day09

集合框架

集合间的操作

集合提供了如取并集,删交集,判断包含子集等操作

```
package collection;

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

/**
 * 集合间的操作
 */
public class CollectionDemo4 {
    public static void main(String[] args) {
//        Collection c1 = new ArrayList();
        Collection c1 = new HashSet();//不可重复元素
        c1.add("java");
        c1.add("c");
        c1.add("c++");
        System.out.println("c1:"+c1);
        Collection c2 = new ArrayList();
        c2.add("android");
        c2.add("ios");
        c2.add("java");
        System.out.println("c2:"+c2);
         /*
            boolean addAll(Collection c)
            将给定集合中的所有元素添加到当前集合中。当前集合若发生了改变则返回true
         */
        boolean tf = c1.addAll(c2);
        System.out.println(tf);
```

```
        System.out.println("c1:"+c1);
        System.out.println("c2:"+c2);

        Collection c3 = new ArrayList();
        c3.add("ios");
        c3.add("c++");
        c3.add("php");
        System.out.println("c3:"+c3);
        /*
            boolean containsAll(Collection c)
            判断当前集合是否包含给定集合中的所有元素
         */
        boolean contains = c1.containsAll(c3);
        System.out.println("包含所有元素:"+contains);

        /*
            boolean removeAll(Collection c)
            删除当前集合中与给定集合中的共有元素
         */
        c1.removeAll(c3);
        System.out.println("c1:"+c1);
        System.out.println("c3:"+c3);
    }
}
```

集合的遍历

Collection提供了统一的遍历集合方式:迭代器模式

Iterator iterator()

该方法会获取一个用于遍历当前集合元素的迭代器.

java.util.Iterator接口

迭代器接口,定义了迭代器遍历集合的相关操作.

不同的集合都实现了一个用于遍历自身元素的迭代器实现类,我们无需记住它们的名字,用多态的角度把他们看做为Iterator即可.

迭代器遍历集合遵循的步骤为:问,取,删.其中删除元素不是必要操作

```
package collection;

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

/**
 * Collection接口没有定义单独获取某一个元素的操作，因为不通用。
 * 但是Collection提供了遍历集合元素的操作。该操作是一个通用操作，无论什么类型的
 * 集合都支持此种遍历方式:迭代器模式。
 *
 * Iterator iterator()            die(二声)
 * 该方法会获取一个用于遍历当前集合元素的迭代器
 *
 * java.util.Iterator接口，是迭代器接口，规定了迭代器遍历集合的相关操作，不同的
 * 集合都提供了一个用于遍历自身元素的迭代器实现类，不过我们不需要直到它们的名字，以
 * 多态的方式当成Iterator使用即可。
 * 迭代器遍历集合遵循的步骤为:问->取->删
 * 其中删除不是必须操作。
 *
 */
public class IteratorDemo {
    public static void main(String[] args) {
        Collection c = new ArrayList();
        c.add("one");
        c.add("two");
        c.add("three");
        c.add("four");
        c.add("five");
        System.out.println(c);
        //获取迭代器
        Iterator it = c.iterator();
        /*
            迭代器提供的相关方法:
            boolean hasNext()
```

```
            判断集合是否还有元素可以遍历

            E next()
            获取集合下一个元素(第一次调用时就是获取第一个元素，以此
类推)
         */
        while(it.hasNext()){
            String str = (String)it.next();
            System.out.println(str);
        }
        System.out.println(c);

    }
}
```

迭代器遍历过程中不得通过集合的方法增删元素

```
package collection;

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

/**
 * Collection接口没有定义单独获取某一个元素的操作，因为不通用。
 * 但是Collection提供了遍历集合元素的操作。该操作是一个通用操作，无
论什么类型的
 * 集合都支持此种遍历方式:迭代器模式。
 *
 * Iterator iterator()            die(二声)
 * 该方法会获取一个用于遍历当前集合元素的迭代器
 *
 * java.util.Iterator接口，是迭代器接口，规定了迭代器遍历集合的相
关操作，不同的
 * 集合都提供了一个用于遍历自身元素的迭代器实现类，不过我们不需要直到
它们的名字，以
 * 多态的方式当成Iterator使用即可。
 * 迭代器遍历集合遵循的步骤为:问->取->删
 * 其中删除不是必须操作。
 *
```

```
 */
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);
        //获取迭代器
        Iterator it = c.iterator();
        /*
            迭代器提供的相关方法：
            boolean hasNext()
            判断集合是否还有元素可以遍历

            E next()
            获取集合下一个元素(第一次调用时就是获取第一个元素，以此
类推)
         */
        while(it.hasNext()){
            String str = (String)it.next();
            System.out.println(str);
            if("#".equals(str)){
                /*
                    迭代器要求遍历的过程中不得通过集合的方法增删元
素
                    否则会抛出异
常:ConcurrentModificationException
                 */
//                c.remove(str);
                /*
                    迭代器的remove方法可以将通过next方法获取的元
素从集合
                    中删除。
```

```
                 */
                it.remove();
            }
        }
        System.out.println(c);

    }
}
```

增强型for循环

JDK5之后推出了一个特性:增强型for循环

- 也称为新循环,使得我们可以使用相同的语法遍历集合或数组.
- 语法:

```
for(元素类型 变量名 : 集合或数组){
    循环体
}
```

```
package collection;

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

/**
 * JDK5推出时，推出了一个新的特性:增强型for循环
 * 也称为新循环，它可以用相同的语法遍历集合或数组。
 *
 * 新循环是java编译器认可的，并非虚拟机。
 */
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);
        }
```

```
        for(String str : array){
            System.out.println(str);
        }


        Collection c = new ArrayList();
        c.add("one");
        c.add("two");
        c.add("three");
        c.add("four");
        c.add("five");
        //迭代器遍历
        Iterator it = c.iterator();
        while(it.hasNext()){
            String str = (String)it.next();
            System.out.println(str);
        }
        //新循环遍历
        for(Object o : c){
            String str = (String)o;
            System.out.println(str);
        }

    }
}
```

泛型

JDK5之后推出的另一个特性:泛型

泛型也称为参数化类型,允许我们在使用一个类时指定它当中属性,方法参数或返回值的类型.

- 泛型在集合中被广泛使用,用来指定集合中的元素类型.
- 有泛型支持的类在使用时若不指定泛型的具体类型则默认为原型Object

```
package collection;
```

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

/**
 * JDK5推出时，推出了一个新的特性:增强型for循环
 * 也称为新循环，它可以用相同的语法遍历集合或数组。
 *
 * 新循环是java编译器认可的，并非虚拟机。
 */
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);
        }

        for(String str : array){
            System.out.println(str);
        }

        /*
         *  泛型 JDK5之后推出的另一个特性。
         * 泛型也称为参数化类型，允许我们在使用一个类时指定它里面属
性的类型，
         * 方法参数或返回值的类型，使得我们使用一个类时可以更灵活。
         * 泛型被广泛应用于集合中，用来指定集合中的元素类型。
         * 支持泛型的类在使用时如果未指定泛型，那么默认就是原型
Object
         *
         * Collection接口的定义
         * public interface Collection<E> ... {
         *
         * Collection<E> 这里的<E>就是泛型
         *
         * Collection中add方法的定义，参数为E
         * boolean add(E e)
```

```
         */
        Collection<String> c = new ArrayList<>();
        c.add("one");//编译器会检查add方法的实参是否为String类型
        c.add("two");
        c.add("three");
        c.add("four");
        c.add("five");
//        c.add(123);//编译不通过
        //迭代器遍历
        //迭代器也支持泛型，指定的与其遍历的集合指定的泛型一致即可
        Iterator<String> it = c.iterator();
        while(it.hasNext()){
            //编译器编译代码时会根据迭代器指定的泛型补充造型代码
            String str = it.next();//获取元素时无需在造型
            System.out.println(str);
        }
        //新循环遍历
        for(String str : c){
            System.out.println(str);
        }

    }
}
```

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.*;

/**
 * 数组转换为集合
 * 数组的工具类java.util.Arrays提供了一个静态方法:asList
 * 该方法可以将一个数组转换为一个List集合
 */
public class ArrayToListDemo {
```

```
    public static void main(String[] args) {
        String[] array =
{"one","two","three","four","five"};
        System.out.println("array:"+
Arrays.toString(array));

        //asList方法会返回Arrays定义的内部类ArrayList,该集合内部直接引用给定数组array
        List<String> list = Arrays.asList(array);
        System.out.println("list:"+list);
        //因为集合直接引用数组array,所以对该集合操作就是对array数组的操作
        list.set(1,"six");
        System.out.println("list:"+list);
        System.out.println("array:"+
Arrays.toString(array));
        //对数组操作后,集合也会改到改变.
        array[2] = "seven";
        System.out.println("array:"+
Arrays.toString(array));
        System.out.println("list:"+list);
        //添加元素相当于要对数组扩容,数组是定长的不可以真实的扩容,因此会抛出不支持该操作的异常.删除也是一样的
//        list.add("!!!!!");

        /*
            如果我们需要增删元素,可另行创建一个集合同时包含该集合元素即可.
         */
        List<String> list2 = new ArrayList<>(list);//等同于先new再addAll()
//        Set<String> set = new HashSet<>(list);//等同于先new再addAll()

        System.out.println("list2:"+list2);
        list2.add("!!!!");
        System.out.println("list2:"+list2);


    }
```

```
}
```

总结

Collection常用方法:

boolean addAll(Collection c):将给定集合所有元素添加到当前集合中。

boolean removeAll(Collection c):删除当前集合中与给定集合的公有元素。

boolean containsAll(Collection c):判断当前集合是否包含给定集合中的所有元素。

Iterator iterator():获取用于遍历当前集合的迭代器

T[] toArray(T[] t):将当前集合转换为一个数组。参数为要转换的数组。

迭代器

java.util.Iterator

迭代器用于遍历集合，不同的集合都提供了一个用于遍历自身元素的迭代器实现类。

使用迭代器遍历集合遵循的过程为:问->取->删。其中删除不是必要操作。

常用方法

boolean hasNext():判断集合是否还有"下一个"元素可以遍历

E next():获取集合下一个元素

void remove():从集合中删除迭代器当前位置的元素(通过next获取的元素)

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。

List集合常用方法(特点:通过下标操作)

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)

集合转换为数组的操作

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。