第十五章 集合类库（下）

15.1 泛型机制（熟悉）

15.1.1 基本概念

- 通常情况下集合中可以存放不同类型的对象，是因为将所有对象都看做Object类型放入的，因此从集合中取出元素时也是Object类型，为了表达该元素真实的数据类型，则需要强制类型转换，而强制类型转换可能会引发类型转换异常。
- 为了避免上述错误的发生，从Java5开始增加泛型机制，也就是在集合名称的右侧使用<数据类型>的方式来明确要求该集合中可以存放的元素类型，若放入其它类型的元素则编译报错。
- 泛型只在编译时期有效，在运行时期不区分是什么类型。

15.1.2 底层原理

- 泛型的本质就是参数化类型，也就是让数据类型作为参数传递，其中E相当于形式参数负责占位，而使用集合时<>中的数据类型相当于实际参数，用于给形式参数E进行初始化，从而使得集合中所有的E被实际参数替换，由于实际参数可以传递各种各样广泛的数据类型，因此得名为泛型。
- 如：

```
//其中i叫做形式参数，负责占位
//int i = 10;
//int i = 20;
public static void show(int i) {
    ...
}
//其中10叫做实际参数，负责给形式参数初始化
show(10);
show(20);
```

```
其中E叫做形式参数，负责占位
E = String;
E = Integer;
public interface List {
    ...
}
//其中String叫做实际参数
List lt1 = ...;
List lt2 = ...;
```

15.1.3 自定义泛型接口

- 泛型接口和普通接口的区别就是后面添加了类型参数列表，可以有多个类型参数，如：<E, T, .. >等。

15.1.4 自定义泛型类

- 泛型类和普通类的区别就是类名后面添加了类型参数列表，可以有多个类型参数，如：<E, T, .. >等。
- 实例化泛型类时应该指定具体的数据类型，并且是引用数据类型而不是基本数据类型。
- 父类有泛型，子类可以选择保留泛型也可以选择指定泛型类型。
- 子类必须是“富二代”，子类除了指定或保留父类的泛型，还可以增加自己的泛型。

15.1.5 自定义泛型方法

- 泛型方法就是我们输入参数的时候，输入的是泛型参数，而不是具体的参数。我们在调用这个泛型方法的时需要对泛型参数进行实例化。
- 泛型方法的格式：
  [访问权限] <泛型> 返回值类型 方法名([泛型标识 参数名称]) { 方法体; }
- 在静态方法中使用泛型参数的时候，需要我们把静态方法定义为泛型方法。

15.1.6 泛型在继承上的体现

- 如果B是A的一个子类或子接口，而G是具有泛型声明的类或接口，则G**并不是G的子类型！**
  比如：String是Object的子类，但是List并不是List的子类。

15.1.7 通配符的使用

- **有时候我们希望传入的类型在一个指定的范围内，此时就可以使用泛型通配符了。**
- **如：之前传入的类型要求为Integer类型，但是后来业务需要Integer的父类Number类也可以传入。**
- **泛型中有三种通配符形式：**

  <?> 无限制通配符：表示我们可以传入任意类型的参数。
  <? extends E> 表示类型的上界是E，只能是E或者是E的子类。
  <? super E> 表示类型的下界是E，只能是E或者是E的父类。

15.2 Set集合（熟悉）

15.2.1 基本概念

- **java.util.Set集合是Collection集合的子集合，与List集合平级。**
- **该集合中元素没有先后放入次序，且不允许重复。**
- **该集合的主要实现类是：HashSet类 和 TreeSet类以及LinkedHashSet类。**
- **其中HashSet类的底层是采用哈希表进行数据管理的。**
- **其中TreeSet类的底层是采用红黑树进行数据管理的。**
- **其中LinkedHashSet类与HashSet类的不同之处在于内部维护了一个双向链表，链表中记录了元素的迭代顺序，也就是元素插入集合中的先后顺序，因此便于迭代。**

15.2.2 常用的方法

- **参考Collection集合中的方法即可！**
- **案例题目**

  准备一个Set集合指向HashSet对象，向该集合中添加元素"two"并打印，再向集合中添加元素"one"并打印，再向集合中添加元素"three"并打印，再向集合中添加"one"并打印。

15.2.3 元素放入HashSet集合的原理

- **使用元素调用hashCode方法获取对应的哈希码值，再由某种哈希算法计算出该元素在数组中的索引位置。**
- **若该位置没有元素，则将该元素直接放入即可。**
- **若该位置有元素，则使用新元素与已有元素依次比较哈希值，若哈希值不相同，则将该元素直接放入。**
- **若新元素与已有元素的哈希值相同，则使用新元素调用equals方法与已有元素依次比较。**
- **若相等则添加元素失败，否则将元素直接放入即可。**

- **思考：为什么要求重写equals方法后要重写hashCode方法呢？**

- 解析：
  当两个元素调用equals方法相等时证明这两个元素相同，重写hashCode方法后保证这两个元素得到的哈希码值相同，由同一个哈希算法生成的索引位置相同，此时只需要与该索引位置已有元素比较即可，从而提高效率并避免重复元素的出现。

15.2.5 TreeSet集合的概念

- 二叉树主要指每个节点最多只有两个子节点的树形结构。
- 满足以下3个特征的二叉树叫做有序二叉树。
  a.左子树中的任意节点元素都小于根节点元素值；
  b.右子树中的任意节点元素都大于根节点元素值；
  c.左子树和右子树的内部也遵守上述规则；
- 由于TreeSet集合的底层采用红黑树进行数据的管理，当有新元素插入到TreeSet集合时，需要使用新元素与集合中已有的元素依次比较来确定新元素的合理位置。
- 比较元素大小的规则有两种方式：
  使用元素的自然排序规则进行比较并排序，让元素类型实现java.lang.Comparable接口；
  使用比较器规则进行比较并排序，构造TreeSet集合时传入java.util.Comparator接口；
- 自然排序的规则比较单一，而比较器的规则比较多元化，而且比较器优先于自然排序；

15.3 Map集合（重点）

15.3.1 基本概念

- java.util.Map<K,V>集合中存取元素的基本单位是：单对元素，其中类型参数如下：
  K - 此映射所维护的键(Key)的类型，相当于目录。
  V - 映射值(Value)的类型，相当于内容。
- 该集合中key是不允许重复的，而且一个key只能对应一个value。
- 该集合的主要实现类有：HashMap类、TreeMap类、LinkedHashMap类、Hashtable类、Properties类。
- 其中HashMap类的底层是采用哈希表进行数据管理的。
- 其中TreeMap类的底层是采用红黑树进行数据管理的。
- 其中LinkedHashMap类与HashMap类的不同之处在于内部维护了一个双向链表，链表中记录了元素的迭代顺序，也就是元素插入集合中的先后顺序，因此便于迭代。
- 其中Hashtable类是古老的Map实现类，与HashMap类相比属于线程安全的类，且不允许null作为key或者value的数值。
- 其中Properties类是Hashtable类的子类，该对象用于处理属性文件，key和value都是String类型的。
- Map集合是面向查询优化的数据结构, 在大数据量情况下有着优良的查询性能。
- 经常用于根据key检索value的业务场景。

15.3.2 常用的方法

方法声明	功能介绍
V put(K key, V value)	将Key-Value对存入Map，若集合中已经包含该Key，则替换该Key所对应的Value，返回值为该Key原来所对应的Value，若没有则返回null
V get(Object key)	返回与参数Key所对应的Value对象，如果不存在则返回null
boolean containsKey(Object key);	判断集合中是否包含指定的Key
boolean containsValue (Object value);	判断集合中是否包含指定的Value
V remove(Object key)	根据参数指定的key进行删除
Set keySet()	返回此映射中包含的键的Set视图
Collection values()	返回此映射中包含的值的Set视图
Set<Map.Entry<K,V>> entrySet()	返回此映射中包含的映射的Set视图

15.3.3 元素放入HashMap集合的原理

- **使用元素的key调用hashCode方法获取对应的哈希码值，再由某种哈希算法计算在数组中的索引位置。**
- **若该位置没有元素，则将该键值对直接放入即可。**
- **若该位置有元素，则使用key与已有元素依次比较哈希值，若哈希值不相同，则将该元素直接放入。**
- **若key与已有元素的哈希值相同，则使用key调用equals方法与已有元素依次比较。**
- **若相等则将对应的value修改，否则将键值对直接放入即可。**

15.3.4 相关的常量

- **DEFAULT_INITIAL_CAPACITY : HashMap的默认容量是16。**
- **DEFAULT_LOAD_FACTOR：HashMap的默认加载因子是0.75。**
- **threshold：扩容的临界值，该数值为：容量*填充因子，也就是12。**
- **TREEIFY_THRESHOLD：若Bucket中链表长度大于该默认值则转化为红黑树存储，该数值是8。**
- **MIN_TREEIFY_CAPACITY：桶中的Node被树化时最小的hash表容量，该数值是64。**

- **案例题目**

  准备一个HashMap集合，统计字符串"123,456,789,123,456"中每个数字字符串出现的次数并打印出来。
  如：
  123 出现了 2次
  456 出现了 2次
  789 出现了 1次

15.4 Collections类

15.4.1 基本概念

- **java.util.Collections类主要提供了对集合操作或者返回集合的静态方法。**

15.4.2 常用的方法

方法声明	功能介绍
static <T extends Object & Comparable<? super T>> T max(Collection<? extends T> coll)	根据元素的自然顺序返回给定集合的最大元素
static T max(Collection<? extends T> coll, Comparator<? super T> comp)	根据指定比较器引发的顺序返回给定集合的最大元素
static <T extends Object & Comparable<?super T>> T min(Collection<? extends T> coll)	根据元素的自然顺序返回给定集合的最小元素
static T min(Collection<? extends T> coll, Comparator<? super T> comp)	根据指定比较器引发的顺序返回给定集合的最小元素
static void copy(List<? super T> dest, List<? extends T> src)	将一个列表中的所有元素复制到另一个列表中

方法声明	功能介绍
static void reverse(List<?> list)	反转指定列表中元素的顺序
static void shuffle(List<?> list)	使用默认的随机源随机置换指定的列表
static <T extends Comparable<? super T>> void sort(List list)	根据其元素的自然顺序将指定列表按升序排序
static void sort(List list, Comparator<? super T> c)	根据指定比较器指定的顺序对指定列表进行排序
static void swap(List<?> list, int i, int j)	交换指定列表中指定位置的元素