Java语言程序设计（第2版）

第 6章 集合

郑莉

导学

目录

- Java集合框架介绍
- 接口及常用类概述
- 常用算法
- 数组实用方法
- 基于动态数组的类型(Vector, ArrayList)
- 遍历Collection
- Map接口及其实现

Java集合框架介绍

<7.1>

Java集合框架

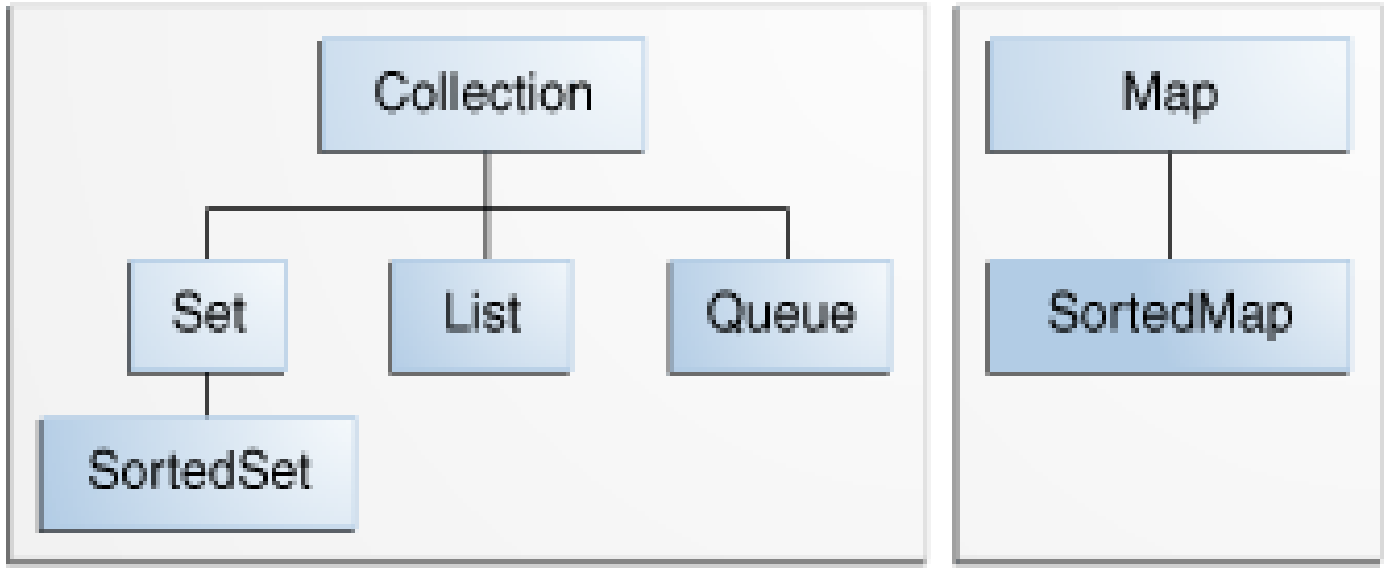
- 为表示和操作集合而规定的一种统一的标准的体系结构
- 提供了一些现成的数据结构可供使用，程序员可以利用集合框架快速编写代码，并获得优良性能
- 将具有相同性质的一类对象，汇聚成一个整体
- 可动态改变其大小

Java集合框架

- 对外的接口：表示集合的抽象数据类型
- 接口的实现：指实现集合接口的Java类，是可重用的数据结构
- 对集合运算的算法：是指执行运算的方法，例如在集合上进行查找和排序

集合框架接口

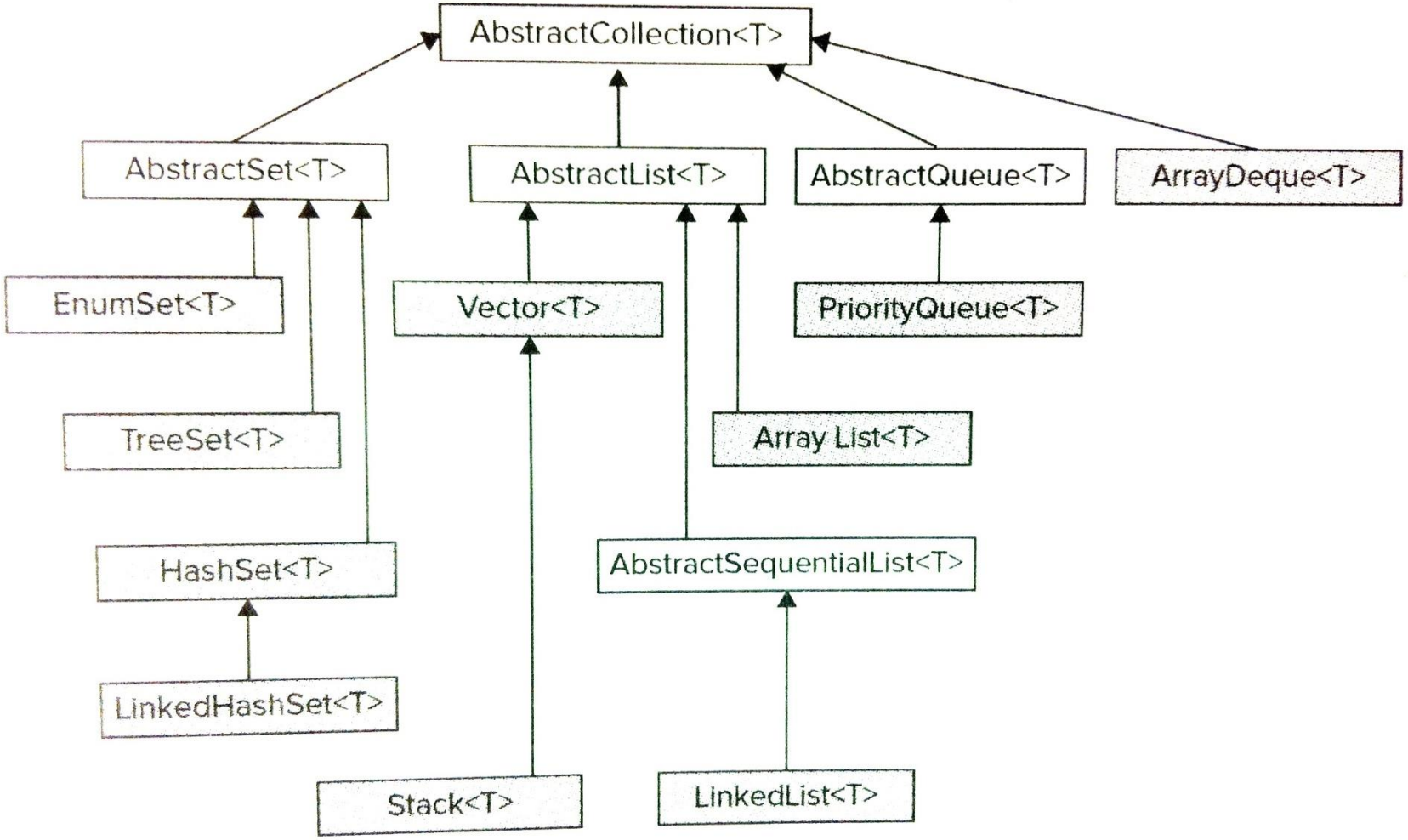
- 声明了对各种集合类型执行的一般操作
- 包括Collection、Set、List、Queue、SortedSet、Map、SortedMap
- 基本结构如图(图重画，颜色和背景不能用原图)



Collection接口

- 声明了一组操作成批对象的抽象方法
- 实现它的类：AbstractCollection

AbstractCollection类（图重新画）



Collection接口常用方法

- 查询方法
  - int size() – 返回集合对象中包含的元素个数
  - boolean isEmpty() – 判断集合对象中是否还包含元素，如果没有任何元素，则返回true
  - boolean contains(Object obj) – 判断对象是否在集合中
  - boolean containsAll(Collection c) – 判断方法的接收者对象是否包含集合中的所有元素

Collection接口常用方法

- 修改方法包括
  - boolean add(Object obj) – 向集合中增加对象
  - boolean addAll(Collection<?> c) – 将参数集合中的所有元素增加到接收者集合中
  - 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收者集合中删除
  - boolean retainAll(Collection c) – 在接收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

结束语

主要接口及常用的实现类

Set接口

- 禁止重复的元素，是数学中“集合”的抽象

Set接口

- 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，二者便是相等的

实现Set接口的类

- 哈希集合(HashSet)及树集合(TreeSet)
- 其他：AbstractSet, ConcurrentSkipListSet, CopyOnWriteArraySet, EnumSet, JobStateReasons, LinkedHashSet

SortedSet接口

- 一种特殊的Set
- 其中的元素是升序排列的，还增加了与次序相关的操作
- 通常用于存放词汇表这样的内容
- 实现它的类：ConcurrentSkipListSet, TreeSet

List接口

- 可包含重复元素；
- 元素是有顺序的，每个元素都有一个index值（从0开始）标明元素在列表中的位置。

实现List接口的类

- Vector
- ArrayList：一种类似数组的形式进行存储，因此它的随机访问速度极快
- LinkedList：内部实现是链表，适合于在链表中间需要频繁进行插入和删除操作
- 栈Stack
- 其他：AbstractList, AbstractSequentialList, AttributeList, CopyOnWriteArrayList, RoleList, RoleUnresolvedList

Queue 接口

- 除了Collection 的基本操作，队列接口另外还有插入、移除和查看操作。
- FIFO (先进先出，first-in-first-out)

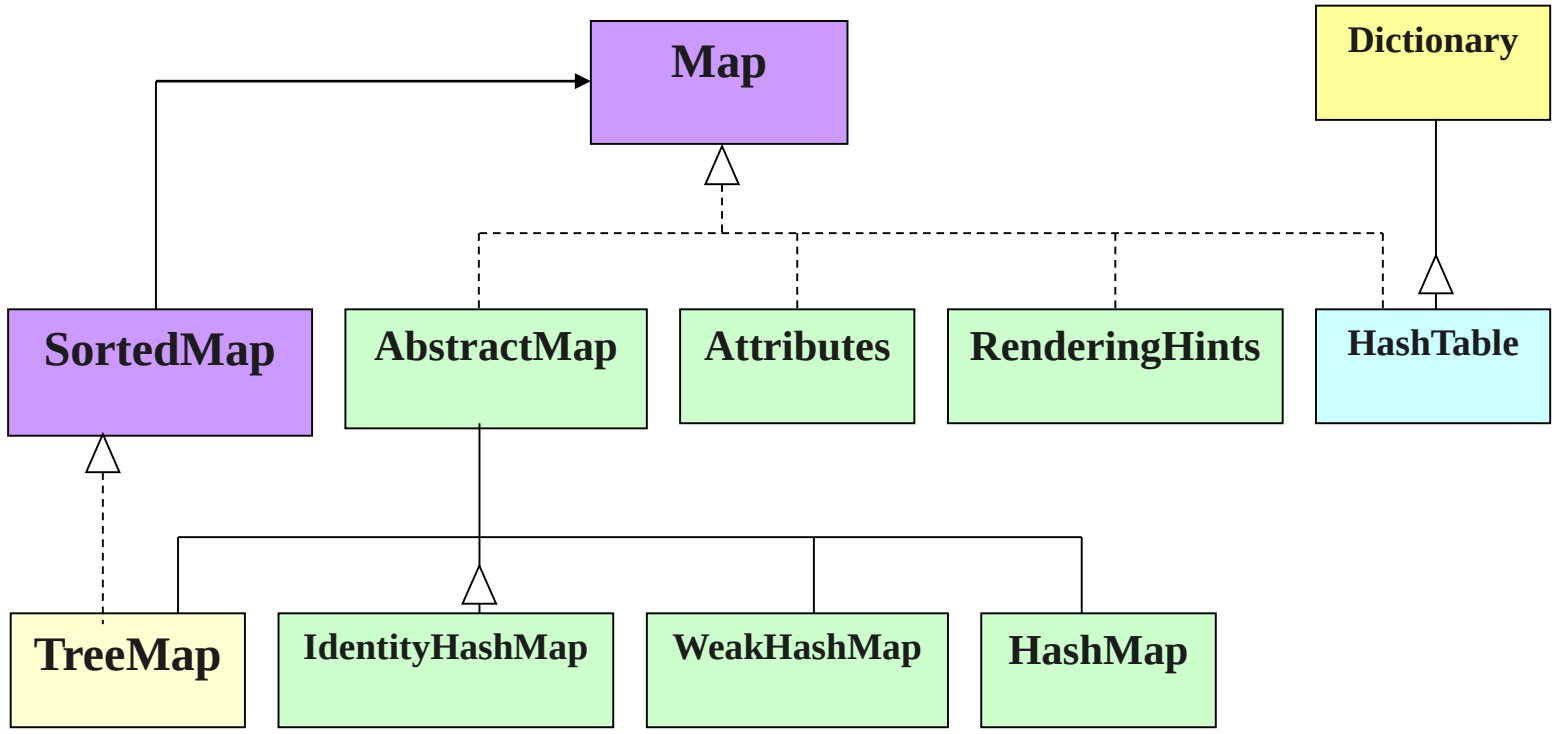
实现Queue接口的类

- LinkedList
  - 同时也实现了List，前进先出
- PriorityQueue
  - 按元素值排序的队列
- 其他：AbstractQueue, ArrayBlockingQueue, ArrayDeque, ConcurrentLinkedQueue, DelayQueue, LinkedBlockingDeque, LinkedBlockingQueue, PriorityBlockingQueue, SynchronousQueue

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

Map接口及其子接口和实现



实现Map接口的类

- HashMap, TreeMap,
- 其他：AbstractMap, Attributes, AuthProvider, ConcurrentHashMap, ConcurrentSkipListMap, EnumMap, Hashtable, IdentityHashMap, LinkedHashMap, PrinterStateReasons, Properties, Provider, RenderingHints, SimpleBindings, TabularDataSupport, UIDefaults, WeakHashMap

SortedMap接口

- Map的子接口；
- 一种特殊的Map，其中的关键字是升序排列的；
- 通常用于词典和电话目录等 。

SortedMap接口

- 在声明时可以带有两个类型参数，即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
  - TreeMap
  - ConcurrentSkipListMap（支持并发）

结束语

常用算法

对集合运算的算法

- 大多数算法都是用于操作List对象
- 有两个(min和max)可用于任意集合对象

排序算法sort

- 对List排序，使其中的元素按照某种次序关系升序排列。

排序算法sort

- 使List元素按照某种次序关系升序排列。
- 有两种形式
  - 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；
  - 第二种形式需要一个附加的Comparator对象作为参数，用于规定比较规则，可用于实现反序或特殊次序排序。
- 算法性能
  - 快：时间复杂度nlog(n)；
  - 稳定。

洗牌算法shuffle

- 其作用与排序算法恰好相反，它打乱List中的任何次序。

洗牌算法shuffle

- 以随机方式重排List元素，任何次序出现的几率都是相等的；
- 在实现偶然性游戏的时候，这个算法很有用，例如洗牌。

常规数据处理算法

- reverse：将一个List中的元素反向排列 。
- 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。
- 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，则多余的部分内容不受影响。

查找算法binarySearch

- 二分法查找算法

查找算法binarySearch

- 使用二分法在一个有序的List中查找指定元素
- 有两种形式
  - 第一种形式假定List是按照自然顺序升序排列的
  - 第二种形式需要增加一个Comparator对象，表示比较规则，并假定List是按照这种规则排序的。
- 检查集合是否实现了RandomAccess接口。是：二分法查找。否：线性查找

寻找最值

寻找最值

- 用于任何集合对象。。
- min和max算法返回指定集合中的最小值和最大值。
- 这两个算法分别都有两种形式
  - 简单形式按照元素的自然顺序返回最值；
  - 另一种形式需要附加一个Comparator对象作为参数，并按照Comparator对象指定的比较规则返回最值。

结束语

数组实用方法

数组实用方法

- Java集合框架提供了一套专门用于操作数组的实用方法，它们作为静态方法存在该类中。

Arrays类

- java.util.Arrays
- 常用方法
  - 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；
  - equals (type[] a, type[] b)：实现两个数组的比较，相等时返回true；
  - sort (type[] a)： 对数组排序；
  - binarySearch ( )： 对数组元素进行二分法查找；
  - asList(T... a)： 实现数组到ArrayList的转换；
  - toString（基本类型或object数组引用）： 实现数组到string的转换。

例：数组的填充和复制

例：数组的填充和复制

```
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);
    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];
    Arrays.fill(u, new Integer(47));
    Arrays.fill(v, new Integer(99));
    System.arraycopy(v, 0,  u, u.length/2, v.length);
  }
}
```

例：数组的比较

例：数组的比较

```
import java.util.*;
public class ComparingArrays{
        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] = 11;
                System.out.println(Arrays.equals(a1, a2)); //false
                String[] s1 = new String[5];
                Arrays.fill(s1, "Hi");
                String[] s2 = {"Hi", "Hi", "Hi", "Hi", "Hi"};
                System.out.println(Arrays.equals(s1, s2));//true
        }
}
```

结束语

基于动态数组的类型(Vector, ArrayList)

Vector, ArrayList

- 实现了Collection接口。
- 能够存储相同类型（或具有相同的父类或接口）的对象。
- 不能存储基本类型（primitive）的数据，要将基本类型数据包裹在包裹类中。
- 其容量能够根据空间需要自动扩充。
- 增加元素方法的效率较高，除非空间已满（在这种情况下，在增加之前需要先扩充容量）。
- Vector：集合框架中的遗留类，旧线程安全集合。
- ArrayList方法是非同步的，效率较高。
- Java提供了线程安全集合：Java.util.concurrent包，映像、有序集、队列
  - 任何集合类通过使用同步包装器可以变成线程安全的：

```
List<E> synchArrayList=Collections.synchronisedList(new ArrayList<E>());
```

ArrayList构造方法

- ArrayList()
  - 构造一个空表，默认容量为10。
- ArrayList(Collection<? extends E> c)
  - 用参数集合元素为初始值构造一个表。
- ArrayList(int initialCapacity)
- 构造一个空表，容量为initialCapacity 。

ArrayList其他方法

- boolean add(E e)
- void add(int index, E element)
- boolean addAll(Collection<? extends E> c)
- boolean addAll(int index, Collection<? extends E> c)
- void clear()
- Object clone()
- boolean contains(Object o)
- void ensureCapacity(int minCapacity)
- void forEach(Consumer<? super E> action)
- E get(int index)
- int indexOf(Object o)
- boolean isEmpty()
- Iterator<E> iterator()

ArrayList其他方法（续）

- int lastIndexOf(Object o)
- ListIterator<E> listIterator()
- ListIterator<E> listIterator(int index)
- E remove(int index)
- boolean remove(Object o)
- boolean removeAll(Collection<?> c)
- boolean removeIf(Predicate<? super E> filter)
- protected void removeRange(int fromIndex, int toIndex)
- void replaceAll(UnaryOperator<E> operator)
- boolean retainAll(Collection<?> c)
- E set(int index, E element)
- int size()

ArrayList其他方法

- boolean add(E e)
- void add(int index, E element)
- boolean addAll(Collection<? extends E> c)
- boolean addAll(int index, Collection<? extends E> c)
- void clear()
- Object clone()
- boolean contains(Object o)
- void ensureCapacity(int minCapacity)
- void forEach(Consumer<? super E> action)
- E get(int index)
- int indexOf(Object o)
- boolean isEmpty()
- Iterator<E> iterator()

ArrayList其他方法（续）

- void sort(Comparator<? super E> c)
- Spliterator<E> spliterator()
- List<E> subList(int fromIndex, int toIndex)
- Object[] toArray()
- <T> T[] toArray(T[] a)
- void trimToSize()

结束语

遍历Collection

遍历实现了Collection接口的集合

- 通过Enumeration及Iterator接口遍历集合；
- 增强for循环遍历集合；
- 通过聚集操作遍历集合。

通过Enumeration及Iterator接口遍历

- Enumeration / Iterator
  - 能够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中的许多方法（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接口既可以用于ArrayList对象，也可以用于Vector对象

Iterator接口

- 是对Enumeration接口的改进，因此在遍历集合元素时，优先选用Iterator接口。
- 具有从正在遍历的集合中去除对象的能力。
- 具有如下三个实例方法
  - hasNext() —— 判断是否还有元素。
  - next() —— 取得下一个元素。
  - 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：Iterator举例

例：Iterator举例

```
import java.util.Vector;
import java.util.Iterator;
public class IteratorTester {
  public static void main(String args[]) {
    String[]  num = {"one", "two", "three", "four", "five",
                     "six", "seven", "eight", "nine", "ten"};
    Vector<String>   aVector = new Vector<String> (java.util.Arrays.asList(num));
    System.out.println("Before Vector: " + aVector);
    Iterator<String> nums = aVector.iterator();
    while(nums.hasNext()) {
      String aString = (String)nums.next();
      System.out.println(aString);
      if (aString.length() > 4)    nums.remove();
    }
    System.out.println("After Vector: " + aVector);
  }
}
```

例：Iterator举例

- 运行结果

Before Vector: [one, two, three, four, five, six, seven, eight, nine, ten]
one
two
three
four
five
six
seven
eight
nine
ten
After Vector: [one, two, four, five, six, nine, ten]

用增强for循环遍历

- 格式
  - for (Type a : 集合对象)

例：增强for循环遍历集合

```
import java.util.Vector;
import java.util.Enumeration;
public class ForTester {
    public static void main(String args[]) {
        Enumeration<String> days;
        Vector<String> dayNames = new Vector();
        dayNames.add("Sunday");        dayNames.add("Monday");
        dayNames.add("Tuesday");       dayNames.add("Wednesday");
        dayNames.add("Thursday");     dayNames.add("Friday");
        dayNames.add("Saturday");
        days = dayNames.elements();
        for (String day : dayNames) {
            System.out.println(day);
        }
    }
}
```

通过聚集操作遍历

- 聚集操作与lambda表达式一起使用。

通过聚集操作遍历

- 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：

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

结束语

Map接口及其实现

Map接口意义

- 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。
- 当需要通过关键字实现对值的快速存取时使用。

Map接口

- 抽象方法主要有
  - 查询方法
  - 修改方法
- 两个主要实现类
  - HashTable （1.0）
  - HashMap （1.2）

Map接口的查询方法

- int size() —— 返回Map中的元素个数
- boolean isEmpty() —— 返回Map中是否包含元素，如不包括任何元素，则返回true
- boolean containsKey(Object key) —— 判断给定的参数是否是Map中的一个关键字(key)
- boolean containsValue(Object val) —— 判断给定的参数是否是Map中的一个值(value)
- Object get(Object key) —— 返回Map中与给定关键字相关联的值(value)
- Collection values() —— 返回包含Map中所有值(value)的Collection对象
- Set keySet() ——返回包含Map中所有关键字(key)的Set对象
- Set entrySet() —— 返回包含Map中所有项的Set对象

Map接口的修改方法

- Object put(Object key, Object val) —— 将给定的关键字(key)/值(value)对加入到Map对象中。其中关键字(key)必须唯一，否则，新加入的值会取代Map对象中已有的值
- void putAll(Map m) —— 将给定的参数Map中的所有项加入到接收者Map对象中
- Object remove(Object key) —— 将关键字为给定参数的项从Map对象中删除
- void clear() —— 从Map对象中删除所有的项

哈希表(HashTable, HashMap)

- 也称为散列表，是用来存储群体对象的集合类结构。
- 其两个常用的类
  - HashTable
  - HashMap

哈希表(HashTable, HashMap)

- 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(Hash)函数。它使每个对象与一个唯一的存储位置相对应。因而在查找时，只要根据待查对象的关键属性k，计算f(k)的值即可知其存储位置

哈希表相关的主要概念

- 容量（capacity）—— 哈希表的容量不是固定的，随对象的加入，其容量可以自动扩充。

- 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，也可以是对象的一部分（如对象的某一个属性）。

- 哈希码（hash code）—— 要将对象存储到HashTable，就需要将其关键字key映射到一个整型数据，称为key的哈希码（hash code）。

- 哈希函数（hash function）——返回对象的哈希码。

- 项（item）—— 哈希表中的每一项都有两个域：关键字域key及值域value（即存储的对象）。key及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。

- 装填因子（load factor）—— (表中填入的项数) / (表的容量)。

HashMap构造方法

- HashMap()
  - 默认容量16，默认装填因字0.75。
- HashMap(int initialCapacity)
  - 容量initialCapacity，默认装填因字0.75。
- HashMap(int initialCapacity, float loadFactor)
  - 容量initialCapacity，装填因字loadFactor。
- HashMap(Map<? extends K,? extends V> m)
  - 以参数m为初值构造新的HashMap.

HashMap其他方法

- void clear()
- Object clone()
- V compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)
- V computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction)
- V computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction)
- boolean containsKey(Object key)
- boolean containsValue(Object value)
- Set<Map.Entry<K,V>> entrySet()

HashMap其他方法（续）

- void forEach(BiConsumer<? super K,? super V> action)
- V get(Object key)
- V getOrDefault(Object key, V defaultValue)
- boolean isEmpty()
- Set<K> keySet()
- V merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction)
- V put(K key, V value)
- void putAll(Map<? extends K,? extends V> m)

HashMap其他方法（续）

- V putIfAbsent(K key, V value)
- V remove(Object key)
- boolean remove(Object key, Object value)
- V replace(K key, V value)
- boolean replace(K key, V oldValue, V newValue)
- void replaceAll(BiFunction<? super K,? super V,? extends V> function)
- int size()
- Collection<V> values()

结束语

第6章小结

本章内容回顾

- Java集合框架介绍
- 接口及常用类概述
- 常用算法
- 数组实用方法
- 基于动态数组的类型(Vector, ArrayList)
- 遍历Collection
- Map接口及其实现

结束语