# 集合类说明

Collection  
├List  
│├LinkedList  
│├ArrayList  
│└Vector  
│　└Stack  
└Set  
Map  
├Hashtable  
├HashMap  
└WeakHashMap

## 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](http://lib.csdn.net/base/17" \o "Java EE知识库" \t "http://blog.csdn.net/softwave/article/details/4166598/_blank)** 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。  
　　所有实现Collection接口的类都必须提供两个标准的构造函数：无参数的构造函数用于创建一个空的Collection，有一个 Collection参数的构造函数用于创建一个新的Collection，这个新的Collection与传入的Collection有相同的元素。后 一个构造函数允许用户复制一个Collection。  
　　如何遍历Collection中的每一个元素？不论Collection的实际类型如何，它都支持一个iterator()的方法，该方法返回一个迭代子，使用该迭代子即可逐一访问Collection中每一个元素。典型的用法如下：  
　　　　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接口

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。  
　　除了具有Collection接口必备的iterator()方法外，List还提供一个listIterator()方法，返回一个 ListIterator接口，和标准的Iterator接口相比，ListIterator多了一些add()之类的方法，允许添加，删除，设定元素， 还能向前或向后遍历。  
　　实现List接口的常用类有LinkedList，ArrayList，Vector和Stack。

## 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](http://lib.csdn.net/base/31" \o "算法与数据结构知识库" \t "http://blog.csdn.net/softwave/article/details/4166598/_blank)** 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。

## 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接口

Set是一种不包含重复的元素的Collection，即任意的两个元素e1和e2都有e1.equals(e2)=false，Set最多有一个null元素。  
　　很明显，Set的构造函数有一个约束条件，传入的Collection参数不能包含重复的元素。  
　　请注意：必须小心操作可变对象（Mutable Object）。如果一个Set中的可变元素改变了自身状态导致Object.equals(Object)=true将导致一些问题。

## Map接口

请注意，Map没有继承Collection接口，Map提供key到value的映射。一个Map中不能包含相同的key，每个key只能映射一个 value。Map接口提供3种集合的视图，Map的内容可以被当作一组key集合，一组value集合，或者一组key-value映射。

## 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。

**总结**　　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。

**相互区别**

**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%.如过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，都是0(1),这个时候使用vector和arraylist都可以。而

如果移动一个指定位置的数据花费的时间为0(n-i)n为总长度，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

所花费的时间为0(1),而查询一个指定位置的数据时花费的时间为0(i)。

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！

**arraylist和linkedlist**

1.ArrayList是实现了基于动态数组的**[数据结构](http://lib.csdn.net/base/31" \o "算法与数据结构知识库" \t "http://blog.csdn.net/softwave/article/details/4166598/_blank)**，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**        (注)  
       文章出处：http://www.diybl.com/course/3\_program/java/javaxl/200875/130233.html  
  
       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）。

HashMap中元素的排列顺序是不固定的）。

        2、  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）。集合框架”提供两种常规的Map实现：HashMap和TreeMap (TreeMap实现SortedMap接口)。

         3、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。　　这个TreeMap没有调优选项，因为该树总处于平衡状态。

      结过研究，在原作者的基础上我还发现了一点，二树map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**

一.历史原因:Hashtable是基于陈旧的Dictionary类的，HashMap是Java 1.2引进的Map接口的一个实现

二.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

三.值：只有HashMap可以让你将空值作为一个表的条目的key或value