JAVA
40 期

集合 宝典

Collection

传说在很久以前的江湖

有一本《葵花宝典》

这是一本魔书

因为这本书的每一次出现

都将引起一段腥风血雨

而今

这本书的姊妹篇

首现江湖

江湖恩怨又将四起！

毕向东
北京传智播客
JAVA 预热班 40 期

理想出版社

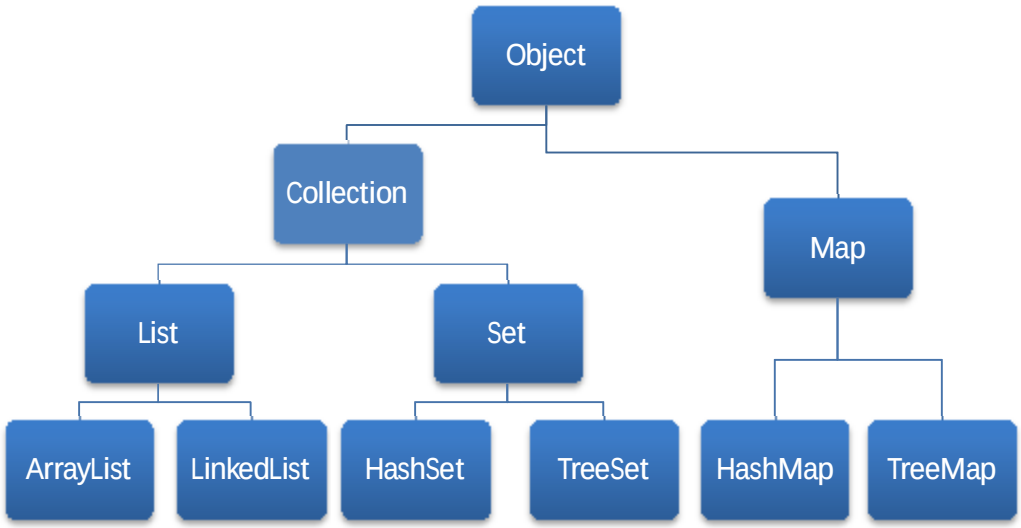
集合（Collection）文档

毕向东 编写
Lenovo 整理

集合：

用于存储对象的可变长度的容器。
因为容器中的数据结构不同，所以出现了容器的体系，也就是不断的向上抽取。

集合中常用集合类的继承关系



Collection：该接口中定义了集合的共性方法。

1，添加：
boolean add(obj);
2，删除：
boolean remove(obj);
void clear();
3，判断：
boolean contains(obj)
boolean isEmpty():判断容器中是否有元素，依据的是 size 方法。
4，获取：Iterator iterator():迭代器。
5，交集：retainAll(Collection)； al.retainAll(a2); //将 al 集合中只保留与 a2 集合共同的元素。
6，个数：int size();
7，转成数组： T[] toArray(T[] t):

|——List:有序，可以重复，有角标。
|——Set:无序，不可以重复。

List：中特有的方法。依据角标进行操作的元素的方法。

1，添加(插入):add(index,obj)
2，删除：remove(index);
3，获取：get(index);
4，索引：
indexOf(obj);
lastIndexOf(obj);
5，取子列表：subList(start,end);
6，修改：set(index,obj);
7，list 集合支持列表迭代器 ListIterator：
Iterator 在迭代时，只能对元素进行获取(next())和删除(remove())的操作。
对于 Iterator 的子接口 ListIterator 在迭代 list 集合时，还可以对元素进行添加(add(obj))，修改 set(obj)的操作。

List：

|--Vector: 底层是数组数据结构，jdk1.0 出现，线程是同步的。被 jdk1.2 版本后出现的 ArrayList 替代，因为效率低。
|--ArrayList: 底层是数组数据结构，jdk1.2 出现，线程是不同步的。查询的速度快。
|--LinkedList: 底层是链表数据结构，线程是不同步的。增删的速度很快。
可以使用该集合去模拟出 队列(先进先出) 或者 堆栈(后进先出) 数据结构。
示例：day14:\LinkedListDemo.java;

Vector：该集合支持一种特有的取出方式：枚举(Enumeration)
枚举的功能和迭代器的功能是一致的。
因为枚举的功能方法名称过长，被迭代器取代。

LinkedList：特有的方法。
addFirst(): -->jdk1.6 offerFirst();
addLast(): -->jdk1.6 offerLast();

getFirst(); -->jdk1.6 peekFirst();
getLast(): -->jdk1.6 peekLast();

removeFirst(): -->jdk1.6 pollFirst();
removeLast(): -->jdk1.6 pollFirst();

对于获取和删除，如果容器中没有元素，get remove 会抛出异常 NoSuchElementException。
到了 1.6 的新方法，容器中没有元素，返回 null;

Set：该集合中没有特有的方法，直接继承自 Collection。

|--HashSet：数据结构是：哈希表。如何保证元素唯一性的呢？

通过元素的两个方法：hashCode(),equals();
判断元素是否相同，先要判断元素的 hashCode（注：hashCodea()返回值为 int 型）值是否一致，
只有在该值一致的情况下，才会去判断 equals()方法。
注意：复写 hashCode()方法时，尽量依据元素的判断相同的条件来定义每一个元素的哈希值。
示例：day14:\HashSetDemo.java;

例：

```
public int hashCode()
{
    reutrn name.hashCode()+age*36;//尽量保证哈希值唯一性。
}
public boolean equals(Object obj)
{
    //这句是为了验证什么时候调用 equals()方法的。
    System.out.println(this+"----equals---"+obj);
    if(!(obj instanceof Student))
        return false;
    Student s = (Student)obj;
    return this.name.equals(s.name) && this.age==s.age;
}
```

|--LinkedHashSet：该子类基于哈希表又融入了链表。可以对 Set 集合进行增删提高效率。

|--TreeSet: 数据结构是：二叉树。如何保证元素唯一性的呢？

通过比较方法的 return 0 来判断元素是否相同。
treeSet 可以对 Set 集合中的元素进行排序。
排序的两种方式：
一，让元素自身具备比较性。
也就是元素需要实现 Comparable 接口，覆盖 compareTo 方法。
这种方式也作为元素的自然排序，也可称为默认排序 。
二，让容器自身具备比较性，自定义比较器。
需求：当元素自身不具备比较性，或者元素自身具备的比较性不是所需的。

那么这时只能让容器自身具备。
定义一个类实现 Comparator 接口，覆盖 compare 方法。
并将该接口的子类对象作为参数传递给 TreeSet 集合的构造函数。
当 Comparable 比较方式，及 Comparator 比较方式同时存在，以 Comparator 比较方式为主。

注意：在覆盖 compareTo 方法，或者 compare 方法时，
必须要明确比较的主要条件相等时，需要参阅次要条件。
示例：day14:\TreeSetDemo2.java;

第一种：元素（类）本身就具有比较方法。

```
class Person implements Comparable
{
    public int compareTo(Object obj)
    {
        Person p = (Person)obj;
        if(this.age>p.age)
            return -1;
        if(this.age<p.age)
            return 1;
        return
this.name.compareTo(p.name);
    }
}
```

第二种方法：建立一个新的类继承 Comparator 接口，并覆盖 compare(Object o1,Object o2)方法。

```
class MyCompare implements Comparator
{   public int compare(Object o1,Object o2)
    {
        Person p1 = (Person)o1;
        Person p2 = (Person)o2;
        if(p1.getAge()>p2.getAge())
            return 1;
        if(p1.getAge()<p2.getAge())
            return -1;
        return 0;
    }
}
{
    TreeSet ts =
    new TreeSet(new MyCompare());
}
```

第二种方法也可以采用匿名内部类的方法，如右：

```
TreeSet ts = new TreeSet(new Comparator()
        {
            public int compare(Object o1,Object o2)
            {
                String s1   = (String)o1;
                String s2   = (String)o2;
                if(s1.length()>s2.length())
                    return 1;
                if(s1.length()<s2.length())
                    return -1;
                return s1.compareTo(s2);

            }
        });
```

技巧：
如何判断这些容器的数据结构？
通过每一个容器的名称即可明确其数据结构：
ArrayList:　　数组 array。
LinkedList:　　链表：link。
HashSet:　　哈希表：hash。
TreeSet:　　二叉树：tree。
HashMap:　　哈希表。hash。
TreeMap:　　二叉树。tree。

看到 array，就要想到角标。
看到 link，就要想到 first，last。
看到 hash，就要想到 hashCode,equals.
看到 tree，就要想到两个接口。Comparable，Comparator。

迭代器：Iterator。

所有 Collection 集合共性的取出方式。

每一个容器都有取出功能。这些功能定义都是一样，只不过实现的具体方式不同。
因为每一个容器的数据结构不一样。
所以这些定义的声明就进行了抽取，从而出现了 Iterator 接口。
而每一个容器都在其内部对该接口进行了内部类的实现。
也就是将取出方式的细节进行封装。通过 iterator 方法对外提供了一个取出元素的对象。

迭代器就如同街头抓布娃娃的游戏机里的夹子。游戏机里面的夹子可能不同（有三个钩子的，有两个钩子的），但人操作游戏机的方法都一样，通过手摇动摇杆来控制里面的夹子。尽管我们不知道里面夹子的具体机械实现方式（具体实现方法被封装了），但这并不影响我们玩游戏和减少游戏带给我们的快乐。

在使用时，通常通过两个方法来完成。
1，判断容器中是否有元素。hasNext
2，取出元素。next.

需要取出所有元素时，可以通过循环，
java 建议使用 for 循环。
因为可以对内存进行一下优化。

注意：在迭代时，next 方法每调用一次，
内部指针就会自定往下走。
在循环，next 方法调用一次即可。

```
ArrayList    al = new ArrayList();
      al.add("haha1");
      al.add("haha2");
for(Iterator it = al.iterator(); it.hasNext() ; )
      {
            String s = (String)it.next();
            System.out.println(s);

      //System.out.println(it.next()+"..."+in.ne
xt());不要这样写。
      }
```

JDK1.5 以后，认为还可以对迭代器进行一下简化。
所以出现了增强的 for 循环。

```
ArrayList<String>    al = new ArrayList<String>();
al.add("haha1");
al.add("haha2");
for(String s : al)
{
    System.out.println(s);
}
```

一定要知道的就是：集合中其实存储的都是对象的引用。
而迭代器取出的也是集合中的对象引用。
集合中可以直接存储基本数据类型。
因为自动装箱拆箱，会将基本数据类型都先封装成对象后才存入集合。

```
al.add(3);//al.add(new Integer(3));
```

Map 集合

Map:一次存入一对元素，以键值对（Key,Value）形式存在，必须保证键的唯一性。

1，添加：
put(K key,V value):返回的是被覆盖的 Value，如果键没有重复，返回的是 null。

2，删除：
remove(key);
clear();

3，获取：value get(key);
也可以用于判断键是否存在的情况。当指定的键不存在的时候，返回的是 null。
对于 HashMap 集合，可以存入 null 键 null 值。
hm.put("haha",null);
hm.get("haha");==>null.//要注意一下。

4，判断：
boolean containsKey(key);
boolean containsValue(value);
boolean isEmpty();

5，长度：int size();

6，取出：原理：将 map 集合转成 Set 集合后，在通过迭代器取出。
Set<k> keySet()：将 map 集合中所有的键取出存入 set 集合。再通过 get 方法获取键对应的值。
Set<Map.Entry<k,v>> entrySet()：将 map 集合中的键值映射关系打包成一个对象 Map.Entry 对象。

将该对象存入 set 集合。取出是可以通过 Map.Entry 对象的 getKey，getValue 获取其键和值。

7，获取所有的值。

Collection<v> values();

Map：

|--Hashtable:底层是哈希表数据结构，线程是同步的，不可以存入 null 键，null 值。效率较低，被 HashMap 替代。

|--HashMap:底层是哈希表数据结构，**线程是不同步的**，可以存入 null 键，null 值。要保证键的唯一性，需要覆盖 hashCode 方法，和 equals 方法。

|--LinkedHashSet：该子类基于哈希表又融入了链表。可以 Map 集合进行增删提高效率。

|--TreeMap:底层是二叉树数据结构。可以对 map 集合中的键进行排序。需要使用 Comparable 或者 Comparator 进行比较排序。return 0，来判断键的唯一性。

其实 set 集合，底层使用的就是 map 集合。

Map 集合中有一个特殊的对象是可以和 IO 对象相结合的：Properties。它是 Hashtable 的子类。

该集合对象不存在泛型，键和值都是字符串。

什么时候使用 map 集合？当对象之间存在着映射关系时，就要先想到 map 集合。

集合框架中的工具类：特点：该工具类中的方法都是静态的。

Collections：常见方法：

1，对 list 进行二分查找：前提该集合一定要有序。

int binarySearch(list,key);//要求 list 集合中的元素都是 Comparable 的子类。

int binarySearch(list,key,Comparator);

2，对 list 集合进行排序。

sort(list);

sort(list,comaprator);

3，对集合去最大值或者最小值。

max(Collection)

max(Collection,comparator)

min(Collection)

min(Collection,comparator)

4，对 list 集合进行反转。

reverse(list);

5，对比较方式进行强行逆转。

Comparator reverseOrder();

Comparator reverseOrder(Comparator);

6，对 list 集合中的元素进行位置的置换。

swap(list,x,y);

7，对 list 集合进行元素的替换。如果被替换的元素不存在，那么原集合不变。

replaceAll(list,old,new);

8，可以将不同步的集合变成同步的集合。

Set synchronizedSet(Set<T> s)

Map synchronizedMap(Map<K,V> m)

List synchronizedList(List<T> list)

需求：替换 list 集合中部分元素为指定元素。

```
public    void myFill(List list,int start,int end,Object obj)
{
      for(int x=start; x<end; x++)
      {
            list.set(x,obj);
      }
}
```

Arrays:用于对数组操作的工具类。

1，binarySearch(int[])

binarySearch(double[])……

2，sort(int[])

sort(char[])……

3，toString(int[])……:将数组变成字符串。

4，copyOf();复制数组。

5，copyOfRange():复制部分数组。

6，equals(int[],int[]);比较两个数组是否相同。

7，List asList(T[]);将数组变成集合。

这样可以通过集合的操作来操作数组中元素，

但是不可以使用增删方法，add，remove。因为数组长度是固定的，会出现 UnsupportOperationExcetion。

可以使用的方法：contains，indexOf。。。

如果数组中存入的基本数据类型，那么 asList 会将数组实体作为集合中的元素。

如果数组中的存入的应用数据类型，那么 asList 会将数组中的元素作为集合中的元素。

如果想要将集合变数组：
可以使用 Collection 中的 toArray 方法。
传入指定的类型数组即可，该数组的长度最好为集合的 size。

在 JDK1.5 版本之后出现一个新特性。

泛型：(Generic)

提供一个安全机制。

好处：

1，将运行时期出现的问题(ClassCastException)转移到了编译时期。

2，避免了强制转换的麻烦。

格式：通过<>的形式接收指定的类型。

也就是在定义集合容器时，要先明确该集合存入的元素类型。

1，泛型类。泛型定义在类上，在 new 对象的时候传入具体类型。该类型作用于整个类。

```
class Demo<T>
{
	public void show(T t){}
}
```

2，泛型方法。泛型定义在方法上，该泛型只在方法上有效。

```
class Demo
{
	public <T> void show(T t)
	{}
}
```

泛型即定义在类上又定义在方法上。

```
class Demo<T>
{
	public <Q> void show(Q q)
	{}
	public void function(T t)
	{}
}
```

注意：当方法被静态修饰时，静态不可以访问类上定义的泛型。
而且在静态方法上定义泛型时，泛型只能定义在 static 关键字之后。

3，泛型接口。泛型定义在接口上。

```
interface Demo<T>
{
	void show(T t);
}

//1,使用接口时明确具体类型。
class DemoImpl implements Demo<String>
{
	public void show(String s){}
}

//2，使用接口时，不明确具体类型，需要建立接口的子类对象在明确。
class DemoImpl<T> implements Demo<T>
{
	public void show(T t)
	{
		System.out.println(t);
	}
}

DemoImpl<String> di = new DemoImpl<String>();
d1.show("haha");
```

泛型的限定：

通配符：？

```
public    void show(List<?> list)
{

}
```

可以对类型进行限定范围。

？ extends E: 接收 E 类型或者 E 的子类型。

? super E: 接收 E 类型或者 E 的父类型。

泛型嵌套：

<T extends Object & Comparable<? super T>>

在定义类 T，并限定 T 的类型范围。

T 类型必须是 Object 的子类，并该类型实现了 Comparable 接口。

泛型的细节：

1，保证左右两变泛型一致。

2，在泛型的方法中，不可以使用该泛型类型的特有方法。

可以使用 Object 类中的方法。

JDK1.5 的新特性：

1，可变参数。

简化数组参数的定义。

```
public void show(int[] arr)
{}

public void show(int... arr)
{
}
```

注意：可变参数要定义在参数列表的最后。

2，静态导入。

简化静态方法的书写。

```
import static java.lang.System.*;

System.out.println("hello");
out.println("hello");

import static java.util.Collections.*;

Collections.sort(list);
sort(list);
max(collection);
```

为简化书写而存在的两个特性。

集合学完，你应该能完成以下作业！

集合练习：

1，去除 ArrayList 中的重复元素。要求存入的是自定义元素如 Person。

2，定义 自定义元素存入 HashSet 集合。刻意存入相同元素。

3，对自定义元素进行排序。

4，获取字符串中每一个字母出现的次数。"df3g6h-j+kl"

if(chs[x]>='a' && chs[x]<='z' || chs[x]>='A' && chs[x]<='Z')

5，定义一个列表容器，可以使用 LinkedList 完成。

集合的常见问题：

1，ArrayList 和 Vector 的区别。

2，HashMap 和 Hashtable 的区别

3，Collection 和 Collections 的区别。

如本书内有错误，系整理者粗心所致，欢迎批评指正。
请发邮件至 wangboak@126.com ，希望共同切磋，共同进步。
欢迎加入专为 40 期预热班同学创建的 QQ 群：78297986。