();
        map.put("语文", 90);
        map.put("数学", 99);
        map.put("物理", 80);
        map.put("化学", 85);
        System.out.println(map);
        System.out.println("------------------单独遍历key------------------");
        //将map中所有的key值都存在Set集合中
        Set<String> keySet = map.keySet();
        for (String k : keySet) {
            System.out.println("k = " + k);
        }
        System.out.println("------------------单独遍历value------------------");
        //将map中的所有value都存在Collection集合中
        Collection<Integer> values = map.values();
        for (Integer value : values) {
            System.out.println("value = " + value);
        }
        System.out.println("------------------遍历每一组键值对------------------");
        //将map中的所有的entry存储在Set集合中 entry就是键值对实例对象
        Set<Map.Entry<String, Integer>> entries = map.entrySet();
        for (Map.Entry<String, Integer> entry : entries) {
            String key = entry.getKey();
            Integer value = entry.getValue();
            System.out.println(key + " = " + value);
        }
        System.out.println("------------------Lambda表达式遍历Map------------------");
        map.forEach(
                (k, v) -> System.out.println(k + " = " + v)
        );
        System.out.println("------------------Lambda表达式遍历集合------------------");
        Collection<String> c = new ArrayList<>();
        c.add("A");
        c.add("B");
        c.add("C");
        c.add("D");
        c.forEach(
```

```
                (e) -> System.out.println(e)
        );
        //参数只有一个时,可以省略小括号
        c.forEach(
                e -> System.out.println(e)
        );
        //如果输出的参数和传入的参数是同一个时,就可以省略参数,替换::
        c.forEach(
                System.out::println
        );
    }
}
```