#集合 Collection和Map两大类； Collection又分为：Set/Queue/List三类

##Collection **Collection接口定义的方法**，所有集合可以使用的方法：

> int size();长度 isEmpty();判空 boolean contains(Object o);是否包含元素 boolean contains(Collection o);是否包含集合中所有元素 Iterator iterator();获取迭代器 boolean add(E e);添加元素 boolean addAll(Collection<? extends E> boolean remove(Object o);移出元素 boolean removeAll(Collection<?> c); void clear();清除所有元素 Object[] toArray();转变成数组 重写了toString()方法，输出所有元素，格式为：**[.....]**

遍历集合的方法

1. 使用Lambda表达式遍历

> forEach(Consumer<? super T> action)是一个实例方法（**Iterable接口的方法，不是迭代器！**），action是函数式接口，有一个accept(T t)方法，程序会把元素传给这个方法。

2. 使用Iterator迭代器

> 获取迭代器后，可以使用迭代器的方法： boolean hasNext();判断是否有下一个 E next();返回下一个元素 void remove();删除一个元素 把集合元素的值传给迭代变量Iterator.next()；在使用迭代器的过程中，不能使用集合Collection的remove()方法，因为迭代器采用快速失败(fail-fast)机制，如果检测到修改，就会爆异常ConcurrentModificationException。

3. 使用Lambda表达式遍历Iterator迭代器

> 迭代器有一个void forEachRemaining(Consumer<? super E> action)方法，与1方法类似。

4. 使用增强for循环来遍历集合元素

> 与迭代器类似，不能使用collection.remove()删除集合元素。

使用Predicate函数式接口，批量删除符合条件的元素 boolean removeIf(Predicate<? super E> filter)

> **Predicate有一个boolean test(T t);方法**，可以通过这个方法定义查看元素是否符合自定义的标准。
> 如：

```
public static int CallAll(Collection c,  Predicate p){
        int number = 0;
        for (Object j : c){
            if (p.test(j)){
                number++;
            }
        }
        return number;
    }
```

然后可以通过调用该方法，传入不同的lambda表达式查看符合条件的元素有多少个。

##Set集合

> 不允许包含相同元素，且无序；

HashSet

> 存取与查找功能 无序；不是线程安全；**元素值可以为NULL**； 判断元素相等的标准是**equals()相等且hashCode()也相同**。 **尽量不要修改参与equals()与hashCode()两个方法计算的实例变量，不然会导致HashSet无法正确操作这些元素。**

LinkedHsahSet类

> 同样使用hashcode来决定存储位置，同时使用链表维护元素次序，**当遍历集合时，按元素添加顺序访问**。

TreeSet类

> **使元素处于排序状态（不是添加顺序）**。 采用**红黑树的数据结构**来进行存储数据。支持两种排序方法：**自然排序和定制排序**。 元素值可以为空吗？

自然排序：

> 根据实际值大小进行**升序**排序； **集合会调用元素的compareTo(Object obj)方法来比较大小**； **所以添加的元素必须实现Comparable接口，不然会抛出异常**，该接口定义了compareTo(Object obj)方法；该方法比较时，**返回0则相等**；返回正数为大于；返回负数为小于。 TreeSet判断两个对象是否相等的**唯一**标准为：compareTo(Object obj)方法返回0； 定制排序 自己定义一个排序规则，使用Comparator接口，此接口包含一个compare(T o1,T o2)的方法，**返回0则相等**；返回正数为大于；返回负数为小于。 在new的时候（使用构造器），参数传入lambda表达式。 好处：元素不需要实现Comparable接口；限制：仍不可以添加不同类型的对象。

EnumSet类

> 专门为枚举类设计的，所有元素必须是**指定枚举类类型的枚举值**，按枚举值定义顺序排序。 **元素值不能为空**

性能分析：

> **hashSet性能总要比TreeSet要好，所以只有当需要进行排序时，才使用TreeSet，其余都用hashSet**； LinkedHsahSet类由于有链表，对于**遍历**会比hashSet要快。 EnumSet性能最好。 hashSet、TreeSet、EnumSet**线程不安全**；需要使用Collections工具类；

##List集合

> 代表元素有序、可重复的集合；通过索引来访问指定位置的元素；默认使用添加顺序设置索引； 其通用方法：

1. void add(int index, E element);把某个元素插入到index处，**注意不是覆盖了index处原有的数据，而是插入！！**
2. boolean addAll(int index, Collection<? extends E> c);两个都是添加元素进List中，**增加了索引式添加**；
3. get(int index);
4. indexOf(Object o);返回**第一次**出现的索引；
5. int lastIndexOf(Object o);返回**最后一次**出现的索引；
6. remove(int index);
7. set(int index, E element);把index处的元素替换；

8. List subList(int fromIndex, int toIndex);获取片段list；

> 主要是增加了索引来进行各种操作 添加了两个特殊的方法： 用来对元素进行重新排序

1. default void replaceAll(UnaryOperator operator) ;
2. default void sort(Comparator<? super E> c);

list判断两个对象相等只需要通过equals()方法返回true即可；

> list提供了ListIterator listIterator();方法，返回一个list迭代器，可以进行向前迭代和添加元素（原先只有**remove**）。

ArrayList和Vector类

> 都是基于数组实现的List；默认长度为**10**； public void trimToSize()方法，可以直接调整**List**集合长度为当前元素个数，减少占用的内存空间； Vector类线程安全但缺点多不推荐使用；ArrayList是线程不安全的。

LinkedList类

> 根据索引随机访问集合元素； 实现了Deque接口，所以能当栈和队列使用； 以链表形式实现；

性能分析

1. 所有基于数组实现的集合，在随机访问（即获取某个集合元素的值时）比使用**Iterator**迭代访问性能好。
2. ArrayList总体来说性能要比LinkedList强；LinkedList插入和删除操作性能好，ArrayList访问性能好；
3. 对于**ArrayList**随机访问（**get**方法）遍历集合元素好，**LinkedList**采用迭代器来遍历集合元素好；
4. 对于需要线程安全的需求，使用工具类**Collections**包装比较好；

Stack类

> Vector类的子类，模拟栈的结构。peek()出栈、pop()出栈并返回元素、push(object i)入栈； 尽量少用，可以考虑使用ArrayDeque来实现栈。

Arrays.asList(T... a)返回一个Arrays.ArrayList类，这个类是固定长度的，不可增加删除元素；

Queue集合

> 队列数据结构；先进先出； 方法：

1. boolean add(E e);添加到队尾
2. E element();获取队头元素，但不删除；
3. E peek();获取队头元素；如果为空队列则返回**null**。
4. E poll();获取队头元素；并删除元素,如果为空队列则返回**Null**；
5. E remove();获取队头元素；并删除元素；
6. boolean offer(E e);将指定元素添加至队尾，在有容量限制的队列中会比add方法性能好（？）；

PriorityQueue类

> 与TreeSet类似，元素顺序默认为从小到大，不再是先进先出原则；元素都要实现**Comparable**接口；不允许插入NULL元素；

Deque接口和ArrayDeque类

> Deque接口是Queue的子接口，定义了双端队列，即可以从队列两端添加获取元素。 所以可以当做栈来使用。 ArrayDeque类是基于数组实现的，底层长度为16； ArrayDeque既可以当做队列也可以当做栈来使用，其实际操作区别为：
>
> 1. 当做栈时：使用**push(E e)**方法入栈，（**把元素放入队列队头**）；
> 2. 当做队列时：使用**offer()或者add()**把元素添加入队列中。
> 3. 栈出栈是使用**pop()**;而队列把元素出队列是用**poll()**； 相同点：**都可以使用peek()获取元素值，不把元素取出集合**；

##Map集合

> 键值对；**所有的key组成Set集合，使用keySet()方法可以返回一个key组成的Set集合**； Map和Set:Map提供了一个Entry内部类来封装键值对，而Set实际上是一个**value都为null的Map**； 遍历Map集合：
>
> 1. 使用keySet()方法获取一个Set集合；
> 2. 使用增强for循环来遍历map;

HashMap类与Hashtable类

> Hashtable线程安全；HashMap线程不安全； **HashMap可以把null值作为key或者value**；只能有一个key为null，但可以有很多value为null； HashMap类与Hashtable类元素一定要实现**hashcode()方法和equals()方法**； 判断相等标准（**分为两部分**）：1.key值相等条件：hashcode一致和equals()方法返回true；2.value值相等：equals()方法返回true；

HashMap的实现原理

> HashMap基于hashing原理，java1.8后采用**位桶数组+链表+红黑树**的实现方式；我们通过put()和get()方法储存和获取对象。当我们将键值对传递给**put()方法**时，它调用键对象的hashCode()方法来**计算hashcode**，让后找到在**位桶数组bucket位置**来储存值对象。当**获取对象get()**时，先通过计算key的Hash值，然后找到链表位置，**遍历链表用键对象的equals()方法用来找到键值对**。。HashMap使用链表来解决碰撞问题，当发生碰撞了，它会在同一个bucket位置生成链表，当链表长度大于阈值（8）时会将链表转换为红黑树。

HashMap的数据结构

> 位桶数组：Node<K,V>[]; 数组元素：Node<K,V>实现了Map.Entry<K,V>接口； 红黑树：TreeNode<K,V> **如果没有实现对比接口，那怎么实现红黑树结构？（估计要看看源码）**

LinkedHashMap

> 使用双向链表来维护键值对的次序（插入顺序）**使用forEach()遍历才会有？**； 在迭代返回元素有较好的性能；

IdentityHashMap

> 是特殊的hashmap，在**判断key相等**时比较特殊，当且仅当key1==key2时才相等。

SortedMap接口和TreeMap实现类

> TreeMap是红黑树结构；根据key对节点进行排序； 也分为自然排序和定制排序（即实现Comparable接口）； 提供了几个访问第一个、前一个、后一个、最后一个键值对的方法，并提供了几个截取子TreeMap方法；

EnumMap

> 每个key都是单个枚举类的枚举值；**根据枚举值（key）在枚举类的定义顺序来维护键值对的顺序** 以数组的形式保存元素；

Map性能分析

1. TreeMap通常比HashMap要慢；但好处在于键值对始终保持有序状态无序进行排序操作；
2. 由第一点可以由以下操作：通过keySet()获取key的Set集合，通过toArray()方法获取对应数组，在使用Arrays.binarySearch(数组,数组元素)快速查找对象；
3. hashMap就是为了快速查询设计的。

Collections工具类

1. 对List集合进行排序操作：

> void reverse(List<?> list)：把**反转**list集合元素； void sort(List list)与void sort(List list, Comparator<? super T> c)：按自然排序或者指定顺序对List集合进行排序； void rotate(List<?> list, int distance)：**当distance为正数**时，把集合后distance个元素移到前面（这几个元素顺序不变）；**当distance为负数**时，把集合前distance个元素移到后面；

2. 查找、替换集合元素操作

> int binarySearch(list, T key):使用二分法搜索指定List集合元素，**必须先保证List集合处于有序状态**； 找最大最小值：max，min； fill(list，obj)：用obj替换list集合中所有元素；

3. **同步控制**

> 提供多个synchronizedXXX(),把对应集合包装成线程安全的集合。