Java 基础：集合总结（7）

各类集合对比总结

集(Set)：集里的对象不按任何特定的方式排列，按索引值来操作数据，不能有重复的元素；
列表(List)：序列中的对象以线性方式存储，按索引值来操作数据，可以有重复的元素；
映射(Map)：映射的每一项为“名称—数值”对，名称不可以重复，值可以重复，一个名称对应一个唯一的值。

迭代器 Iterator

迭代器是按次序一个一个地获取集合中所有的对象，是访问集合中每个元素的标准机制。
迭代器的创建：Collection 接口的 iterator()方法返回一个 Iterator

```
Iterator it=test.iterator(); //将 test 集合对象转为迭代器
```

迭代器的常用方法：

```
hasNext() //判断迭代器中是否有下一个元素
next() //返回迭代的下一个元素
Remove() //将迭代器新返回的元素删除
```

```
public interface Iterator {
    boolean hasNext();
    Object next();
    void remove(); // Optional
}
```

在调用 remove()方法的时候，必须调用一次 next()方法.
remove()方法实际上是删除上一个返回的元素.

List 常用方法

```
void add(int index, Object element) ：添加对象 element 到位置 index 上
boolean addAll(int index, Collection collection) ：在 index 位置后添加容器 collection 中所有的元素
Object get(int index) ：取出下标为 index 的位置的元素
int indexOf(Object element) ：查找对象 element 在 List 中第一次出现的位置
int lastIndexOf(Object element) ：查找对象 element 在 List 中最后出现的位置
Object remove(int index) ：删除 index 位置上的元素
```

```
ListIterator listIterator(int startIndex) ：返回一个 ListIterator 跌代器，开始位置为 startIndex

List subList(int fromIndex, int toIndex) ：返回一个子列表 List ，元素存放为从 fromIndex 到 toIndex 之前的一个元素
```

ArrayList

可以将其看作是能够自动增长容量的数组。
利用 ArrayList 的 toArray()返回一个数组。
Arrays.asList()返回一个列表。
迭代器(Iterator) 给我们提供了一种通用的方式来访问集合中的元素。
ArrayList 可以自动扩展容量
ArrayList.ensureCapacity(int minCapacity)
首先得到当前 elementData 属性的长度 oldCapacity。
然后通过判断 oldCapacity 和 minCapacity 参数谁大来决定是否需要扩容，如果 minCapacity 大于 oldCapacity，那么我们就对当前的 List 对象进行扩容。
扩容的的策略为：取(oldCapacity * 3)/2 + 1 和 minCapacity 之间更大的那个。然后使用数组拷 贝的方法，把以前存放的数据转移到新的数组对象中如果 minCapacity 不大于 oldCapacity 那么就不进行扩容。

LinkedList

LinkedList 是采用双向循环链表实现的。
利用 LinkedList 可以实现栈(stack)、队列(queue)、双向队列(double-ended queue)。
它具有方法 addFirst()、addLast()、getFirst()、getLast()、removeFirst()、removeLast()等。

ArrayList 和 LinkedList 的比较

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。尽量避免同时遍历和删除集合。因为这会改变集合的大小；

```
for( Iterator<ComType> iter = ComList.iterator(); iter.hasNext();){
    ComType com = iter.next();
    if ( !com.getName().contains("abc")){
        ComList.remove(com);}
}
```

推荐：

```
for( Iterator<ComType> iter = ComList.iterator(); iter.hasNext();){
```

```
ComType com = iter.next();
if ( !com.getName().contains("abc")){
iter.remove(com);
 }
}
```

无限制的在 list 中 add element，势必会造成 list 占用内存过高.

Map 常用方法

```
Object put(Object key,Object value)：用来存放一个键-值对 Map 中
Object remove(Object key)：根据 key(键)，移除键-值对，并将值返回
void putAll(Map mapping) ：将另外一个 Map 中的元素存入当前的 Map 中
void clear() ：清空当前 Map 中的元素
Object get(Object key) ：根据 key(键)取得对应的值
boolean containsKey(Object key) ：判断 Map 中是否存在某键（key）
boolean containsValue(Object value):判断 Map 中是否存在某值(value)
public Set keySet() ：返回所有的键（key），并使用 Set 容器存放
public Collection values() ：返回所有的值（Value），并使用 Collection 存放
public Set entrySet() ：返回一个实现 Map.Entry 接口的元素 Set
```

HashMap

Map 主要用于存储键(key)值(value)对，根据键得到值，因此键不允许重复,但允许值重复。

HashMap 是一个最常用的 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。

HashMap 最多只允许一条记录的键为 Null;允许多条记录的值为 Null;

HashMap 不支持线程的同步，即任一时刻可以有多个线程同时写 HashMap;可能会导致数据的不一致。如果需要同步，可以用 Collections 的 synchronizedMap 方法使 HashMap 具有同步的能力，或者使用 ConcurrentHashMap。

使用 HashMap ，当一个对象被当作键值需要对 equals()和 hashCode()同时覆写。

LinkedHashMap 和 HashMap，TreeMap 对比

Hashtable 与 HashMap 类似,它继承自 Dictionary 类，不同的是:它不允许记录的键或者值为空;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 Hashtable 在写入时会比较慢。

Hashmap 是一个最常用的 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。**

LinkedHashMap 保存了记录的插入顺序，**在用 Iterator 遍历 LinkedHashMap 时，先得到的记录肯定是先插入的.也可以在构造时用带参数，按照应用次数排序。在遍历的时候会比 HashMap 慢，不过有种情况例外，当 HashMap 容量很大，实际数据较少时，遍历起来可能会比 LinkedHashMap 慢，因为 LinkedHashMap 的遍历速度只和实际数据有关，和容量无关，而 HashMap 的遍历速度和他的容量有关。**

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 ,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。

我们用的最多的是 HashMap,HashMap 里面存入的键值对在取出的时候是随机的,在 Map 中插入、删除和定位元素，HashMap 是最好的选择。

TreeMap 取出来的是排序后的键值对。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。

LinkedHashMap 是 HashMap 的一个子类 ,如果需要输出的顺序和输入的相同,那么用 LinkedHashMap 可以实现,它还可以按读取顺序来排列，像连接池中可以应用。

Set 的使用

- 不允许重复元素
- 对 add()、equals() 和 hashCode() 方法添加了限制
- HashSet 和 TreeSet 是 Set 的实现
- Set—》HashSet LinkedHashSet
- SortedSet —》 TreeSet

HashSet

```
public boolean contains(Object o) ：如果 set 包含指定元素，返回 true

public Iterator iterator()返回 set 中元素的迭代器

public Object[] toArray() ：返回包含 set 中所有元素的数组 public Object[]
toArray(Object[] a) ：返回包含 set 中所有元素的数组，返回数组的运行时类型是指定数组的运
行时类型

public boolean add(Object o) ：如果 set 中不存在指定元素，则向 set 加入

public boolean remove(Object o) ：如果 set 中存在指定元素，则从 set 中删除

public boolean removeAll(Collection c) ：如果 set 包含指定集合，则从 set 中删除指定集合
的所有元素

public boolean containsAll(Collection c) ：如果 set 包含指定集合的所有元素，返回
true。如果指定集合也是一个 set，只有是当前 set 的子集时，方法返回 true
```

实现 Set 接口的 HashSet，依靠 HashMap 来实现的。

我们应该为要存放到散列表的各个对象定义 hashCode()和 equals()。

HashSet 如何过滤重复元素

调用元素 HashCode 获得哈希码–》判断哈希码是否相等，不相等则录入—》相等则判断 equals()后是否相等，不相等在进行 hashcode 录入，相等不录入

TreeSet

- TreeSet 是依靠 TreeMap 来实现的。
- TreeSet 是一个有序集合，TreeSet 中元素将按照升序排列，缺省是按照自然顺序进行排列 ,意味着 TreeSet 中元素要实现 Comparable 接口 ,我们可以在构造 TreeSet 对象时，传递实现了 Comparator 接口的比较器对象。

HashSet 与 TreeSet 与 LinkedHashSet 对比

HashSet 不能保证元素的排列顺序，顺序有可能发生变化，不是同步的，集合元素可以是 null,但只能放入一个 null；

TreeSet 是 SortedSet 接口的唯一实现类，TreeSet 可以确保集合元素处于排序状态。TreeSet 支持两种排序方式，自然排序 和定制排序，其中自然排序为默认的排序方式。向 TreeSet 中加入的应该是同一个类的对象。

TreeSet 判断两个对象不相等的方式是两个对象通过 equals 方法返回 false，或者通过 CompareTo 方法比较没有返回 0。

自然排序

自然排序使用要排序元素的 CompareTo(Object obj)方法来比较元素之间大小关系，然后将元素按照升序排列。

定制排序

自然排序是根据集合元素的大小 ,以升序排列 ,如果要定制排序 ,应该使用 Comparator 接口，实现 int compare(To1,To2)方法。

LinkedHashSet 集合同样是根据元素的 hashCode 值来决定元素的存储位置 ,但是它同时使用链表维护元素的次序。这样使得元素看起 来像是以插入顺 序保存的，也就是说，当遍历该集合时候，LinkedHashSet 将会以元素的添加顺序访问集合的元素。

LinkedHashSet 在迭代访问 Set 中的全部元素时，性能比 HashSet 好，但是插入时性能稍微逊色于 HashSet。