# Java基础：

## 简述java程序执行的过程

由java源文件，通过javac.exe可执行文件，编译为字节码的class文件，然后通过解析器解析执行class文件，解析为计算机识别的二进制编码。

这几年的开发,我基本采用的是1.7版本,当然也使用过1.8版本,在新版的eclipse,必须使用1.8版本,才能运行 lamda

## 各种排序方式哪个效率高一些

测试数据100000条进行排序时，实现效率如下

冒泡排序：13098ms

选择排序：2334ms

插入排序：1264ms

快速排序：23ms

通过测试，几种排序效率差距有些大！快速排序效率最高

## new的线程放在堆里还是栈里?

堆里，对象实例放到堆里边，引用放到栈中

## Overload和Override的区别。Overloaded的方法是否可以改变返回值的类型?

方法的重写Overriding和重载Overloading是Java多态性的不同表现。

重写Overriding是父类与子类之间多态性的一种表现，重载Overloading是一个类中多态性的一种表现。

如果在子类中定义某方法与其父类有相同的名称和参数，我们说该方法被重写 (Overriding)。子类的对象使用这个方法时，将调用子类中的定义，对它而言，父类中的定义如同被“屏蔽”了。

如果在一个类中定义了多个同名的方法，它们或有不同的参数个数或有不同的参数类型，则称为方法的重载(Overloading)。

Overloaded的方法是可以改变返回值的类型,它只是与函数名和参数相关,与返回值类型无关。

## HashCode和equals()方法的作用是什么？有什么区别？是否必须存在？请说明理由

 hashCode()方法和equal()方法的作用其实一样，在[Java](http://lib.csdn.net/base/java)里都是用来对比两个对象是否相等一致;

 在实际操作经验中,我发现重写的equal（）里一般比较的比较全面比较复杂，这样效率就比较低，而利用hashCode()进行对比，则只要生成一个hash值进行比较就可以了，效率很高.

然而,hashCode()并不是完全可靠，有时候不同的对象他们生成的hashcode也会一样（当然不排除生成hash值得公式可能存在的问题）

总结起来就是:

1.equal()相等的两个对象他们的hashCode()肯定相等，也就是用equal()对比是绝对可靠的。

2.hashCode()不相等的两个对象他们的equal()肯定不相等

3.equal()相等的两个对象他们的hashCode()也可能相等

4.hashCode()相等的两个对象他们的equal()不一定相等，也就是hashCode()不是绝对可靠的。

所以,重写equals(Object obj)方法，有必要重写hashcode()方法,对于需要大量并且快速的对比的话如果都用equal()去做显然效率太低，所以解决方式是，每当需要对比的时候，首先用hashCode()去对比，如果hashCode()不一样，则表示这两个对象肯定不相等（也就是不必再用equal()去再对比了）,如果hashCode()相同，此时再对比他们的equal()，如果equal()也相同，则表示这两个对象是真的相同了，这样既能大大提高了效率也保证了对比的绝对正确性！

## Java集合

Java里边主要有两种集合，collection接口和map接口，其中collection下又包含list和set两个子接口：

**List子接口**：有序，可以有重复元素。和数组类似，List可以动态增长，查找元素效率高，相对的插入删除元素效率低，因为会引起其他元素位置改变。

**Set子接口**:无序，不允许重复。检索元素效率低下，删除和插入效率高，插入和删除不会引起元素位置改变。set集合中的元素**不按特定方式排序**，只是简单的把对象加入集合中，就像往口袋里放东西。

Map：双列集合，用于存放键值对。键值是唯一的，不可重复

**List接口下有三个实现类ArrayList 、LinkedList和Vector**

**Vector**是一种老的动态数组，是线程同步的，效率很低，一般不赞成使用

**ArrayList**实现一个动态数组，它的规模可变并且能像链表一样被访问。它提供的功能类似Vector类但**不同步**，它是以Array方式实现的List，**允许快速随机存取**。特点是读快改慢；

**ArrayList**是基于数组实现的，是一个动态数组（长度可变的数据结构，允许null当元素，默认长度是10，一次增长0.5倍，其容量为原来的1.5倍。

**LinkedList**实现一个链表，提供最佳顺序存取，适合插入和移除元素。由这个类定义的链

表也可以像栈或队列一样被使用。提供**最佳顺序存取**，**适合插入和移除元素**。特点是改快读慢

**Set接口有HashSet和TreeSet两个实现类**

**HashSet**能够快速定位一个元素，

要注意的是：**存入HashSet中的对象必须实现HashCode()方法**；

HashSet实现了Set接口，它不允许集合中有重复的值，当我们提到HashSet时，第一件事情就是在将对象存储在HashSet之前，要先确保对象重写equals()和hashCode()方法，这样才能比较对象的值是否相等，以确保set中没有储存相等的对象。如果我们没有重写这两个方法，将会使用这个方法的默认实现。

**TreeSet** 将放入其中的元素**按序存放**。它实现的是SortedSet接口，也就是加入了对象比较的方法。通过对集中的对象迭代，我们可以得到一个升序的对象集合。

Map接口的实现类主要有**HashMap** 、**HashTable和TreeMap；**

**当元素的顺序很重要时选用TreeMap，当元素不必以特定的顺序进行存储时，使用HashMap。**HashMap不是同步的，**Hashtable是同步的，但Hashtable不推荐使用，因为HashMap提供了所有类似的功能，并且速度更快。当需要在多线程环境下使用时，HashMap也可以转换为同步的。**HashMap没法保证映射的顺序一直不变，但是作为HashMap的子类LinkedHashMap可以。

HashMap可以通过Map m = Collections.synchronizedMap（hashMap）来达到同步的效果。



## set为什么不可以有重复？

set是一个特殊的Map，value值都是一个object，key值就是set的值，当key一样时，hash算法会计算到同一个节点，所以不能重复

## 数据结构

常用的有四类基本结构：集合、线性结构、树形结构、图状结构

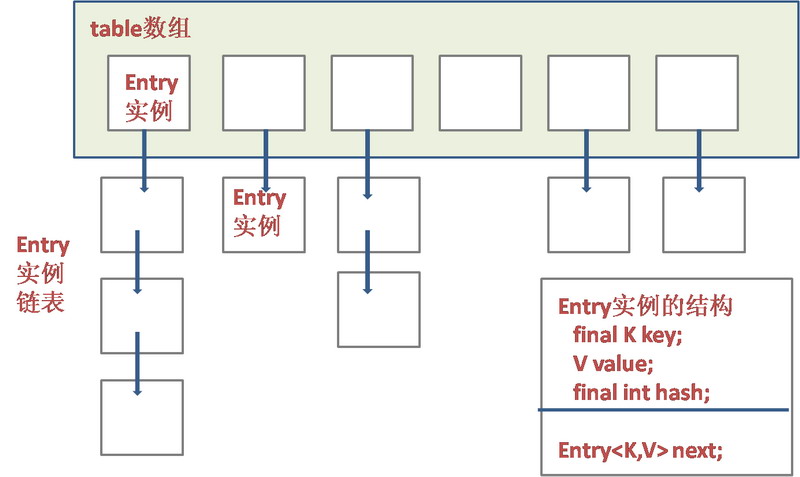
## 说出ArrayList,Vector, LinkedList的存储性能和特性？

ArrayList 和Vector他们底层的实现都是一样的，都是使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，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以查询数据快而插入,删改数据慢。  当Vector或ArrayList中的元素超过它的初始大小时,Vector会将它的容量翻倍,而ArrayList只增加50%的大小，这样,ArrayList就有利于节约内存空间

Vector中的方法由于添加了synchronized修饰，因此Vector是线程安全的容器，但性能上较ArrayList差，因此已经是Java中的遗留容器。

LinkedList使用双向链表实现存储（将内存中零散的内存单元通过附加的引用关联起来，形成一个可以按序号索引的线性结构，这种链式存储方式与数组的连续存储方式相比，内存的利用率更高），按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以增删改,查询慢些。

## HashMap底层原理

  
 HashSet是按照哈希算法来存取集合中的对象，具有很好的存取和查找性能，当向集合中加入一个对象时，HashSet会调用对象的hashCode()方法来获取哈希码，然后根据这个哈希吗来计算对象在集合中的存放位置。  
在Object类中定义了hashCode()和equal()，equal()是按照内存地址比较对象是否相同，如果object1.equal(object1)w为true时，则表明了object1和object2变量实际上引用了同一个对象，那么object1和object2的哈希码也是肯定相同。

## 堆和栈的区别？

heap是堆,空间是很大的自由区;stack是栈,空间有限。  
从存储内容和分配空间上看:

栈：基本类型的变量和对象的引用变量都在函数的栈内存中分配, 当在一段代码块定义一个变量时，Java就在栈中为这个变量分配内存空间，当超过变量的作用域后，Java会自动释放掉为该变量所分配的内存空间，该内存空间可以立即被另作他用。

堆：　堆内存用来存放由 new 创建的对象和数组，在堆中分配的内存，由 Java 垃圾回收器来管理。

## Java中有几种方法可以实现一个线程？用什么关键字修饰同步方法？

(1)继承Thread类，重写run函数

(2)实现Runnable接口，重写run函数

(3)实现Callable接口，重写call函数

Callable是类似于Runnable的接口，实现Callable接口的类和实现Runnable的类都 是可被其它线程执行的任务。

  Synchronized

改类的目的。

## sleep() 和 wait() 有什么区别?

① 这两个方法来自不同的类分别是，sleep来自Thread类，和wait来自Object类。

sleep不出让系统资源,依然保持监控状态，到时后会自动恢复；wait是进入线程等待池等待，出让系统资源，其他线程可以占用CPU。一般wait不会加时间限制，要等待其他线程调用notify/notifyAll唤醒等待池中的所有线程，才会进入就绪队列等待OS分配系统资源。sleep(milliseconds)可以用时间指定使它自动唤醒过来，如果时间不到只能调用interrupt()强行打断。

Thread.sleep(0)的作用是“触发操作系统立刻重新进行一次CPU竞争”。

② 使用范围：wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而sleep可以在任何地方使用。

   synchronized(x){   
      x.notify()   
     //或者wait()

## java 中操作字符串都有哪些类？它们之间有什么区别？

操作字符串的类有：String、StringBuffer、StringBuilder。

String 和 StringBuffer、StringBuilder 的区别在于 String 声明的是不可变的对象，每次操作都会生成新的 String 对象，然后将指针指向新的 String 对象，而 StringBuffer、StringBuilder 可以在原有对象的基础上进行操作，所以在经常改变字符串内容的情况下最好不要使用 String。

StringBuffer 和 StringBuilder 最大的区别在于，StringBuffer 是线程安全的，而 StringBuilder 是非线程安全的，但 StringBuilder 的性能却高于 StringBuffer，所以在单线程环境下推荐使用 StringBuilder，多线程环境下推荐使用 StringBuffer。

## 什么是线程池？

线程池是指在初始化一个多线程应用程序过程中创建一个线程 集合，然后在需要执行新的任务时重用这些线程而不是新建一个线程。线程池中线程的数量通常完全取决于可用内存数量和应用程序的需求。然而，增加可用线程数量是可能的。线程池中的每个线程都有被分配一个任务，一旦任务已经完成了，线程回到池子中并等待下一次分配任务。

## 线程的状态以及状态之间的相互转换

1、新建状态(New)：新创建了一个线程对象。

2、就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取CPU的使用权。

3、运行状态(Running)：就绪状态的线程获取了CPU，执行程序代码。

4、阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

①等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

②同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

③其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

1. 死亡状态(Dead)：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

## 线程安全的概念？

当多个线程执行一个并发任务时,始终都能得到我们想要的结果,就是线程安全的.

在某一段代码块或方法上使用synchronized关键字,使这段代码成为线程安全的代码.

但并不是使用了synchronized关键字,一定能保证线程安全.因为如果有多个线程,多把锁,两个锁互不影响,这时需要在具有synchronized关键字的方法上加上static,将锁变成类级别的锁(一把锁).

## 终止线程的方法有哪些?

使用退出标志,使线程正常退出,也就是当run方法完成后线程终止. 2.使用stop方法强制终止线程(这个方法不推荐使用,因为stop和suspend、resume一样,也可能发生不可预测的结果). 3.使用interrupt方法中断线程.

## String 类的常用方法都有那些？

**indexOf()：返回指定字符的索引。**

**charAt()：返回指定索引处的字符。**

**replace()：字符串替换。**

**trim()：去除字符串两端空白。**

**split()：分割字符串，返回一个分割后的字符串数组。**

**getBytes()：返回字符串的 byte 类型数组。**

**length()：返回字符串长度。**

**toLowerCase()：将字符串转成小写字母。**

**toUpperCase()：将字符串转成大写字符。**

**substring()：截取字符串。**

**equals()：字符串比较。**

## 普通类和抽象类有哪些区别？

**普通类不能包含抽象方法，抽象类可以包含抽象方法。**

**抽象类不能直接实例化，普通类可以直接实例化。**

## 接口和抽象类有什么区别？

**实现：抽象类的子类使用 extends 来继承；接口必须使用 implements 来实现接口。**

**构造函数：抽象类可以有构造函数；接口不能有。**

**main 方法：抽象类可以有 main 方法，并且我们能运行它；接口不能有 main 方法。**

**实现数量：类可以实现很多个接口；但是只能继承一个抽象类。**

**访问修饰符：接口中的方法默认使用 public 修饰；抽象类中的方法可以是任意访问修饰符。**

## java 中 IO 流分为几种？

**按功能来分：输入流（input）、输出流（output）。**

**按类型来分：字节流和字符流。**

**字节流和字符流的区别是：字节流按 8 位传输以字节为单位输入输出数据，字符流按 16 位传输以字符为单位输入输出数据。**

## BIO、NIO、AIO 有什么区别？

**BIO：Block IO 同步阻塞式 IO，就是我们平常使用的传统 IO，它的特点是模式简单使用方便，并发处理能力低。**

**NIO：New IO 同步非阻塞 IO，是传统 IO 的升级，客户端和服务器端通过 Channel（通道）通讯，实现了多路复用。**

**AIO：Asynchronous IO 是 NIO 的升级，也叫 NIO2，实现了异步非堵塞 IO ，异步 IO 的操作基于事件和回调机制。**

## Files的常用方法都有哪些？

**Files.exists()：检测文件路径是否存在。**

**Files.createFile()：创建文件。**

**Files.createDirectory()：创建文件夹。**

**Files.delete()：删除一个文件或目录。**

**Files.copy()：复制文件。**

**Files.move()：移动文件。**

**Files.size()：查看文件个数。**

**Files.read()：读取文件。**

**Files.write()：写入文件。**

## 多线程用途

1、常见的浏览器、Web服务(现在写的web是中间件帮你完成了线程的控制)，web处理请求，各种专用服务器(如游戏服务器)

2、servlet多线程

3、FTP下载，多线程操作文件

4、[数据库](http://lib.csdn.net/base/mysql)用到的多线程

6、tomcat，tomcat内部采用多线程，上百个客户端访问同一个WEB应用，tomcat接入后就是把后续的处理扔给一个新的线程来处理，这个新的线程最后调用我们的servlet程序，比如doGet或者dpPost方法

7、后台任务：如定时向大量(100W以上)的用户发送邮件；定期更新配置文件、任务调度(如quartz)，一些监控用于定期信息采集

8、自动作业处理：比如定期备份日志、定期备份数据库

9、异步处理：如发微博、记录日志

10、页面异步处理：比如大批量数据的核对工作(有10万个手机号码，核对哪些是已有用户)  
线程间的通信方式

**①同步**

这里讲的同步是指多个线程通过synchronized关键字这种方式来实现线程间的通信。

由于线程A和线程B持有同一个MyObject类的对象object，尽管这两个线程需要调用不同的方法，但是它们是同步执行的，比如：线程B需要等待线程A执行完了methodA()方法之后，它才能执行methodB()方法。这样，线程A和线程B就实现了 通信。

这种方式，本质上就是“共享内存”式的通信。多个线程需要访问同一个共享变量，谁拿到了锁（获得了访问权限），谁就可以执行。

**②while轮询的方式**

在这种方式下，线程A不断地改变条件，线程ThreadB不停地通过while语句检测这个条件(list.size()==5)是否成立 ，从而实现了线程间的通信。但是这种方式会浪费CPU资源。之所以说它浪费资源，是因为JVM调度器将CPU交给线程B执行时，它没做啥“有用”的工作，只是在不断地测试 某个条件是否成立。就类似于现实生活中，某个人一直看着手机屏幕是否有电话来了，而不是： 在干别的事情，当有电话来时，响铃通知TA电话来了。关于线程的轮询的影响，可参考：JAVA多线程之当一个线程在执行死循环时会影响另外一个线程吗？

这种方式还存在另外一个问题：

轮询的条件的可见性问题，关于内存可见性问题，可参考：JAVA多线程之volatile 与 synchronized 的比较中的第一点“一，volatile关键字的可见性”

线程都是先把变量读取到本地线程栈空间，然后再去再去修改的本地变量。因此，如果线程B每次都在取本地的 条件变量，那么尽管另外一个线程已经改变了轮询的条件，它也察觉不到，这样也会造成死循环。

**③wait/notify机制**

代码如下：

线程A要等待某个条件满足时(list.size()==5)，才执行操作。线程B则向list中添加元素，改变list 的size。

A,B之间如何通信的呢？也就是说，线程A如何知道 list.size() 已经为5了呢？

这里用到了Object类的 wait() 和 notify() 方法。

当条件未满足时(list.size() !=5)，线程A调用wait() 放弃CPU，并进入阻塞状态。---不像②while轮询那样占用CPU

当条件满足时，线程B调用 notify()通知 线程A，所谓通知线程A，就是唤醒线程A，并让它进入可运行状态。

这种方式的一个好处就是CPU的利用率提高了。

但是也有一些缺点：比如，线程B先执行，一下子添加了5个元素并调用了notify()发送了通知，而此时线程A还执行；当线程A执行并调用wait()时，那它永远就不可能被唤醒了。因为，线程B已经发了通知了，以后不再发通知了。这说明：通知过早，会打乱程序的执行逻辑

**④**管道通信就是使用java.io.PipedInputStream 和 java.io.PipedOutputStream进行通信

具体就不介绍了。分布式系统中说的两种通信机制：共享内存机制和消息通信机制。感觉前面的①中的synchronized关键字和②中的while轮询 “属于” 共享内存机制，由于是轮询的条件使用了volatile关键字修饰时，这就表示它们通过判断这个“共享的条件变量“是否改变了，来实现进程间的交流。而管道通信，更像消息传递机制，也就是说：通过管道，将一个线程中的消息发送给另一个。

## 反射机制主要功能

在运行时判断任意一个对象所属的类； 

在运行时构造任意一个类的对象； 

在运行时判断任意一个类所具有的成员变量和方法； 

在运行时调用任意一个对象的方法； 

生成动态代理。

## 说明: jvm原理图JVM虚拟机

## 说明: jvm原理图堆和栈

**①栈的空间比较小，**存放函数的参数值，局部变量的值等**。**

**②堆的空间比较大，用于存放new的对象。**

**③**与C++不同，Java自动管理栈和堆，程序员不能直接地设置栈或堆。

**④**堆（数据结构）：堆可以被看成是一棵树，如：二叉树。

**⑤**栈（数据结构）：一种先进后出的数据结构。

## gc是什么，几种方法?

gc是垃圾回收机制，垃圾回收器要负责三项任务：分配内存、确保被引用对象的内存不被错误地回收以及回收不再被引用的对象的内存空间。垃圾回收器的存在一方面把开发人员从释放内存的复杂工作中解脱出来，提高了开发人员的生产效率；另一方面，对开发人员屏蔽了释放内存的方法，可以避免因开发人员的错误地操作内存而导致应用程序的崩溃，保证了程序的稳定性。但是，垃圾回收也带来的问题，为了实现垃圾回收，垃圾回收器必须跟踪内存情况，释放没用的对象，在完成内存释放后还需要处理堆中的碎片，这些操作增加了JVM的负担，从而降低程序的执行效率。

  System.gc()方法,

  finalize()方法

# Javaweb基础：

## 存储过程

存储过程（procedure）是为了完成特定功能的SQL 语句集，存储在数据库中，经过第一次编译后再次调用不需要再次编译。存储过程里边可以定义变量、写if判断、写循环。他不能return返回，但是可以通过参数返回，有三种参数（in输入、our输出 还有 inout 即可输入又可输出。因为存储过程只在创造时进行编译，以后每次执行存储过程都不需再重新编译，而一般SQL语句每执行一次就编译一次,所以使用存储过程可提高数据库执行速度。当对数据库进行复杂操作时(如对多个表进行Update,Insert,Query,Delete时），可将此复杂操作用存储过程封装起来与数据库提供的事务处理结合一起使用。.存储过程可以重复使用,可减少数据库开发人员的工作量。安全性高,可设定只有某此用户才具有对指定存储过程的使用权。

当然他也有缺点，最大的缺点就是可移植性查，另外学习起来也有一定的难度。还有就是采用存储过程后也就意味着你的系统有一些业务逻辑不是在应用程序里处理，这种架构会增加一些系统维护和调试成本。所以，个人认为，普通业务逻辑尽量不要使用存储过程，定时性的ETL任务或报表统计函数可以根据团队资源情况采用存储过程处理。存储过程可以快速解决问题，但是移植性、维护性、扩展性不好，它有时会约束软件的架构，约速程序员的思维，在你的系统没有性能问题时不建议用存储过程。如果你要完成的功能只是一次或有限次的工作，如数据订正、数据迁移等等，存储过程也可以拿上用场。如果你的系统很小，并且有50%的开发人员熟练掌握PL/SQL，人员结构稳定，那存储过程可以减少很多代码量，并且性能不错。当系统变复杂了，开发人员多了，存储过程的弊端就会呈现。mysql存储过程用call调用，Oracle用excecute调用；mysql写存储过程，一般先要用delimiter $$，设置 $$为命令终止符号，代替分号 ,因为分号在存储过程里边要用；创建好以后， 在用delimiter ；恢复；

我之前写过一些存储过程，但最近接触的比较多的还是电商类的互联网项目，这类项目中，存储过程并不建议使用。最近一次存储过程记得是，为测试人员写的一个创建测试数据的存储过程。当时测试人员为了进行报表统计功能的测试，需要经常重复性的生成大量的数据，还设计到了很多张表。如果手动一条条加数据特别麻烦，于是找我想个办法，我就帮他写了一个存储过程；好久没写了，手比较生，当时也是一边找资料，一边写的。很简单的东西，写了近2个小时；

函数（function）与存储过程的结构类似，但是存储过程中不能用return返回值，函数必须有一个return子句，用于返回函数值；当然函数也可以用类似存储过程的参数返回；函数除了直接调用以外，可以直接放在select语句中使用，存储过程不行，存储过程定义关键字用procedure，函数定义用function。一般情况下如果要求有且有一个返回值，用函数，否则，用存储过程。

触发器（trigger）一般是执行sql语句的增删改查时使用，比如修改A表的一条数据时，让他自动触发一个操作，在B表里边新增一条记录；触发器因为需要绑定在表上，所以有数量限制，他里边的语法和存储过程、函数类似。

视图(View)实际上是多张表之间进行关联查询，把查询结果虚拟成一张表，就叫视图。创建视图也非常简单，create view viewname as + 查询语句，就可以了。视图的使用和表的使用完全一样，但不要在视图上进行增删改的操作，因为他本身就是一个查询结果的集合。利用视图，可以使我们把复杂的查询简单化，可以增强数据的安全性，屏蔽用户对基表的操作；他一般无法提高查询效率。