容器类

孙聪

网络与信息安全学院

2019-10-21

课程内容

- Java概述
- 面向对象程序设计概念
- Java语言基础
- Java面向对象特性
- Java高级特征
- 容器类
- 常用预定义类
- 异常处理
- 输入输出
- 线程

提要

提要

泛型的基本概念

- 泛型：又称为参数化类型，通过定义含有一个或多个类型参数的类或接口，对具有类似特征与行为的类进行抽象
  - 类型参数：可以指代任何具体类型
  - 例：JDK中java.util.ArrayList<E>的定义，ArrayList<E>定义了一个泛型，E为类型参数，它代表了能够放入ArrayList中的对象的类型

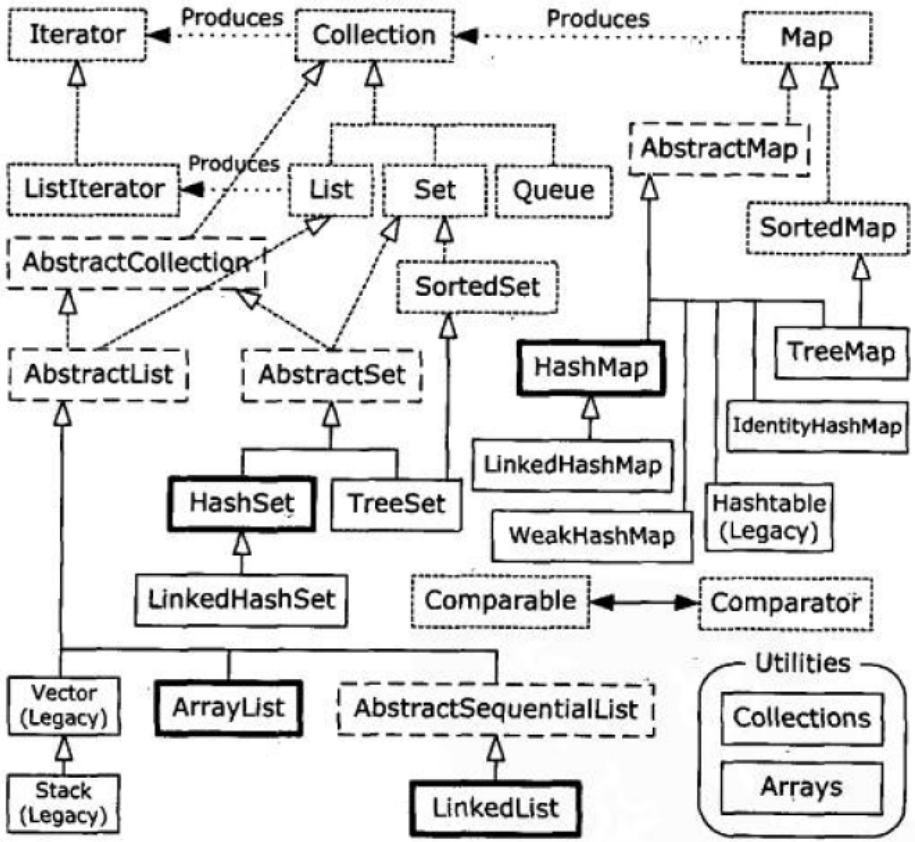
泛型的基本概念

- 泛型：又称为参数化类型，通过定义含有一个或多个类型参数的类或接口，对具有类似特征与行为的类进行抽象
  - 类型参数：可以指代任何具体类型
  - 例：JDK中java.util.ArrayList<E>的定义，ArrayList<E>定义了一个泛型，E为类型参数，它代表了能够放入ArrayList中的对象的类型
- 如何使用一个预定义的泛型？
  - 对泛型中的类型参数进行具体化，即可得到具体的类
  - 例：如果希望创建一个能够存放String对象的ArrayList，应使用ArrayList<String>进行声明。ArrayList<String>总是可以被看作一个具体的类，该类的实例作为容器可以存放String类型的对象

容器类

- 一个容器类的实例（容器、容器对象）表示了一组对象，容器对象存放指向其他对象的引用

完整的容器分类（可见容器接口树的结构）



简化的容器分类

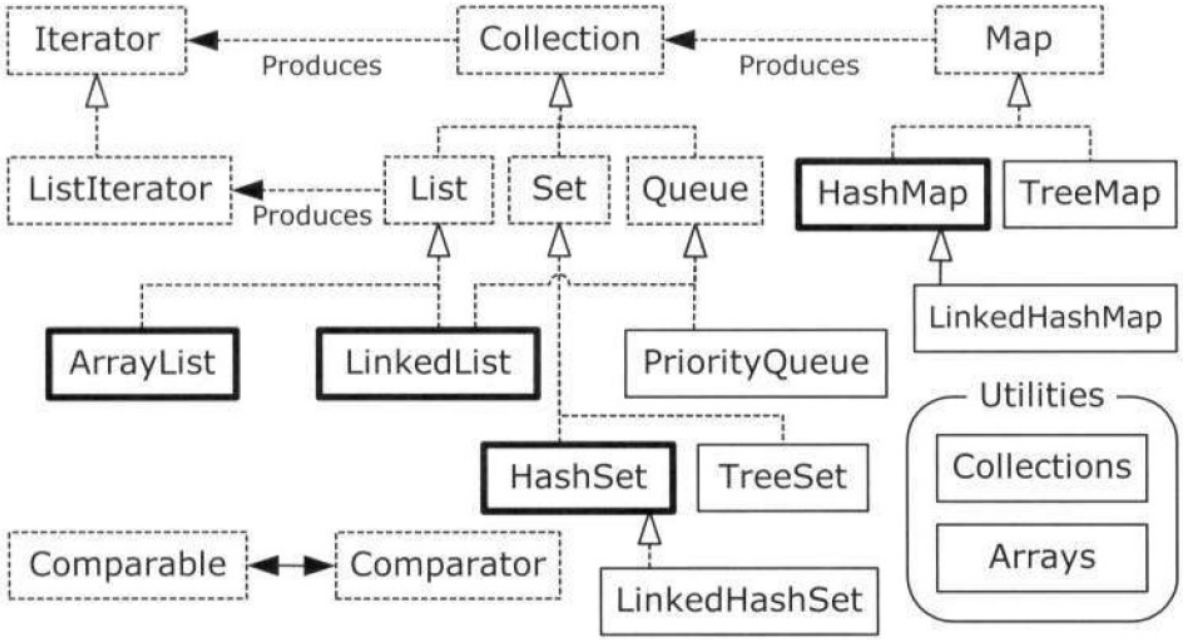
- 抽象类Abstract*：方便创建自己的容器类，多数情况下，已有容器类库足够，可忽略Abstract*
- legacy：Java 1.0/1.1容器（尽量不用）

Java 1.0/1.1的容器（legacy）	Java SE 5的容器
Vector	ArrayList
Enumeration	Iterator
Hashtable	HashMap
Stack	LinkedList
BitSet	EnumSet

简化的容器分类

- 去除Abstract*，legacy类，中间接口等，得到简化的容器分类

简化的容器分类



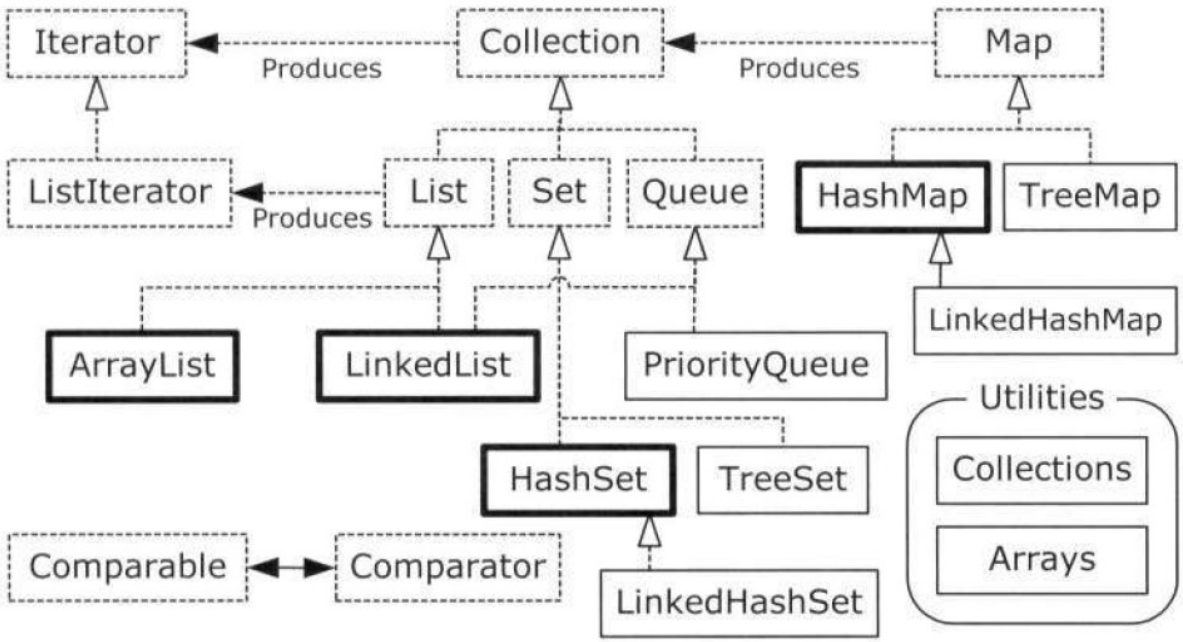
容器接口的基本特征

- Collection：集合接口树的根，定义通用的集合操作API
  - Set：集合。无序，不能包含重复元素
  - List：列表。有序，可包含重复元素，可通过索引序号访问元素
  - Queue：队列。必须按照排队规则来确定元素顺序
- Map：由一系列“键值对”组成的序列，允许通过键查找值
  - 不能包含重复的键
  - 每个键至多只映射到一个值
- SortedSet，SortedMap：具有排序功能的Set和Map

提要

Set接口

- 三种接口实现：HashSet，TreeSet，LinkedHashSet



Set接口

- 三种接口实现：HashSet，TreeSet，LinkedHashSet
- HashSet：采用Hash表实现Set接口，元素无固定顺序，对元素的访问效率高
- TreeSet：实现了SortedSet接口，采用有序树结构存储集合元素，元素按照比较结果的升序排列
- LinkedHashSet：采用Hash表和链表结合的方式实现Set接口，元素按照被添加的先后顺序保存

Set与Collection的主要接口方法

接口方法	用法
boolean add(E e)	如果给定元素不存在于当前Set中，则将其加入当前Set
boolean addAll(Collection<? extends E> c)	对于一个容器中的所有元素，如果它不存在于当前Set中，则将其加入当前Set。只要添加了任意元素就返回true，否则返回false
void clear()	移除当前Set中的所有元素
boolean contains(Object o)	如果指定元素在当前Set中，则返回true；否则返回false
boolean containsAll(Collection<?> c)	对于一个容器c，如果当前Set包含该容器中的所有元素，则返回true；否则返回false
boolean isEmpty()	如果当前Set不包含任何元素,则返回true;否则返回false
boolean remove(Object o)	如果指定元素在当前Set中，则移除该元素
boolean removeAll(Collection<?> c)	从当前Set中将指定容器中包含的所有元素都移除。只要有移除动作发生就返回true；否则返回false
boolean retainAll(Collection<?> c)	只保留当前Set中同时也包含于指定容器中的元素。集合发生了改变就返回true；否则返回false
int size()	返回当前Set中元素的数量
Object[] toArray()	返回一个数组，该数组由当前Set中所有元素组成
<T> T[] toArray(T[] a)	返回一个数组，该数组由当前Set中所有元素组成，返回数组的运行时类型由参数指定

```
public class TestSet{
    public static void main(String[] args){
        Random rand=new Random(47);
        Set<Integer> s=new HashSet<Integer>();
        for(int i=0;i<5000;i++){ s.add(rand.nextInt(40)); }
        System.out.println(s);

        s=new TreeSet<Integer>();
        for(int i=0;i<5000;i++){ s.add(rand.nextInt(40)); }
        System.out.println(s);

        s=new LinkedHashSet<Integer>();
        for(int i=0;i<5000;i++){ s.add(rand.nextInt(40)); }
        System.out.println(s);
    }
}
```

Set接口

- 注意
  - 如果要将T类型的实例放入HashSet或TreeSet中，需要为T定义equals()方法
  - 如果要将T的实例放入HashSet或LinkedHashSet中，需要为T定义hashCode()方法

equals()方法与对象等价性

- 如何测试对象的等价性？
  - “==”和“!=”比较的是对象的引用而非对象的内容
    （引用相等：指向同一块堆空间）
  - 所有对象都拥有从Object类继承的equals()方法，
    equals()方法默认也是比较对象的引用

equals()方法与对象等价性

- 如何测试对象的等价性？
  - “==”和“!=”比较的是对象的引用而非对象的内容
    （引用相等：指向同一块堆空间）
  - 所有对象都拥有从Object类继承的equals()方法，
    equals()方法默认也是比较对象的引用
- 如何比较两个不同对象的内容是否相等？
  - 需要对equals()方法进行重写，以比较对象内容
  - 很多Java类库都已通过重写equals()实现了对于对象内容的比较

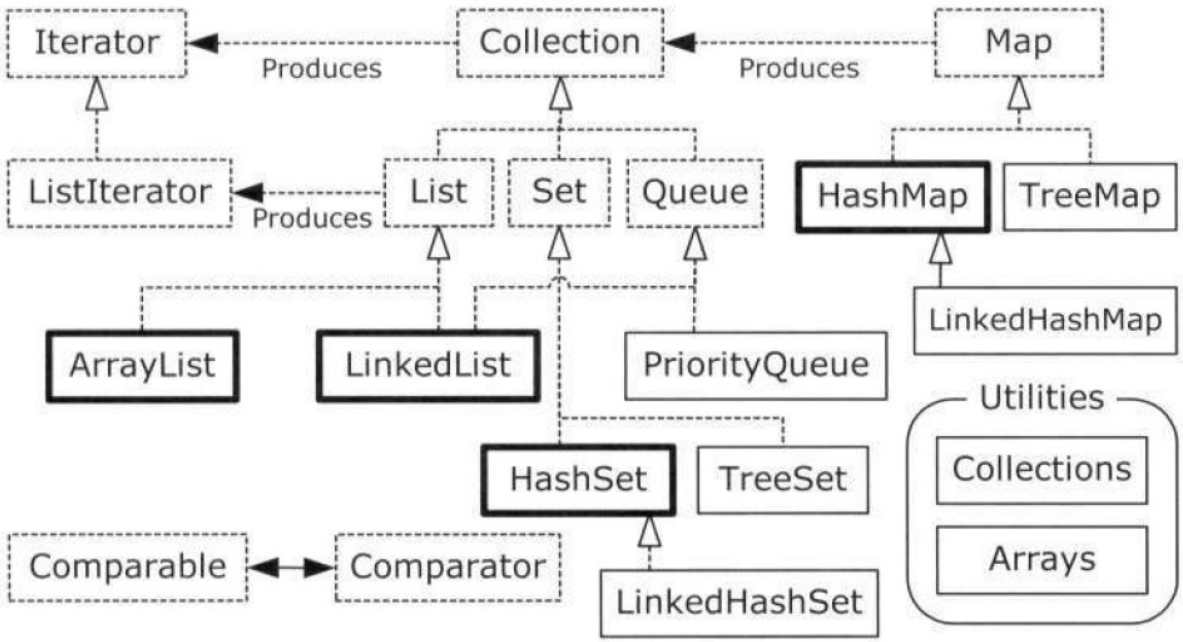
equals()方法与对象等价性

```
class Value {
    int i;
    public boolean equals(Value v) {
        return (this.i == v.i) ? true : false;
    }
}
public class Equivalence {
    public static void main(String[] args) {
        Integer n1=new Integer(47);
        Integer n2=new Integer(47);
        System.out.println("n1==n2: " + (n1==n2));                    //对引用的比较
        System.out.println("n1!=n2: " + (n1!=n2));
        System.out.println("n1.equals(n2): "+ n1.equals(n2)); //Integer已重写equals()
        Value v1 = new Value();
        Value v2 = new Value();
        v1.i = v2.i = 10;
        System.out.println("v1.equals(v2): "  + v1.equals(v2));
    }
}
```

提要

List接口

- 两种接口实现：ArrayList, LinkedList



List接口

- 两种接口实现：ArrayList，LinkedList
- ArrayList：采用可变大小的数组实现List接口
  - 无需声明上限，随着元素的增加，长度自动增加
  - 对元素的随机访问速度快，插入/移除元素较慢
  - 该类是非同步的，相对于Vector（legacy）效率高
- LinkedList：采用链表结构实现List接口
  - 实际上实现了List接口、Queue接口和双端队列Deque接口，因此可用来实现堆栈、队列或双端队列
  - 插入/移除元素快，对元素的随机访问较慢
  - 该类是非同步的

List接口方法

接口方法	用法
void add(int index, E element)	向当前List的指定位置插入特定元素
boolean addAll(int index, Collection<? extends E> c)	向当前List的指定位置插入一个容器中的所有元素
E get(int index)	返回当前List中指定位置的元素
int indexOf(Object o)	返回指定元素在当前List中第一次出现时的索引值，如果指定元素在当前List中不存在，则返回-1
int lastIndexOf(Object o)	返回指定元素在当前List中最后一次出现时的索引值，如果指定元素在当前List中不存在，则返回-1
E remove(int index)	移除当前List中指定位置的元素，后续对象依次前提
E set(int index, E element)	用指定的元素替换当前List中指定位置的元素
List<E> subList(int fromIndex, int toIndex)	返回一个作为当前List一部分的子List的视图，子List的索引范围是[fromIndex, toIndex)。子视图不是一个新的List，对子List的更改和删除将能反映到原List

ArrayList的构造方法

构造方法	用法
public ArrayList()	创建一个空ArrayList，初始长度为10
public ArrayList(int initialCapacity)	创建一个空ArrayList，初始长度为initialCapacity
public ArrayList(Collection<E> c)	由指定集合的元素创建ArrayList，顺序由集合的迭代器决定

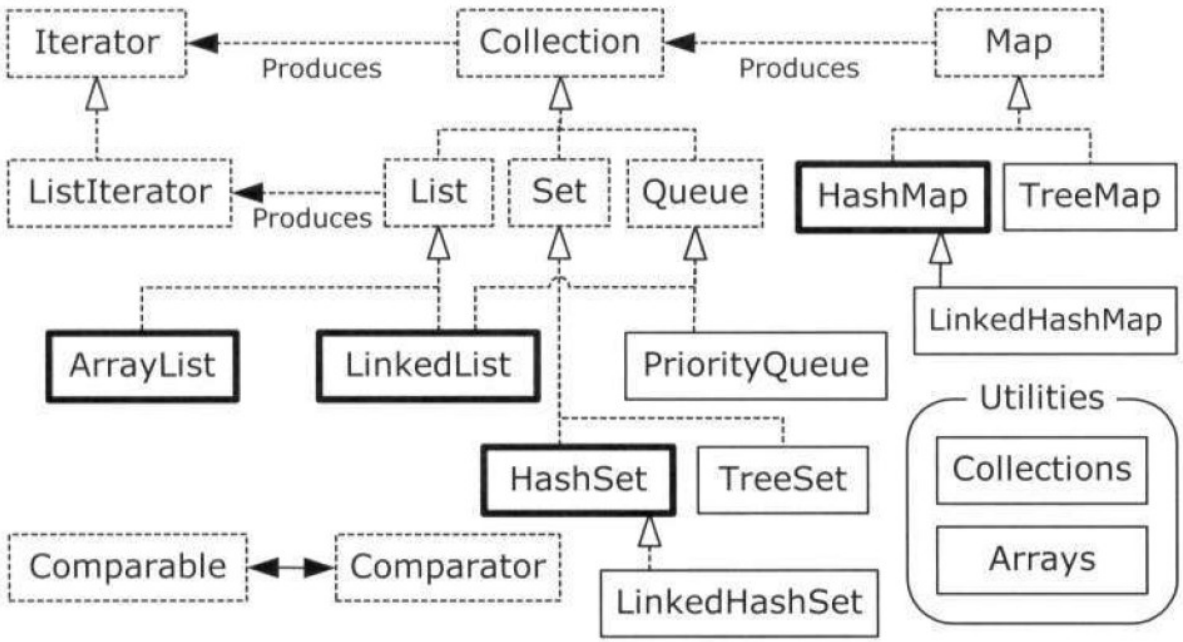
```
public class UseArrayList {
    public static void main(String[] args) {
        List<String> scores = new ArrayList<String>();
        scores.add("86");          // 添加元素
        scores.add("98");          // 添加元素
        scores.add(1, "99");       // 插入元素
        for (int i = 0; i < scores.size(); i++) {
            System.out.print(scores.get(i) + " ");  // 输出结果
        }
        scores.set(1, "77");       // 修改第二个元素
        scores.remove(0);          // 删除第一个元素
        System.out.println("\n修改并删除之后");
        for (int i = 0; i < scores.size(); i++) {
            System.out.print(scores.get(i) + " ");
        }
        System.out.println(" \n按字符串输出\n" + scores.toString());
    }
}
```

- 又例：TestShapesArrayList.java

提要

Queue接口

- 两种接口实现：LinkedList, PriorityQueue



Queue接口方法

接口方法	用法
boolean offer(E e)	向队列末尾加入指定元素，如果成功则返回true，否则返回false
boolean add(E e)	向队列末尾加入指定元素，如果加入失败则抛出异常
E peek()	获取但不删除队首元素（LinkedList的第一个元素），如果队列为空，则返回null
E element()	获取但不删除队首元素（LinkedList的第一个元素），如果队列为空，则抛出异常
E poll()	获取并删除队首元素（LinkedList的第一个元素），如果队列为空，则返回null
E remove()	获取并删除队首元素（LinkedList的第一个元素），如果队列为空，则抛出异常

Queue接口方法

接口方法	用法
boolean offer(E e)	向队列末尾加入指定元素，如果成功则返回true，否则返回false
boolean add(E e)	向队列末尾加入指定元素，如果加入失败则抛出异常
E peek()	获取但不删除队首元素（LinkedList的第一个元素），如果队列为空，则返回null
E element()	获取但不删除队首元素（LinkedList的第一个元素），如果队列为空，则抛出异常
E poll()	获取并删除队首元素（LinkedList的第一个元素），如果队列为空，则返回null
E remove()	获取并删除队首元素（LinkedList的第一个元素），如果队列为空，则抛出异常

- Queue提供两种形式的插入、删除和元素检查

操作	功能说明	异常情况	抛出异常的方法	返回特定值的方法
插入	向队尾加入一个元素	有界队列满	add(e)	offer(e),返回false
删除	从队首移去一个元素	队列空	remove()	poll(),返回null
元素检查	返回队首元素但不删除	队列空	element()	peek(),返回null

```
public class TestQueue{
    public static void printQueue(Queue q){
        while(q.peek()!=null)
            System.out.print(q.remove()+" ");
        System.out.println();
    }
    public static void main(String[] args){
        Queue<Integer> q=new LinkedList<Integer>();
        Random rand=new Random(37);
        for(int i=0;i<10;i++)
            q.offer(rand.nextInt(i+10));
        TestQueue.printQueue(q);

        Queue<Character> qc=new LinkedList<Character>();
        for(char c: "JavaLanguage".toCharArray())
            qc.offer(c);
        TestQueue.printQueue(qc);
    }
}
```

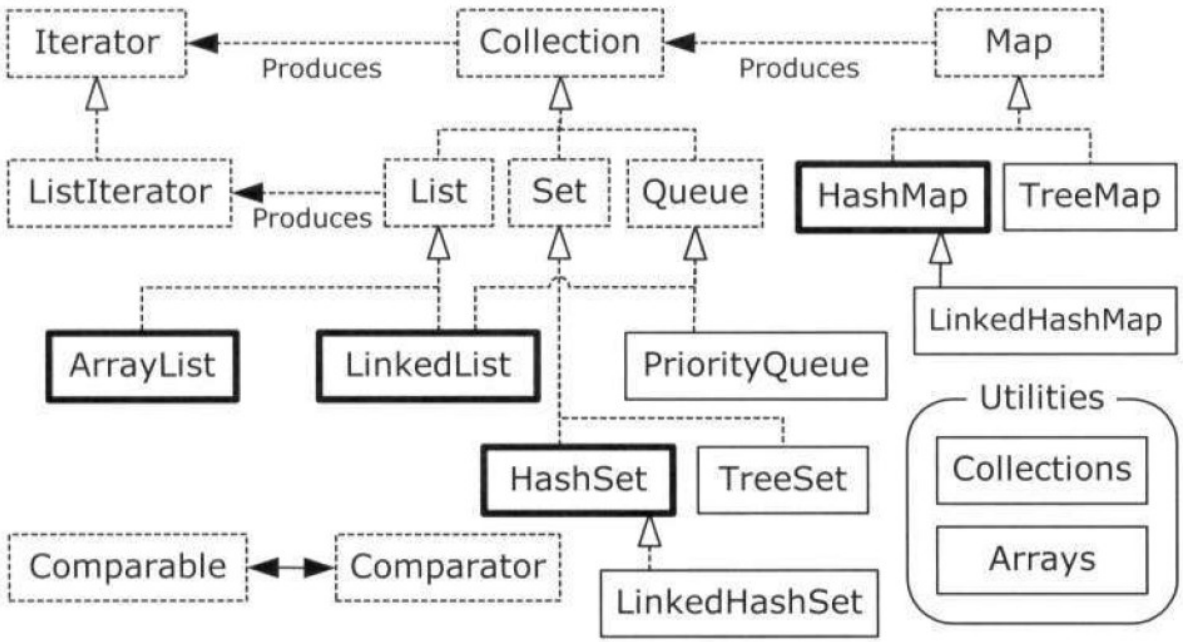
优先队列（PriorityQueue）

- 队列规则：给定一组队列中的元素的情况下，确定下一个弹出队列的元素的规则
  - 一般队列：FIFO
  - 优先队列：选择优先级最高的元素
- 当调用offer()插入对象时，对象在队列中排序
  - Integer、String、Character等类型的默认排序采用“自然顺序”
  - 可通过提供自己的Comparator修改排序顺序
  - 例：TestPriorityQueue.java

提要

Map接口

- 接口实现：HashMap, TreeMap, LinkedHashMap



Map接口

- 接口实现：HashMap，TreeMap，LinkedHashMap
- 把键映射到某个值
  - 一个键最多只能映射一个值
  - 一个值可对应多个键

Map接口

- 接口实现：HashMap，TreeMap，LinkedHashMap
- 把键映射到某个值
  - 一个键最多只能映射一个值
  - 一个值可对应多个键
- 常用：HashMap（无序）和TreeMap（有序）
- HashMap：使用Hash表实现Map接口
  - 无序，非同步且允许空的键与值
  - 相比Hashtable（legacy）效率高
- TreeMap：与TreeSet类似，使用有序树实现SortedMap接口
  - 保证“键”始终处于排序状态

Map接口方法

接口方法	用法
V put(K key, V value)	将指定的值value关联到当前Map中的指定的键key
void putAll(Map<? extends K,? extends V> m)	将参数Map中的所有键值对都添加到当前Map中
V get(Object key)	返回当前Map中指定的键所映射到的值，如果没有与该键相对应的映射，就返回null
V remove(Object key)	移除当前Map中指定的键所对应的键值对
void clear()	从当前Map中移除所有键值对
boolean containsKey(Object key)	如果当前Map中存在从指定的键所发出的映射，则返回true；否则返回false
boolean containsValue(Object value)	如果当前Map中存在一个或多个键，能够映射到指定的值，那么返回true；否则返回false
boolean isEmpty()	如果当前Map中不含任何键值对，则返回true；否则返回false
Set<Map.Entry<K,V>> entrySet()	返回一个由当前Map中所有键值对所组成的Set
Set<K> keySet()	返回一个Set，其中包含了当前Map中的所有键
Collection<V> values()	返回一个Collection，其中包含了当前Map中的所有值
int size()	返回当前Map中的键值对的数量

HashMap的构造方法

构造方法	用法
public HashMap()	构造空HashMap，默认初始容量为16，默认load因子为0.75
public HashMap(int initialCapacity)	构造空HashMap，指定初始容量为initialCapacity，默认load因子为0.75
public HashMap(int initialCapacity, float loadFactor)	构造空HashMap，指定初始容量为initialCapacity，指定load因子为loadFactor

```
public class Freq {
    public static void main(String args[]) {
        String[] words = { “if”, “it”, “is”, “to”, “be”, “it”,
                “is”, “up”, “to”, “me”, “to”, “delegate” };
        Integer freq;
        Map<String, Integer> m = new TreeMap<String, Integer>();

        for (String a : words) {    //以(单词,词频)为键值对, 构造频率表
            freq = m.get(a);        // 获取指定单词的词频。
            if (freq == null) {     // 词频递增
                freq = new Integer(1);
            } else {
                freq = new Integer(freq + 1);   // .intValue()
            }
            m.put(a, freq);         // 在Map中更改词频
        }
        System.out.println(m.size() + “ distinct words detected:”);
        System.out.println(m);
    }
}
```

- 又例：TestMap.java

提要

迭代器(Iterator)

- Iterator是一个轻量级对象，用于遍历并选择序列中的对象
- 用法
  - 使用容器的iterator()方法返回容器的迭代器，该迭代器准备返回容器的第一个元素
  - 迭代器只能单向移动
  - next()：获得序列的下一个元素
  - hasNext()：检查序列中是否还有元素
  - remove()：将迭代器新近返回的元素（即由next()产生的最后一个元素）删除，因此在调用remove()之前必须先调用next()

```
public class TestIterator{
    public static void main(String[] args){
      String sentence=“I believe I can fly, I believe I can touch the sky.”;
       String[] strs=sentence.split(“ ”);
       List<String> list=new ArrayList<String>( Arrays.asList(strs) );
       Iterator<String> it=list.iterator();
       while(it.hasNext())
           System.out.print(it.next()+“_”);
       System.out.println();

       it=list.iterator();
       while(it.hasNext()){
           if(it.next().equals(“I”))
               it.remove();
       }
       it=list.iterator();
       while(it.hasNext())
           System.out.print(it.next()+“ ”);
       System.out.println();
    }
}
```

ListIterator

- Iterator的子类，只能用于各种List类的访问
- 用法
  - List容器的listIterator()方法产生一个指向List开始处的ListIterator对象
  - 可以双向移动
  - hasPrevious()：检查序列中是否有前一个元素
  - previous()：返回序列中的前一个元素
  - nextIndex()：返回下一次next()方法调用将要返回的元素的索引
  - previousIndex()：返回下一次previous()方法调用将要返回的元素的索引
  - set(E e)：将上一次next()或previous()所返回的元素替换为e
  - add(E e)：向序列中下一个next()被访问元素之前（亦即下一个previous()被访问元素之后）加入e

```
public class TestListIterator{
    public static void main(String[] args){
        String sentence=“I believe I can fly, I believe I can touch the sky.”;
        String[] strs=sentence.split(“ ”);
        List<String> list=new ArrayList<String>( Arrays.asList(strs) );

        ListIterator li=list.listIterator();
        while(li.hasNext())
            System.out.print(li.next()+“_”);
        System.out.println();
        while(li.hasPrevious())
            System.out.print(li.previous()+“_”);
        System.out.println();

        while(li.hasNext()){
            if(li.next().equals(“I”)){ li.set(“You”); }
        }
        li=list.listIterator();
        while(li.hasNext())
            System.out.print(li.next()+“ ”);
        System.out.println();

        li=list.listIterator();
        while(li.hasNext()){
            if(li.next().equals(“You”)){ li.add(“and I”); }
        }
        li=list.listIterator();
        while(li.hasNext())
            System.out.print(li.next()+“ ”);
        System.out.println();
    }
}
```

Collection与增强型for循环

- 通过增强型for循环遍历Collection中的元素

```
public class CollectionWithForeach{
    public static void main(String[] args){
      String sentence=“I believe I can fly, I believe I can touch the sky.”;
       String[] strs=sentence.split(“ ”);
       Collection<String> c=new ArrayList<String>( Arrays.asList(strs) );

       for(String s: c){
           System.out.print(s+“_”);
       }
    }
}
```