## sql.Date和util.Date

1） java.sql.Date 是 java.util.Date 的子类，是一个包装了毫秒值的瘦包装器，允许JDBC将毫秒值标识为 SQL DATE 值

2）java.sql.Date是针对 SQL 语句使用的，只包含日期而没有时间部分。

以下操作中容易出现不易被发现的 BUG：获得一个 JAVA 里的日期对象。 从数据库里读取日期 试图比较两个日期对象是否相等。如果毫秒部分丢失，本来认为相等的两个日期对象用 Equals 方法可能返回 false。sql.Timestamp比util.Date类精确度要高

## 匿名内部类可不可以继承或实现接口？

匿名内部类是没有名字的内部类,不能继承其它类,但内部类可以作为接口,由另一个内部类实现.

1、由于匿名内部类没有名字，所以它没有构造函数,所以它必须完全借用父类的构造函数来实例化，换言之：匿名内部类完全把创建对象的任务交给了父类去完成。

2、在匿名内部类里创建新的方法没有太大意义，但它可以通过覆盖父类的方法达到神奇效果

3、匿名内部类没有名字，所以无法向下强转，持有对一个匿名内部类对象引用的变量类型一定是它的直接或间接父类类型。

## 静态内部类和内部类有什么区别

静态内部类不需要有指向外部类的引用。但非静态内部类需要持有对外部类的引用。

静态内部类可以有静态成员(方法，属性)，而非静态内部类则不能有静态成员(方法，属性)。

静态内部类只能访问外部类的静态成员。非静态内部类能够访问外部类的静态和非静态成员。

实例化方式不同：

1) 静态内部类：不依赖于外部类的实例，直接实例化内部类对象

2) 非静态内部类：通过外部类的对象实例生成内部类对象

### Math.round(11.5)和Math.round(-11.5)

round()是+0.5再向下取整

所以是12 -11

### float f=3.4是否正确?

答:不正确。3.4 是双精度数，将双精度型（double）赋值给浮点型（float）

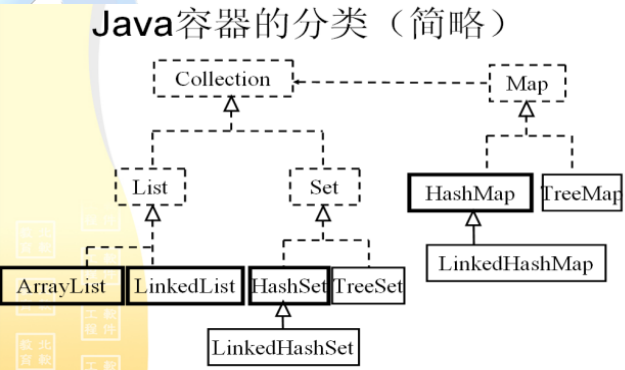
属于下转型 ,会造成精度损失，因此需要强

制类型转换 float f =(float)3.4; 或者写成 float f =3.4F;。

# 集合

## Collection（List、Set、Collection、Map的区别联系）

Java 中集合主要分为两种：Collection 和 Map



1、Collection不唯一，无序

2、List不唯一，有序

3、Set唯一，无序

4、Map Key无序，唯一 value不要求有序，允许重复

## ArrayList 和 LinkedList 的区别和联系

相同点：

都实现List接口，有序、不唯一

不同点：

ArrayList数组实现,长度可变，查询效率高 虽然是数组,初始容量10,但是当满了时,就会新建一个2倍容量的数组,并把原数组复制过去,实现了扩容

LinkedList 采用双向链表存储方式。插入、删除元素时效率高

#### List list = new ArrayList() 与 ArrayList alist = new ArrayList()

List接口有多个实现类，现在你用的是ArrayList，也许哪一天需要换成LinkedList或者Vector等等，这时你只要改变一行就行 这就是面向接口编程

LinkedList和ArrayList都实现了add等方法,都实现了List接口

在List list时,并不知道实例化了Linked还是Array,但是这个list都是要去add()的

这也是多态的体现,父类引用指向子类对象

## HashSet 的使用和原理（hashCode()和 equals()）

1）哈希表的查询速度特别快，时间复杂度为 O（1）。

2）HashMap、Hashtable、HashSet 这些集合采用的是哈希表结构，需要用hashCode

3）系统类已经覆盖了hashCode 方法 自定义类放入hash类集合，必须重写 hashcode。如果不重写，调用的是 Object 的 hashcode，而Object 的 hashCode 实际上是地址。

4）向哈希表中添加数据的原理：当向集合Set中增加对象时，首先集合计算要增加对象的 hashCode 码，得到一个位置用来存放当前对象，如在该位置没有一个对象存在的话，那么集合 Set 认为该对象在集合中不存在，直接增加进去。如果在该位置有一个对象存在的话，接着将准备增加到集合中的对象与该位置上的对象进行equals方法比较，如果该 equals 方法返回 false,那么集合认为集合中不存在该对象，在进行一次散列，将该对象放到散列后计算出的新地址里。如果equals方法返回 true，那么集合认为集合中已经存在该对象了，不会再将该对象增加到集合中了。

5）在哈希表中判断两个元素是否重复要使用到hashCode()和equals()。hashCode决定数据在表中的存储位置，而equals判断是否存在相同数据。

## TreeSet 的原理和使用（Comparable和comparator）

1）TreeSet 集合，元素不允许重复且有序(自然顺序)

2）TreeSet 采用树结构存储数据，存入元素时需要和树中元素进行对比，需

要指定比较策略。

3）可以通过 Comparable(外部比较器)和 Comparator(内部比较器)来指定

比较策略，实现了 Comparable 的系统类可以顺利存入 TreeSet。自定

义类可以实现 Comparable 接口来指定比较策略。

4）可创建 Comparator 接口实现类来指定比较策略，并通过 TreeSet 构造

方法参数传入。这种方式尤其对系统类非常适用。

## 集合和数组的比较（为什么引入集合）

数组不是面向对象的，存在明显的缺陷，集合完全弥补了数组的一些缺点，比数组更灵活实用，可大大提高软件的开发效率.不同的集合框架类可适用于不同场合

1）数组的效率高于集合类.

2）数组能存放基本数据类型和对象，而集合类中只能放对象

3）数组容量固定且无法动态改变，集合类容量动态改变

4）数组无法判断实际有多少元素，length只告诉了array的容量

5）集合有多种实现方式和不同的适用场合，而不像数组仅采用顺序表方式。

6）集合以类的形式存在，具有封装、继承、多态等类的特性，通过简单的方法和属性调用即可实现各种复杂操作，提高效率

## Collection 和 Collections 的区别

1）Collection集合接口，存储一组不唯一，无序的对象。两个子接口List和Set

2）Collections类专门用来操作集合类 ，它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。

Map 的实现类中，哪些是有序的，哪些是无序的，有序的是

如何保证其有序性，你觉得哪个有序性性能更高，你有没有更好

或者更高效的实现方式？

答：

1. Map 的实现类有 HashMap,LinkedHashMap,TreeMap

2. HashMap 是有无序的，LinkedHashMap 和 TreeMap 都是有序的（LinkedHashMap记录了添加数据的顺序；TreeMap默认是自然升序）

3. LinkedHashMap 底层存储结构是哈希表+链表，链表记录了添加数据的顺序

4. TreeMap 底层存储结构是二叉树，二叉树的中序遍历保证了数据的有序性

5. LinkedHashMap 性能比较高，因为底层数据存储结构采用的哈希表

## TreeMap 和 TreeSet 在排序时如何比较元素？Collections工具类中的 sort（）方法如何比较元素？

TreeSet 要求存放的对象所属的类必须实现 Comparable 接口，该接口提供了比较元素的 compareTo()方法，当插入元素时会 回调该方法比较元素的大小。

TreeMap 要求存放的键值对映射的键必须实现 Comparable接口从而根据键对元素进行排序。Collections 工具类的 sort 方法有两种重载的形式，第一种要求传入的待排序容器中存放的对象比较实现

Comparable 接口以实现元素的比较；第二种不强制性的要求容器中的元素必须可比较，但是要求传入第二个参数，参数是 Comparator 接口的子类型（需要重写 compare 方法实现元素的比较），相当于一个临时定义的排序规则，其实就是是通过接口注入比较元素大小的算法，也是对回调模式的应用。

## List、Map、Set 三个接口，存取元素时，各有什么特点？

List 以特定索引来存取元素，可有重复元素。

Set 不能存放重复元素（用对象的 equals()方法来区分元素是否重复）

Map 保存键值对映射，映射关系可以是一对一或多对一。

Set 和 Map 容器都有基于哈希存储和排序树（红黑树）的两种实现版本，基于哈希存储的版本理论存取时间复杂度为O(1)，而基于排序树版本的实现在插入或删除元素时会按照元素或元素的键（key）构成排序树从而达到排序和去重的效果。

# 数据结构

## 堆

堆就是用数组实现的二叉树，所有它没有使用父指针或者子指针。堆根据“堆属性”来排序，“堆属性”决定了树中节点的位置。

堆分为两种：最大堆和最小堆，两者的差别在于节点的排序方式。

在最大堆中，父节点的值比每一个子节点的值都要大。在最小堆中，父节点的值比每一个子节点的值都要小。这就是所谓的“堆属性”，并且这个属性对堆中的每一个节点都成立。

### 堆和普通树的区别

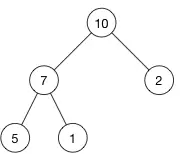
节点顺序。在二叉搜索树中，左子节点必须比父节点小，右子节点必须必比父节点大。但是在堆中并非如此。在最大堆中两个子节点都必须比父节点小，而在最小堆中，它们都必须比父节点大。

内存占用。普通树占用的内存空间比它们存储的数据要多。你必须为节点对象以及左/右子节点指针分配额为是我内存。堆仅仅使用一个数据来存储数组，且不使用指针。

平衡。二叉搜索树必须是“平衡”的情况下，其大部分操作的复杂度才能达到O(log n)。你可以按任意顺序位置插入/删除数据，或者使用 AVL 树或者红黑树，但是在堆中实际上不需要整棵树都是有序的。我们只需要满足对属性即可，所以在堆中平衡不是问题。因为堆中数据的组织方式可以保证O(log n) 的性能。

搜索。在二叉树中搜索会很快，但是在堆中搜索会很慢。在堆中搜索不是第一优先级，因为使用堆的目的是将最大（或者最小）的节点放在最前面，从而快速的进行相关插入、删除操作。

### 堆的存储方式

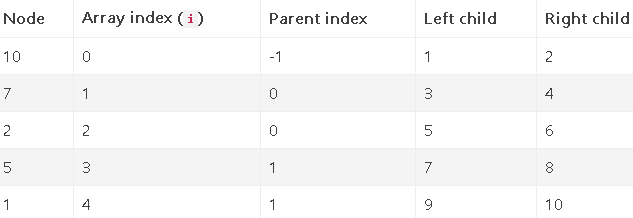
[ 10, 7, 2, 5, 1 ]

如果 i 是节点的索引，那么下面的公式就给出了它的父节点和子节点在数组中的位置

parent(i) = floor((i - 1)/2)

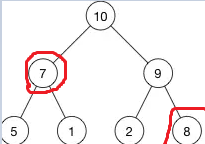
left(i) = 2i + 1

right(i) = 2i + 2



根节点(10)没有父节点，因为 -1 不是一个有效的数组索引。同样，节点 (2)，(5)和(1) 没有子节点，因为这些索引已经超过了数组的大小，所以我们在使用这些索引值的时候需要保证是有效的索引值。

堆并不一定是有序数组！要将堆转换成有序数组，需要使用堆排序。

堆中低层节点不一定小于高层节点！7<8

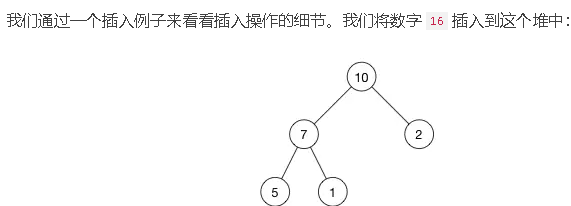
### 两个保证插入或删除节点后依然是最大/小堆的方法

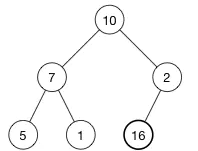
shiftUp(): 如果一个节点比它的父节点大（最大堆）或者小（最小堆），那么需要将它同父节点交换位置。这样是这个节点在数组的位置上升。

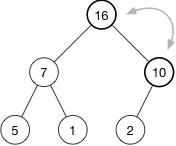
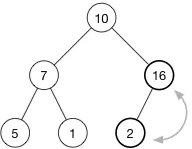
shiftDown(): 如果一个节点比它的子节点小（最大堆）或者大（最小堆），那么需要将它向下移动。这个操作也称作“堆化（heapify）”。

shiftUp 或者 shiftDown 是一个递归的过程，所以它的时间复杂度是 O(log n)

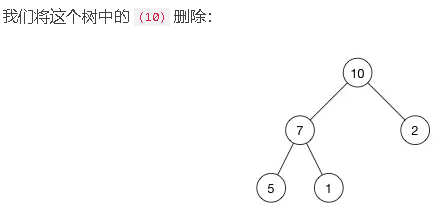
### 插入

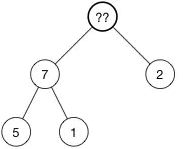


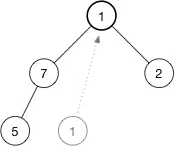
变成了[ 10, 7, 2, 5, 1, 16 ] 

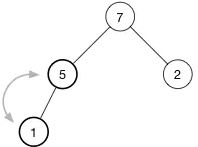
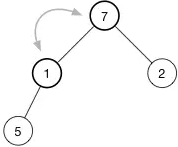
为了恢复堆属性,交换两次

### 删除根节点



此时顶部为空节点

取出最后一个节点,代替空节点

之后开始交换

### 删除子节点

remove() 的通用版本，它可能会使用到 shiftDown 和 shiftUp。

[ 10, 7, 2, 5, 1 ] 删除7

-> [ 10, 1, 2, 5, 7 ] 将最后的1与7交换,7就是需要返回的元素,用removeLast()删除,而1的位置不对,用shiftDown()调整位置