day12

集合(续)

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

```java
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;
```

```java
/**
 *  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))
;
    }
}
```

变长参数

JDK5时推出的另一个特性:变长参数

一个方法中只能声明一个变长参数,并且必须是最后一个参数

```
package collection;

import java.util.Arrays;

public class ArgDemo {
    public static void main(String[] args)
{
        dosome(1,"a");
        dosome(1,"a","b");

 dosome(1,"a","b","a","b","a","b","a","b","
a","b");
        dosome(1,new String[]
{"1","2","3"});
    }

    public static void dosome(int
i,String... s){
        /*
            变长参数在方法中实际上就是一个数组.给
变长参数传入了几个
            实参,该数组长度与实参个数一致.
         */
        System.out.println(s.length);
```

```
        System.out.println("s:"+
Arrays.toString(s));
    }

}
```

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;
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);
        }
}
```