第六章 对象群体的组织

6.1 Java 集合框架介绍

Java 集合框架是为了表示和操作集合类型而规定的统一体系结构。

1.框架内容

（1）对外接口：表示集合的抽象数据类型，规定了所有集合类型都应该具有的对外服务功能；

（2）接口实现：实现集合接口的 Java 类，是可重用的数据结构；

（3）对集合运算的算法：执行运算的方法，例如在集合上进行查找和排序。

2.Java 集合框架接口基本结构

图 1

接口声明了对各种集合类型执行的一般操作。我们看这个接口有两个系列，一个根接口，也就是最顶层的父接口是 Collection，它接下来的子接口包括 Set、List、Queue 和 SortedSet。另外一个是 Map 系列，下面接口是 SortedMap。

（1）Collection 接口

声明了一组操作成批对象的抽象方法；实现它的类是 AbstractCollection 类，它的结构如图 2 所示。如果我们要使用它的规定的功能的话，我们只能使用图 2 中的子类实现。比如说 Vector ArrayList 都是继承自 AbstractCollection 的具体的类。

图 2

常用方法：

①查询方法：不改变集合对象内容。它有：

int size()：返回集合对象中包含的元素个数。

isEmpty()：判断集合对象中是否有元素，如果空的，返回 true。

Boolean contains(Object obj)：判断对象是否在集合中。

Boolean containsAll(Collection c)：判断方法的接受者对象是否包含集合中所有的元素。

②修改方法：改变集合对象里面的内容。它有：

boolean add(Object obj)：增加对象。

Boolean addAll(Collection<?> c)：将参数集合中的所有元素增加到接受者集合中。

Boolean remove(Object obj)：从集合中删除对象

Boolean removeAll(Collection c)：将参数集合中的所有元素从接受者集合中删除。

Boolean retainAll(Collection c)：在接受者集合中保留参数集合中的所有元素，其它元素都删除。

void clear()：删除集合中的所有元素。

（2）Set 接口：Set 接口实现数学中的集合功能，在 Set 接口是**禁止出现重复元素**的。它对 equals 和 hashCode 操作有了更强的约定，如果两个 Set 对象包含同样的元素，两者便是相等的。

实现 set 接口的类有 HashSet 还有 TreeSet 等。

（3）SortedSet 接口：其中元素是升序排列，还增加了与次序相关的操作。通常用于存储词

汇表。

（4）List 接口：可包含重复元素，元素是有顺序的，每一个元素都有一个 index 值，从 0 开始，用以标明元素在列表中的位置。实现 List 接口的类，主要是 Vector，ArrayList，LinkedList、栈结构 Stack 等。

LinkedList 内部实现是链表，适合于链表中间需要频繁进行插入和删除操作。

（5）Queue 接口：FIFO（先进先出）接口。除了 Collection 基本操作，队列接口还有插入、移除和查看操作。

实现类主要有 LinkedList，同时也实现了 List；PriorityQueue 也是。

（6）Map 接口：用于维护键值对（key/value pairs）。不能有重复的关键字，每个关键字最多能够映射到一个值。声明时可以带有两个参数，即 Map<K, V>，其中 K 表示关键字的类型，V 表示值的类型。如图 3 所示是 Map 接口以及它的子接口和实现的类。

图 3

SortedMap 是 Map 的子接口，是升序排列的，通常用于字典和电话目录等。SortedMap<K,V>，其中 K 表示关键字的类型，V 表示值的类型。实现它的类包括：TreeMap 和 ConcurrentSkipListMap（支持并发）。**关于跳表，学习数据结构!!!**

6.2 常用算法

对集合预算的算法，大多数用来操作 List 对象，但是有两个（min 和 max）课用于任何集合对象。

1.排序算法 sort：使 List 元素按照某种关系**升序**排列。它主要有两种形式：

（1）简单形式只是将元素按照自然次序排列，或者集合实现了 Comparable 接口，定义了如何比较大小的操作；

（2）附加一个 Comparator 对象作为参数，用于规定比较规则，课用于实现反序或者特殊次序排序。

排序算法的算法性能来说比较快，时间复杂度是 nlog(n)；它是稳定的，相等元素排序过程中顺序不改变。

2.洗牌算法 shuffle：用于打乱 List 中的任何次序，以随机方式重排 List 元素，任何次序出现可能性都是相等的，在实现偶然性游戏的时候，这个算法有用，例如洗牌。

3.其它常规数据处理算法：

（1）reverse：将一个 List 中的元素反向排列。

（2）fill：用指定的值覆写 List 中的每一个元素，这个操作在重新初始化 List 时有用。

（3）copy：接受两个参数，目标 List 和源 List，将源中的元素复制到目标，覆写其中的内容。目标 List 必须至少与源一样长，如果更长，则多余部分内容不受影响。

4.查找算法——二分法查找算法：使用二分法在一个**有序**的 List 中查找指定元素。查找有两种形式：

（1）假定 List 是按照自然顺序升序排列的。首先 List 中要有自然顺序，并且排好了。

（2）增加了一个对象 Comparator，表示比较规则，并且假定 list 是按照这种规则排序的，并且元素能够随机访问。

算法首先检查集合是否实现了 RandomAccess 接口，如果是，执行二分法查找，否则线性查找。

5.寻找最值：用于任何集合算法，min 和 max 分别返回最小值和最大值。这两个算法分别有两种形式：

（1）简单形式按照元素的自然顺序返回最值；

（2）另一种需要附加一个 Comparator 对象作为参数，并且按照该对象指定的比较规则返回最值。

6.3 数组实用方法

Arrays 类：java.util.Arrays

1.常用方法：

（1）fill(type[] a, type val)：给数组填充，就是简单地把一个数组全部或者某段数据填成一个特殊的值；

（2）equals(type[]a, type[] b)：实现两个数组的比较，相等时返回 true；

（3）sort(type[] a)：对数据排序；

（4）binarySearch()：对数组元素进行二分法查找；

（5）asList(T...a)：实现数组到 ArrayList 的转换，参数就是一个长度不确定的数组，数组元素类型是抽象的 T 类型，也就是说任何数组都能用这个；

（6）toString(基本类型或者 object 数组引用)：往 String 类型转换。

例：数组的填充和复制

代码：

```
import java.util.*;
public class CopyingArrays{
    public static void main(String args[]){
    int[] i = new int[25];
    int[] j = new int[25];
    Arrays.fill(i,47);//可以给数组填充基本类型数值，例如 47/99/103
    Arrays.fill(j.99);
    System.arraycopy(i,0,j,0,i.length);//
    int[] k = new int[10];
    Arrays.fill(k,103);
    System.arraycopy(i,0,k,0,k.length);
    Arrays.fill(k,103);
    System.arraycopy(k,0,i,0,k.length);
    Integer[] u = new Integer[10];
    Integer[] v = new Integer[5];
    Array.fill(u, new Integer(47));//也可以给对象数组去统一填充对象
    Array.fill(v, new Integer(99));
    System.arraycopy(v,0,u,u.length/2,v.length);
    }
}
```

例：数组的比较

代码：

```
import java.util.*;
public class Arrays{
    public static void main(String args[]){
    int[] a1 = new int[10];
```

```
        int[] a2 = new int[10];
        Arrays.fill(a1,47);//
        Arrays.fill(a2.47);
        System.out.println(Arrays.equals(a1,a2));//true
        a2[3] = 12;
        System.out.println(Arrays.equals(a1,a2));//false
        String[] s1 = new String[5];
        Arrays.fill(s1,"hi");
        String[] s1 = {"hi","hi","hi","hi","hi"};
        System.out.println(Arrays.equals(s1,s2));//true
        }
}
```

6.4 基于动态数组的类型（Vcetor，ArrayList）

它们实现了 Collection 接口，都能够存储相同类型（或相同父类或接口）的对象，不能存储基本类型(primitive)的数据，要讲基本类型数据包裹在包裹类中，例如我们要存一组 int 数据，那么首先要把它包裹成 integer 这种类型的对象再存进去。

由于它们是基于动态数组这种存储结构，所以其容量能够空间需要自动扩充，增加元素方法效率较高，除非空间已满。若满了，则需要多消耗一点运行时间来扩充容量。

Vector：集合框架中的遗留类，旧线程安全集合。（**不鼓励使用**）

ArrayList：是非同步的，效率较高。

在新版本的 JDK 中，提供了线程安全的集合的包在 Java.util.concurrent 这个包，有映像、序集、队列等。

任何集合类也可以通过使用同步包装器变成线程安全的。

ArrayList 构造方法：

1.ArrayList()：构造一个空表，默认容量为 10。

2.ArrayList(Collection<? extends E>c)：用参数集合元素为初始值构造一个表。

3.ArrayList(int initialCapacity)：构造一个空表，容量为 initialCapacity。

其它方法见图 4，图 5 等。

图 4

图 5

6.5 遍历 Collection

遍历实现了 Collection 接口的集合方法：

1.通过 Enumeration 及 Iterator 接口遍历集合

Enumeration/Iterator 都可以从集合类对象中提取每一个元素，并提供了用于遍历元素的方法。

Java 中许多方法（如 elements()）都返回 Enumeration 类型的对象，而不是返回集合类对象。

Enumeration 接口不能用于 ArrayList 对象，而 Iterator 接口即可以用于 ArrayList 也可以用于 Vector 对象。

Iterator 是对 Enumeration 接口的改进，**优先选用此接口**。它具有从正在遍历的集合中去除对象的能力。它具有如下三个实例方法：

（1）hasNext()：判断是否还有元素。

（2）next()：取得下一个元素。

（3）remove()：去除一个元素。注意是从集合中去除**最后调用 next()返回的元素**，而不是从 Iterator 类中去除。

例：使用 Iterator 遍历元素,滤除长度大于 4 的元素，其它的显示出来。

代码：

```
import java.util.*;
public class IteratorTester{
    public static void main(String args[]){
    String[] num = {"one","two",...,"ten"};
    //为了举例，故省略中间元素
    Vector<String> aVector = new Vector<String>(
    java.util.Array.asList(num));
    //asList 实现数组到 ArrayList 的转换
    System.out.println("Before Vector: " + aVector);
    Iterator<String> nums = aVector.iterator();
    while(nums.hasNext()){
        String aString = (String)num.next();
        System.out.println(aString);
        if(aString.length()>4)
            nums.remove();
    }
    System.out.println("After Vector: " + aVector);
    }
}
```

运行结果：

After Vector:{//正确答案}

2.增强 for 循环：

例：增强 for 循环遍历集合

代码：

```
import java.util.*;
public class ForTester{
    public static void main(String args[]){
    Enumeration<String> days;
    Vector<String> dayNames = new Vector();
    dayNames.add("Sunday");
    dayNames.add("Monday");
    //一直增加到 Saturday
    days = dayNames.elements();
    for(String day : dayNames){
```

```
            System.out.println(day);
            }
        }
}
```

运行结果：

//显示 Sunday，Monday，……

3.聚集操作遍历：

聚集操作与 lambda 表达式一起使用。

例：假设有一个实现了 Collection 接口的 myShapesCollection 集合对象，有 getColor()可以返回对象的颜色，getName()方法返回对象的名字，则遍历并输出红色对象的名字：

代码：

```
myShapesCollection.stream()
.filter(e->e.getColor) == Color.RED)
.forEach(e->System.out.println(e.getName()));
```

先取得这个流，然后利用一个过滤器取得所有 color 为 red 的对象，然后将这些对象输出。

6.6 Map 接口及其实现

Map 接口用于存储“关键字”（key）和“值”（value）的元素对，其中每个关键字映射到一个值。当我们需要通过关键字实现对值的快速存取的时候使用该接口。

Map 接口的抽象方法主要有**查询方法**和**修改方法**两种，两个主要实现的类是**HashTable(1.0)**和 **HashMap(2.0)**两种。

Map 接口中的查询方法如表 1 所示。

表 1 Map 接口的查询方法

查询方法	作用
int size()	返回 map 中的元素个数
boolean isEmpty()	返回 map 中是否包含元素，若没有返回 true
boolean containsKey(Object key)	判断给定参数是否是 map 中的一个关键字
boolean containsValue(Object val)	判断给定参数是否是 map 中的一个值
Object get(Object key)	返回 map 中与给定关键字关联的值
Collection values()	返回包含 map 中所有值的 Collection 对象
Set keyset()	返回包含 Map 中所有关键字的 Set 对象
Set entrySet()	返回包含 Map 中所有项的 Set 对象（键值对）

Map 接口的修改方法：

1. Object put(Object key, Object va)：将给定的键值对加入到 Map 对象中。其中关键字必须唯一，否则新加入的值会取代 Map 对象中已有的值。

2. void putAll(Map m)：将给你的参数 Map 中的所有项加入到接受者 Map 对象中。

3.Object remove(Object key)：将关键字为给定参数的项从 map 中删除。

4.void clear()：从 map 对象中删除所有的项。

常用类：HashTable（老版本中的），HashMap（新版本中的）

哈希表存储对象的方式：对象的位置和对象的关键属性 k 之间有一个特定的对应关系 f,

我们称之为哈希函数。它使每一个对象与一个唯一的存储位置相对应。因而在查找的时候，只要根据待查对象的关键属性 k，计算 f(jk)的值即可知道其存储位置。

哈希表的几个概念：

1.容量（capacity）：哈希表容量不固定，随对象加入，容量自动扩充。

2.关键字/键（key）：每个存储的对象都需要有一个关键字 key，key 可以是对象本身也可以是对象的一个部分（如对象的某一属性）。

3.哈希码：要将对象存储到哈希表中，需要将其关键字映射到整形数据，称为关键字的哈希码。

4.哈希函数：返回对象的哈希码。

5.项（item）：哈希表中的每一个项都有两个域：关键字域 key 和值域 value（即存储的对象）。key 和 value 都可以是任意的 object 类型的对象，但不能为空（null）。HashTable 中的所有关键字都是唯一的。

6.装填因子（load factor）：（表中填入的项数）/(表的容量)。如果装填因子越大，说明要存的越多，容量越少。如果过大，说明存的项数超出容量，肯定发生冲突，需要解决冲突。

HashMap 的构造方法：

1.HashMap()：默认容量为 16，默认装填因子 0.75。

2. HashMap(int initialCap)：容量为 Initialcap，装填因子为 0.75.

3. HashMap(int init, float loadFactor)：容量为 init，装填因子为 loadFactor。

4. HashMap(Map<? extends K,? extends V> m)：以参数 m 为初值构造新的 HashMap.

其它方法不再累述。