day01

集合

List集

java.util.List接口,继承自Collection.

List集合是可重复集,并且有序,提供了一套可以通过下标操作元素的方法

常用实现类:

- 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,首尾增删元素性能更好.

List集合常见方法

get()与set()

```
package collection;

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

/**
 *  List集合
 *  List是Collection下面常见的一类集合。
 *  java.util.List接口是所有List的接口，它继承自Collection。
 *  常见的实现类:
 *  java.util.ArrayList:内部由数组实现，查询性能更好。
 *  java.util.LinkedList:内部由链表实现，增删性能更好。
 *
 *  List集合的特点是:可以存放重复元素，并且有序。其提供了一套可以通过下标
 *  操作元素的方法。
 */
public class ListDemo {
    public static void main(String[] args) {
        List<String> list = new ArrayList<>();
//        List<String> list = new LinkedList<>();

        list.add("one");
        list.add("two");
        list.add("three");
        list.add("four");
        list.add("five");

        /*
            E get(int index)
            获取指定下标对应的元素
         */
        //获取第三个元素
        String e = list.get(2);
```

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

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

        /*
            E set(int index,E e)
            将给定元素设置到指定位置，返回值为该位置原有的元素。
            替换元素操作
         */
        //[one,six,three,four,five]
        String old = list.set(1,"six");
        System.out.println(list);
        System.out.println("被替换的元素是:"+old);
    }
}
```

重载的add()和remove()

```
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
 */
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)
            将给定元素插入到指定位置
         */
        //[one,two,six,three,four,five]
        list.add(2,"six");
        System.out.println(list);

        /*
            E remove(int index)
            删除并返回指定位置上的元素
         */
        //[one,six,three,four,five]
        String e = list.remove(1);
        System.out.println(list);
```

```
        System.out.println("被删除的元素:"+e);
    }
}
```

subList()方法

```
package collection;

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

/**
 *  List subList(int start,int end)
 *  获取当前集合中指定范围内的子集。两个参数为开始与结束的下标(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);
        //获取3-7这部分
        List<Integer> subList = list.subList(3,8);
        System.out.println(subList);
        //将子集每个元素扩大10倍
        for(int i=0;i<subList.size();i++){
            subList.set(i,subList.get(i) * 10);
        }
        //[30,40,50,60,70
        System.out.println(subList);
        /*
            对子集元素的操作就是对原集合对应元素的操作
         */
        System.out.println(list);

        //删除list集合中的2-8
        list.subList(2,9).clear();
        System.out.println(list);

    }
}
```

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)

```
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(1,2));
        list.add(new Point(97,88));
        list.add(new Point(7,6));
        list.add(new Point(9,9));
        list.add(new Point(5,4));
        list.add(new Point(2,3));
        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(1,2));
        list.add(new Point(97,88));
        list.add(new Point(7,6));
        list.add(new Point(9,9));
        list.add(new Point(5,4));
        list.add(new Point(2,3));

        System.out.println(list);
        /*
            Collections.sort(List list)在排序List集合时要求集合元素必须实现了
            Comparable接口。实现了该接口的类必须重写一个方法compareTo用与定义比较
            大小的规则，从而进行元素间的比较后排序。否则编译不通过。

            侵入性:
            当我们调用某个API时，其反过来要求我们为其修改其他额外的代码，这种现象就
            成为侵入性。侵入性不利于程序后期的维护，尽可能避免。
            compare:比较
         */
//        Collections.sort(list);

         //匿名内部类的形式创建一个比较器
        Comparator<Point> com = new Comparator<Point>() {
            @Override
            /**
             * 实现比较器接口后必须重写方法compare.
             * 该方法用来定义参数o1与参数o2的比较大小规则
             * 返回值用来表示o1与o2的大小关系
             */
            public int compare(Point o1, Point o2) {
                int len1 = o1.getX() * o1.getX() + o1.getY() * o1.getY();
                int len2 = o2.getX() * o2.getX() + o2.getY() * o2.getY();
                return len1-len2;
            }
        };
        Collections.sort(list,com);//回调模式



        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(1,2));
        list.add(new Point(97,88));
        list.add(new Point(7,6));
        list.add(new Point(9,9));
        list.add(new Point(5,4));
        list.add(new Point(2,3));

        System.out.println(list);
        /*
            Collections.sort(List list)在排序List集合时要求集合元素必须实现了
            Comparable接口。实现了该接口的类必须重写一个方法compareTo用与定义比较
            大小的规则，从而进行元素间的比较后排序。否则编译不通过。

            侵入性:
            当我们调用某个API时，其反过来要求我们为其修改其他额外的代码，这种现象就
            称为侵入性。侵入性不利于程序后期的维护，尽可能避免。
            compare:比较
         */
//        Collections.sort(list);

        //匿名内部类的形式创建一个比较器
//        Comparator<Point> com = new Comparator<Point>() {
//            @Override
//            /**
//             * 实现比较器接口后必须重写方法compare.
//             * 该方法用来定义参数o1与参数o2的比较大小规则
//             * 返回值用来表示o1与o2的大小关系
//             */
//            public int compare(Point o1, Point o2) {
//                int len1 = o1.getX() * o1.getX() + o1.getY() * o1.getY();
//                int len2 = o2.getX() * o2.getX() + o2.getY() * o2.getY();
```

```
//                return len1-len2;
//            }
//        };
//        Collections.sort(list,com);//回调模式

//        Collections.sort(list,new Comparator<Point>() {
//            public int compare(Point o1, Point o2) {
//                int len1 = o1.getX() * o1.getX() + o1.getY() * o1.getY();
//                int len2 = o2.getX() * o2.getX() + o2.getY() * o2.getY();
//                return len1-len2;
//            }
//        });

        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);
    }
}
```

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");
```

```
        list.add("小泽老师");
        list.add("苍老师");
        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);
    }
}
```

总结

List集合

list集合有两个常用的实现类:

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。

性能没有苛刻要求时，通常使用ArrayList。

常用方法

E get(int index):获取指定下标index处对应的元素

E set(int index, E e):将给定元素设置到index指定的位置，返回值为该位置被替换的元素。

void add(int index,E e):将给定元素插入到index指定的位置

E remove(int index):删除并返回下标index处对应的元素。

List subList(int start,int end):获取当前集合中start到end之间的子集。(含头不含尾)

集合与数组的互转操作

集合转换为数组，使用集合的toArray方法即可。

数组转换为集合，只能转换为List集合，使用的是Arrays.asList()方法。

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("./demo1.2.3.423.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);

    }

}
```