**6.集合类**

* 集合概述
  + JDK中提供了一系列特殊的类，这些类可以存储任意类型的对象，并且长度可变，在Java中这些类被统称为集合。集合类都位于java.util包中，在使用时一定要注意导包的问题，否则会出现异常。
* Collection接口
  + Collection是所有单列集合的父接口，因此在Collection中定义了单列集合（List和Set）通用的一些方法，这些方法可用于操作所有的单列集合。
  + boolean add(Object o) : 向集合中添加一个元素。
  + boolean addAll(Object o) : 将指定Collection中的所有元素添加到该集合中。
  + void clear() : 删除该集合中的所有元素。
  + boolean remove(Object o) : 删除该集合中指定的元素。
  + boolean removeAll(Collection c) : 删除指定集合中的所有元素。
  + boolean isEmpty() : 判断该集合是否为空。
  + boolean contains(Object o) : 判断该集合中是否包含某个元素。
  + boolean containsAll(Collection c) : 判断该集合中是否包含给定集合中的所有元素。
  + Iterator iterator() : 返回在该集合的元素上进行迭代的迭代器（Iterator），用于遍历该集合所有元素。
  + int size() : 获取该集合元素个数。
* List接口
  + List接口简介
    - List接口继承自Collection接口，是单列集合的一个重要分支，习惯性地将实现了List接口的对象称为List集合。在List集合中允许出现重复的元素，所有的元素都是以一种线性方式存储的，在程序中可以通过索引来访问集合中的指定元素。另外，List集合中还有一个特点就是元素有序，即元素的存入顺序和取出顺序一致。
    - List集合特有方法：
    - void add(int index, Object element) : 将元素element插入在List集合的index处。
    - bool boolean addAll(int index, Collection c) : 将集合C处所包含的所有元素插入到List集合的index处。
    - Object get(int index) : 返回集合索引index处的元素。
    - Object remove(int index) : 删除index索引处的元素。
    - Object set(int index, Object element) : 将索引index处元素替换成element对象，并将替换后的集合返回。
    - int indexOf(object o) : 返回对象o在List集合中出现的位置索引。
    - int lastIndexOf(Object o) : 返回对象o在List集合中最后一次出现的位置索引。
    - List subList(int fromIndex, int toIndex) : 返回从索引fromIndex（包括）到toIndex（不包括）处所有元素集合组成的子集合。
  + ArrayList集合
    - ArrayList集合是List接口的一个实现类，它是程序中最常见的一种集合。它的add（）方法和get（）方法用于实现元素的存取。此数据结构长于便利查询。
  + LinkedList集合
    - LinkedList作为List接口的另一个实现类，长于元素的增删。
    - 针对元素的增删操作，LinkedList集合定义了一些特有的方法：
    - void add(int index, E element) : 在此列表中指定的位置插入指定的元素。
    - void addFirst(Object o) : 将指定元素插入此列表的开头。
    - void addLast(Object o) : 将指定元素添加到此列表的结尾。
    - Object getFirst() : 返回此列表的第一个元素。
    - Object getLast() : 返回此列表的最后一个元素。
    - Object removeFirst() : 移除并返回此列表的第一个元素。
    - Object removeLast() : 移除并返回此列表的最后一个元素。
  + Iterator接口
    - Iterator主要用于迭代访问（即遍历）Collection中的元素，因此Iterator对象也被称为迭代器。
    - 由于Iterator的某种特性，很明显这种特性产生于它是独立于List而被单独构造出来的新对象，故在应用Iterator进行遍历数组的同时如果调用了集合的remove（）方法导致集合的结构发生了变化，Iterator如果继续向下迭代的话就会报错。此时有两个处理方法：
    - 1、在调用了集合的remove（）方法之后就break退出Iterator的向下迭代。
    - 2、抛弃集合的remove（）方法，而使用Iterator迭代器中的remove（）方法对元素进行删改。
  + foreach循环
    - foreach循环是一种更加简洁的for循环，也称增强for循环。但是需要注意的是在应用foreach语句对字符串数组进行遍历的同时，应用其临时变量去修改字符串中的值的话，在逻辑上就是不能达成的，因为在foreach中并没有真正引用到字符串数组而是通过一个字符串变量来间接达成了遍历的目的，此时修改的话只会改变临时变量的引用指向，而对真实的字符串数组中存储的值没有影响。
* Set接口
  + Set接口简介
    - Set借口和List接口一样，同样继承自Collection接口，它与Collection接口中的方法基本一致，并没有对Collection接口进行功能上的扩充，只是比Collection接口更加严格了。与List接口不同的是，Set接口中元素无序，并且都会以某种规则保证存入的元素不会重复。
    - Set接口主要有两个实现类，一个是HashSet，另一个是TreeSet。其中HashSet根据对象的哈希值来确定元素在集合中的存储位置，因此具有良好的存取和查找性能。TreeSet则是以二叉树的方式来存储元素，它可以实现对集合中的元素进行排序。
  + HashSet集合
    - HashSet是Set接口的一个实现类，它所存储的元素是不可重复的，并且元素都是无序的。当向HashSet集合中添加一个对象时，首先会调用该对象的hashCode（）方法来计算对象的哈希值，从而确定元素的存储位置。如果此时哈希值相同，在调用对象的equals（）方法来确保该位置没有重复元素。
    - 当向集合中存入元素时，为了保证HashSet正常工作，要求在存入对象时，重写Object类中的hashCode（）和equals（）方法。
* Map接口
  + Map接口简介
    - Map接口是一种双列集合，它的每个元素都包含一个键对象Key和值对象Value。键和值之间存在一种对应关系，称为映射。
    - Map接口常用方法：
      * void put(Object key, Object value) : Associates the specified value with the specified key in this map (optional operation).
      * Object get(Object key) : Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
      * boolean containsKey(Object key) : Returns true if this map contains a mapping for the specified key.
      * boolean containsValue(Object value) : Returns true if this map maps one or more keys to the specified value.
      * Set keySet() : Returns a Set view of the keys contained in this map.
      * Collection<V> values() : Returns a Collection view of the values contained in this map.
      * Set<Map.Entry<K,V>> entrySet() : Returns a Set view of the mappings contained in this map.
  + HashMap集合
    - HashMap集合是Map集合的一个实现类，它用于存储键值映射关系，同时Map中的键必须是唯一的，简而言之就是键相同，值覆盖。HashMap集合迭代出来元素的顺序和存入的顺序是不一致的。如果想让这两个顺序一致，可以使用Java中提供的LinkedHashMap类。
    - 如何遍历Map集合中所有的键值对呢？有两种方法可以实现：
      * 1、先遍历Map集合中所有的键（通过keySet（）方法实现），再根据键获取相应的值。
      * 2、先获取集合中所有的映射关系，然后从映射关系中取出键和值。（Set<Map.Entry<K,V>> entrySet() ：Returns a Set view of the mappings contained in this map.）
  + Properties集合
    - Map接口中还有一个实现类HashTable，它和HashMap十分相似，区别在于HashTable是线程安全的。Hashtable存取元素时速度很慢，目前基本上被HashMap所取代，但Hashtable类有一个子类Properties，在实际应用中非常重要。
    - Hashtable中有一些独特的存取方法：Object setProperty(String key, String value) ： Calls the Hashtable method put. String getProperty(String key) ： Searches for the property with the specified key in this property list.Enumeration<?> propertyNames() ： Returns an enumeration of all the keys in this property list, including distinct keys in the default property list if a key of the same name has not already been found from the main properties list.
      * public interface Enumeration<E>Method Summaryboolean hasMoreElements() : Tests if this enumeration contains more elements.E nextElement() : Returns the next element of this enumeration if this enumeration object has at least one more element to provide.
* 泛型
  + 泛型，即“参数化类型”。一提到参数，最熟悉的就是定义方法时有形参，然后调用此方法时传递实参。那么参数化类型怎么理解呢？顾名思义，就是将类型由原来的具体的类型参数化，类似于方法中的变量参数，此时类型也定义成参数形式（可以称之为类型形参），然后在使用/调用时传入具体的类型（类型实参）。泛型的本质是为了参数化类型（在不创建新的类型的情况下，通过泛型指定的不同类型来控制形参具体限制的类型）。也就是说在泛型使用过程中，操作的数据类型被指定为一个参数，这种参数类型可以用在类、接口和方法中，分别被称为泛型类、泛型接口、泛型方法。

JDK中提供了一系列特殊的类，这些类可以存储任意类型的对象，并且长度可变，在Java中这些类被统称为集合。集合类都位于java.util包中，在使用时一定要注意导包的问题，否则会出现异常。

