**1、Set、Map**

Set：无序、不可重复。

Map：多个key-value对组成的集合，本质是关联数组（key数组和value数组）。

联系：Map的所有key集中起来，可组成一个Set集合。

**2、HashMap、HashSet**

对于HashSet而言，系统采用Hash算法决定集合元素的存储位置，这样可以保证快速存、取集合元素；对于HashMap而言，系统将value当成key的附属，系统根据Hash算法来决定key的存储位置，而完全没有考虑Entry中的value。

这里得提到一个重要接口Map.Entry，每个Map.Entry其实就是一个key-value对。当程序试图将一个key-value放入HashMap中时，首先根据该key的hashCode()返回值决定Entry的存储位置：如果两个Entry的key的hashCode()返回值相同，那它们的存储位置相同；如果这两个Entry的key通过equals比较返回true，新添加的Entry的value将覆盖集合中原有Entry的value，但key不会覆盖；如果这两个Entry的key通过equals比较返回false，新添加的Entry将与集合中原有Entry形成Entry链，而且新添加的Entry位于Entry链的头部。即：key的hashCode()决定位置，位置相同则由key通过equals()比较值决定采用覆盖行为（返回true），还是生成Entry链（返回false）。

当创建HashMap时，有一个默认的负载因子（load factor），其默认值为0.75。这是时间和空间上的一种折中：增大负载因子可以减少Hash表（即Entry数组）所占用的内存空间，但会增加查询数据的时间开销，而查询是最频繁的操作（HashMap的get和put方法都要用到查询）；减小负载因子会提高查询的性能，但会增加Hash表所占用的内存空间。负载因子的作用是，当load factor（= HashMap中的数据量/HashMap总容量）达到指定值时，总容量自动扩展1倍。

**3、TreeSet、TreeMap**

与HashSet完全类似，TreeSet里绝大部分方法都是直接调用TreeMap的方法实现的。对TreeMap而言，它采用“红黑树”的排序二叉树来保存Map中的每个Entry——每个Entry被当做“红黑树”的一个节点对待，这就意味着，TreeMap通过循环查找到合适位置并添加元素、取出元素的性能都比HashMap低。但TreeMap、TreeSet相比HashMap、HashSet的优势在于：TreeMap中所有的Entry总是按key根据指定排序规则保持有序状态，TreeSet中所有元素总是根据指定排序规则保持有序状态。

类比而言，HashMap、HashSet的存储方式类似“妈妈放东西”，不同东西放在不同位置，需要时可以快速找到。TreeSet、TreeMap的存储方式类似“体育课站队”，第一个人（相当于元素）自成一队，以后每添加一个人都要找到这个人应该插入的位置（使这个位置前的所有人比新插入的人矮，这个位置后的所有人比新插入的高），然后在该位置插入即可。这样保证了该队伍总是由矮到高地排列。

**4、ArrayList**

ArrayList底层是基于数组实现，每次创建ArrayList时传入的int参数就是它所包装的数组的长度；如果未传入int参数，那么ArrayList的初始长度为10，也就是它底层所封装Object[]数组的长度为10。

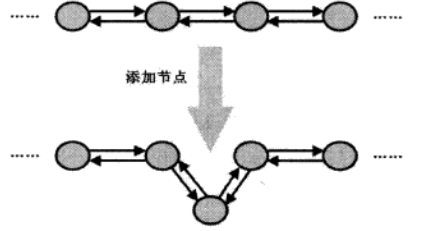
注意：就像引用类型的数组一样，当把Java对象放入数组中，并不是把真正的Java对象放入数组，而是把对象的引用放入数组中，每个数组元素都是一个引用变量。

**5、ArrayList和LinkedList的性能分析和使用场景**

ArrayList是一种顺序存储的线性表，底层采用数组来保存每个集合的元素。LinkedList则是一种链式存储的线性表，其本质是一个双向链表，它不仅实现了List接口，还实现了Deque接口。也就是说LinkedList既可以当双向链表使用，也可以当队列使用，可以当成栈使用。（Deque代表双端队列，既具有队列的特征，又具有栈的特征）

对于ArrayList而言，当程序向集合中添加、删除元素时，ArrayList底层都需要对数组进行“整体搬家”，如果添加元素导致集合长度将要超过底层数组长度，ArrayList必须创建一个长度为原来长度1.5倍的数组，再由垃圾回收机制回收原有数组，因此性能非常差。但如果程序调用get(int index)方法取出集合中的元素时，性能和数组几乎相同——非常快！

由于LinkedList采用双向链表保存集合元素，因此在添加集合元素的时候，只要对链表进行如下图所示的操作即可添加一个新的节点。因此LinkedList可以非常方便地在指定节点之前插入新节点，但若要在特定位置index处插入元素，则LinkedList必须一个一个元素地搜索，直到找到第index个元素为止。即便如此，当程序调用add(int index,Object obj)时，LinkedList方法的性能依然高于ArrayList。



就经验来讲，ArrayList的性能总体上优于LinkedList。因此绝大多数场景考虑使用ArrayList集合，但如果程序要经常添加、删除元素，尤其是需要调用add(E e)方法向集合中添加元素时，则应该考虑使用LinkedList。