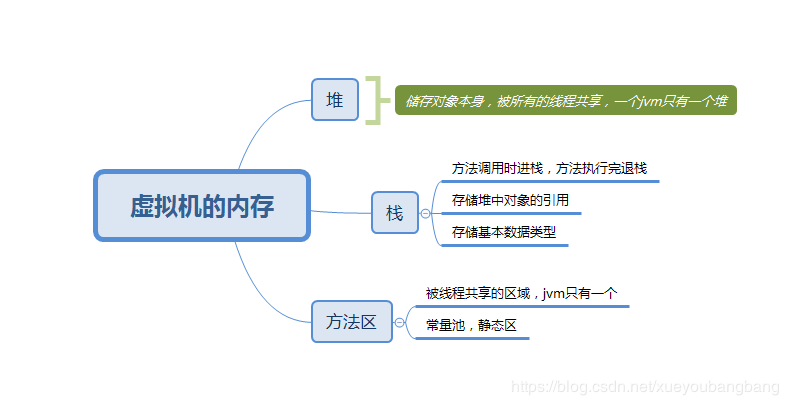
## 堆和栈

栈的效率高.由操作系统自动分配释放,在硬件层级对栈提供支持：分配专门的寄存器存放栈的地址，压栈出栈都有专门的指令执行

按照先后定义的顺序依次压入栈中，也就是说相邻变量的地址之间不会存在其它变量。栈的内存地址生长方向与堆相反，由高到底，后定义的变量地址低于先定义的变量。栈中存储的数据的生命周期随着函数的执行完成而结束

栈是运行时单位，用来解决程序运行时的问题，堆是存储单位，解决数据存储的问题。



堆: (heap)

1.存储的全部是对象，每个对象都包含一个与之对应的class的信息。(class的目的是得到操作指令)

2.只有一个堆,伴随着JVM的启动而创建。

3.堆的申请和释放工作由程序员控制，容易产生内存泄漏--己动态分配的堆内存未释放或无法释放

栈区:

1.每个线程一个栈，栈中只保存基本数据类型和自定义对象的引用(不存放对象)

2.每个栈中的数据(原始类型和对象引用)都是私有的，其他栈不能访问。

3.栈分为3个部分：基本类型变量区、执行环境上下文、操作指令区(存放操作指令)

方法区:



1.又叫静态区，跟堆一样，被所有的线程共享。方法区包含所有的class文件信息和static变量

2.方法区中包含的都是在整个程序中永远唯一的元素，如class，static变量。

常量池

1.常量池中包含基本类型和对象型的常量值（如String及数组）。

　　1.1 这些类是Byte,Short,Integer,Long,Character,Boolean,另外两种浮点数类型的包装类则没有实现。另 外Byte,Short,Integer,Long,Character这5种整型的包装类也只是在对应值小于等于127时才可使用常量池，也即对象不负责创建和管理大于127的这些类的对象。大于127的对象存放在堆中。

　　1.2 String类也实现了常量池技术

2.常量池还包含一些以文本形式出现的符号引用，比如：

　　2.1 类和接口的全限定名；

　　2.2 字段的名称和描述符；

　　2.3 方法和名称和描述符。

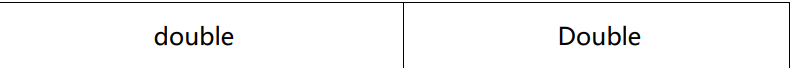
### 拆箱装箱JDK1.5

装箱：基本数据类型🡪包装器类型 valueOf方法

拆箱：包装器类型🡪基本数据类型 xxxValue方法

基本数据类型和包装类





基本数据类型不是面向对象（没有属性、方法），实际使用时存在很多的不便（比如集合的元素只能是 Object）。所以需要包装类。

## 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

## 下列说法错误的有（）(多选)

A. Java 面向对象语言容许单独的过栈与函数存在

B. Java 面向对象语言容许单独的方法存在

C. Java 语言中的方法属于类中的成员（member）

D. Java 语言中的方法必定隶属于某一类（对象），调用方法与过程或函

数相同

答案：ABC

B. Java 不允许单独的方法，过程或函数存在，需要隶属于某一类中；

C. 静态方法属于类的成员，非静态方法属于对象的成员。

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

round()是+0.5再向下取整

所以是12 -11

### static修饰变量,其值能改变吗?

对于静态变量在内存中只有一个拷贝（节省内存），JVM只为静态分配一次内存，在加载类的过程中完成静态变量的内存分配，可用类名直接访问（方便），当然也可以通过对象来访问（但是这是不推荐的）。

　对于实例变量，每创建一个实例，就会为实例变量分配一次内存，实例变量可以在内存中有多个拷贝，互不影响（灵活）。

所以一般在需要实现以下两个功能时使用静态变量：

1.在对象之间共享值时

2.方便访问变量时

static final修饰的成员变量可理解为“全局常量”，常量的值才是不能更改的，才是不可修改的！

静态变量并不是说其就不能改变值，不能改变值的量叫常量。 其拥有的值是可变的 ，而且它会保持最新的值。

说其静态，是因为它不会随着函数的调用和退出而发生变化。即上次调用的时候，如果我们给静态变量赋予某个值的话，下次函数调用时，这个值保持不变。

### 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（down-casting，也称为窄化）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;。

### short s1 = 1; s1 = s1 + 1;有错吗?short s1 = 1; s1 += 1;有错吗?

答：对于 short s1 = 1; s1 = s1 + 1;由于 1 是 int 类型，因此 s1+1 运算结

果也是 int 型，需要强制转换类型才能赋值给 short 型。而 short s1 = 1; s1

+= 1;可以正确编译，因为 s1+= 1;相当于 s1 = (short)(s1 + 1);其中有隐含的强制类型转换。

# 反射

## Class 类的作用？生成 Class 对象的方法有哪些？

Class 类是 Java 反射机制的起源和入口，用于获取与类相关的各种信息，提供了获取类信息的相关方法。Class 类继承自 Object 类Class 类是所有类的共同的图纸。每个类有自己的对象，好比图纸和实物的关系；每个类也可看做是一个对象，有共同的图纸 Class，存放类的 结构信息，能够通过相应方法取出相应信息：类的名字、属性、方法、构造方法、父类和接口

## 反射的使用场合和作用、及其优缺点

在编译时根本无法知道该对象或类可能属于哪些类，程序只依靠运行时信息来发现该对象和类的真实信息。通过反射可以使程序代码访问装载到 JVM 中的类的内部信息，获取已装载类的属性信息，获取已装载类的方法，获取已装载类的构造方法信息

反射提高了 Java 程序的灵活性和扩展性，降低耦合性，提高自适应能力。它允许程序创建和控制任何类的对象，无需提前硬编码目标类；

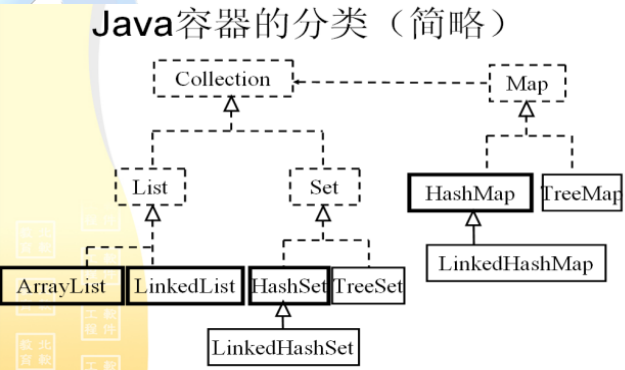
Java 反射技术应用领域很广，如软件测试等；许多流行的开源框架例如Struts、Hibernate、Spring 在实现过程中都采用了该技术

反射基本上是一种解释操作，用于字段和方法接入时要远慢于直接代码。因此 Java 反射机制主要应用在对灵活性和扩展性要求很高的系统框架上,普通程序不建议使用。使用反射会模糊程序内部逻辑：程序人员希望在源代码中看到程序的逻辑，反射等绕过了源代码的技术，因而会带来维护问题。反射代码比相应的直接代码更复杂。

# 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

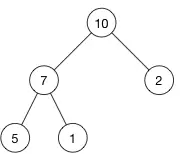
节点顺序。在二叉搜索树中，左子节点必须比父节点小，右子节点必须必比父节点大。但是在堆中并非如此。在最大堆中两个子节点都必须比父节点小，而在最小堆中，它们都必须比父节点大。

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

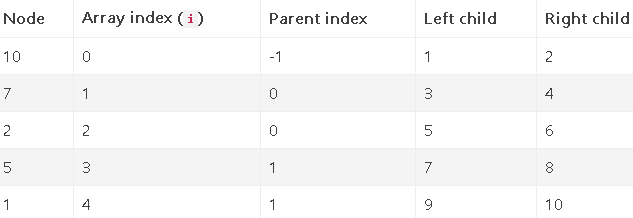
[ 10, 7, 2, 5, 1 ]

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)

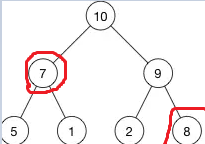
left(i) = 2i + 1

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

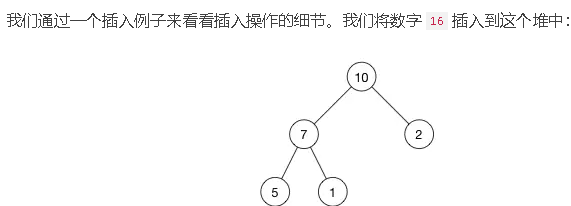
### 两个保证插入或删除节点后依然是最大/小堆的方法

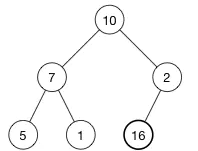
shiftUp(): 如果一个节点比它的父节点大（最大堆）或者小（最小堆），那么需要将它同父节点交换位置。这样是这个节点在数组的位置上升。

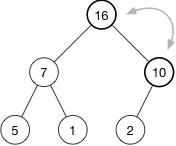
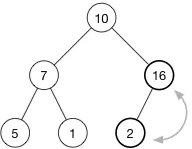
shiftDown(): 如果一个节点比它的子节点小（最大堆）或者大（最小堆），那么需要将它向下移动。这个操作也称作“堆化（heapify）”。

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)

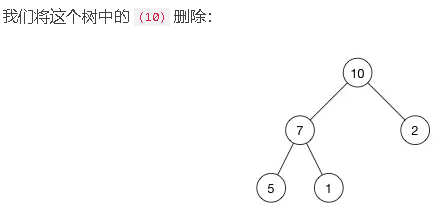
### 插入

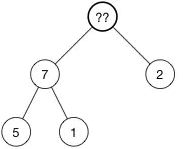


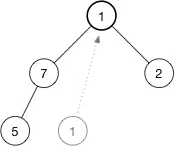
变成了[ 10, 7, 2, 5, 1, 16 ] 

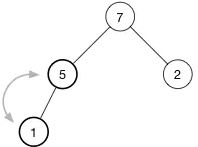
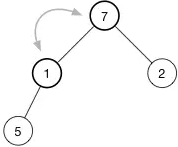
为了恢复堆属性,交换两次

### 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