JAVA 集合类汇总

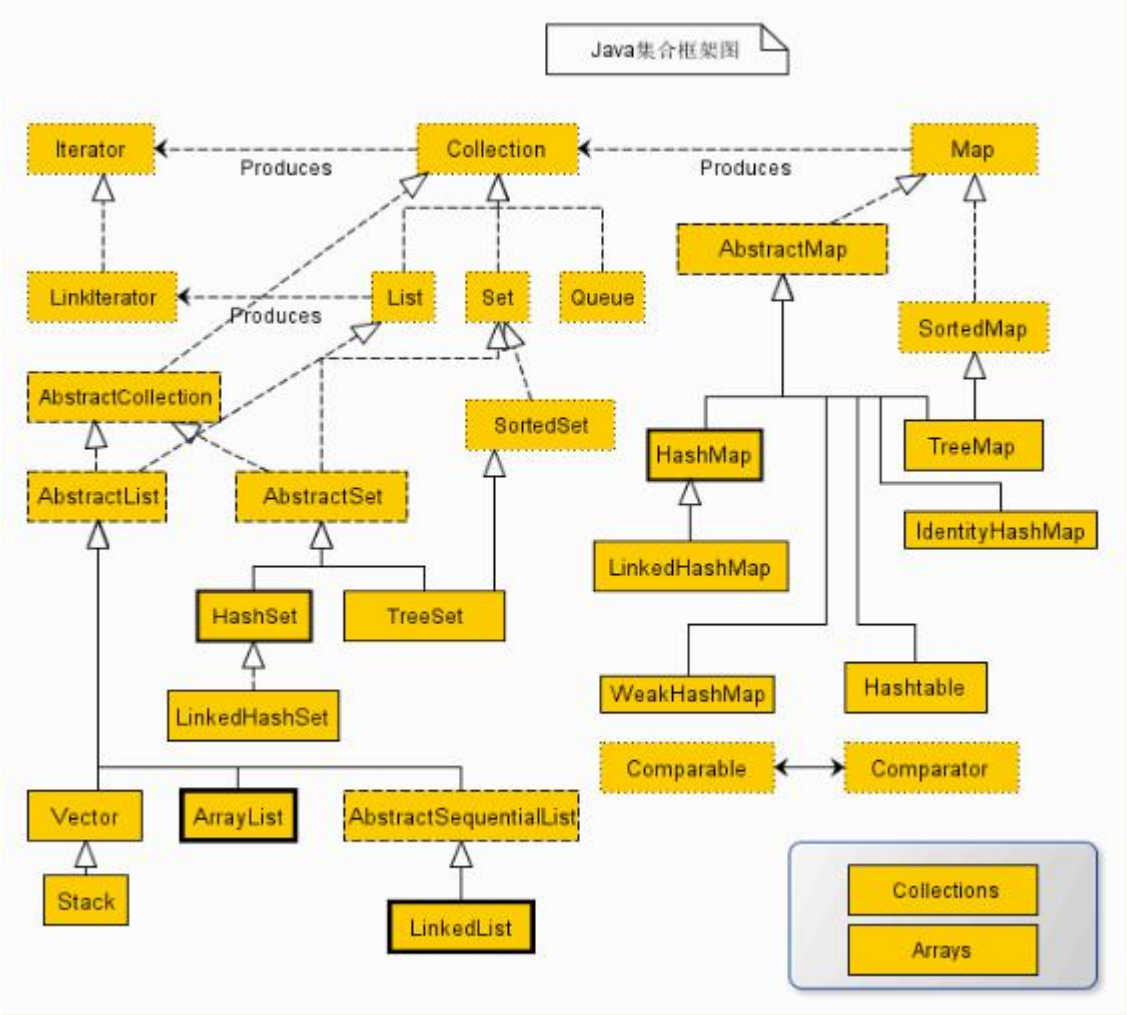
一、集合与数组

数组（可以存储基本数据类型）是用来存现对象的一种容器，但是数组的长度固定，不适合在对象数量未知的情况下使用。

集合（只能存储对象，对象类型可以不一样）的长度可变，可在多数情况下使用。

二、层次关系

如图所示：图中，实线边框的是实现类，折线边框的是抽象类，而点线边框的是接口



Collection 接口是集合类的根接口，Java 中没有提供这个接口的直接的实现类。但是却让其被继承产生了两个接口，就是 Set 和 List。Set 中不能包含重复的元素。List 是一个有序的集合，可以包含重复的元素，提供了按索引访问的方式。

Map 是 Java.util 包中的另一个接口，它和 Collection 接口没有关系，是相互独立的，但是都属于集合类的一部分。Map 包含了 key-value 对。Map 不能包含重复的 key，但是可以包含相同的 value。

Iterator，所有的集合类，都实现了 Iterator 接口，这是一个用于遍历集合中元素的接口，主要包含以下三种方法：

1.hasNext()是否还有下一个元素。

2.next()返回下一个元素。

3.remove()删除当前元素。

三、几种重要的接口和类简介

1、List（有序、可重复）

List 里存放的对象是有序的，同时也是可以重复的，List 关注的是索引，拥有一系列和索引相关的方法，查询速度快。因为往 list 集合里插入或删除数据时，会伴随着后面数据的移动，所有插入删除数据速度慢。

2、Set（无序、不能重复）

Set 里存放的对象是无序，不能重复的，集合中的对象不按特定的方式排序，只是简单地把对象加入集合中。

3、Map（键值对、键唯一、值不唯一）

Map 集合中存储的是键值对，键不能重复，值可以重复。根据键得到值，对 map 集合遍历时先得到键的 set 集合，对 set 集合进行遍历，得到相应的值。

对比如下：

		是否有序	是否允许元素重复
Collection			
List		是	是
Set	AbstractSet	否	否
	HashSet		
	TreeSet	是（用二叉排序树）	
Map	AbstractMap	否	使用 key-value 来映射和存储数据，key 必须唯一，value 可以重复
	HashMap		
	TreeMap	是（用二叉排序树）	

四、遍历

在类集中提供了以下四种的常见输出方式：

1）Iterator：迭代输出，是使用最多的输出方式。

2）ListIterator：是 Iterator 的子接口，专门用于输出 List 中的内容。

3）foreach 输出：JDK1.5 之后提供的新功能，可以输出数组或集合。

4）for 循环

代码示例如下：

for 的形式：for（int i=0;i<arr.size();i++）{...}

foreach 的形式： for（int i：arr）{...}

iterator 的形式：

```
Iterator it = arr.iterator();
while(it.hasNext()){ object o =it.next(); ...}
```

五、ArrayList 和 LinkedList

ArrayList 和 LinkedList 在用法上没有区别，但是在功能上还是有区别的。LinkedList 经常用在增删操作较多而查询操作很少的情况下，ArrayList 则相反。

六、Map 集合

实现类：HashMap、Hashtable、LinkedHashMap 和 TreeMap

HashMap

HashMap 是最常用的 Map，它根据键的 HashCode 值存储数据，根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。因为键对象不可以重复，所以 HashMap 最多只允许一条记录的键为 Null，允许多条记录的值为 Null，是非同步的

Hashtable

Hashtable 与 HashMap 类似，是 HashMap 的线程安全版，它支持线程的同步，即任一时刻只有一个线程能写 Hashtable，因此也导致了 Hashtale 在写入时会比较慢，它继承自 Dictionary 类，不同的是它不允许记录的键或者值为 null，同时效率较低。

ConcurrentHashMap

线程安全，并且锁分离。ConcurrentHashMap 内部使用段(Segment)来表示这些不同的部分，每个段其实就是一个小的 hash table，它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并发进行。

LinkedHashMap

LinkedHashMap 保存了记录的插入顺序，在用 Iteraor 遍历 LinkedHashMap 时，先得到的记录肯定是先插入的，在遍历的时候会比 HashMap 慢，有 HashMap 的全部特性。

TreeMap

TreeMap 实现 SortMap 接口，能够把它保存的记录根据键排序，默认是按键值的升序排序（自然顺序），也可以指定排序的比较器，当用 Iterator 遍历 TreeMap 时，得到的记录是排过序的。不允许 key 值为空，非同步的；

map 的遍历

第一种：KeySet()

将 Map 中所有的键存入到 set 集合中。因为 set 具备迭代器。所有可以迭代方式取出所有的键，再根据 get 方法。获取每一个键对应的值。keySet():迭代后只能通过 get()取 key 。取到的结果会乱序，是因为取得数据行主键的时候，使用了 HashMap.keySet()方法，而这个方法返回的 Set 结果，里面的数据是乱序排放的。

典型用法如下：

```
Map map = new HashMap();
map.put("key1","lisi1");
map.put("key2","lisi2");
map.put("key3","lisi3");
map.put("key4","lisi4");
//先获取 map 集合的所有键的 set 集合，keyset（）
Iterator it = map.keySet().iterator();
 //获取迭代器
while(it.hasNext()){
Object key = it.next();
System.out.println(map.get(key));
}
```

第二种：entrySet（）

Set<Map.Entry<K,V>> entrySet() //返回此映射中包含的映射关系的 Set 视图。（一个关系就是一个键-值对），就是把(key-value)作为一个整体一对一对地存放到 Set 集合当中的。Map.Entry 表示映射关系。entrySet()：迭代后可以 e.getKey()，e.getValue()两种方法来取 key 和 value。返回的是 Entry 接口。

典型用法如下：

```
Map map = new HashMap();
map.put("key1","lisi1");
map.put("key2","lisi2");
map.put("key3","lisi3");
map.put("key4","lisi4");
//将 map 集合中的映射关系取出，存入到 set 集合
Iterator it = map.entrySet().iterator();
while(it.hasNext()){
Entry e =(Entry) it.next();
System.out.println("键"+e.getKey () + "的值为" + e.getValue());
}
```

推荐使用第二种方式，即 entrySet()方法，效率较高。

对于 keySet 其实是遍历了 2 次，一次是转为 iterator，一次就是从 HashMap 中取出 key

所对于的 value。而 entryset 只是遍历了第一次，它把 key 和 value 都放到了 entry 中，所以快了。两种遍历的遍历时间相差还是很明显的。

七、主要实现类区别小结

Vector 和 ArrayList

1，vector 是线程同步的，所以它也是线程安全的，而 arraylist 是线程异步的，是不安全的。如果不考虑到线程的安全因素，一般用 arraylist 效率比较高。

2，如果集合中的元素的数目大于目前集合数组的长度时，vector 增长率为目前数组长度的 100%，而 arraylist 增长率为目前数组长度的 50%。如果在集合中使用数据量比较大的数据，用 vector 有一定的优势。

3，如果查找一个指定位置的数据，vector 和 arraylist 使用的时间是相同的，如果频繁的访问数据，这个时候使用 vector 和 arraylist 都可以。而如果移动一个指定位置会导致后面的元素都发生移动，这个时候就应该考虑到使用 linklist,因为它移动一个指定位置的数据时其它元素不移动。

ArrayList 和 Vector 是采用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，都允许直接序号索引元素，但是插入数据要涉及到数组元素移动等内存操作，所以索引数据快，插入数据慢，Vector 由于使用了 synchronized 方法（线程安全）所以性能上比 ArrayList 要差，LinkedList 使用双向链表实现存储，按序号索引数据需要进行向前或向后遍历，但是插入数据时只需要记录本项的前后项即可，所以插入数度较快。

arraylist 和 linkedlist

1.ArrayList 是实现了基于动态数组的数据结构，LinkedList 基于链表的数据结构。

2.对于随机访问 get 和 set，ArrayList 觉得优于 LinkedList，因为 LinkedList 要移动指针。

3.对于新增和删除操作 add 和 remove，LinedList 比较占优势，因为 ArrayList 要移动数据。 这一点要看实际情况的。若只对单条数据插入或删除，ArrayList 的速度反而优于 LinkedList。但若是批量随机的插入删除数据，LinkedList 的速度大大优于 ArrayList. 因为 ArrayList 每插入一条数据，要移动插入点及之后的所有数据。

HashMap 与 TreeMap

1、 HashMap 通过 hashcode 对其内容进行快速查找，而 TreeMap 中所有的元素都保持着某种固定的顺序，如果你需要得到一个有序的结果你就应该使用 TreeMap（HashMap 中元素的排列顺序是不固定的）。

2、在 Map 中插入、删除和定位元素，HashMap 是最好的选择。但如果您要按自然顺序或自定义顺序遍历键，那么 TreeMap 会更好。使用 HashMap 要求添加的键类明确定义了 hashCode()和 equals()的实现。

两个 map 中的元素一样，但顺序不一样，导致 hashCode()不一样。

同样做测试：

在 HashMap 中，同样的值的 map,顺序不同，equals 时，false;

而在 treeMap 中，同样的值的 map,顺序不同,equals 时，true，说明，treeMap 在 equals()时是整理了顺序了的。

HashTable 与 HashMap

1、同步性:Hashtable 是线程安全的，也就是说是同步的，而 HashMap 是线程序不安全的，不是同步的。

2、HashMap 允许存在一个为 null 的 key，多个为 null 的 value 。

3、hashtable 的 key 和 value 都不允许为 null。

Java Dictionary 类

Dictionary 类是一个抽象类，用来存储键/值对，作用和 Map 类相似。

给出键和值，你就可以将值存储在 Dictionary 对象中。一旦该值被存储，就可以通过它的键来获取它。所以和 Map 一样， Dictionary 也可以作为一个键/值对列表。

Dictionary 定义的抽象方法如下表所示：

序号	方法描述
1	Enumeration elements( ) 返回此 dictionary 中值的枚举。
2	Object get(Object key) 返回此 dictionary 中该键所映射到的值。
3	boolean isEmpty( ) 测试此 dictionary 是否不存在从键到值的映射。
4	Enumeration keys( ) 返回此 dictionary 中的键的枚举。
5	Object put(Object key, Object value) 将指定 key 映射到此 dictionary 中指定 value。
6	Object remove(Object key) 从此 dictionary 中移除 key （及其相应的 value）。
7	int size( ) 返回此 dictionary 中条目（不同键）的数量。

Dictionary 类已经过时了。在实际开发中，你可以实现 Map 接口来获取键/值的存储功能。

Java Map 接口

Map 接口中键和值一一映射. 可以通过键来获取值。

给定一个键和一个值，你可以将该值存储在一个 Map 对象. 之后，你可以通过键来访问对应的值。

当访问的值不存在的时候，方法就会抛出一个 NoSuchElementException 异常.

当对象的类型和 Map 里元素类型不兼容的时候，就会抛出一个 ClassCastException 异常。

当在不允许使用 Null 对象的 Map 中使用 Null 对象，会抛出一个 NullPointerException 异常。

当尝试修改一个只读的 Map 时，会抛出一个 UnsupportedOperationException 异常。

序号	方法描述
1	void clear( ) 从此映射中移除所有映射关系（可选操作）。
2	boolean containsKey(Object k) 如果此映射包含指定键的映射关系，则返回 true。
3	boolean containsValue(Object v)

	如果此映射将一个或多个键映射到指定值，则返回 true。
4	Set entrySet( ) 返回此映射中包含的映射关系的 Set 视图。
5	boolean equals(Object obj) 比较指定的对象与此映射是否相等。
6	Object get(Object k) 返回指定键所映射的值；如果此映射不包含该键的映射关系，则返回 null。
7	int hashCode( ) 返回此映射的哈希码值。
8	boolean isEmpty( ) 如果此映射未包含键-值映射关系，则返回 true。
9	Set keySet( ) 返回此映射中包含的键的 Set 视图。
10	Object put(Object k, Object v) 将指定的值与此映射中的指定键关联（可选操作）。
11	void putAll(Map m) 从指定映射中将所有映射关系复制到此映射中（可选操作）。
12	Object remove(Object k) 如果存在一个键的映射关系，则将其从此映射中移除（可选操作）。
13	int size( ) 返回此映射中的键-值映射关系数。
14	Collection values( ) 返回此映射中包含的值的 Collection 视图。

实例

下面的例子来解释 Map 的功能

```
import java.util.*;
public class CollectionsDemo {

   public static void main(String[] args) {
      Map m1 = new HashMap();
      m1.put("Zara", "8");
      m1.put("Mahnaz", "31");
      m1.put("Ayan", "12");
      m1.put("Daisy", "14");
      System.out.println();
      System.out.println(" Map Elements");
      System.out.print("\t" + m1);
   }}
```

以上实例编译运行结果如下：

```
Map Elements
        {Mahnaz=31, Ayan=12, Daisy=14, Zara=8}
```

Java Hashtable 类
Hashtable 是原始的 java.util 的一部分， 是一个 Dictionary 具体的实现 。
然而，Java 2 重构的 Hashtable 实现了 Map 接口，因此，Hashtable 现在集成到了集合框架中。它和 HashMap 类很相似，但是它支持同步。
像 HashMap 一样，Hashtable 在哈希表中存储键/值对。当使用一个哈希表，要指定用作键的对象，以及要链接到该键的值。
然后，该键经过哈希处理，所得到的散列码被用作存储在该表中值的索引。
Hashtable 定义了四个构造方法。第一个是默认构造方法：
Hashtable()
第二个构造函数创建指定大小的哈希表：
Hashtable(int size)
第三个构造方法创建了一个指定大小的哈希表，并且通过 fillRatio 指定填充比例。
填充比例必须介于 0.0 和 1.0 之间，它决定了哈希表在重新调整大小之前的充满程度：
Hashtable(int size,float fillRatio)
第四个构造方法创建了一个以 M 中元素为初始化元素的哈希表。
哈希表的容量被设置为 M 的两倍。
Hashtable(Map m)
Hashtable 中除了从 Map 接口中定义的方法外，还定义了以下方法：

序号	方法描述
1	void clear( ) 将此哈希表清空，使其不包含任何键。
2	Object clone( ) 创建此哈希表的浅表副本。
3	boolean contains(Object value) 测试此映射表中是否存在与指定值关联的键。
4	boolean containsKey(Object key) 测试指定对象是否为此哈希表中的键。
5	boolean containsValue(Object value) 如果此 Hashtable 将一个或多个键映射到此值，则返回 true。
6	Enumeration elements( ) 返回此哈希表中的值的枚举。
7	Object get(Object key) 返回指定键所映射到的值，如果此映射不包含此键的映射，则返回 null. 更确切地讲，如果此映射包含足 (key.equals(k)) 的从键 k 到值 v 的映射，则此方法返回 v；否则，返回 null。
8	boolean isEmpty( ) 测试此哈希表是否没有键映射到值。
9	Enumeration keys( ) 返回此哈希表中的键的枚举。
10	Object put(Object key, Object value)

	将指定 key 映射到此哈希表中的指定 value。
11	void rehash( ) 增加此哈希表的容量并在内部对其进行重组，以便更有效地容纳和访问其元素。
12	Object remove(Object key) 从哈希表中移除该键及其相应的值。
13	int size( ) 返回此哈希表中的键的数量。
14	String toString( ) 返回此 Hashtable 对象的字符串表示形式，其形式为 ASCII 字符 ", " （逗号加空格）分隔开的、括 中的一组条目。

实例

下面的程序说明这个数据结构支持的几个方法：

```
import java.util.*;
public class HashTableDemo {

   public static void main(String args[]) {
      // Create a hash map
      Hashtable balance = new Hashtable();
      Enumeration names;
      String str;
      double bal;

      balance.put("Zara", new Double(3434.34));
      balance.put("Mahnaz", new Double(123.22));
      balance.put("Ayan", new Double(1378.00));
      balance.put("Daisy", new Double(99.22));
      balance.put("Qadir", new Double(-19.08));

      // Show all balances in hash table.
      names = balance.keys();
      while(names.hasMoreElements()) {
         str = (String) names.nextElement();
         System.out.println(str + ": " +
         balance.get(str));
      }
      System.out.println();
      // Deposit 1,000 into Zara's account
      bal = ((Double)balance.get("Zara")).doubleValue();
      balance.put("Zara", new Double(bal+1000));
      System.out.println("Zara's new balance: " +
      balance.get("Zara"));
   }}
```

以上实例编译运行结果如下：

```
Qadir: -19.08Zara: 3434.34Mahnaz: 123.22Daisy: 99.22Ayan: 1378.0
Zara's new balance: 4434.34
```