day04

集合与数组的转换

集合转换为数组

Collection提供了一个方法:**toArray**,可以将当前集合转换为一个数组

```
package collection;

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

/**
 * 集合转换为数组
 * Collection提供了方法toArray可以将当前集合转换为一个数组
 */
public class CollectionToArrayDemo {
    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);

//        Object[] array = list.toArray();
        /*
            重载的toArray方法要求传入一个数组，内部会将集合所有元素存入该数组
            后将其返回（前提是该数组长度>=集合的size）。如果给定的数组长度不足，
            则方法内部会自行根据给定数组类型创建一个与集合size一致长度的数组并
            将集合元素存入后返回。
         */
        String[] array = list.toArray(new String[list.size()]);
        System.out.println(array.length);
        System.out.println(Arrays.toString(array));
    }
}
```

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一个List集合

```
package collection;

import java.util.ArrayList;
import java.util.Arrays;
```

```java
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，可以将数组转换为一个List集合。
 */
public class ArrayToListDemo {
    public static void main(String[] args) {
        String[] array = {"one","two","three","four","five"};
        System.out.println(Arrays.toString(array));
        List<String> list = Arrays.asList(array);
        System.out.println(list);

        list.set(1,"six");
        System.out.println(list);
        //数组跟着改变了。注意:对数组转换的集合进行元素操作就是对原数组对应的操作
        System.out.println(Arrays.toString(array));

        /*
            由于数组是定长的，因此对该集合进行增删元素的操作是不支持的，会抛出
            异常:java.lang.UnsupportedOperationException
         */
//        list.add("seven");

        /*
            若希望对集合进行增删操作，则需要自行创建一个集合，然后将该集合元素
            导入。
         */
//        List<String> list2 = new ArrayList<>();
//        list2.addAll(list);
        /*
            所有的集合都支持一个参数为Collection的构造方法，作用是在创建当前
            集合的同时包含给定集合中的所有元素
         */
        List<String> list2 = new ArrayList<>(list);
        System.out.println("list2:"+list2);
        list2.add("seven");
        System.out.println("list2:"+list2);
    }
}
```

集合的排序

java.util.Collections类

Collections是集合的工具类,里面定义了很多静态方法用于操作集合.

Collections.sort(List list)方法

可以对List集合进行自然排序(从小到大)

```java
package 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:java.util.Collections提供了一个静态方法sort,可以对List集合
 * 进行自然排序
 */
public class SortListDemo1 {
    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);
        System.out.println(list);
    }
}
```

java.util.Collections类

Collections是集合的工具类,里面定义了很多静态方法用于操作集合.

Collections.sort(List list)方法

可以对List集合进行自然排序(从小到大)

```
package 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:java.util.Collections提供了一个静态方法sort,可以对List集合
 * 进行自然排序
 */
public class SortListDemo1 {
    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);
        System.out.println(list);
    }
}
```

排序自定义类型元素

```
package collection;

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

/**
 * 排序自定义类型元素
 */
public class SortListDemo2 {
    public static void main(String[] args) {
        List<Point> list = new ArrayList<>();
        list.add(new Point(3,5));
        list.add(new Point(7,9));
        list.add(new Point(1,1));
        list.add(new Point(8,3));
        list.add(new Point(2,6));
        System.out.println(list);
        /*
            编译不通过的原因:
            Collections.sort(List list)该方法要求集合中的元素类型必须实现接口:
            Comparable,该接口中有一个抽象方法compareTo,这个方法用来定义元素之间比较
            大小的规则.所以只有实现了该接口的元素才能利用这个方法比较出大小进而实现排序
            操作.
         */
        Collections.sort(list);//编译不通过 compare比较  comparable可以比较的
        System.out.println(list);
    }
}
```

实际开发中,我们并不会让我们自己定义的类(如果该类作为集合元素使用)去实现Comparable接口,因为这对我们的程序有**侵入性**.

侵入性:当我们调用某个API功能时,其要求我们为其修改其他额外的代码,这个现象就是侵入性.侵入性越强的API越不利于程序的后期可维护性.应当尽量避免.

重载的Collections.sort(List list,Comparator c)方法

```
package collection;

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

/**
 * 排序自定义类型元素
 */
public class SortListDemo2 {
    public static void main(String[] args) {
        List<Point> list = new ArrayList<>();
```

```
        list.add(new Point(3,5));
        list.add(new Point(7,9));
        list.add(new Point(1,1));
        list.add(new Point(8,3));
        list.add(new Point(2,6));
        System.out.println(list);
        /*
            编译不通过的原因:
            Collections.sort(List list)该方法要求集合中的元素类型必须实现接口:
            Comparable,该接口中有一个抽象方法compareTo,这个方法用来定义元素之间比较
            大小的规则.所以只有实现了该接口的元素才能利用这个方法比较出大小进而实现排序
            操作.

            当我们调用某个API时,它反过来要求我们为其修改其他额外的代码时就是侵入性.
            侵入性不利于程序后期的维护,应当在实际开发中尽量避免.
         */
//        Collections.sort(list);//编译不通过 compare比较  comparable可以比较的

        /*
            Collections.sort(List list,Comparator c)
            重载的sort方法要求我们再传入一个Comparator"比较器",该比较器用来为集合元素
            临时定义一种比较规则,从而将List集合中的元素通过该比较器比较大小后进行排序.
            Comparator是一个接口,实际应用中我们需要实现该接口为集合元素提供比较规则.
         */
        Comparator<Point> c = new Comparator<Point>() {
            /**
             * compare方法用来定义两个参数o1,o2的大小关系
             * 返回值用来表示o1与o2的大小关系
             * 当返回值>0时,应当表示的含义是o1>o2
             * 当返回值<0时,表示o1<o2
             * 当返回值=0时,表示o1与o2相等
             */
            public int compare(Point o1, Point o2) {
                int olen1 = o1.getX()*o1.getX()+o1.getY()*o1.getY();
                int olen2 = o2.getX()*o2.getX()+o2.getY()*o2.getY();
                return olen1-olen2;
            }
        };
        Collections.sort(list,c);
        System.out.println(list);
    }
}
```

最终没有侵入性的写法

```
package collection;

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

/**
```

```
 * 排序自定义类型元素
 */
public class SortListDemo2 {
    public static void main(String[] args) {
        List<Point> list = new ArrayList<>();
        list.add(new Point(3,5));
        list.add(new Point(7,9));
        list.add(new Point(1,1));
        list.add(new Point(8,3));
        list.add(new Point(2,6));
        System.out.println(list);
        /*
            编译不通过的原因:
            Collections.sort(List list)该方法要求集合中的元素类型必须实现接口:
            Comparable,该接口中有一个抽象方法compareTo,这个方法用来定义元素之间比较
            大小的规则.所以只有实现了该接口的元素才能利用这个方法比较出大小进而实现排序
            操作.

            当我们调用某个API时,它反过来要求我们为其修改其他额外的代码时就是侵入性.
            侵入性不利于程序后期的维护,应当在实际开发中尽量避免.
         */
//        Collections.sort(list);//编译不通过 compare比较  comparable可以比较的

        /*
            Collections.sort(List list,Comparator c)
            重载的sort方法要求我们再传入一个Comparator"比较器",该比较器用来为集合元素
            临时定义一种比较规则,从而将List集合中的元素通过该比较器比较大小后进行排序.
            Comparator是一个接口,实际应用中我们需要实现该接口为集合元素提供比较规则.
         */
//        Comparator<Point> c = new Comparator<Point>() {
//            /**
//             * compare方法用来定义两个参数o1,o2的大小关系
//             * 返回值用来表示o1与o2的大小关系
//             * 当返回值>0时,应当表示的含义是o1>o2
//             * 当返回值<0时,表示o1<o2
//             * 当返回值=0时,表示o1与o2相等
//             */
//            public int compare(Point o1, Point o2) {
//                int olen1 = o1.getX()*o1.getX()+o1.getY()*o1.getY();
//                int olen2 = o2.getX()*o2.getX()+o2.getY()*o2.getY();
//                return olen1-olen2;
//            }
//        };
//        Collections.sort(list,c);

        Collections.sort(list,new Comparator<Point>() {
            public int compare(Point o1, Point o2) {
                int olen1 = o1.getX()*o1.getX()+o1.getY()*o1.getY();
                int olen2 = o2.getX()*o2.getX()+o2.getY()*o2.getY();
                return olen1-olen2;
            }
        });
        System.out.println(list);
    }
}
```

排序字符串

java中提供的类,如:String,包装类都实现了Comparable接口,但有时候这些比较规则不能满足我们的排序需求时,同样可以临时提供一种比较规则来进行排序.

```
package collection;

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

public class SortListDemo3 {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
//        list.add("Tom");
//        list.add("jackson");
//        list.add("rose");
//        list.add("jill");
//        list.add("ada");
//        list.add("hanmeimei");
//        list.add("lilei");
//        list.add("hongtaoliu");
//        list.add("Jerry");

        list.add("传奇老师");
        list.add("王克晶");
        list.add("刘桑");
        System.out.println(list);

        //按照字符多少排序
//        Collections.sort(list);
//        Collections.sort(list, new Comparator<String>() {
//            public int compare(String o1, String o2) {
////                return o1.length()-o2.length();
//                return o2.length()-o1.length();//反过来减就是降序
//            }
//        });

        Collections.sort(list,(o1,o2)->o2.length()-o1.length());
        System.out.println(list);
    }
}
```

lambda表达式

Lambda表达式-JDK8之后推出的新特性

- 语法:

```
(参数列表)->{
    方法体
}
```

- 当使用匿名内部类创建时，如果实现的接口只有一个抽象方法，则可以使用lambda表达式代替，使代码更简洁优雅。
- 在java中可以使用lambda表达式代替匿名内部类创建所需要实现的接口时，该接口上都有一个注解:**@FunctionalInterface**

lambda创建比较器用于排序集合

```
public class LambdaDemo {
    public static void main(String[] args) {
        //自定义比较器的匿名内部类写法
        Comparator<String> c1 = new Comparator<String>() {
            public int compare(String o1, String o2) {
                return o1.length()-o2.length();
            }
        };
        //使用lambda表达式
        //lambda表达式就是省去了匿名内部类创建是接口与方法名部分
        Comparator<String> c2 = (String o1, String o2)->{
                return o1.length()-o2.length();
        };

        //lambda表达式可以忽略参数的类型
        Comparator<String> c3 = (o1,o2)->{
            return o1.length()-o2.length();
        };

        //如果方法中只有一句代码时，该方法体的"{}"可以忽略不写
        //如果这句代码含有return关键字时，也要一同忽略return
        Comparator<String> c4 = (o1,o2)->o1.length()-o2.length();


        List<String> list = new ArrayList<>();
        list.add("王克晶");
        list.add("传奇老师");
        list.add("刘桑");
        //lambda表达式实际上是编译器认可的，最终会被改回为内部类方式创建
        //源代码中使用lambda可以更突出重点-原匿名内部类中重写方法的逻辑。
//        Collections.sort(list,(o1,o2)->o1.length()-o2.length());
        /*
            JDK8之后，List集合自己推出了一个sort方法，可以排序自身元素
            并且需要传入一个比较器来定义比较规则。
         */
        list.sort((o1,o2)->o1.length()-o2.length());
        System.out.println(list);
    }
}
```

基于lambda表达式的集合遍历

```
package collection;

import java.util.ArrayList;
import java.util.Collection;
import java.util.function.Consumer;

/**
 * JDK8之后，java在集合Collection接口中添加了一个用于遍历集合元素的forEach
 * 方法。可以基于lambda表达式遍历集合元素。
 */
public class ForEachDemo {
    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");
        //新循环方式(迭代器方式)
        for(String e : c){
            System.out.println(e);
        }

        //tips:当lambda中只有一个参数时，参数列表的"()"可以忽略不写
        c.forEach(e->System.out.println(e));
        /*
            JDK8中出现的lambda表达式的变种写法:方法引用
            对象::方法
            当lambda表达式的参数与方法体中调用方法的参数一致时
            例如:
            (e)->System.out.println(e);
            那么就可以写作:
            System.out::println;
                  对象::方法
            现在以了解为主即可
         */
        c.forEach(System.out::println);


    }
}
```

File类

File类的每一个实例可以表示硬盘(文件系统)中的一个文件或目录(实际上表示的是一个抽象路径)

使用File可以做到:

- 1:访问其表示的文件或目录的属性信息,例如:名字,大小,修改时间等等
- 2:创建和删除文件或目录

- 3:访问一个目录中的子项

但是File不能访问文件数据.

```
public class FileDemo {
    public static void main(String[] args) {
        //使用File访问当前项目目录下的demo.txt文件
        /*
            创建File时要指定路径，而路径通常使用相对路径。
            相对路径的好处在于有良好的跨平台性。
            "./"是相对路径中使用最多的，表示"当前目录"，而当前目录是哪里
            取决于程序运行环境而定，在idea中运行java程序时，这里指定的
            当前目录就是当前程序所在的项目目录。
         */
//        File file = new File("c:/xxx/xxx/xx/xxx.txt");
        File file = new File("./demo.txt");
        //获取名字
        String name = file.getName();
        System.out.println(name);
        //获取文件大小(单位是字节)
        long len = file.length();
        System.out.println(len+"字节");
        //是否可读可写
        boolean cr = file.canRead();
        boolean cw = file.canWrite();
        System.out.println("是否可读:"+cr);
        System.out.println("是否可写:"+cw);
        //是否隐藏
        boolean ih = file.isHidden();
        System.out.println("是否隐藏:"+ih);

    }

}
```

创建一个新文件

createNewFile()方法，可以创建一个新文件

```
package file;

import java.io.File;
import java.io.IOException;

/**
 * 使用File创建一个新文件
 */
public class CreateNewFileDemo {
    public static void main(String[] args) throws IOException {
        //在当前目录下新建一个文件:test.txt
        File file = new File("./test.txt");
        //boolean exists()判断当前File表示的位置是否已经实际存在该文件或目录
        if(file.exists()){
            System.out.println("该文件已存在!");
        }else{
```

```
            file.createNewFile();//将File表示的文件创建出来
            System.out.println("文件已创建!");
        }

    }
}
```

删除一个文件

delete()方法可以将File表示的文件删除

```
package file;

import java.io.File;

/**
 * 使用File删除一个文件
 */
public class DeleteFileDemo {
    public static void main(String[] args) {
        //将当前目录下的test.txt文件删除
        /*
            相对路径中"./"可以忽略不写，默认就是从当前目录开始的。
         */
        File file = new File("test.txt");
        if(file.exists()){
            file.delete();
            System.out.println("文件已删除!");
        }else{
            System.out.println("文件不存在!");
        }
    }
}
```

创建目录

mkDir():创建当前File表示的目录

mkDirs():创建当前File表示的目录，同时将所有不存在的父目录一同创建

```
package file;

import java.io.File;

/**
 * 使用File创建目录
 */
public class MkDirDemo {
    public static void main(String[] args) {
        //在当前目录下新建一个目录:demo
//        File dir = new File("demo");
        File dir = new File("./a/b/c/d/e/f");

        if(dir.exists()){
```

```
            System.out.println("该目录已存在!");
        }else{
//            dir.mkdir();//创建目录时要求所在的目录必须存在
            dir.mkdirs();//创建目录时会将路径上所有不存在的目录一同创建
            System.out.println("目录已创建!");
        }
    }
}
```

删除目录

delete()方法可以删除一个目录，但是只能删除空目录。

```
package file;

import java.io.File;

/**
 * 删除一个目录
 */
public class DeleteDirDemo {
    public static void main(String[] args) {
        //将当前目录下的demo目录删除
        File dir = new File("demo");
//        File dir = new File("a");
        if(dir.exists()){
            dir.delete();//delete方法删除目录时只能删除空目录
            System.out.println("目录已删除!");
        }else{
            System.out.println("目录不存在!");
        }
    }
}
```