JAVA基本数据结构和排序算法

**Email:** [joyfly2006@yahoo.com.cn](mailto:joyfly2006@yahoo.com.cn)

**QQ:** 448086006

# Java容器类

## 容器作用和概念

### 数组

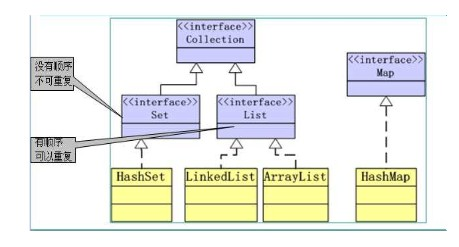
数组是一种容器，以线性序列放置对象和基本类型数据，可快速访问数组元素，但不灵活，容量必须事先定义好，不能随着需求的变化而扩容。基于此JAVA中提供容器类。

### 容器的架构

其层次如下图，都是从Object派生而来。需要注意的是Set、List、Collcetion和Map都是接口，不是具体的类实现。

  在Java中提供了Collection和Map接口。其中List和Set继承了Collection接口；同时用Vector、ArrayList、LinkedList三个类实现List接口，HashSet、TreeSet实现Set接口。直接有HashTable、HashMap、TreeMap实现Map接口。

List和set都是放单独的对象的，map则是方一个名值对，就是通过一个key找到一个value；list存放东西是有序的，set是没有顺序的;list允许重复存入，set不可以。



### List接口

有序的Collection，此接口用户可以对列表中每个元素的插入位置进行精确地控制，用户可以根据元素的整数索引（在列表中的位置）访问元素，并搜索列表中的元素，与Set不同，列表通常允许重复的元素，更确切地讲，列表通常允许满足e1.equals(e2)的元素对e1和e2，并且如果列表本身允许null元素。其方法主要包括：

//添加

boolean add(E e);

void add(int index, E element);

boolean addAll(Collection<? extends E> c);

boolean addAll(int index, Collection<? extends E> c);

//删除

boolean remove(Object o);

E remove(int index);

boolean removeAll(Collection<?> c);

//获取某个元素

E get(int index);

//获取某个元素的索引

int indexOf(Object o);

int lastIndexOf(Object o);

//是否相同

boolean equals(Object o);

//将某个元素指定添加到某个位置

E set(int index, E element);

//获取索引范围之内的元素

List<E> subList(int fromIndex, int toIndex);

//迭代器

ListIterator<E> listIterator();

ListIterator<E> listIterator(int index);

1. ArrayList

底层用数组实现的List，特点，查询效率高，增删效率低，线程不安全。

其扩容算法如下：

int newCapacity = (oldCapacity \* 3)/2 + 1;

1. Vector

底层用数组实现List，其实现方法与ArrayList类似，不同之处在于线程安全。其扩容算法如下：

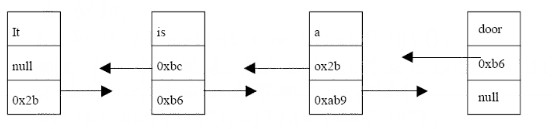
int newCapacity = (capacityIncrement > 0) ? (oldCapacity+capacityIncrement) : (oldCapacity \* 2);

capacityIncrement：设置的扩容步进值。

1. LinkedList

底层采用双向链表实现的List，特点，查询效率低，增删效率高，线程不安全。链表是由若干个称作节点的对象组成的一种数据结构，每个节点含有一个数据和下一个节点对象的引用，或含有一个数据并含有上一个节点对象的引用和下一个节点对象的引用(双向链表)。

LinkedList其实内部采用一个双向链表，如下图所示：



LinkedList继承了抽象的序列链表类，并实现了List、Queue、Cloneable、Serializable接口，使LinkedList可以像队列一样进行操作，同时可以克隆和串化，使其能保存到文件中。

### Set接口

Set是一种不包含重复的元素的无序Collection，每个具体Set实现类依赖添加的对象equals()方法来检查独一性。

1. HashSet

HashSet采用哈希算法实现的Set，其底层是用HashMap实现的，因此，查询效率高，增删效率也高。线程不安全。无序容器，在HashMap的键为HashSet加入的内容，而值则为HashSet本身。

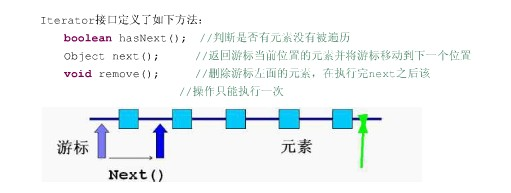
1. TreeSet

TreeSet依靠TreeMap来实现的，是一个有序集合，其元素按照升序排列，默认是按照自然顺序排列，也就是说TreeSet中的对象需要实现Comparable接口。TreeSet不能存在相同的元素。

### Iterator接口

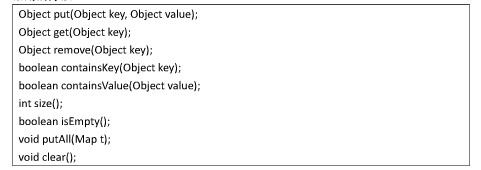
所有实现了Collection接口的容器类都有一个iterator方法用以返回一个实现了Iterator接口的对象，Iterator对象称作迭代器，用以方便的实现对容器内元素的遍历操作。

Iterator接口定义如下方法：



### Map接口

实现Map接口的类用来存储“键(key) – 值(value)”对,Map类中存储的键值对通过键来标识，所以键值不能重复，其方法有：



1. HashMap

线程不安全，效率高，允许key和value为null。

如果有多个元素被Hash函数定位到同一个桶内，我们称为hash冲突，桶内的元素组成单向链表。

1. HashTable

线程安全，效率低，不允许key或value为null。

备注： hashmap和hashtable的区别？

Hashmap是将键映射到值的对象，其中键和值都是对象，并且不能包含**重复键**，但可以包含重复值；Hashmap允许将null作为一个entry的key或者value，而hashtable不允许；Hashtable方法是synchronize的，而hashmap不是，在多个线程访问hashtable时，不需要自己为他的方法实现同步，而hashmap就必须为之提供外同步。

Hash表key的构造：

如果你想把一个原始类型用作一个key，你必须创建一个同等类型的对象，例如，如果你想用一个证书key，你应该用构造器Integer(int)从整数中生成一个对象，所有的封装类如Integer、Boolean、Float等把原始值看作是对象，他们重载了**equals()和hashCode()**方法。

　　hashtable和hashmap都应用广泛，二者比较类似。那么二者之间到底有哪些不同。  
　　1.hashtable是继承自陈旧的Dictionary类的，而hashmap继承自AbstractMap类的同时对Java1.2引进的Map接口进行了实现。  
　　2.hashtable的方法是同步的，而hashmap不是，所以在需要多线程应用中就可以使用hashtable,而在一个单线程应用时应采用hashmap,因为同步，hashtable的系统的开销相比hashmap自然加大。如果要实现hashmap的线程同步，就要用synchronized来块来处理。  
**3.hashtable的key 和value都不可以为null,而hashmap可以。**  
　　4.hashtable的contains方法，在hashmap中已不存在，而是用containsKey来代替相同的功能。  
　　5.二者的迭代子不同，hashtable的迭代子是Enumeration,hashmap是Iterator.  
　　6.hashtable和hashmap的默认size不同，前者默认size是11，后者是16.  
　　7.hashtable 和hashMap size增加方式不同。hashtable 的size增加方式是oldsize\*2+1,而HashMap是2的指数。  
　　8.hashcode的使用不同，，HashTable直接使用对象的hashCode，代码是这样的：  
　　int hash = key.hashCode();  
　　int index = (hash & 0x7FFFFFFF) % tab.length;  
　　而HashMap重新计算hash值，而且用与代替求模：  
　　int hash = hash(k);  
　　int i = indexFor(hash, table.length);  
　　static int hash(Object x) {  
　　int h = x.hashCode();  
　　h += ~(h << 9);  
　　h ^= (h >>> 14);  
　　h += (h << 4);  
　　h ^= (h >>> 10);  
　　return h;  
　　}  
　　static int indexFor(int h, int length) {  
　　return h & (length-1);  
　　}

1. TreeMap

TreeMap的实现就是红黑树数据结构，也就是说一颗自平衡的排序的二叉树，这样就可以保证当需要快速检索指定节点。

附：红黑树是一种自平衡排序二叉树，树中每个节点的值，都大于或等于在它的左子树中的所有节点的值，并且小于或等于在它的有子树中的所有节点的值，这确保红黑树运行时可以快速地在树中查找和定位到所需要节点。

**注意：在TreeMap中不能有相同的关键字**

## 容器类使用方法

### 遍历方法

1. List接口遍历方法

方法一：

For( int i = 0; I < list.size(); i++)

{

Entry temp = (Entry) list.get(i);

}

方法二：

For(E temp:list)

{

Temp即为访问的对象；

}

方法三：

For( Iterator iter = list.iterator(); iter.hasNext();Iter = iter.next())

{

/\*\*iter指示当前的内容\*/

}

方法四：

Iterator iter = list.iterator()

While(iter.hasNext())

{

Iter = iter.next()

}

1. Set接口遍历方法



1. Map接口遍历方法

key-set的方法遍历:

Set<String> key = map.**keySet**();

for (Iterator it = key.iterator(); it.hasNext();) {

String s = (String) it.**next**(); //得到key

map.**get**(s);//根据key得到value

}

value列表的方法遍历:

Student：为map的value类型

**Collection**<Student> c = map.**values**(); //获取值的列表

Iterator it = c.iterator();

for (; it.hasNext();) {

it.**next**();获取当前value的内容

}

Map内部的Entry的方法遍历:

Set<**Map.Entry**<String, Student>> set = map.**entrySet**();

for (Iterator<Map.Entry<String, Student>> it = set.iterator(); it.hasNext();) {

**Map.Entry<String, Student> entry** = (Map.Entry<String, Student>) it.next();

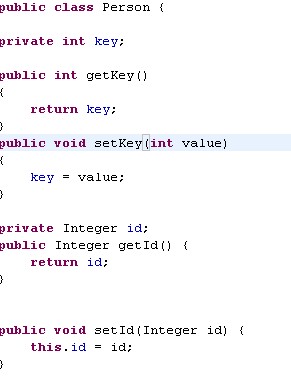
通过entry能得到任何信息。

}

# 排序算法

排序方法分为两大类：内部排序和外部排序，内部排序是指在排序过程中，所有数据都容纳在内存之中，内部排序适用数据量不太大的情形；外部排序用于数据量大的情形，在排序过程中全部数据不能同时存放在内存之中，下面先套路内部排序，包括：插入排序，快速排序、归并排序、堆排序、基数排序，然后在讨论外部排序。

以下讨论Person的内容如下：



## 插入排序

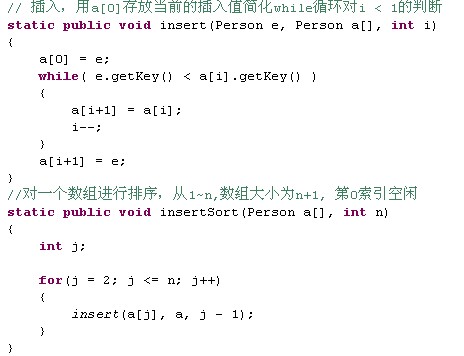
基本思路：在已排序的i条记录中插入一条新记录，得到有序的i+1条记录。

**特别提示：可以牺牲数组０的空间来作为插入的中间变量。**

**改进插入顺序：如果在插入过程中奖顺序查找改为折半查找，那么关键字的比较次数可以减少，记录的移动次数不变。**

**链式插入排序：不用数组而用链表存储数据，就不需要移动数据而仅仅需要改变链即可以实现。**

**数组插入排序如下：**



## 希尔排序

(1) 希尔排序的核心是以某个增量h 为步长跳跃分组进行插入排序，由于分组的步长h 逐步缩小，所以也叫“缩小增量排序”插入排序。其关键是如何选取分组的步长序列ht ,. . . , hk ,. . . , h1 , h0 才能使得希尔方法的时间效率最高；

(2) 待排序列记录的个数n 、跳跃分组步长逐步减小直到为1时所进行的扫描次数T、增量的和、记录关键字比较的次数以及记录移动的次数或各子序列中的反序数等因素都影响希尔算法的时间复杂度：其中记录关键字比较的次数是重要因素，它主要取决于分组步长序列的选择；

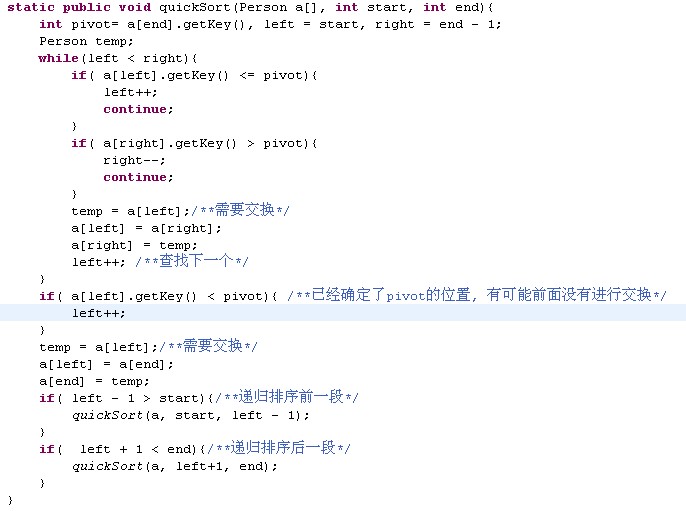
(3) 希尔方法是一种不稳定排序算法，因为其排序过程中各趟的步长不同，在第k 遍用hk作为步长排序之后，第k +1 遍排序时可能会遇到多个逆序存在，影响排序的稳定性。

试验结果表明,SHELL 算法的时间复杂度受增量序列的影响明显大于其他因素，选取恰当的增量序列能明显提高排序的时间效率，我们认为第k 趟排序扫描的增量步长为 2^k - 1 ,即增量序列为. . . 2^k - 1 ,. . . ,15 ,7 ,3 ,1时较为理想，但它并不是唯一的最佳增量序列，这与其关联函数目前尚无确定解的理论结果是一致的。

## 快速排序

基本思路：从待排序的记录中选一个分割记录，接着调整整个待排序记录，将分割记录放置在一个位置上，使得分割记录左边所有记录的关键字不大于分割记录，分割记录右边大于分割记录，最后将把分割记录左边的那些记录与分割记录右边的那些记录分别看作两个独立表，再对这两个表分别进行排序。

快速排序有最好的**平均性能**。

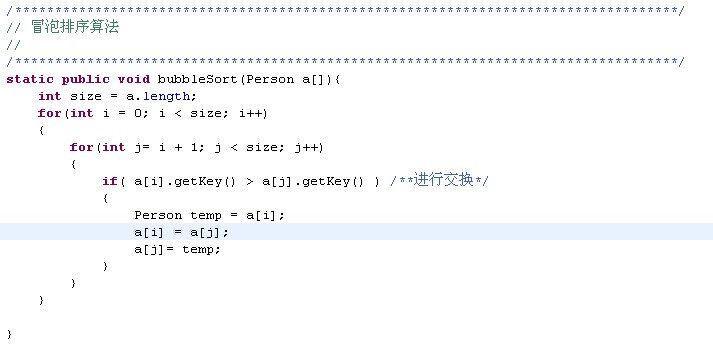


## 归并排序

内容：将两个有序的列表归并成一个有序列表的方法。

## 冒泡排序

最简单的排序方法是冒泡排序方法。基本思想：将待排序的元素看作是竖着排列的“气泡”，较小的元素比较轻，从而要往上浮。在冒泡排序算法中我们要对这个“气泡”序列处理若干遍。所谓一遍处理，就是自底向上检查一遍这个序列，并时刻注意两个相邻的元素的顺序是否正确。如果发现两个相邻元素的顺序不对，即“轻”的元素在下面，就交换它们的位置。显然，处理一遍之后，“最轻”的元素就浮到了最高位置；处理二遍之后，“次轻”的元素就浮到了次高位置。在作第二遍处理时，由于最高位置上的元素已是“最轻”元素，所以不必检查。一般地，第i遍处理时，不必检查第i高位置以上的元素，因为经过前面i-1遍的处理，它们已正确地排好序。这个算法可实现如下。



## 堆排序

堆实质是满足如下性质的完全二叉树，树中任一非叶节点的关键字不大于（或不小于）其左右孩子结点的关键字。

根节点的关键字是堆里所有节点关键字中最小者的堆称为小顶堆；

堆是一种完全二叉树，一般使用数组来实现。

