Collection  
List 代表有序、重复的集合  
ArrayList 允许null、非同步  
Vector Vector非常类似ArrayList，但是Vector是同步的。使用了synchronized方法（线程安全）所以性能上比ArrayList要 差  
├LinkedList 允许null、非同步。如果多个线程同时访问一个List则必须自己实现访问同步。List list = Collections.synchronizedList(new LinkedList(...));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！  
Stack继承自Vector，实现一个后进先出的堆栈。  
Set 代表无序、不可重复的集合

├[HashSet](mk:@MSITStore:J:\\面试\\JDK_API_1_6_zh_CN.CHM::/java/util/HashSet.html" \o "java.util 中的类)  
├[LinkedHashSet](mk:@MSITStore:J:\\面试\\JDK_API_1_6_zh_CN.CHM::/java/util/LinkedHashSet.html" \o "java.util 中的类) **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**  
[TreeSet](mk:@MSITStore:J:\面试\JDK_API_1_6_zh_CN.CHM::/java/util/TreeSet.html) **TreeSet可以确保集合元素处于排序状态。**  
Map  
HashMap HashMap和Hashtable类似，不同之处在于 HashMap是非同步的，并且允许null  
Hashtable 同步、非空

├[TreeMap](mk:@MSITStore:J:\\面试\\JDK_API_1_6_zh_CN.CHM::/java/util/TreeMap.html" \o "java.util 中的类)  
WeakHashMap WeakHashMap是一种改进的HashMap，它对key实行“弱引用”，如果一个key不再被外部所引用，那么该key可以被GC回收。

Collections是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

Collection接口  
　　Collection是最基本的集合接口，一个Collection代表一组Object，即Collection的元素（Elements）。一些 Collection允许相同的元素而另一些不行。一些能排序而另一些不行。Java SDK不提供直接继承自Collection的 类，Java SDK提供的类都是继承自Collection的“子接口”如List和Set。

遍历通用Collection

Iterator it = collection.iterator(); // 获得一个迭代子  
　　　　while(it.hasNext()) {  
　　　　　　Object obj = it.next(); // 得到下一个元素  
　　　　}  
　　由Collection接口派生的两个接口是List和Set。

List代表有序、重复的集合，它的具体实现类常用的有ArrayList和LinkedList。你可以将任何东西放到一个 List容器中，并在需要时从中取出。ArrayList从其命名中可以看出它是一种类似数组的形式进行存储，因此它的随机访问速度极快，而 LinkedList的内部实现是链表，它适合于在链表中间需要频繁进行插入和删除操作。

List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在List中的位置，类似于数组下标）来访问List中的元素，这类似于Java的数组。  
和下面要提到的Set不同，List允许有相同的元素。

实现List接口的常用类有LinkedList，ArrayList，Vector和Stack。

ArrayList类  
　　ArrayList实现了可变大小的数组。它允许所有元素，包括null。ArrayList没有同步。  
size，isEmpty，get，set方法运行时间为常数。但是add方法开销为分摊的常数，添加n个元素需要O(n)的时间。其他的方法运行时间为线性。  
　　每个ArrayList实例都有一个容量（Capacity），即用于存储元素的数组的大小。这个容量可随着不断添加新元素而自动增加，但是增长算法 并没有定义。当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率。  
　　和LinkedList一样，ArrayList也是非同步的（unsynchronized）。  
  
Vector类  
　　Vector非常类似ArrayList，但是Vector是同步的。由Vector创建的Iterator，虽然和ArrayList创建的Iterator是同一接口，但是，因为Vector是同步的，当一个Iterator被创建而且正在被使用，另一个线程改变了Vector的状态（例如，添加或删除了一些元素），这时调用Iterator的方法时将抛出ConcurrentModificationException，因此必须捕获该异常。

LinkedList类  
　　LinkedList实现了List接口，允许null元素。此外LinkedList提供额外的get，remove，insert方法在 LinkedList的首部或尾部。这些操作使LinkedList可被用作堆栈（stack），队列（queue）或双向队列（deque）。  
　　注意LinkedList没有同步方法。如果多个线程同时访问一个List，则必须自己实现访问同步。一种解决方法是在创建List时构造一个同步的List：  
　　　　List list = Collections.synchronizedList(new LinkedList(...));

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，但是插入数据要设计到数组元素移动 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！  
  
Stack 类  
　　Stack继承自Vector，实现一个后进先出的堆栈。Stack提供5个额外的方法使得Vector得以被当作堆栈使用。基本的push和pop方法，还有peek方法得到栈顶的元素，empty方法测试堆栈是否为空，search方法检测一个元素在堆栈中的位置。Stack刚创建后是空栈。

Set代表无序、不可重复的集合，而与List不同的是，在Set中的对象元素不能重复，也就是说你不能把同样的东西两次放入同一个Set容器中。它的常用具体实现有HashSet和TreeSet类。HashSet能快速定位一个元素，但是你放到HashSet中的对象需要实现 hashCode()方法，它使用了前面说过的哈希码的算法。而TreeSet则将放入其中的元素按序存放，这就要求你放入其中的对象是可排序的，这就用到了集合框架提供的另外两个实用类Comparable和Comparator。

Map没有继承Collection接口

Map是一种把键对象和值对象映射进行关联的容器，而一个值对象又可以是一个 Map，依次类推，这样就可形成一个多级映射。对于键对象来说，像Set一样，一个Map容器中的键对象不允许重复，这是为了保持查找结果的一致性; Map 有两种比较常用的实现：HashMap和TreeMap。HashMap也用到了哈希码的算法，以便快速查找一个键，TreeMap则是对键按序存放，因此它便有一些扩展的方法，比如firstKey(),lastKey()等，你还可以从TreeMap中指定一个范围以取得其子Map。

Hashtable类  
　　Hashtable继承Map接口，实现一个key-value映射的哈希表。任何非空（non-null）的对象都可作为key或者value。  
　　添加数据使用put(key, value)，取出数据使用get(key)，这两个基本操作的时间开销为常数。  
Hashtable通过initial capacity和load factor两个参数调整性能。通常缺省的load factor 0.75较好地实现了时间和空间的均衡。增大load factor可以节省空间但相应的查找时间将增大，这会影响像get和put这样的操作。  
使用Hashtable的简单示例如下，将1，2，3放到Hashtable中，他们的key分别是”one”，”two”，”three”：  
　　　　Hashtable numbers = new Hashtable();  
　　　　numbers.put(“one”, new Integer(1));  
　　　　numbers.put(“two”, new Integer(2));  
　　　　numbers.put(“three”, new Integer(3));  
　　要取出一个数，比如2，用相应的key：  
　　　　Integer n = (Integer)numbers.get(“two”);  
　　　　System.out.println(“two = ” + n);  
　　由于作为key的对象将通过计算其散列函数来确定与之对应的value的位置，因此任何作为key的对象都必须实现hashCode和equals方 法。hashCode和equals方法继承自根类Object，如果你用自定义的类当作key的话，要相当小心，按照散列函数的定义，如果两个对象相 同，即obj1.equals(obj2)=true，则它们的hashCode必须相同，但如果两个对象不同，则它们的hashCode不一定不同，如 果两个不同对象的hashCode相同，这种现象称为冲突，冲突会导致操作哈希表的时间开销增大，所以尽量定义好的hashCode()方法，能加快哈希 表的操作。  
　　如果相同的对象有不同的hashCode，对哈希表的操作会出现意想不到的结果（期待的get方法返回null），要避免这种问题，只需要牢记一条：要同时复写equals方法和hashCode方法，而不要只写其中一个。  
　　Hashtable是同步的。  
  
HashMap类  
　　HashMap和Hashtable类似，不同之处在于 HashMap是非同步的，并且允许null，即null value和null key。，但是将HashMap视为Collection时 （values()方法可返回Collection），其迭代子操作时间开销和HashMap的容量成比例。因此，如果迭代操作的性能相当重要的话，不要 将HashMap的初始化容量设得过高，或者load factor过低。  
  
WeakHashMap类  
　　WeakHashMap是一种改进的HashMap，它对key实行“弱引用”，如果一个key不再被外部所引用，那么该key可以被GC回收。

线性表，链表，哈希表是常用的数据结构，在进行Java开发时，JDK已经为我们提供了一系列相应的类来实现基本的数据结构。这些类均在java.util包中。本文试图通过简单的描述，向读者阐述各个类的作用以及如何正确使用这些类。   
如果涉及到堆栈，队列等操作，应该考虑用List，对于需要快速插入，删除元素，应该使用LinkedList，如果需要快速随机访问元素，应该使用ArrayList。  
　　如果程序在单线程环境中，或者访问仅仅在一个线程中进行，考虑非同步的类，其效率较高，如果多个线程可能同时操作一个类，应该使用同步的类。  
　　要特别注意对哈希表的操作，作为key的对象要正确复写equals和hashCode方法。  
　　尽量返回接口而非实际的类型，如返回List而非ArrayList，这样如果以后需要将ArrayList换成LinkedList时，客户端代码不用改变。这就是针对抽象编程。  
**同步性  
Vector是同步的。这个类中的一些方法保证了Vector中的对象是线程安全的。而ArrayList则是异步的，因此ArrayList中的对象并不是线程安全的。因为同步的要求会影响执行的效率，所以如果你不需要线程安全的集合那么使用ArrayList是一个很好的选择，这样可以避免由于同步带来的不必要的性能开销。  
数据增长  
从内部实现机制来讲ArrayList和Vector都是使用数组(Array)来控制集合中的对象。当你向这两种类型中增加元素的时候，如果元素的数目超出了内部数组目前的长度它们都需要扩展内部数组的长度，Vector缺省情况下自动增长原来一倍的数组长度，ArrayList是原来的50%,所以最后你获得的这个集合所占的空间总是比你实际需要的要大。所以如果你要在集合中保存大量的数据那么使用Vector有一些优势，因为你可以通过设置集合的初始化大小来避免不必要的资源开销。  
使用模式  
在ArrayList和Vector中， 从一个指定的位置（通过索引）查找数据或是在集合的末尾增加、移除一个元素所花费的时间是一样的，这个时间我们用O(1)表示。但是，如果在集合的其他位 置增加或移除元素那么花费的时间会呈线形增长：O(n-i)，其中n代表集合中元素的个数，i代表元素增加或移除元素的索引位置。为什么会这样呢？以为在 进行上述操作的时候集合中第i和第i个元素之后的所有元素都要执行位移的操作。这一切意味着什么呢？  
这意味着，你只是查找特定位置的元素或只在集合的末端增加、移除元素，那么使用Vector或 ArrayList都可以。如果是其他操作，你最好选择其他的集合操作类。比如，LinkList集合类在增加或移除集合中任何位置的元素所花费的时间都 是一样的?O(1)，但它在索引一个元素的使用缺比较慢－O(i),其中i是索引的位置.使用ArrayList也很容易，因为你可以简单的使用索引来代 替创建iterator对象的操作。LinkList也会为每个插入的元素创建对象，所有你要明白它也会带来额外的开销。**

**最后，在《Practical Java》一书中Peter Haggar建议使用一个简单的数组（Array）来代替Vector或ArrayList。尤其是对于执行效率要求高的程序更应如此。因为使用数组(Array)避免了同步、额外的方法调用和不必要的重新分配空间的操作。**