[一．共享锁和排它锁 2](#_Toc908)

[二．Tcp协议作用 2](#_Toc5510)

[四．数据库的设计总体上分为6个阶段： 3](#_Toc23685)

[数据库范式 3](#_Toc16566)

[线程池 6](#_Toc6343)

[十一．内存分配 8](#_Toc12820)

[十二．线程和进程 9](#_Toc13147)

[文件类型： 9](#_Toc29421)

[进程和线程 11](#_Toc10549)

[十六．Object类 13](#_Toc10708)

[各种排序方式的复杂度 14](#_Toc30275)

[实例方法和类方法 15](#_Toc15360)

[二十、死锁 15](#_Toc26586)

[二十四、数据库 17](#_Toc12933)

[三十三、设计模式 22](#_Toc22413)

[给初学者的设计模式面试问题 28](#_Toc19021)

[三十九、外部碎片和内部碎片 31](#_Toc14770)

[四十一、](#_Toc3337)**[Java虚拟机 运行时数据区](#_Toc3337)** [32](#_Toc3337)

[四十二、死锁 35](#_Toc1314)

[五、哈希索引和B树索引的区别 36](#_Toc620)

[四十四、反射机制 37](#_Toc26109)

**[1反射机制是什么](#_Toc17515)** [37](#_Toc17515)

**[3反射机制的相关API](#_Toc31602)** [37](#_Toc31602)

[四十六、面向对象的特征 38](#_Toc22106)

[四十七、接口和抽象类 38](#_Toc8410)

[四十八、数据库事务的四个特性及含义 39](#_Toc24096)

[四十九、](#_Toc1197)**[drop,delete与truncate的区别](#_Toc1197)** [39](#_Toc1197)

[五十、索引的工作原理及其种类 40](#_Toc6103)

[五十二、](#_Toc9239)**[数据库优化的思路](#_Toc9239)** [43](#_Toc9239)

**[1、SQL语句优化](#_Toc24461)** [43](#_Toc24461)

[2.索引优化 44](#_Toc14390)

[3.数据库结构优化 44](#_Toc18956)

**[五十四、多态](#_Toc19016)** [44](#_Toc19016)

[1、请求行：请求方法URI协议/版本 46](#_Toc19658)

[2. 请求头 47](#_Toc3669)

[二、 HTTP响应格式  49](#_Toc13937)

[1、状态行 50](#_Toc24598)

[2、响应正文 51](#_Toc1190)

[3 、响应头信息 51](#_Toc23596)

**[1.什么时候自动装箱](#_Toc18670)** [54](#_Toc18670)

[六十六、sleep()和wait()有什么区别? 56](#_Toc8154)

[六十七、notify()和notifyAll()区别 56](#_Toc7659)

[六十八、举例说明同步和异步。 56](#_Toc19436)

[六十九、简述synchronized 和java.util.concurrent.locks.Lock的异同？ 57](#_Toc30529)

[七十二、String 和StringBuilder、StringBuffer 的区别? 58](#_Toc2923)

[七十三、java.nio相关问题 58](#_Toc10723)

[七十四、依赖注入和控制反转 60](#_Toc9544)

[七十七、TCP的拥塞控制 62](#_Toc14239)

[八十、谈谈对jvm的理解 70](#_Toc6195)

## 一．共享锁和排它锁

　　1.共享锁

　　又称读锁，若事务T对数据对象A加上S锁，则事务T可以读A但不能修改A，其他事务只能再对A加S锁，而不能加X锁，直到T释放A上的S锁。这保证了其他事务可以读A，但在T释放A上的S锁之前不能对A做任何修改。

　　2.排他锁

又称写锁。若事务T对数据对象A加上X锁，事务T可以读A也可以修改A，其他事务不能再对A加任何锁，直到T释放A上的锁。 这保证了其他事务在T释放A上的锁之前不能再读取和修改A

## 二．Tcp协议作用

（1）在[流量控制](https://www.baidu.com/s?wd=%E6%B5%81%E9%87%8F%E6%8E%A7%E5%88%B6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1Rzm17bmWfsuyczPvf40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnsPWTknjbL" \t "https://zhidao.baidu.com/question/_blank)上，采用[滑动窗口协议](https://www.baidu.com/s?wd=%E6%BB%91%E5%8A%A8%E7%AA%97%E5%8F%A3%E5%8D%8F%E8%AE%AE&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1Rzm17bmWfsuyczPvf40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnsPWTknjbL" \t "https://zhidao.baidu.com/question/_blank)，协议中规定，对于窗口内未经确认的分组需要重传。  
　（2）在[拥塞控制](https://www.baidu.com/s?wd=%E6%8B%A5%E5%A1%9E%E6%8E%A7%E5%88%B6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1Rzm17bmWfsuyczPvf40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnsPWTknjbL" \t "https://zhidao.baidu.com/question/_blank)上，采用广受好评的TCP[拥塞控制](https://www.baidu.com/s?wd=%E6%8B%A5%E5%A1%9E%E6%8E%A7%E5%88%B6&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Ykm1Rzm17bmWfsuyczPvf40ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EPjnsPWTknjbL" \t "https://zhidao.baidu.com/question/_blank)算法（也称AIMD算法），该算法主要包括三个主要部分：1，加性增、乘性减；2，慢启动；3，对超时事件做出反应。

1. ****接口**** 只能用 ****public****和 ****abstract****修饰

## 四．数据库的设计总体上分为6个阶段：

****1、**** ****需求分析阶段****

    准确了解用户的需求，撰写需求说明

****2、概念设计阶段****

    它是整个数据库设计的关键，通过对用户需求进行综合，归纳与抽象，形成一个独立于具体DBMS的概念模型。**E-R图的设计**在此阶段。

****3、逻辑结构设计阶段****

   将概念结果转换为某个DBMS所支持的数据模型。也就是**指E-R图和关系模型的转换**，具体为将实体，实体的属性和实体之间的联系转换为关系模式。

****4、数据库物理设计阶段****

   为逻辑结果选取一个最适合应用环境的物理结构，包括**存储结构和存取方法。**

****5、数据库实施阶段****

  此阶段利用SQL语句实现**逻辑结构设计和物理设计阶**段的内容，包括建立数据库，编制与调试应用程序等。

****6、数据库运行和维护阶段****

  运行过程中不断的调整，修改和优化数据库系统。

## 数据库范式

第一范式（1NF）：强调的是列的原子性，即列不能够再分成其他几列。

第二范式（2NF）：首先是 1NF，另外包含两部分内容，一是表必须有一个主键；二是没有包含在主键中的列必须完全依赖于主键，而不能只依赖于主键的一部分。

第三范式（3NF）：首先是 2NF，另外非主键列必须直接依赖于主键，不能存在传递依赖。即不能存在：非主键列 A 依赖于非主键列 B，非主键列 B 依赖于主键的情况。

第二范式（2NF）和第三范式（3NF）的概念很容易混淆，区分它们的关键点在于，2NF：非主键列是否完全依赖于主键，还是依赖于主键的一部分；3NF：非主键列是直接依赖于主键，还是直接依赖于非主键列。

第一范式(1NF)   
定义：如果关系模式R的每个关系r的属性都是不可分的数据项，那么就称R是第一范式的模式。   
简单的说，每一个属性都是原子项，不可分割。   
1NF是关系模式应具备的最起码的条件，如果数据库设计不能满足第一范式，就不称为关系型数据库。关系数据库设计研究的关系规范化是在1NF之上进行的。   
  
例如(学生信息表)：   
学生编号　　姓名　　性别　　联系方式   
20080901　　张三　　男　　　email:zs@126.com,phone:88886666   
20080902　　李四　　女　　　email:ls@126.com,phone:66668888   
  
以上的表就不符合，第一范式：联系方式字段可以再分，所以变更为正确的是：   
  
学生编号　　姓名　　性别　　电子邮件　　　电话   
20080901　　张三　　男　　　zs@126.com　　88886666   
20080902　　李四　　女　　　ls@126.com　　 66668888   
  
第二范式（2NF）   
定义：如果关系模式R是1NF，且每个非主属性 完全函数依赖(往后看) 于候选键，那么就称R是第二范式。   
简单的说，第二范式要满足以下的条件：首先要满足第一范式，其次每个非主属性要完全函数依赖与候选键，或者是主键。也就是说，每个非主属性是由整个主键函数决定的，而不能由主键的一部分来决定。   
  
例如(学生选课表):  
学生　　　　课程　　　教师　　　　教师职称　　教材　　　　　　　　　教室　　上课时间  
李四　　　　Spring　　张老师　　　java讲师　　《Spring深入浅出》　　３０１　08:00  
张三　　　　Struts　　杨老师　　　java讲师　　《Struts in Action》　３０２　13:30  
  
这里通过（学生，课程）可以确定教师、教师职称，教材，教室和上课时间，所以可以把（学生，课程）作为主键。但是，教材并不完全依赖于（学生，课程），只拿出课程就可以确定教材，因为一个课程，一定指定了某个教材。这就叫不完全依赖，或者部分依赖。出现这种情况，就不满足第二范式。   
  
修改后，选课表：   
学生　　　　课程　　　教师　　　　教师职称　　教室　　上课时间  
李四　　　　Spring　　张老师　　　java讲师　　３０１　08:00  
张三　　　　Struts　　杨老师　　　java讲师　　３０２　13:30  
  
课程表：   
课程　　　教材   
Spring　　《Spring深入浅出》   
Struts　　《Struts in Action》   
  
所以，第二范式可以说是消除部分依赖。第二范式可以减少插入异常，删除异常和修改异常。

第三范式（3NF）：符合2NF，并且，消除传递依赖（也就是每个非主属性都不传递依赖于候选键，判断传递函数依赖，指的是如果存在"A → B → C"的决定关系，则C传递函数依赖于A。）

BC范式（BCNF）：符合3NF，并且，主属性不依赖于主属性(也就是不存在任何字段对任一候选关键字段的传递函数依赖)

BC范式既检查非主属性，又检查主属性。当只检查非主属性时，就成了第三范式。满足BC范式的关系都必然满足第三范式。

还可以这么说：若一个关系达到了第三范式，并且它只有一个候选码，或者它的每个候选码都是单属性，则该关系自然达到BC范式。

给你举个例子：假设仓库管理关系表 (仓库ID, 存储物品ID, 管理员ID, 数量)，且有一个管理员只在一个仓库工作；一个仓库可以存储多种物品。

这个数据库表中存在如下决定关系：

(仓库ID, 存储物品ID) →(管理员ID, 数量)

(管理员ID, 存储物品ID) → (仓库ID, 数量)

所以，(仓库ID, 存储物品ID)和(管理员ID, 存储物品ID)都是StorehouseManage的候选关键字，表中的唯一非关键字段为数量，它是符合第三范式的。但是，由于存在如下决定关系：

(仓库ID) → (管理员ID)

(管理员ID) → (仓库ID)

即存在关键字段决定关键字段的情况，所以其不符合BCNF范式。

第四范式：要求把同一表内的多对多关系删除。

第五范式：从最终结构重新建立原始结构。

七．



## 线程池

Java的四种线程池的使用：

Java通过Executors提供四种线程池，分别为：  
newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。  
newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。  
newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。  
newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。+++++

1. HTML5 提供了两种在客户端存储数据的新方法： localStorage - 没有时间限制的数据存储 sessionStorage - 针对一个 session 的数据存储 之前。sessionStorage 、localStorage 和 cookie 之间的区别 共同点：都是保存在浏览器端，且同源的。 区别： 1:cookie数据始终在同源的http请求中携带（即使不需要），即cookie在浏览器和服务器间来回传递。而sessionStorage和localStorage不会自动把数据发给服务器，仅在本地保存。cookie数据还有路径（path）的概念，可以限制cookie只属于某个路径下。 2:存储大小限制也不同，cookie数据不能超过4k，同时因为每次http请求都会携带cookie，所以cookie只适合保存很小的数据，如会话标识。sessionStorage和localStorage 虽然也有存储大小的限制，但比cookie大得多，可以达到5M或更大。 3:数据有效期不同，sessionStorage：仅在当前浏览器窗口关闭前有效，自然也就不可能持久保持；localStorage：始终有效，窗口或浏览器关闭也一直保存，因此用作持久数据；cookie只在设置的cookie过期时间之前一直有效，即使窗口或浏览器关闭。 4:作用域不同，sessionStorage不在不同的浏览器窗口敏感词享，即使是同一个页面；localStorage在所有同源窗口中都是共享的；cookie也是在所有同源窗口中都是共享的。 5:Web Storage支持事件通知机制，可以将数据更新的通知发送给监听者。 6:Web Storage 的 api 接口使用更方便。
2. GET和POST的区别：

(1)Get是从服务器上获取数据，Post是向服务器传送数据。

(2)Get是把参数数据队列加到提交表单的Action属性所指向的URL中，值和表单内各个字段一一对应，在URL中科院看到。

(3)Get传送的数据量小，不能大于2KB；post传送的数据量较大，一般被默认为不受限制。

(4)根据HTTP规范，GET用于信息获取，而且应该是安全的和幂等的。

　　I.所谓**安全的**意味着该操作用于获取信息而非修改信息。换句话说，GET 请求一般不应产生副作用。就是说，它仅仅是获取资源信息，就像**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/jxh_123/article/details/_blank)**查询一样，不会修改，增加数据，不会影响资源的状态。

II.**幂等**的意味着对同一URL的多个请求应该返回同样的结果。

1.GET请求的数据会附在URL之后（就是把数据放置在HTTP协议头中），以?分割URL和传输数据，参数之间以&amp;amp;amp;POST把提交的数据则放置在是HTTP包的包体中。 2.&amp;amp;quot;GET方式提交的数据最多只能是1024字节，理论上POST没有限制。 3.POST的安全性要比GET的安全性高。注意：这里所说的安全性和上面GET提到的“安全”不是同个概念。上面“安全”的含义仅仅是不作数据修改，而这里安全的含义是真正的Security的含义，比如：通过GET提交数据，用户名和密码将明文出现在URL上，因为(1)登录页面有可能被浏览器缓存，(2)其他人查看浏览器的历史纪录，那么别人就可以拿到你的账号和密码了，除此之外，使用GET提交数据还可能会造成Cross-site request forgery攻击。

## 十一．内存分配

1） 从静态存储区域分配。内存在程序编译的时候就已经分配好，这块内存在程序的整个运行期间都存在。例如全局变量，static 变量。

2） 在栈上创建。在执行函数时，函数内局部变量的存储单元都可以在栈上创建，函数执行结束时这些存储单元自动被释放。栈内存分配运算内置于处理器的指令集。

3） 从堆上分配，亦称动态内存分配。程序在运行的时候用malloc 或new 申请任意多少的内存，程序员自己负责在何时用free 或delete 释放内存。动态内存的生存期由程序员决定，使用非常灵活，但问题也最多。

栈：存放基本类型的变量数据和对象的引用，但对象本身不存放在栈中，而是存 放在堆（new 出来的对象）或者常量池中（字符串常量对象存放在常量池中。）

堆：存放所有new出来的对象或数组。  
静态域：存放静态成员（static定义的）  
常量池：存放字符串常量和基本类型常量（public static final）

堆和常量池，对于 栈和常量池中的对象可以共享，对于堆中的对象不可以共享。**栈中的数据大小和生命周期是可以确定的，当没有引用指向数据时，这个数据就会消失。堆中的对象的由垃圾回收器负责回收**，因此大小和生命周期不需要确定 ，具有很大的灵活性。  
对于字符串：其对象的引用都是存储在栈中的，如果是 编译期已经创建好(直接用双引号定义的)的就存储在常量池中，如果是运行期（new出来的）才能确定的就存储在堆中 。对于equals相等的字符串，在常量池中永远只有一份，在堆中有多份。

局部变量的数据存在于栈内存中 。栈内存中的局部变量随着方法的消失而消失。  
成员变量存储在堆中的对象里面 ，由垃圾回收器负责回收

**堆是由垃圾回收来负责的，堆的优势是可以动态地分配内存 大小，**生存期也不必事先告诉编译器**，因为它是在运行时动态分配内存的，Java的垃圾收集器会自动收走这些不再使用的数据。但缺点是，由于要在运行时动态 分配内存，存取速度较慢。**   
  
**栈的优势是，存取速度比堆要快，仅次于寄存器，栈数据可以共享。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。**

内存分配的方法：

****最优适应算法：通常将空闲区按长度递增顺序排列****。查找时总是从最小一个空闲区开始，直到找到满足要求的分区为止。此算法保证不会分割一个更大的区域，使得装入大作业的要求容易得到满足。

****最先适应算法：通常将空闲区按地址从小到大排列****。查找时总是从低地址开始，可使高地址尽量少用，以保持一个大空闲区，有利于大作业的装入；缺点是内存低地址和高地址两端的分区利用不平衡，回收分区较麻烦。

****最坏适应算法****：****通常将空闲区按长度递减顺序排列**** 。查找时从最大的一个空闲区开始，总是挑选一个最大的空闲区分割给作业使用，其优点是使剩下的空闲区不致于太小，这样有利于中小型作业，但不利于大作业。

这些都属于可变分区分配算法，当然还有下次适应分配算法和快速适应分配算法。

为文件分配外存空间时，常用的分配方法有连续分配、链接分配、索引分配三类

## 十二．线程和进程

在引入线程的操作系统中，线程是进程中的一个实体，是系统独立调度和分派的基本单位。但是线程自己基本上不拥有系统资源，所以它不是资源分配的基本单位，它只拥有一部分在运行中必不可少的与处理机相关的资源，如线程状态、寄存器上下文和栈等，它同样有就绪、阻塞和执行三种基本状态。它可与同属一个进程的其他线程共享进程所拥有的全部资源。一个线程可以创建和撤销另一个线程；同一个进程中的多个线程之间可以并发执行。由于用户线程不依赖于操作系统内核，因此，操作系统内核是不知道用户线程的存在的，用户线程是由用户来管理和调度的，用户利用线程库提供的API来创建、同步、调度和管理线程。所以，用户线程的调度在用户程序内部进行，通常采用非抢先式和更简单的规则，也无须用户态和核心态切换，所以速度很快。由于操作系统不知道用户线程的存在，所以，操作系统把CPU的时间片分配给用户进程，再由用户进程的管理器将时间分配给用户线程。那么，用户进程能得到的时间片即为所有用户线程共享。

## 文件类型：

顺序文件：每条记录都使用同一种固定的格式，所有记录具有相同的长度，不可随机存取；

索引文件：

**索引文件**由数据文件组成，它是带**索引**的顺序文件。**索引**本身非常小，只占两个字段；顺序文件的键和在磁盘上相应记录的地址。存取文件中的记录需按以下步骤：

（1）整个索引文件都载入到内存中（文件很小，只占用很小的内存空间）。

（2）搜索项目，用高效的算法（如折半查询法）查找目标键。

（3）检索记录的地址。

（4）按照地址，检索数据记录并返回给用户。[1] 

由**索引表**和**主文件**两部分构成。

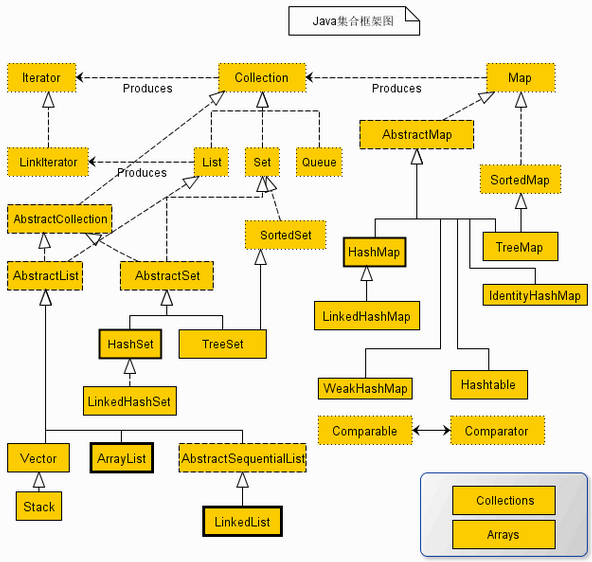
**索引表**是一张指示逻辑记录和物理记录之间对应关系的表。索引表中的每项称作索引项。索引项是按键（或逻辑[记录号](http://baike.baidu.com/item/%E8%AE%B0%E5%BD%95%E5%8F%B7" \t "http://baike.baidu.com/_blank)）顺序排列。若文件本身也是按关键字顺序排列，则称为索引[顺序文件](http://baike.baidu.com/item/%E9%A1%BA%E5%BA%8F%E6%96%87%E4%BB%B6" \t "http://baike.baidu.com/_blank)。否则，称为索引非[顺序文件](http://baike.baidu.com/item/%E9%A1%BA%E5%BA%8F%E6%96%87%E4%BB%B6" \t "http://baike.baidu.com/_blank)。

主文件按[主关键字](http://baike.baidu.com/item/%E4%B8%BB%E5%85%B3%E9%94%AE%E5%AD%97" \t "http://baike.baidu.com/_blank)有序的文件称索引[顺序文件](http://baike.baidu.com/item/%E9%A1%BA%E5%BA%8F%E6%96%87%E4%BB%B6" \t "http://baike.baidu.com/_blank)。在索引[顺序文件](http://baike.baidu.com/item/%E9%A1%BA%E5%BA%8F%E6%96%87%E4%BB%B6" \t "http://baike.baidu.com/_blank)中，可对一组记录建立一个索引项。这种索引表称为稀疏索引。

主文件按[主关键字](http://baike.baidu.com/item/%E4%B8%BB%E5%85%B3%E9%94%AE%E5%AD%97" \t "http://baike.baidu.com/_blank)无序的文件称索引非[顺序文件](http://baike.baidu.com/item/%E9%A1%BA%E5%BA%8F%E6%96%87%E4%BB%B6" \t "http://baike.baidu.com/_blank)。在索引非[顺序文件](http://baike.baidu.com/item/%E9%A1%BA%E5%BA%8F%E6%96%87%E4%BB%B6" \t "http://baike.baidu.com/_blank)中，必须为每个记录建立一个索引项，这样建立的索引表称为稠密索引。

聚集索引和非聚集索引的根本区别是表记录的排列顺序和与索引的排列顺序是否一致

，聚集索引表记录的排列顺序与索引的排列顺序一致，优点是查询速度快，因为一旦具有第一个索引值的纪录被找到，具有连续索引值的记录也一定物理的紧跟其后。聚集索引的缺点是对表进行修改速度较慢，这是为了保持表中的记录的物理顺序与索引的顺序一致，而把记录插入到数据页的相应位置，必须在数据页中进行数据重排，

1. Java集合框架图

## 进程和线程

1.定义

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.

线程是进程的一个实体,是CPU调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

2.关系

一个线程可以创建和撤销另一个线程;同一个进程中的多个线程之间可以并发执行.

相对进程而言，线程是一个更加接近于执行体的概念，它可以与同进程中的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行序列。

进程和线程的主要差别在于它们是不同的**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/yaosiming2011/article/details/_blank)**资源管理方式。进程有独立的地址空间，一个进程崩溃后，在保护模式下不会对其它进程产生影响，而线程只是一个进程中的不同执行路径。线程有自己的堆栈和局部变量，但线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉，所以多进程的程序要比多线程的程序健壮，但在进程切换时，耗费资源较大，效率要差一些。**但对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程。**

**1) 简而言之,一个程序至少有一个进程,一个进程至少有一个线程.**

2) 线程的划分尺度小于进程，使得多线程程序的并发性高。

3) 另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。

4) 线程在执行过程中与进程还是有区别的。每个独立的线程有一个程序运行的入口、顺序执行序列和程序的出口。**但是线程不能够独立执行，**必须依存在应用程序中，由应用程序提供多个线程执行控制。

5) 从逻辑角度来看，多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。但操作系统并没有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配。**这就是进程和线程的重要区别。**

3、进程间通讯的方式有哪些，各有什么优缺点：

1）管道：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程之间使用。进程的亲缘关系通常是指父子进程关系。

2）有名管道（FIFO）：有名管道也是半双工的通信方式，但是允许在没有亲缘关系的进程之间使用，管道是先进先出的通信方式。

3）信号量：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

4）消息队列：消息队列是有消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

5）信号 ( sinal ) ：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

6）共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。  
7）套接字( socket ) ：套接字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同机器间的进程通信。

## 十六．Object类

Object类存储在java.lang包中，是所有java类(Object类除外)的终极父类。当然，数组也继承了Object类。然而，接口是不继承Object类的

1. clone()方法

Object将clone()作为一个本地方法来实现，这意味着它的代码存放在本地的库中。当代码执行的时候，将会检查调用对象的类(或者父类)是否实现了java.lang.Cloneable接口(Object类不实现Cloneable)。如果没有实现这个接口，clone()将会抛出一个检查异常()——java.lang.CloneNotSupportedException,如果实现了这个接口，clone()会创建一个新的对象，并将原来对象的内容复制到新对象，最后返回这个新对象的引用。

理解浅克隆和深克隆，主要是针对引用类型的成员变量。

1. euqals()方法

Object类默认的eqauls()函数进行比较的依据是：调用它的对象和传入的对象的引用是否相等。也就是说，默认的equals()进行的是引用比较。如果两个引用是相同的，equals()函数返回true；否则，返回false. 我觉得和==一样。

1. finalize()方法

finalize()方法可以被子类对象所覆盖，然后作为一个终结者，当GC被调用的时候完成最后的清理工作（例如释放系统资源之类）。这就是终止。默认的finalize()方法什么也不做，当被调用时直接返回。对于任何一个对象，它的finalize()方法都不会被JVM执行两次。如果你想让一个对象能够被再次调用的话（例如，分配它的引用给一个静态变量），注意当这个对象已经被GC回收的时候，finalize()方法不会被调用第二次。

1. getClass()

通过getClass()方法可以得到一个和这个类有关的java.lang.Class对象。返回的Class对象是一个被static synchronized方法封装的代表这个类的对象

1. toString()方法

当 toString() 没有被覆盖的时候，返回的字符串格式是 类名@哈希值，哈希值是十六进制的。

1. Hashcode()方法

hashCode()方法返回给调用者此对象的哈希码（其值由一个hash函数计算得来）。这个方法通常用在基于hash的集合类中，像java.util.HashMap,java.until.HashSet和java.util.Hashtable.

1、hashCode的存在主要是用于查找的快捷性，如Hashtable，HashMap等，hashCode是用来在散列存储结构中确定对象的存储地址的；

2、如果两个对象相同，就是适用于equals(**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/fenglibing/article/details/8905007/_blank)**.lang.Object) 方法，那么这两个对象的hashCode一定要相同；

3、如果对象的equals方法被重写，那么对象的hashCode也尽量重写，并且产生hashCode使用的对象，一定要和equals方法中使用的一致，否则就会违反上面提到的第2点；

4、两个对象的hashCode相同，并不一定表示两个对象就相同，也就是不一定适用于equals(java.lang.Object) 方法，只能够说明这两个对象在散列存储结构中，如Hashtable，他们**“存放在同一个篮子里”**。

十七．方法重写应遵循“三同一小一大”原则：

“三同”：即方法名相同，形参列表相同，返回值类型相同；

“一小”：子类方法声明抛出的异常比父类方法声明抛出的异常更小或者相等；

“一大”:子类方法的访问修饰符应比父类方法更大或相等。

## 各种排序方式的复杂度



## 实例方法和类方法

1、类方法是属于整个类，而不属于某个对象。

2、类方法只能访问类成员变量，不能访问实例变量，而实例方法可以访问类成员变量和实例变量。

3、 类方法的调用可以通过类名.类方法和对象.类方法，而实例方法只能通过对象.实例方法访问。

4、类方法只能访问类方法，而实例方法可以访问类方法和实例方法。

5、类方法不能被覆盖，实例方法可以被覆盖。

## 二十、死锁

产生死锁的原因主要是：  
（1） 因为系统资源不足。  
（2） 进程运行推进的顺序不合适。  
（3） 资源分配不当等。  
如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则  
就会因争夺有限的资源而陷入死锁。其次，进程运行推进顺序与速度不同，也可能产生死锁。

产生死锁的四个必要条件：  
（1） 互斥条件：一个资源每次只能被一个进程使用。  
（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。  
（3） 不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。  
（4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

二十一、网桥和交换机的区别

**网桥（Bridge)**也称为桥接器，是连接两个局域网的存储转发设备，用它可以使完全具有相同或相似体系结构网络系统的连接，这样不但能扩展网络的距离或范围，而且可提高网络的性能、可靠性和安全性。网桥工作在OSI参考模型的数据链路层（第二层），将两个LAN连起来，根据MAC地址来转发帧，可以看作一个“低层的路由器”（路由器工作在网络层，根据网络地址如IP地址进行转发）。

**交换机**是主导网络系统的集线设备，大部分交换机是在OSI参考模型的数据链路层（第二层）操作。

**交换机与网桥的区别**  
  
局域网交换机的基本功能与网桥一样，具有帧转发、帧过滤和生成树**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/fivedoumi/article/details/_blank)**功能。但是，交换机与网桥相比还是存在以下不同：

1、交换机工作时，实际上允许许多组端口间的通道同时工作。所以，交换机的功能体现出不仅仅是一个网桥的功能，而是多个网桥功能的集合。即网桥一般分有两个端口，而交换机具有高密度的端口。   
  
2、分段能力的区别   
  
由于交换机能够支持多个端口，因此可以把网络系统划分成为更多的物理网段，这样使得整个网络系统具有更高的带宽。而网桥仅仅支持两个端口，所以，网桥划分的物理网段是相当有限的。   
  
3、传输速率的区别   
  
交换机与网桥数据信息的传输速率相比，交换机要快于网桥。   
  
4、数据帧转发方式的区别

网桥在发送数据帧前，通常要接收到完整的数据帧并执行帧检测序列FCS后，才开始转发该数据帧。交换机具有存储转发和直接转发两种帧转发方式。直接转发方式在发送数据以前，不需要在接收完整个数据帧和经过32bit循环冗余校验码CRC的计算检查后的等待时间。

二十二、二叉树

二叉树中度为２的结点数目＝叶子结点数目-1

二十三、String的方法

1. length()字符串的长度。

2. chartAt()截取一个字符。

　例：char ch;  
　　　ch="abc".charAt(1); 返回'b'

3. getCharts()截取多个字符

char buf[]=new char[20];  
　　 s.getChars(10,14,buf,0);

4. getBytes()

　 替代getChars()的一种方法是将字符存储在字节数组中，该方法即getBytes()。

5. toChartArray()

6. indexOf和lastIndexOf

indexOf() 查找字符或者子串第一次出现的地方。  
　 lastIndexOf() 查找字符或者子串是后一次出现的地方。

7. Replace()

两种形式，第一种形式用一个字符在调用字符串中所有出现某个字符的地方进行替换，形式如下：  
　　String replace(char original,char replacement)  
　　例如：String s="Hello".replace('l','w');

　 第二种形式是用一个字符序列替换另一个字符序列，形式如下：  
　　String replace(CharSequence original,CharSequence replacement)

8. toLowerCase() 转换为小写

9. toUpperCase() 转换为大写

## 二十四、数据库

DML（data manipulation language）：   
       它们是SELECT、UPDATE、INSERT、DELETE，就象它的名字一样，这4条命令是用来对 [数据库](http://lib.csdn.net/base/mysql" \t "https://www.nowcoder.com/test/question/_blank) 里的数据进行操作的语言   
DDL（data definition language）：   
       DDL比DML要多，主要的命令有CREATE、ALTER、DROP等，DDL主要是用在定义或改变表（TABLE）的结构，数据类型，表之间的链接和约束等初始化工作上，他们大多在建立表时使用   
DCL（Data Control Language）：   
       是数据库控制功能。是用来设置或更改数据库用户或角色权限的语句，包括（grant,deny,revoke等）语句。在默认状态下，只有sysadmin,dbcreator,db\_owner或db\_securityadmin等人员才有权力执行DCL

二十五、进程控制块

存放进程的管理和控制信息的数据结构称为进程控制块。它是进程管理和控制的最重要的数据结构，每一个进程均有一个PCB，在创建进程时，建立PCB，伴随进程运行的全过程，直到进程撤消而撤消。

在不同的操作系统中对进程的控制和管理机制不同，PCB中的信息多少也不一样，通常PCB应包含如下一些信息。  
  
1、进程标识符 name：  
每个进程都必须有一个唯一的标识符，可以是字符串，也可以是一个数字。UNIX系统中就是一个整型数。在进程创建时由系统赋予。  
  
2、进程当前状态 status：  
说明进程当前所处的状态。为了管理的方便，系统设计时会将相同的状态的进程组成一个队列，如就绪进程队列，等待进程则要根据等待的事件组成多个等待队列，如等待打印机队列、等待磁盘I/O完成队列等等。  
  
3、进程相应的程序和数据地址，以便把PCB与其程序和数据联系起来。  
  
4、进程资源清单。列出所拥有的除CPU外的资源记录，如拥有的I/O设备，打开的文件列表等。  
  
5、进程优先级 priority：  
进程的优先级反映进程的紧迫程序，通常由用户指定和系统设置。UNIX系统采用用户设置和系统计算相结合的方式确定进程的优先级 。  
  
6、CPU现场保护区 cpustatus：  
当进程因某种原因不能继续占用CPU时（等待打印机），释放CPU，这时就要将CPU的各种状态信息保护起来，为将来再次得到处理机恢复CPU的各种状态，继续运行。  
  
7、进程同步与通信机制 用于实现进程间互斥、同步和通信所需的信号量等。

8、进程所在队列PCB的链接字     
根据进程所处的现行状态，进程相应的PCB参加到不同队列中。PCB链接字指出该进程所在队列中下一个进程PCB的首地址。

1. 与进程有关的其他信息。   
   如进程记账信息，进程占用CPU的时间等。

僵尸进程：一个子进程在其父进程还没有调用wait()或waitpid()的情况下退出。这个子进程就是僵尸进程。

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。

二十六、http协议

https与http区别在于前者的传输采用了加密方式，web服务器申请了证书并与服务器绑定，传输更安全

二十七.命令

- kill    无参数表示向进程输入关闭的指令（一般关闭后台进程）

- ^C    关闭当前（前台）进程

- ^Z    暂停当前（前台）进程

- shutdown    关机

- halt    就是调用shutdown -h

二十八、静态的方法属性初始化，是在加载类的时候初始化。而非静态方法属性初始化，是new类实例对象的时候加载。实例化对象的时候先初始化非静态参数然后调用构造方法。

当我们编写一个新的java类时,JVM就会帮我们编译成class对象,存放在同名的.class文件中。在运行时，当需要生成这个类的对象，JVM就会检查此类是否已经装载内存中。若是没有装载，则把.class文件装入到内存中。若是装载，则根据class文件生成实例对象。

Class对象的生成方式如下：

1.类名.class           说明： JVM将使用类装载器, 将类装入内存(前提是:类还没有装入内存),不做类的初始化工作.返回Class的对象

2.Class.forName("类名字符串")  （注：类名字符串是包名+类名）  说明：装入类,并做类的静态初始化，返回Class的对象

3.实例对象.getClass()  说明：对类进行静态初始化、非静态初始化；返回引用o运行时真正所指的对象(因为:子对象的引用可能会赋给父对象的引用变量中)所属的类的Class的对象

二十九、两段锁协议（Two-Phase Locking――2PL）  
　　两段锁协议规定所有的事务应遵守的规则：  
　　① 在对任何数据进行读、写操作之前，首先要申请并获得对该数据的封锁。  
　　② 在释放一个封锁之后，事务不再申请和获得其它任何封锁。  
　　即事务的执行分为两个阶段：  
　　第一阶段是获得封锁的阶段，称为扩展阶段。  
　　第二阶段是释放封锁的阶段，称为收缩阶段。

三十、广度优先遍历用队列

深度优先遍历用栈

三十一、事物的隔离级别

**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/qq_33290787/article/details/_blank)**事务的隔离级别有4种，由低到高分别为Read uncommitted 、Read committed 、Repeatable read 、Serializable

**Read uncommitted**

读未提交，顾名思义，就是一个事务可以读取另一个未提交事务的数据。

事例：老板要给程序员发工资，程序员的工资是3.6万/月。但是发工资时老板不小心按错了数字，按成3.9万/月，该钱已经打到程序员的户口，但是事务还没有提交，就在这时，程序员去查看自己这个月的工资，发现比往常多了3千元，以为涨工资了非常高兴。但是老板及时发现了不对，马上回滚差点就提交了的事务，将数字改成3.6万再提交。

分析：实际程序员这个月的工资还是3.6万，但是程序员看到的是3.9万。他看到的是老板还没提交事务时的数据。这就是脏读。

**Read committed**

读提交，顾名思义，就是一个事务要等另一个事务提交后才能读取数据。

事例：程序员拿着信用卡去享受生活（卡里当然是只有3.6万），当他埋单时（程序员事务开启），收费系统事先检测到他的卡里有3.6万，就在这个时候！！程序员的妻子要把钱全部转出充当家用，并提交。当收费系统准备扣款时，再检测卡里的金额，发现已经没钱了（第二次检测金额当然要等待妻子转出金额事务提交完）。程序员就会很郁闷，明明卡里是有钱的…

分析：这就是读提交，若有事务对数据进行更新（UPDATE）操作时，读操作事务要等待这个更新操作事务提交后才能读取数据，可以解决脏读问题。但在这个事例中，出现了一个事务范围内两个相同的查询却返回了不同数据，这就是不可重复读。

**Repeatable read**

重复读，就是在开始读取数据（事务开启）时，不再允许修改操作

事例：程序员拿着信用卡去享受生活（卡里当然是只有3.6万），当他埋单时（事务开启，不允许其他事务的UPDATE修改操作），收费系统事先检测到他的卡里有3.6万。这个时候他的妻子不能转出金额了。接下来收费系统就可以扣款了。

分析：重复读可以解决不可重复读问题。写到这里，应该明白的一点就是，不可重复读对应的是修改，即UPDATE操作。但是可能还会有幻读问题。因为幻读问题对应的是插入INSERT操作，而不是UPDATE操作。

**Serializable 序列化**

Serializable 是最高的事务隔离级别，在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。

三十二、数据完整性分为以下四类：  
1) 实体完整性：规定表的每一行在表中是惟一的实体。  
2) 域完整性：是指表中的列必须满足某种特定的数据类型约束，其中约束又包括取值范围、精度等规定。  
3) 参照完整性：是指两个表的主关键字和外关键字的数据应一致，保证了表之间的数据的一致性，防止了数据丢失或无意义的数据在数据库中扩散。  
4) 用户定义的完整性：不同的关系数据库系统根据其应用环境的不同，往往还需要一些特殊的约束条件。用户定义的完整性即是针对某个特定关系数据库的约束条件，它反映某一具体应用必须满足的语义要求。

## 三十三、设计模式

总体来说设计模式分为三大类：

创建型模式，共五种：工厂方法模式、抽象工厂模式、单例模式、建造者模式、原型模式。

结构型模式，共七种：适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式。

行为型模式，共十一种：策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式。

其实还有两类：并发型模式和线程池模式。

1. 工厂方法模式

**11、普通工厂模式**，就是建立一个工厂类，对实现了同一接口的一些类进行实例的创建。

**22、多个工厂方法模式**，是对普通工厂方法模式的改进，在普通工厂方法模式中，如果传递的字符串出错，则不能正确创建对象，而多个工厂方法模式是提供多个工厂方法，分别创建对象。

**33、静态工厂方法模式**，将上面的多个工厂方法模式里的方法置为静态的，不需要创建实例，直接调用即可。

总体来说，工厂模式适合：凡是出现了大量的产品需要创建，并且具有共同的接口时，可以通过工厂方法模式进行创建。在以上的三种模式中，第一种如果传入的字符串有误，不能正确创建对象，第三种相对于第二种，不需要实例化工厂类，所以，大多数情况下，我们会选用第三种——静态工厂方法模式。

1. 抽象工厂模式

工厂方法模式有一个问题就是，类的创建依赖工厂类，也就是说，如果想要拓展程序，必须对工厂类进行修改，这违背了闭包原则，所以，从设计角度考虑，有一定的问题，如何解决？就用到抽象工厂模式，创建多个工厂类，这样一旦需要增加新的功能，直接增加新的工厂类就可以了，不需要修改之前的代码。

1. 单例模式（Singleton）

单例对象（Singleton）是一种常用的设计模式。在Java应用中，单例对象能保证在一个JVM中，该对象只有一个实例存在。这样的模式有几个好处：

1、某些类创建比较频繁，对于一些大型的对象，这是一笔很大的系统开销。

2、省去了new操作符，降低了系统内存的使用频率，减轻GC压力。

3、有些类如交易所的核心交易引擎，控制着交易流程，如果该类可以创建多个的话，系统完全乱了。（比如一个军队出现了多个司令员同时指挥，肯定会乱成一团），所以只有使用单例模式，才能保证核心交易服务器独立控制整个流程。

一种单例模式，在多线程的时候会有问题

1. public class Singleton {
3. /\* 持有私有静态实例，防止被引用，此处赋值为null，目的是实现延迟加载 \*/
4. private static Singleton instance = null;
6. /\* 私有构造方法，防止被实例化 \*/
7. private Singleton() {
8. }
10. /\* 静态工程方法，创建实例 \*/

11    public static Singleton getInstance() {

12         if (instance == null) {

13             synchronized (instance) {

14                if (instance == null) {

15                    instance = new Singleton();

16                }

17            }

18        }

2. /\* 如果该对象被用于序列化，可以保证对象在序列化前后保持一致 \*/
3. public Object readResolve() {
4. return instance;
5. }
6. }

以下是一种比较好的单例模式。单例模式使用内部类来维护单例的实现，JVM内部的机制能够保证当一个类被加载的时候，这个类的加载过程是线程互斥的。这样当我们第一次调用getInstance的时候，JVM能够帮我们保证instance只被创建一次，并且会保证把赋值给instance的内存初始化完毕，这样我们就不用担心上面的问题。同时该方法也只会在第一次调用的时候使用互斥机制，这样就解决了低性能问题。这样我们暂时总结一个完美的单例模式。

1. public class Singleton {
3. /\* 私有构造方法，防止被实例化 \*/
4. private Singleton() {
5. }
7. /\* 此处使用一个内部类来维护单例 \*/
8. private static class SingletonFactory {
9. private static Singleton instance = new Singleton();
10. }
12. /\* 获取实例 \*/
13. public static Singleton getInstance() {
14. return SingletonFactory.instance;
15. }
17. /\* 如果该对象被用于序列化，可以保证对象在序列化前后保持一致 \*/
18. public Object readResolve() {
19. return getInstance();
20. }
21. }
22. **建造者模式（Builder）**

工厂类模式提供的是创建单个类的模式，而建造者模式则是将各种产品集中起来进行管理，用来创建复合对象，所谓复合对象就是指某个类具有不同的属性，其实建造者模式就是前面抽象工厂模式和最后的Test结合起来得到的。

1. public class Builder {
3. private List<Sender> list = new ArrayList<Sender>();
5. public void produceMailSender(int count){
6. for(int i=0; i<count; i++){
7. list.add(new MailSender());
8. }
9. }
11. public void produceSmsSender(int count){
12. for(int i=0; i<count; i++){
13. list.add(new SmsSender());
14. }
15. }
16. }

从这点看出，建造者模式将很多功能集成到一个类里，这个类可以创造出比较复杂的东西。所以与工程模式的区别就是：工厂模式关注的是创建单个产品，而建造者模式则关注创建符合对象，多个部分

1. **原型模式（Prototype）**

原型模式虽然是创建型的模式，但是与工程模式没有关系，从名字即可看出，该模式的思想就是将一个对象作为原型，对其进行复制、克隆，产生一个和原对象类似的新对象。

1. **适配器模式**

总结一下三种适配器模式的应用场景：

类的适配器模式：当希望将**一个类**转换成满足**另一个新接口**的类时，可以使用类的适配器模式，创建一个新类，继承原有的类，实现新的接口即可。

对象的适配器模式：当希望将一个对象转换成满足另一个新接口的对象时，可以创建一个Wrapper类，持有原类的一个实例，在Wrapper类的方法中，调用实例的方法就行。

接口的适配器模式：当不希望实现一个接口中所有的方法时，可以创建一个抽象类Wrapper，实现所有方法，我们写别的类的时候，继承抽象类即可。

1. **装饰模式（Decorator）**

Source类是被装饰类，Decorator类是一个装饰类，可以为Source类动态的添加一些功能，代码如下：

**[java]** [view plain](http://blog.csdn.net/zhangerqing/article/details/8239539" \o "view plain)[copy](http://blog.csdn.net/zhangerqing/article/details/8239539" \o "copy)

1. public interface Sourceable {
2. public void method();
3. }

**[java]** [view plain](http://blog.csdn.net/zhangerqing/article/details/8239539" \o "view plain)[copy](http://blog.csdn.net/zhangerqing/article/details/8239539" \o "copy)

1. public class Source implements Sourceable {
3. @Override
4. public void method() {
5. System.out.println("the original method!");
6. }
7. }

**[java]** [view plain](http://blog.csdn.net/zhangerqing/article/details/8239539" \o "view plain)[copy](http://blog.csdn.net/zhangerqing/article/details/8239539" \o "copy)

1. public class Decorator implements Sourceable {
3. private Sourceable source;
5. public Decorator(Sourceable source){
6. super();
7. this.source = source;
8. }
9. @Override
10. public void method() {
11. System.out.println("before decorator!");
12. source.method();
13. System.out.println("after decorator!");
14. }
15. }

装饰器模式的应用场景：

1、需要扩展一个类的功能。

2、动态的为一个对象增加功能，而且还能动态撤销。（继承不能做到这一点，继承的功能是静态的，不能动态增删。）

在装饰器模式和代理模式之间还是有很多差别的。装饰器模式关注于在一个对象上动态的添加方法，然而代理模式关注于控制对对象的访问。换句话 说，用代理模式，代理类（proxy class）可以对它的客户隐藏一个对象的具体信息。因此，当使用代理模式的时候，我们常常在一个代理类中创建一个对象的实例。并且，当我们使用装饰器模 式的时候，我们通常的做法是将原始对象作为一个参数传给装饰者的构造器。

   我们可以用另外一句话来总结这些差别：使用代理模式，代理和真实对象之间的的关系通常在编译时就已经确定了，而装饰者能够在运行时递归地被构造。

## 给初学者的设计模式面试问题

这些软件设计和设计模式问题大多在初学者层次时被问起，目的只是了解一下候选人(应聘者)对设计模式知道多少，如， ****设计模式是什么**** 或者 ****一个特定的设计模式做什么**** ?这些问题通过简单地记忆概念就可以回答，但就信息和知识而言还是有价值的。

****1. 什么是设计模式？你是否在你的代码里面使用过任何设计模式？****

设计模式是世界上各种各样程序员用来解决特定设计问题的尝试和测试的方法。设计模式是代码可用性的延伸。

****2. 你可以说出几个在JDK库中使用的设计模式吗？****

[装饰器设计模式（Decorator design pattern）](http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html" \t "http://www.importnew.com/_blank)被用于多个Java IO类中。单例模式（Singleton pattern）用于Runtime，Calendar和其他的一些类中。工厂模式（Factory pattern）被用于各种不可变的类如Boolean，像Boolean.valueOf，观察者模式（Observer pattern）被用于Swing和很多的事件监听中。

****3. Java中什么是单例设计模式？用Java写出线程安全的单例****

单例模式重点在于在整个系统上共享一些创建时较耗资源的对象。整个应用中只维护一个特定类实例，它被所有组件共同使用。Java.lang.Runtime是单例模式的经典例子。你可以在我的文章[Java单例模式的10个问题](http://javarevisited.blogspot.com/2011/03/10-interview-questions-on-singleton.html" \t "http://www.importnew.com/_blank)看到更多的问题和讨论。从Java 5开始你可以使用[枚举（enum）](http://javarevisited.blogspot.com/2011/08/enum-in-java-example-tutorial.html" \t "http://www.importnew.com/_blank)来实现线程安全的单例。

****4. 使用工厂模式最主要的好处是什么？你在哪里使用？****

工厂模式的最大好处是增加了创建对象时的封装层次。如果 你使用工厂来创建对象，之后你可以使用更高级和更高性能的实现来替换原始的产品实现或类，这不需要在调用层做任何修改。可以看我的文章[工厂模式](http://javarevisited.blogspot.com/2011/12/factory-design-pattern-java-example.html" \t "http://www.importnew.com/_blank)得更详细的解释和和了解更多的好处。

****5. 在Java中，什么叫观察者设计模式（observer design pattern）****？

观察者模式是基于对象的状态变化和观察者的通讯，以便他们作出相应的操作。简单的例子就是一个天气系统，当天气变化时必须在展示给公众的视图中进行反映。这个视图对象是一个主体，而不同的视图是观察者。可以在[这篇文章](http://javarevisited.blogspot.sg/2011/12/observer-design-pattern-java-example.html" \t "http://www.importnew.com/_blank)中看到Java观察者模式的完整例子。

****6. 举一个用Java实现的装饰模式(decorator design pattern)？它是作用于对象层次还是类层次？****

装饰模式增加强了单个对象的能力。Java IO到处都使用了装饰模式，经典的例子就是Buffered系列类如BufferedReader和BufferedWriter，它们增强了Reader和Writer对象，以实现提升性能的Buffer层次的读取和写入。可以看[这篇文章](http://javarevisited.blogspot.com/2011/11/decorator-design-pattern-java-example.html" \t "http://www.importnew.com/_blank)了解更多。

****7. 什么是MVC设计模式？举一个MVC设计模式的例子？****

****8, Java中什么是表示层设计模式（FrontController design pattern）？举一个使用表示层设计模式（front controller pattern）的例子？****

****9. 什么是责任链模式（Chain of Responsibility）？****

****10. 什么是适配器模式？举用Java实现适配器模式的例子？****

适配器模式（Adapter）：将一个类的接口转换成客户希望的另外一个接口，使得原本接口不兼容而不能一起工作的那些类可以一起工作。

**三十四、**在C中没有逻辑型数据，在逻辑运算结果中以“1”代表“真”，以“0”代表“假”；在逻辑判断中，以“非0”代表“真”，以“0”代表“假”。在C++中有逻辑型数据，以“true”代表“真”，以“false”代表“假”。

**三十五、**tar是操作.tar的命令   
 gzip是压缩.gz压缩包的命令   
 compress：压缩.Z文件

uncompress：解压缩.Z文件

**三十六、**抢占式与非抢占式的对比：     
非抢占式（Nonpreemptive）   让进程运行直到结束或阻塞的调度方式（容易实现，适合专用系统，不适合通用系统）   
抢占式（Preemptive）   允许将逻辑上可继续运行的在运行过程暂停的调度方式，可防止单一进程长时间独占CPU（系统开销大）

可抢占式调度是严格保证任何时刻，让具有最高优先数（权）的进程占有处理机运行，因此增加了处理机调度的时机，引起为退出处理机的进程保留现场，为占有处理机的进程恢复现场等时间（和空间）开销增大。

三十七、首次适应算法（First Fit）：

从空闲[分区表](http://baike.baidu.com/view/1412956.htm" \t "https://www.nowcoder.com/test/question/_blank)的第一个表目起查找该表，把最先能够满足要求的空闲区分配给作业，这种方法目的在于减少查找时间。为适应这种算法，空闲[分区表](http://baike.baidu.com/view/1412956.htm" \t "https://www.nowcoder.com/test/question/_blank)（空闲区链）中的空闲分区要按地址由低到高进行排序。该算法优先使用低址部分空闲区，在低址空间造成许多小的空闲区，在高[地址空间](http://baike.baidu.com/view/1507129.htm" \t "https://www.nowcoder.com/test/question/_blank)保留大的空闲区。

三十八、

主码：我们在建立数据库的时候，需要为每张表指定一个主码，主码也叫主键。

候选码：若关系中的某一属性组的值能唯一的标识一个元组,而其任何真子集都 不能再标识,则称该属性组为候选码。

主属性：包含在任一候选关键字中的属性称主属性。

超键(super key):在关系中能唯一标识元组的属性集称为关系模式的超键

候选键(candidate key):不含有多余属性的超键称为候选键

主键(primary key):用户选作元组标识的一个候选键程序主键

## 三十九、外部碎片和内部碎片

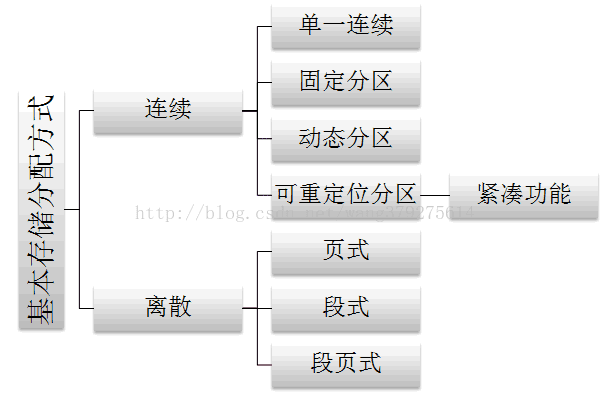
在内存管理中，内部碎片是已经被分配出去的的内存空间大于请求所需的内存空间。

外部碎片是指还没有分配出去，但是由于大小太小而无法分配给申请空间的新进程的内存空间空闲块。

固定分区存在内部碎片，可变式分区分配会存在外部碎片；

页式虚拟存储系统存在内部碎片；段式虚拟存储系统，存在外部碎片

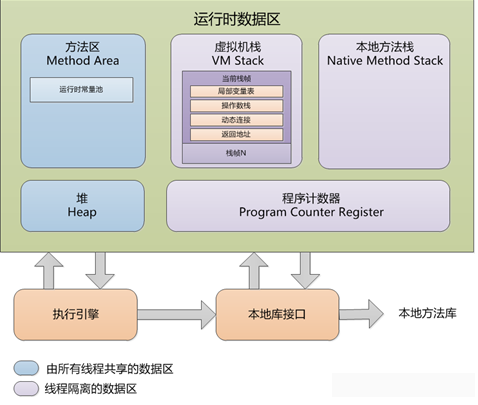
四十、基本的存储分配方式



# 四十一、**[Java虚拟机 运行时数据区](http://www.cnblogs.com/Cratical/archive/2012/08/21/2649985.html)**

Java在执行Java程序的过程中会把它所管理的内存划分为若干个不同的数据区域。这些区域都有各自的用途、创建和销毁的时间，有一些是随虚拟机的启动而创建，随虚拟机的退出而销毁，有些则是与线程一一对应，随线程的开始和结束而创建和销毁。

Java虚拟机所管理的内存将会包括以下几个运行时数据区域



**程序计数器（Program Counter Register）**

它是一块较小的内存空间，它的作用可以看做是当先线程所执行的字节码的信号指示器。

每一条JVM线程都有自己的PC寄存器，各条线程之间互不影响，独立存储，这类内存区域被称为“线程私有”内存

在任意时刻，一条JVM线程只会执行一个方法的代码。该方法称为该线程的当前方法（Current Method）

如果该方法是java方法，那PC寄存器保存JVM正在执行的字节码指令的地址

如果该方法是native，那PC寄存器的值是undefined。

此内存区域是唯一一个在Java虚拟机规范中没有规定任何OutOfMemoryError情况的区域。

**Java虚拟机栈（Java Virtual Machine Stack）**

与PC寄存器一样，Java虚拟机栈也是线程私有的。每一个JVM线程都有自己的java虚拟机栈，这个栈与线程同时创建，它的生命周期与线程相同。

虚拟机栈描述的是Java方法执行的内存模型：每个方法被执行的时候都会同时创建一个栈帧（Stack Frame）用于存储局部变量表、操作数栈、动态链接、方法出口等信息。每一个方法被调用直至执行完成的过程就对应着一个栈帧在虚拟机栈中从入栈到出栈的过程。

**本地方法栈（Native Method Stack）**

本地方法栈与虚拟机栈作用相似，后者为虚拟机执行Java方法服务，而前者为虚拟机用到的Native方法服务。

虚拟机规范对于本地方法栈中方法使用的语言，使用方式和数据结构没有强制规定，甚至有的虚拟机（比如HotSpot）直接把二者合二为一。

这玩意儿抛出的异常跟上面的虚拟机栈一样。

**Java堆（Java Heap）**

虚拟机管理的内存中最大的一块，同时也是被**所有线程所共享**的，它在虚拟机启动时创建，这货存在的意义就是存放对象实例，几乎所有的对象实例以及数组都要在这里分配内存。这里面的对象被自动管理，也就是俗称的GC（Garbage Collector）所管理。用就是了，有GC扛着呢，不用操心销毁回收的事儿。

**方法区（Method Area）**

跟堆一样是被**各个线程共享的内存区域**，用于存储以被虚拟机加载的类信息、常量、静态变量、即时编译器编译后的代码等数据。虽然这个区域被虚拟机规范把方法区描述为堆的一个逻辑部分，但是它的别名叫非堆，用来与堆做一下区别。

这里有一个小例子，来说明堆，栈和方法区之间的关系的

public class Test2 {

public static void main(String[] args) {

public Test2 t2 = new Test2();

//JVM将Test2类信息加载到方法区,new Test2()实例保存在堆区,Test2引用保存在栈区

}

}

## 四十二、死锁

1：****饥饿****是指一个可运行的进程尽管能继续执行，但被调度程序无限期地忽略，而不能被调度执行的情形。

2：****死锁****是两个或两个以上的进程其中每个进程都在等待其它进程做完某些事而不能继续执行。

3：****互斥****是当一个进程在临界区访问共享资源时，其它进程不能进入该临界区访问任何共享资源。

****顺便复习下死锁产生的四个必要条件：****

* 互斥条件：一个资源每次只能被一个进程使用
* 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放
* 不剥夺条件：进程已获得的资源在未使用完之前，不能强行剥夺
* 循环等待条件：若干进程之间形成头尾相接的循环等待资源关系

****相应的预防措施：****

* 采用静态资源分配策略，破坏部分分配条件
* 允许进程剥夺其它进程的资源，破坏不可剥夺的条件
* 采用资源有序分配，破坏环路条件
* ****注意：互斥条件是无法被破坏的****

****四十三、数据库索引查询****

一、什么是索引：

简单的来说，建立索引在进行数据库操作的时候不需要全盘一条条的扫描，删选出符合的记录，索引内部自己有一套优化算法，因此借助索引来对数据库进行操作可以提高查询的效率。

二、什么时候建立的索引将失效或效率不高（情况有很多，这里列举常见的几种， 假设在字段name上建立了索引）：

1、使用了运算符!=，以及关键字not in, not exist等，认为产生的结果集很大，往往导致引擎不走索引而是走全盘扫描

2、对索引字段使用了函数，如where substr(name, 1, 3)=‘mark’， 导致索引无效

3、使用like和通配符，第一个字符是%将导致索引失效，如where name like "%ark“

三、order by与索引

首先利用where进行数据查询，这一步是免不了的，至于这一步有没有利用索引暂时不考虑，关键是在获取所有符合的记录后还需要进行排序，看看order by是如何利用索引的。

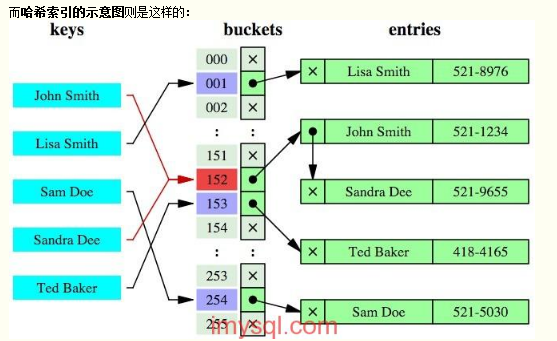
如果order by中的字段有建立索引，同时：

1. 该字段没有出现在where中，则在排序的时候需要正常排序，默认order by是升序排序, 故索引没有对排序产生有利帮助

2、该字段同时同时出现在where中，则在获取记录后不进行排序，而是直接利用索引， 效率变高。

--order by的字段混合ASC和DESC,不使用索引

## 五、哈希索引和B树索引的区别



从上面的图来看，B+树索引和哈希索引的明显区别是：

**1、如果是等值查询，那么哈希索引明显有绝对优势**，因为只需要经过一次算法即可找到相应的键值；当然了，这个前提是，键值都是唯一的。如果键值不是唯一的，就需要先找到该键所在位置，然后再根据链表往后扫描，直到找到相应的数据；

2、从示意图中也能看到，**如果是范围查询检索，这时候哈希索引就毫无用武之地了**，因为原先是有序的键值，经过哈希算法后，有可能变成不连续的了，就没办法再利用索引完成范围查询检索；

3、同理，**哈希索引也没办法利用索引完成排序**，以及like ‘xxx%’ 这样的部分模糊查询（这种部分模糊查询，其实本质上也是范围查询）；

**4、哈希索引也不支持多列联合索引的最左匹配规则**；

5、B+树索引的关键字检索效率比较平均，不像B树那样波动幅度大，**在有大量重复键值情况下，哈希索引的效率也是极低的，因为存在所谓的哈希碰撞问题**

## 四十四、反射机制

## **1反射机制是什么**

1.反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

## **3反射机制的相关API**

 TestReflect testReflect = new TestReflect();

 System.out.println(testReflect.getClass().getName());

 // 一般采用这种形式

 class1 = Class.forName("net.xsoftlab.baike.TestReflect");

 class2 = new TestReflect().getClass();

 class3 = TestReflect.class;

 System.out.println("类名称   " + class1.getName());

 System.out.println("类名称   " + class2.getName());

 System.out.println("类名称   " + class3.getName());

//

  Method method = clazz.getMethod("reflect1");

method.invoke(clazz.newInstance());

四十五、虚拟机

## 四十六、面向对象的特征

三大特性是：封装,继承,多态

所谓封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。封装是面向对象的特征之一，是对象和类概念的主要特性。 简单的说，一个类就是一个封装了数据以及操作这些数据的代码的逻辑实体。在一个对象内部，某些代码或某些数据可以是私有的，不能被外界访问。通过这种方式，对象对内部数据提供了不同级别的保护，以防止程序中无关的部分意外的改变或错误的使用了对象的私有部分。

所谓继承是指可以让某个类型的对象获得另一个类型的对象的属性的方法。它支持按级分类的概念。继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。 通过继承创建的新类称为“子类”或“派生类”，被继承的类称为“基类”、“父类”或“超类”。继承的过程，就是从一般到特殊的过程。要实现继承，可以通过“继承”（Inheritance）和“组合”（Composition）来实现。继承概念的实现方式有二类：实现继承与接口继承。实现继承是指直接使用基类的属性和方法而无需额外编码的能力；接口继承是指仅使用属性和方法的名称、但是子类必须提供实现的能力；

所谓多态就是指一个类实例的相同方法在不同情形有不同表现形式。多态机制使具有不同内部结构的对象可以共享相同的外部接口。这意味着，虽然针对不同对象的具体操作不同，但通过一个公共的类，它们（那些操作）可以通过相同的方式予以调用。

## 四十七、接口和抽象类

1. 接口中只能够有静态的 不能被修改的数据成员（也就是必须是static final的，不过在interface中一般不定义数据成员），所有的成员方法都是abstract的。抽象类中可以有自己的数据成员，也可以有非abstarct的成员方法
2. 抽象类和接口都不能直接实例化，如果要实例化，抽象类变量必须指向实现所有抽象方法的子类对象，接口变量必须指向实现所有接口方法的类对象。
3. 接口只能做方法申明，抽象类中可以做方法申明，也可以做方法实现
4. 抽象类里的抽象方法必须全部被子类所实现，如果子类不能全部实现父类抽象方法，那么该子类只能是抽象类。同样，一个实现接口的时候，如不能全部实现接口方法，那么该类也只能为抽象类。
5. 抽象方法只能申明，不能实现。abstract void abc();不能写成abstract void abc(){}。
6. 抽象方法要被实现，所以不能是静态的，也不能是私有的。

# 四十八、数据库事务的四个特性及含义

**原子性**:整个事务中的所有操作，要么全部完成，要么全部不完成，不可能停滞在中间某个环节。事务在执行过程中发生错误，会被回滚（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。  
 **一致性**:在事务开始之前和事务结束以后，数据库的完整性约束没有被破坏。  
 **隔离性**:隔离状态执行事务，使它们好像是[系统](http://www.2cto.com/os/" \t "http://www.2cto.com/database/201504/_blank)在给定时间内执行的唯一操作。如果有两个事务，运行在相同的时间内，执行 相同的功能，事务的隔离性将确保每一事务在系统中认为只有该事务在使用系统。这种属性有时称为串行化，为了防止事务操作间的混淆，必须串行化或序列化请 求，使得在同一时间仅有一个请求用于同一数据。  
 **持久性**:在事务完成以后，该事务所对数据库所作的更改便持久的保存在数据库之中，并不会被回滚。

# 四十九、****drop,delete与truncate的区别****

drop直接删掉表 truncate删除表中数据，再插入时自增长id又从1开始 delete删除表中数据，可以加where字句。

（1） DELETE语句执行删除的过程是每次从表中删除一行，并且同时将该行的删除操作作为事务记录在日志中保存以便进行进行回滚操作。TRUNCATE TABLE 则一次性地从表中删除所有的数据并不把单独的删除操作记录记入日志保存，删除行是不能恢复的。并且在删除的过程中不会激活与表有关的删除触发器。执行速度快。

（2） 表和索引所占空间。当表被TRUNCATE 后，这个表和索引所占用的空间会恢复到初始大小，而DELETE操作不会减少表或索引所占用的空间。drop语句将表所占用的空间全释放掉。

（3） 应用范围。TRUNCATE 只能对TABLE；DELETE可以是table和view

（4） Truncate table 表名 速度快,而且效率高,TRUNCATE TABLE 删除表中的所有行，但表结构及其列、约束、索引等保持不变。新行标识所用的计数值重置为该列的种子。如果想保留标识计数值，请改用 DELETE。如果要删除表定义及其数据，请使用 DROP TABLE 语句。

## 五十、索引的工作原理及其种类

**数据库索引**，是数据库管理系统中一个排序的数据结构，以协助快速查询、更新数据库表中数据。**索引的实现通常使用B树及其变种B+树**。

创建索引可以大大提高系统的性能。

第一，通过创建唯一性索引，可以保证数据库表中每一行数据的唯一性。

第二，可以大大加快数据的检索速度，这也是创建索引的最主要的原因。

第三，可以加速表和表之间的连接，特别是在实现数据的参考完整性方面特别有意义。

第四，在使用分组和排序子句进行数据检索时，同样可以显著减少查询中分组和排序的时间。

第五，通过使用索引，可以在查询的过程中，使用优化隐藏器，提高系统的性能。

增加索引也有许多不利的方面。

第一，创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加。

第二，索引需要占物理空间，除了数据表占数据空间之外，每一个索引还要占一定的物理空间，如果要建立聚簇索引，那么需要的空间就会更大。

第三，当对表中的数据进行增加、删除和修改的时候，索引也要动态的维护，这样就降低了数据的维护速度。

**一般来说，应该在这些列上创建索引：1、**在经常需要**搜索的列**上，可以加快搜索的速度；2、在作为**主键的列**上，强制该列的唯一性和组织表中数据的排列结构；3、在经常用在**连接的列**上，这些列主要是一些外键，可以加快连接的速度；4、在经常需要根据**范围进行搜索的列**上创建索引，因为索引已经排序，其指定的范围是连续的；5、在经常需要**排序的列**上创建索引，因为索引已经排序，这样查询可以利用索引的排序，加快排序查询时间；6、在经常使用在**WHERE子句中的列**上面创建索引，加快条件的判断速度。

**一般来说，不应该创建索引的的这些列具有下列特点：**

第一，对于那些在查询中很少使用或者参考的列不应该创建索引。这是因为，既然这些列很少使用到，因此有索引或者无索引，并不能提高查询速度。相反，由于增加了索引，反而降低了系统的维护速度和增大了空间需求。

第二，对于那些只有很少数据值的列也不应该增加索引。这是因为，由于这些列的取值很少，例如人事表的性别列，在查询的结果中，结果集的数据行占了表中数据行的很大比例，即需要在表中搜索的数据行的比例很大。增加索引，并不能明显加快检索速度。

第三，对于那些定义为text, image和bit数据类型的列不应该增加索引。这是因为，这些列的数据量要么相当大，要么取值很少。

第四，当修改性能远远大于检索性能时，不应该创建索引。

在[数据库设计](http://www.2cto.com/database/" \t "http://www.2cto.com/database/201504/_blank)器中创建三种索引：**唯一索引、主键索引和聚集索引**

**唯一索引**

唯一索引是不允许其中任何两行具有相同索引值的索引。

当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在employee表中职员的姓(lname)上创建了唯一索引，则任何两个员工都不能同姓。

**主键索引**

数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。 在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。

**聚集索引**

在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。**一个表只能包含一个聚集索引。**如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。**与非聚集索引相比，聚集索引通常提供更快的数据访问速度。**

如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。**与非聚集索引相比，聚集索引通常提供更快的数据访问速度。**

**五十一、外连接**

1. 概念：包括左向外联接、右向外联接或完整外部联接  
     
   2.**左连接**：left join 或 left outer join  
   (1)左向外联接的结果集包括 LEFT OUTER 子句中指定的左表的所有行，而不仅仅是联接列所匹配的行。如果左表的某行在右表中没有匹配行，则在相关联的结果集行中右表的所有选择列表列均为空值(null)。  
   (2)sql 语句  
   select \* from table1 left join table2 on table1.id=table2.id
2. **右连接**：right join 或 right outer join  
   (1)右向外联接是左向外联接的反向联接。将返回右表的所有行。如果右表的某行在左表中没有匹配行，则将为左表返回空值。  
   (2)sql 语句  
   select \* from table1 right join table2 on table1.id=table2.id
3. **完整外部联接**:full join 或 full outer join  
   (1)完整外部联接返回左表和右表中的所有行。当某行在另一个表中没有匹配行时，则另一个表的选择列表列包含空值。**如果表之间有匹配行，则整个结果集行包含基表的数据值。**  
   (2)sql 语句  
   select \* from table1 full join table2 on table1.id=table2.id
4. 内连接  
   1.概念：内联接是用比较运算符比较要联接列的值的联接  
   2.内连接：join 或 inner join  
   3.sql 语句  
   select \* from table1 join table2 on table1.id=table2.id

**注释：只返回符合条件的table1和table2的列**

1. 交叉连接**(完全)**  
     
   1.概念：没有 WHERE 子句的交叉联接将产生联接所涉及的表的笛卡尔积。第一个表的行数乘以第二个表的行数等于笛卡尔积结果集的大小。（table1和table2交叉连接产生3\*3=9条记录）  
   2.交叉连接：**cross join (不带条件where...)**  
   3.sql语句  
   select \* from table1 cross join table2

在使用left jion时，on和where条件的区别如下：

1、 on条件是在生成临时表时使用的条件，它不管on中的条件是否为真，都会返回左边表中的记录。

2、where条件是在临时表生成好后，再对临时表进行过滤的条件。这时已经没有left join的含义（必须返回左边表的记录）了，条件不为真的就全部过滤掉。

# 五十二、****数据库优化的思路****

## ****1、SQL语句优化****

1）应尽量避免在 where 子句中使用!=或<>操作符，否则将引擎放弃使用索引而进行全表扫描。  
2）应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：  
select id from t where num is null  
**可以在num上设置默认值0，确保表中num列没有null值**，然后这样查询：  
select id from t where num=0  
3）很多时候用 exists 代替 in 是一个好的选择  
4）用Where子句替换HAVING 子句 因为HAVING 只会在检索出所有记录之后才对结果集进行过滤

BTREE索引和 HASH 索引的差异：

（1）HASH索引只用于使用 = 或 <=> 操作符的等式比较。如果一定要使用范围查询 的话，只能使用BTREE索引。

（2）优化器不能使用 Hash 索引来加速 order by 操作。

（3）使用 Hash 索引时 MySQL 不能确定在两个值之间大约有多少行。如果将一 个MyISAM表改为的 Hash 索引 memory 表，

会影响一些查询的执行效率。

（4）Hash索引只能使用整个关键字来搜索一行。

## 2.索引优化

上文

## 3.数据库结构优化

五十三、存储过程与触发器的区别

触发器与存储过程非常相似，触发器也是SQL语句集，**两者唯一的区别是触发器不能用EXECUTE语句调用，而是在用户执行Transact-SQL语句时自动触发（激活）执行。触发器是在一个修改了指定表中的数据时执行的存储过程。**通**常通过创建触发器来强制实现不同表中的逻辑相关数据的引用完整性和一致性。**由于用户不能绕过触发器，所以可以用它来强制实施复杂的业务规则，以确保数据的完整性。触发器不同于存储过程，**触发器主要是通过事件执行触发而被执行的**，而**存储过程可以通过存储过程名称名字而直接调用**。当对某一表进行诸如UPDATE、INSERT、DELETE这些操作时，SQLSERVER就会自动执行触发器所定义的SQL语句，从而确保对数据的处理必须符合这些SQL语句所定义的规则。

## ****五十四、多态****

对于Super supsub=   **new** Sub( )的访问特点：成员变量是编译看左，运行看左，所以用的是父类的成员变量；成员方法是编译看左，运行看右，运行的是子类的方法；静态方法同成员变量，编译看左运行看左。静态方法的调用是不具有多态性的，声明的类型是哪个类，就调用那个类的方法。（多态只表现在方法上，而没有表现在属性上）

**五十五、TCP三次握手和四次挥手的全过程**

**三次握手**

第一次握手：客户端发送syn包(syn=x)到服务器，并进入SYN\_SEND状态，等待服务器确认；  
第二次握手：服务器收到syn包，必须确认客户的SYN（ack=x+1），同时自己也发送一个SYN包（syn=y），即SYN+ACK包，此时服务器进入SYN\_RECV状态；  
第三次握手：客户端收到服务器的SYN（x+1）＋ACK包，向服务器发送确认包ACK(ack=y+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。  
握手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连接之前，TCP 连接都将被一直保持下去。

                                  发送                  确认

第一次：SYN=1       SEQ=X               ACK=0（客）

第二次：SYN=1       SEQ=Y              ACK=X+1（服）

第三次：                  SEQ=X+1          ACK=Y+1（客）

SYN 同步位

四次挥手  
 与建立连接的“三次握手”类似，断开一个TCP连接则需要“四次握手”。  
第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不 会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可 以接受数据。  
第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。  
第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。  
第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

五十六、**ARP是**地址解析协议****

1：首先，每个主机都会在自己的ARP缓冲区中建立一个ARP列表，以表示IP地址和MAC地址之间的对应关系。  
2：当源主机要发送数据时，首先检查ARP列表中是否有对应IP地址的目的主机的MAC地址，如果有，则直接发送数据，如果没有，就向本网段的所有主机发送ARP数据包，该数据包包括的内容有：源主机 IP地址，源主机MAC地址，目的主机的IP 地址。  
3：当本网络的所有主机收到该ARP数据包时，首先检查数据包中的IP地址是否是自己的IP地址，如果不是，则忽略该数据包，如果是，则首先从数据包中取出源主机的IP和MAC地址写入到ARP列表中，如果已经存在，则覆盖，然后将自己的MAC地址写入ARP响应包中，告诉源主机自己是它想要找的MAC地址。  
4：源主机收到ARP响应包后。将目的主机的IP和MAC地址写入ARP列表，并利用此信息发送数据。如果源主机一直没有收到ARP响应数据包，表示ARP查询失败。  
广播发送ARP请求，单播发送ARP响应。

五十七、以太网的帧值总是在一定范围内浮动，最大的帧值是1518字节，最小的帧值64字节。



****五十八、http请求格式****

HTTP 请求由三部分组成：请求行、  请求头和请求正文。

 (1)、请求行：请求方法  URI 协议/版本

 (2)、请求头(Request Header)

 (3)、 请求正文



### 1、请求行：请求方法URI协议/版本

 请求的第一行是“方法 URL  协议/版本”，并以 回车换行作为结尾。请求行以空格分隔。格式如下：

POST /index.php HTTP/1.1

以上代码中“POST”代表请求方法，“/index.php”表示URI，“HTTP/1.1代表协议和协议的版本。

        根据HTTP标准，HTTP请求可以使用多种请求方法。例如：HTTP1.1支持7种请求方法：GET、POST、HEAD、OPTIONS、PUT、DELETE和TARCE。在Internet应用中，最常用的方法是GET和POST。

重点介绍 GET、POST 和 HEAD 三个方法：

**（1）GET**

        GET 方法用于获取由 Request-URI 所标识的资源的信息，常见的形式是：

        GET Request-URI HTTP/1.1  
        GET方法是默认的HTTP请求方法，例如当我们通过在浏览器的地址栏中直接输入网址的方式去访问网页的时候，浏览器采用的就是 GET 方法向服务器获取资源。

**（2）POST**

          POST方法是GET方法的一个替代方法，它主要是向Web服务器提交表单数据，尤其是大批量的数据。 在请求头信息结束之后的两个回车换行之后（实际是空一行），就是表单提交的数据。如上面提到的post表单数据：

        username=aa&password=1234

        POST方法克服了GET方法的一些缺点。通过POST方法提交表单数据时，数据不是作为URL请求的一部分而是作为标准数据传送给Web服务器，这就克服了GET方法中的信息无法保密和数据量太小的缺点。

**GET与POST方法有以下区别：**

      1、  在客户端，Get方式在通过URL提交数据，数据在URL中可以看到；POST方式，数据放在HTTP包的body中。

      2、 GET方式提交的数据大小有限制（因为浏览器对URL的长度有限制），而POST则没有此限制。

      3、安全性问题。正如在（1）中提到，使用 Get 的时候，参数会显示在地址栏上，而 Post 不会。所以，如果这些数据是中文数据而且是非敏感数据，那么使用 get；如果用户输入的数据不是中文字符而且包含敏感数据，那么还是使用 post为好。

      4.、服务器取值方式不一样。GET方式取值，如**[PHP](http://lib.csdn.net/base/php" \o "PHP知识库" \t "http://blog.csdn.net/hudashi/article/details/_blank)**可以使用$\_GET来取得变量的值，而POST方式通过$\_POST来获取变量的值。

### 请求头

#### Transport 头域

每个头域由一个域名，冒号（:）和域值三部分组成。域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

**Connection：**

作用：表示是否需要持久连接。

如果服务器看到这里的值为“Keep-Alive”，或者看到请求使用的是HTTP 1.1（HTTP 1.1默认进行持久连接）,它就可以利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。

例如：　Connection: keep-alive   当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的  网页，会继续使用这一条已经建立的连接

例如：  Connection: close  代表一个Request完成后，客户端和服务器之间用于传输HTTP数据的TCP连接会关闭，  当客户端再次发送Request，需要重新建立TCP连接。

**Host（发送请求时，该报头域是必需的）**

Host请求报头域主要用于指定被请求资源的Internet主机和端口号，它通常从HTTP URL中提取出来的。

eg：[http://；localhost/index.html](http://;localhost/index.html" \t "http://blog.csdn.net/hudashi/article/details/_blank)  
浏览器发送的请求消息中，就会包含Host请求报头域，如下：  
Host：[localhos](http://www.guet.edu.cn/" \t "http://blog.csdn.net/hudashi/article/details/_blank)t

此处使用缺省端口号80，若指定了端口号8080，则变成：Host：localhost:8080

#### Client 头域

**Accept：**

作用：浏览器可以接受的媒体类型（MIME类型）,

例如：  Accept: text/html  代表浏览器可以接受服务器回发的类型为 text/html  也就是我们常说的html文档, 如果服务器无法返回text/html类型的数据，服务器应该返回一个406错误(non acceptable)。

通配符 \* 代表任意类型。例如  Accept: \*/\*  代表浏览器可以处理所有类型，(一般浏览器发给服务器都是发这个)

**Accept-Encoding：**

作用： 浏览器申明自己接收的编码方法，通常指定压缩方法，是否支持压缩，支持什么压缩方法（gzip，deflate），（注意：这不是只字符编码）

**Accept-Language：**

作用： 浏览器申明自己接收的语言。

语言跟字符集的区别：中文是语言，中文有多种字符集，比如big5，gb2312，gbk等等；

例如： Accept-Language:zh-cn 。如果请求消息中没有设置这个报头域，服务器假定客户端对各种语言都可以接受。

**User-Agent：**

作用：告诉HTTP服务器， 客户端使用的**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/hudashi/article/details/_blank)**和浏览器的名称和版本.

我们上网登陆论坛的时候，往往会看到一些欢迎信息，其中列出了你的操作系统的名称和版本，你所使用的浏览器的名称和版本，这往往让很多人感到很神奇，实际上， 服务器应用程序就是从User-Agent这个请求报头域中获取到这些信息User-Agent请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器。

#### Cookie/Login 头域

**Cookie:**

作用： 最重要的header, 将cookie的值发送给HTTP 服务器

#### Entity头域

**Content-Length**

作用：发送给HTTP服务器数据的长度。即请求消息正文的长度；

例如： Content-Length: 38

**Content-Type：**

作用：

例如：Content-Type: application/x-www-form-urlencoded

#### Miscellaneous 头域

**Referer:**

作用： 提供了Request的上下文信息的服务器，告诉服务器我是从哪个链接过来的，比如从我主页上链接到一个朋友那里， 他的服务器就能够从HTTP Referer中统计出每天有多少用户点击我主页上的链接访问    他的网站。

#### Cache 头域

**If-Modified-Since：**

作用： 把浏览器端缓存页面的最后修改时间发送到服务器去，服务器会把这个时间与服务器上实际文件的最后修改时间进行对比。如果时间一致，那么返回304，客户端就直接使用本地缓存文件。如果时间不一致，就会返回200和新的文件内容。客户端接到之后，会丢弃旧文件，把新文件缓存起来，并显示在浏览器中。

例如：If-Modified-Since: Thu, 09 Feb 2012 09:07:57 GMT。

**Cache-Control：**

作用: 这个是非常重要的规则。 这个用来指定Response-Request遵循的缓存机制。各个指令含义如下

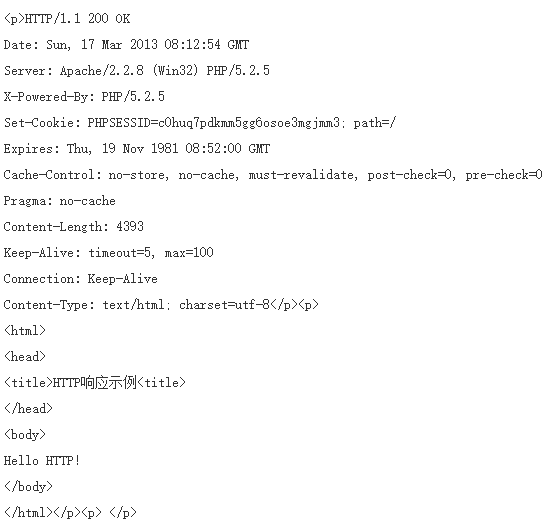
Cache-Control:Public   可以被任何缓存所缓存（）

Cache-Control:Private     内容只缓存到私有缓存中

Cache-Control:no-cache  所有内容都不会被缓存

## 二、 HTTP响应格式

HTTP 响应也是由三个部分组成，分别是：状态行、消息报头和响应正文。



### 1、状态行

       状态行由协议版本、数字形式的状态代码，及相应的状态描述组成，各元素之间以空格分隔，结尾时回车换行符，格式如下：

HTTP-Version Status-Code Reason-Phrase CRLF

HTTP-Version 表示服务器 HTTP 协议的版本，Status-Code 表示服务器发回的响应代码，Reason-Phrase 表示状态代码的文本描述，CRLF 表示回车换行。例如：

HTTP/1.1 200 OK (CRLF)

**状态代码与状态描述**

      状态代码由 3 位数字组成， 表示请求是否被理解或被满足，状态描述给出了关于状态码的简短的文字描述。状态码的第一个数字定义了响应类别，后面两位数字没有具体分类。第一个数字有 5 种取值，如下所示。

* 1xx：指示信息——表示请求已经接受，继续处理
* 2xx：成功——表示请求已经被成功接收、理解、接受。
* 3xx：重定向——要完成请求必须进行更进一步的操作
* 4xx：客户端错误——请求有语法错误或请求无法实现
* 5xx：服务器端错误——服务器未能实现合法的请求。

常见状态代码、状态描述、说明：  
200 OK      //客户端请求成功  
400 Bad Request  //客户端请求有语法错误，不能被服务器所理解  
401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用   
403 Forbidden  //服务器收到请求，但是拒绝提供服务  
404 Not Found  //请求资源不存在，eg：输入了错误的URL  
500 Internal Server Error //服务器发生不可预期的错误  
503 Server Unavailable  //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

### 2、响应正文

响应正文就是服务器返回的资源的内容，响应头和正文之间也必须用空行分隔。

### 3 、响应头信息

HTTP最常见的响应头如下所示：

#### Cache头域

**Date：**

作用：生成消息的具体时间和日期，即当前的GMT时间。

例如：　Date: Sun, 17 Mar 2013 08:12:54 GMT

**Expires：**

作用: 浏览器会在指定过期时间内使用本地缓存，指明应该在什么时候认为文档已经过期，从而不再缓存它。

例如: Expires: Thu, 19 Nov 1981 08:52:00 GMT

**Vary**

作用：

例如: Vary: Accept-Encoding

#### Cookie/Login 头域

#### Cookie/Login 头域

**P3P**

作用: 用于跨域设置Cookie, 这样可以解决iframe跨域访问cookie的问题

例如: P3P: CP=CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA PRE COM NAV OTC NOI DSP COR

**Set-Cookie**

作用： 非常重要的header, 用于把cookie 发送到客户端浏览器， 每一个写入cookie都会生成一个Set-Cookie.

例如: Set-Cookie: PHPSESSID=c0huq7pdkmm5gg6osoe3mgjmm3; path=/

#### Miscellaneous 头域

**Server：**

作用：指明HTTP服务器的软件信息

例如:Apache/2.2.8 (Win32) **[php](http://lib.csdn.net/base/php" \o "PHP知识库" \t "http://blog.csdn.net/hudashi/article/details/_blank)**/5.2.5

**X-Powered-By：**

作用：表示网站是用什么技术开发的

例如： X-Powered-By: PHP/5.2.5

#### Location头域

**Location：**

作用： 用于重定向一个新的位置， 包含新的URL地址

实例请看304状态实例

五十九、通配符

通配符主要以下几种：**%**、**\_**、**[]**、**[^]** 。

1. %：模糊匹配一个或多个字符。如，以ab开头

**[sql]** [view plain](http://blog.csdn.net/ludonqin/article/details/52506427" \o "view plain" \t "http://blog.csdn.net/ludonqin/article/details/_blank) [copy](http://blog.csdn.net/ludonqin/article/details/52506427" \o "copy" \t "http://blog.csdn.net/ludonqin/article/details/_blank)

1. **SELECT** \* **FROM** (
2. **SELECT** 'abc\_2sd\_38u' 'a'
3. **UNION** ALL
4. **SELECT** 'bcf' 'a'
5. ) c
6. **WHERE** c.a LIKE 'ab%'

2. \_：占位符，匹配任意一个字符。如，以abc\_开头

**[sql]** [view plain](http://blog.csdn.net/ludonqin/article/details/52506427" \o "view plain" \t "http://blog.csdn.net/ludonqin/article/details/_blank) [copy](http://blog.csdn.net/ludonqin/article/details/52506427" \o "copy" \t "http://blog.csdn.net/ludonqin/article/details/_blank)

1. **SELECT** \* **FROM** (
2. **SELECT** 'abc\_2sd\_38u' 'a'
3. **UNION** ALL
4. **SELECT** 'abcd\_5d\_3u' 'a'
5. **UNION** ALL
6. **SELECT** 'abc' 'a'
7. ) c
8. **WHERE** c.a LIKE 'abc[\_]%'

3. []：目标匹配字符：匹配单个字符，若写了多个，匹配任一。如，所有以1或者2开头的

**[sql]** [view plain](http://blog.csdn.net/ludonqin/article/details/52506427" \o "view plain" \t "http://blog.csdn.net/ludonqin/article/details/_blank) [copy](http://blog.csdn.net/ludonqin/article/details/52506427" \o "copy" \t "http://blog.csdn.net/ludonqin/article/details/_blank)

1. **SELECT** \* **FROM** (
2. **SELECT** '1\_21' 'a'
3. **UNION** ALL
4. **SELECT** '121' 'a'
5. **UNION** ALL
6. **SELECT** '131' 'a'
7. **UNION** ALL
8. **SELECT** '142' 'a'
9. **UNION** ALL
10. **SELECT** '223' 'a'
11. **UNION** ALL
12. **SELECT** '324' 'a'
13. ) x
14. **WHERE** x.a LIKE '[12]%'

4. [^]：目标匹配字符，[]相反。如，所有非1、2开头的

**[sql]** [view plain](http://blog.csdn.net/ludonqin/article/details/52506427" \o "view plain" \t "http://blog.csdn.net/ludonqin/article/details/_blank) [copy](http://blog.csdn.net/ludonqin/article/details/52506427" \o "copy" \t "http://blog.csdn.net/ludonqin/article/details/_blank)

1. **SELECT** \* **FROM** (
2. **SELECT** '1\_21' 'a'
3. **UNION** ALL
4. **SELECT** '121' 'a'
5. **UNION** ALL
6. **SELECT** '131' 'a'
7. **UNION** ALL
8. **SELECT** '142' 'a'
9. **UNION** ALL
10. **SELECT** '223' 'a'
11. **UNION** ALL
12. **SELECT** '324' 'a'
13. ) x
14. **WHERE** x.a LIKE '[^12]%'

六十、

数据库系统(DBS)由五部分组成：

1.硬件系统；

2.数据库集合；

3.DBMS；

4.及相关软件；

5.数据库管理员与用户；

六十一、类方法和实例方法的区别

1、类方法是属于整个类，而不属于某个对象。

1. 2、类方法只能访问类成员变量，不能访问实例变量，而实例方法可以访问类成员变量和实例变量。
2. 3、类方法的调用可以通过类名.类方法和对象.类方法，而实例方法只能通过对象.实例方法访问。
3. 4、类方法只能访问类方法，而实例方法可以访问类方法和实例方法。
4. 5、类方法不能被覆盖，实例方法可以被覆盖。

六十二、

**物理层：** *透明地传输比特流*

**数据链路层**： *建立逻辑连接、进行硬件地址寻址、差错效验等功能。*

**网络层**： *进行逻辑地址寻址，实现不同网络之间的路径选择。*

**传输层**： *定义传输数据的协议端口号，以及流控和差错效验。*

**会话层**： *建立、管理、终止会话。（在五层模型里面已经合并到了应用层）* *对应主机进程，指本地主机与远程主机正在进行的会话*

**表示层**： *数据的表示、安全、压缩。（在五层模型里面已经合并到了应用层）*

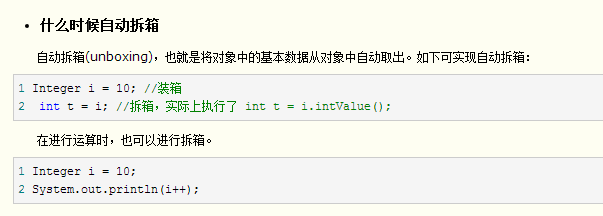
**应用层**： *网络服务与最终用户的一个接口。*

六十三、自动装箱和自动拆箱

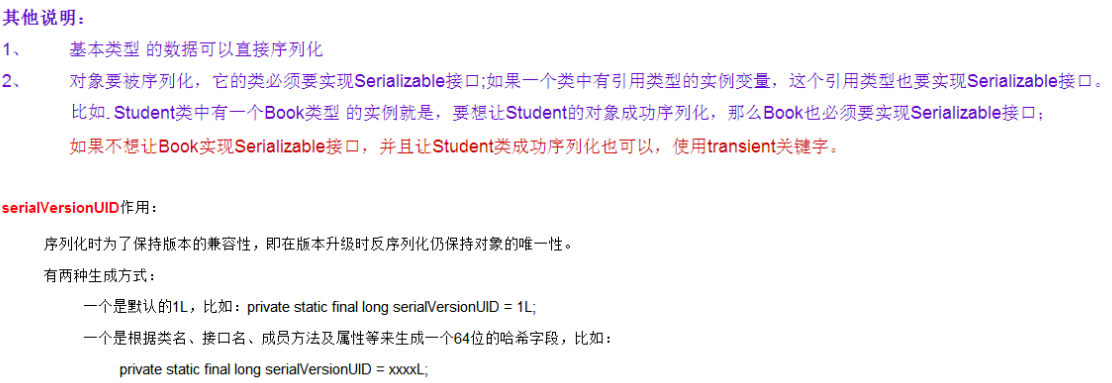
### **1.什么时候自动装箱**

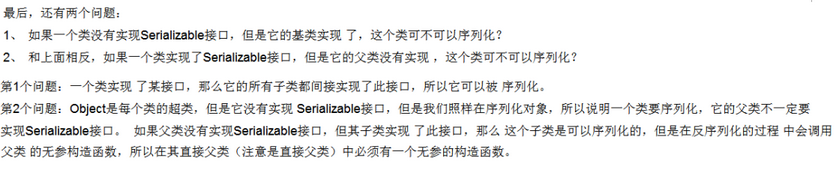
例如：Integer i = 100;

相当于编译器自动为您作以下的语法编译：Integer i = Integer.valueOf(100);



六十三、





六十四、进程的通信机制主要有：管道、有名管道、消息队列、信号量、共享空间、信号、套接字。

六十五、swing和awt的区别

AWT(Abstract Window Toolkit)：抽象窗口工具包，早期编写图形界面应用程序的包。

Swing ：为解决 AWT 存在的问题而新开发的图形界面包。Swing是对AWT的改良和扩展。

AWT和Swing的实现原理不同： AWT的图形函数与[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/iamluole/article/details/_blank)提供的图形函数有着一一对应的关系。为了实现Java语言所宣称的"一次编译，到处运行"的概念，AWT不得不通过牺牲功能来实现平台无关性。因此，AWT 的图形功能是各操作系统图形功能的“交集”。

 因为AWT是依靠本地方法来实现功能的，所以AWT控件称为“重量级控件”。

 而Swing ，不仅提供了AWT 的所有功能，还用纯粹的Java代码对AWT的功能进行了大幅度的扩充。由于 Swing是用纯粹的Java代码来实现的，因此Swing控件在各平台通用。  
因为Swing不使用本地方法，故Swing控件称为“轻量级控件”。

  AWT和Swing之间的区别：  
       1)AWT 是基于本地方法的C/C++程序，其运行速度比较快；Swing是基于AWT的Java程序，其运行速度比较慢。  
       2)AWT的控件在不同的平台可能表现不同，而Swing在所有平台表现一致。

六十五、java并发编程

在并发编程中，我们需要处理两个关键问题：线程之间如何通信及线程之间如何同步（这里的线程是指并发执行的活动实体）。通信是指线程之间以何种机制来交换信息。在命令式编程中，线程之间的通信机制有两种：共享内存和消息传递。

## 六十六、sleep()和wait()有什么区别?

答：sleep()方法是线程类（Thread）的静态方法，导致此线程暂停执行指定时间，将执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复（线程回到就绪（ready）状态），因为调用sleep 不会释放对象锁。wait()是Object 类的方法，对此对象调用wait()方法导致本线程放弃对象锁(线程暂停执行)，进入等待此对象的等待锁定池，只有针对此对象发出notify 方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入就绪状态。

yield()方法对应了如下操作；先检测当前是否有相同优先级的线程处于同可运行状态，如有，则把CPU的占有权交给次线程，否则继续运行原来的线程，所以yield()方法称为“退让”，它把运行机会让给了同等级的其他线程。

补充：这里似乎漏掉了一个作为先决条件的问题，就是**什么是进程，什么是线程**？为什么需要多线程编程？答案如下所示：

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动，是**[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/jackfrued/article/details/_blank)**进行资源分配和调度的一个独立单位；线程是进程的一个实体，是CPU调度和分派的基本单位，是比进程更小的能独立运行的基本单位。线程的划分尺度小于进程，这使得多线程程序的并发性高；进程在执行时通常拥有独立的内存单元，而线程之间可以共享内存。使用多线程的编程通常能够带来更好的性能和用户体验，但是多线程的程序对于其他程序是不友好的，因为它占用了更多的CPU资源。

## 六十七、notify()和notifyAll()区别

notify()和notifyAll()都是Object对象用于通知处在等待该对象的线程的方法。两者的最大区别在于：

notifyAll使所有原来在该对象上等待被notify的线程统统退出wait的状态，变成等待该对象上的锁，一旦该对象被解锁，他们就会去竞争。  
notify则文明得多他只是选择一个wait状态线程进行通知，并使它获得该对象上的锁，但不惊动其他同样在等待被该对象notify的线程们，当第一个线程运行完毕以后释放对象上的锁此时如果该对象没有再次使用notify语句，则即便该对象已经空闲，其他wait状态等待的线程由于没有得到该对象的通知，继续处在wait状态，直到这个对象发出一个notify或notifyAll，它们等待的是被notify或notifyAll，而不是锁。

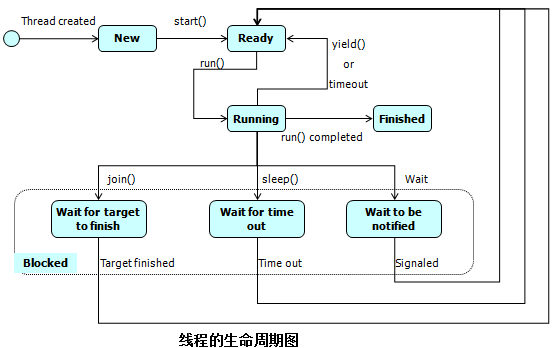
## 六十八、举例说明同步和异步。

答：如果系统中存在临界资源（资源数量少于竞争资源的线程数量的资源），例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就必须进行同步存取（**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/jackfrued/article/details/_blank)**操作中的悲观锁就是最好的例子）。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用异步编程，在很多情况下采用异步途径往往更有效率。事实上，所谓的同步就是指阻塞式操作，而异步就是非阻塞式操作。

## 六十九、简述synchronized 和java.util.concurrent.locks.Lock的异同？

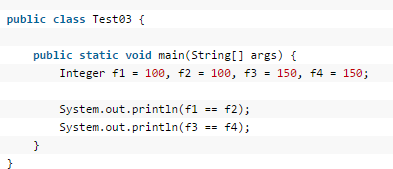
答：Lock是Java 5以后引入的新的API，和关键字synchronized相比主要相同点：Lock 能完成synchronized所实现的所有功能；主要不同点：Lock 有比synchronized 更精确的线程语义和更好的性能。synchronized 会自动释放锁，而Lock 一定要求程序员手工释放，并且必须在finally 块中释放（这是释放外部资源的最好的地方）。

七十、**线程的基本状态以及状态之间的关系？**



除去起始（new）状态和结束（finished）状态，线程有三种状态，分别是：就绪（ready）、运行（running）和阻塞（blocked）。其中就绪状态代表线程具备了运行的所有条件，只等待CPU调度（万事俱备，只欠东风）；处于运行状态的线程可能因为CPU调度（时间片用完了）的原因回到就绪状态，也有可能因为调用了线程的yield方法回到就绪状态，此时线程不会释放它占有的资源的锁，坐等CPU以继续执行；运行状态的线程可能因为I/O中断、线程休眠、调用了对象的wait方法而进入阻塞状态（有的地方也称之为等待状态）；而进入阻塞状态的线程会因为休眠结束、调用了对象的notify方法或notifyAll方法或其他线程执行结束而进入就绪状态。注意：调用wait方法会让线程进入等待池中等待被唤醒，notify方法或notifyAll方法会让等待锁中的线程从等待池进入等锁池，在没有得到对象的锁之前，线程仍然无法获得CPU的调度和执行。

七十一、自动拆箱和装箱



简单的说，如果字面量的值在-128到127之间，那么不会new新的Integer对象，而是直接引用常量池中的Integer对象，所以上面的面试题中f1==f2的结果是true，而f3==f4的结果是false。越是貌似简单的面试题其中的玄机就越多，需要面试者有相当深厚的功力。4

## 七十二、String 和StringBuilder、StringBuffer 的区别?

答：Java 平台提供了两种类型的字符串：String和StringBuffer / StringBuilder，它们可以储存和操作字符串。其中String是只读字符串，也就意味着String引用的字符串内容是不能被改变的。而StringBuffer和StringBuilder类表示的字符串对象可以直接进行修改。StringBuilder是JDK 1.5中引入的，它和StringBuffer的方法完全相同，区别在于它是在单线程环境下使用的，因为它的所有方面都没有被synchronized修饰，因此它的效率也比StringBuffer略高。

补充1：有一个面试题问：有没有哪种情况用+做字符串连接比调用StringBuffer / StringBuilder对象的append方法性能更好？如果连接后得到的字符串在静态存储区中是早已存在的，那么用+做字符串连接是优于StringBuffer / StringBuilder的append方法的。

## 七十三、java.nio相关问题

nio 是 java New IO 的简称，在 jdk1.4 里提供的新 api 。 Sun 官方标榜的特性如下：

–     为所有的原始类型提供 (Buffer) 缓存支持。

–     字符集编码解码解决方案。

–     Channel ：一个新的原始 I/O 抽象。

–     支持锁和内存映射文件的文件访问接口。

–     提供多路 (non-bloking) 非阻塞式的高伸缩性网络 I/O 。

NIO主要有三大核心部分：Channel(通道)，Buffer(缓冲区), Selector。传统IO基于字节流和字符流进行操作，而NIO基于Channel和Buffer(缓冲区)进行操作，数据总是从通道读取到缓冲区中，或者从缓冲区写入到通道中。Selector(选择区)用于监听多个通道的事件（比如：连接打开，数据到达）。因此，单个线程可以监听多个数据通道。

NIO和传统IO（一下简称IO）之间第一个最大的区别是，IO是面向流的，NIO是面向缓冲区的。 Java IO面向流意味着每次从流中读一个或多个字节，直至读取所有字节，它们没有被缓存在任何地方。此外，它不能前后移动流中的数据。如果需要前后移动从流中读取的数据，需要先将它缓存到一个缓冲区。NIO的缓冲导向方法略有不同。数据读取到一个它稍后处理的缓冲区，需要时可在缓冲区中前后移动。这就增加了处理过程中的灵活性。但是，还需要检查是否该缓冲区中包含所有您需要处理的数据。而且，需确保当更多的数据读入缓冲区时，不要覆盖缓冲区里尚未处理的数

IO的各种流是阻塞的。这意味着，当一个线程调用read() 或 write()时，该线程被阻塞，直到有一些数据被读取，或数据完全写入。该线程在此期间不能再干任何事情了。 NIO的非阻塞模式，使一个线程从某通道发送请求读取数据，但是它仅能得到目前可用的数据，如果目前没有数据可用时，就什么都不会获取。而不是保持线程阻塞，所以直至数据变的可以读取之前，该线程可以继续做其他的事情。 非阻塞写也是如此。一个线程请求写入一些数据到某通道，但不需要等待它完全写入，这个线程同时可以去做别的事情。 线程通常将非阻塞IO的空闲时间用于在其它通道上执行IO操作，所以一个单独的线程现在可以管理多个输入和输出通道（channel）。

## 七十四、依赖注入和控制反转

Ioc—Inversion of Control，即“控制反转”，不是什么技术，而是一种设计思想。在Java开发中，Ioc意味着将你设计好的对象交给容器控制，而不是传统的在你的对象内部直接控制。IoC不是一种技术，只是一种思想，一个重要的面向对象编程的法则，它能指导我们如何设计出松耦合、更优良的程序。传统应用程序都是由我们在类内部主动创建依赖对象，从而导致类与类之间高耦合，难于测试；有了IoC容器后，把创建和查找依赖对象的控制权交给了容器，由容器进行注入组合对象，所以对象与对象之间是松散耦合，这样也方便测试，利于功能复用，更重要的是使得程序的整个体系结构变得非常灵活。

DI—Dependency Injection，即“依赖注入”：是组件之间依赖关系由容器在运行期决定，形象的说，即由容器动态的将某个依赖关系注入到组件之中。依赖注入的目的并非为软件系统带来更多功能，而是为了提升组件重用的频率，并为系统搭建一个灵活、可扩展的平台。通过依赖注入机制，我们只需要通过简单的配置，而无需任何代码就可指定目标需要的资源，完成自身的业务逻辑，而不需要关心具体的资源来自何处，由谁实现。

●****谁依赖于谁：****当然是应用程序依赖于IoC容器；

●****为什么需要依赖：****应用程序需要IoC容器来提供对象需要的外部资源；

●****谁注入谁：****很明显是IoC容器注入应用程序某个对象，应用程序依赖的对象；

****●注入了什么：****就是注入某个对象所需要的外部资源（包括对象、资源、常量数据）。

相对IoC 而言，****“依赖注入”明确描述了“被注入对象依赖IoC容器配置依赖对象”。****

在平时的java应用开发中，我们要实现某一个功能或者说是完成某个业务逻辑时至少需要两个或以上的对象来协作完成，在没有使用Spring的时候，每个对象在需要使用他的合作对象时，自己均要使用像new object() 这样的语法来将合作对象创建出来，这个合作对象是由自己主动创建出来的，创建合作对象的主动权在自己手上，自己需要哪个合作对象，就主动去创建，创建合作对象的主动权和创建时机是由自己把控的，而这样就会使得对象间的耦合度高了，A对象需要使用合作对象B来共同完成一件事，A要使用B，那么A就对B产生了依赖，也就是A和B之间存在一种耦合关系，并且是紧密耦合在一起，而使用了Spring之后就不一样了，创建合作对象B的工作是由Spring来做的，Spring创建好B对象，然后存储到一个容器里面，当A对象需要使用B对象时，Spring就从存放对象的那个容器里面取出A要使用的那个B对象，然后交给A对象使用，至于Spring是如何创建那个对象，以及什么时候创建好对象的，A对象不需要关心这些细节问题(你是什么时候生的，怎么生出来的我可不关心，能帮我干活就行)，A得到Spring给我们的对象之后，两个人一起协作完成要完成的工作即可。

　　所以**控制反转IoC(Inversion of Control)是说创建对象的控制权进行转移，以前创建对象的主动权和创建时机是由自己把控的，而现在这种权力转移到第三方**，比如转移交给了IoC容器，它就是一个专门用来创建对象的工厂，你要什么对象，它就给你什么对象，有了 IoC容器，依赖关系就变了，原先的依赖关系就没了，它们都依赖IoC容器了，通过IoC容器来建立它们之间的关系。

****七十五、AOP****

AOP是什么(Aspect   Oriented   Programming)

 AOP是一种编程范式，提供从另一个角度来考虑程序结构以完善面向对象编程（OOP）。

 AOP为开发者提供了一种描述横切关注点的机制，并能够自动将横切关注点织入到面向对象的软件系统中，从而实现了横切关注点的模块化。

 AOP能够将那些与业务无关，却为业务模块所共同调用的逻辑或责任，例如事务处理、日志管理、权限控制等，封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可操作性和可维护性。

nAOP能干什么，也是AOP带来的好处

****1：****降低模块的耦合度

****2：****使系统容易扩展

****3：****设计决定的迟绑定：使用AOP,设计师可以推迟为将来的需求作决定，因为它

可以把这种需求作为独立的方面很容易的实现。

****4：****更好的代码复用性

AOP（Aspect Oriented Programming），即面向切面编程，可以说是OOP（Object Oriented Programming，面向对象编程）的补充和完善。OOP引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过OOP允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对象的核心功能毫无关系对于其他类型的代码，如安全性、异常处理和透明的持续性也都是如此，这种散布在各处的无关的代码被称为横切（cross cutting），在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。

AOP技术恰恰相反，它利用一种称为"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。

使用"横切"技术，AOP把软件系统分为两个部分：**核心关注点**和**横切关注点**。业务处理的主要流程是核心关注点，与之关系不大的部分是横切关注点。横切关注点的一个特点是，他们经常发生在核心关注点的多处，而各处基本相似，比如权限认证、日志、事物。AOP的作用在于分离系统中的各种关注点，将核心关注点和横切关注点分离开来。

****七十六、jvm的四种引用状态****

在JDK1.2之后，Java对引用的概念进行了扩充，将引用分为强引用（Strong Reference）、软引用（Soft Reference）、弱引用（Weak Reference）、虚引用（Phantom Reference）4种，这4中引用强度一次减弱。

* **强引用就是指在程序代码之中普遍存在的，类似"Object obj = new Object()"这类的引用**，只要强引用还存在，垃圾收集器永远不会回收掉被引用的对象
* **软引用是用来描述一些还有用但并非必需的对象**，对于软引用关联着的对象，**在系统将要发生内存溢出异常之前，将会把这些对象列进回收范围进行第二次回收**。如果这次回收还没有足够的内存，才会抛出内存溢出异常。在JDK1.2之后，提供了SoftReference类来实现软引用
* **弱引用也是用来描述非必需对象的，但是它的强度比软引用更弱一些**，被弱引用关联的对象，**只能生存到下一次垃圾收集发生之前**。当垃圾收集器工作时，无论当前内存是否足够，都会回收掉只被弱引用关联的对象。在JDK1.2之后，提供了WeakReference类来实现弱引用
* 虚引用也成为幽灵引用或者幻影引用，它是最弱的一中引用关系。一个对象是否有虚引用的存在，完全不会对其生存时间构成影响，也无法通过虚引用来取得一个对象实例。**为一个对象设置虚引用关联的唯一目的就是能在这个对象被收集器回收时收到一个系统通知**。在JDK1.2之后，提供给了PhantomReference类来实现虚引用

## 七十七、TCP的拥塞控制

1.  拥塞：即对资源的需求超过了可用的资源。若网络中许多资源同时供应不足，网络的性能就要明显变坏，整个网络的吞吐量随之负荷的增大而下降。

    拥塞控制：**防止过多的数据注入到网络中，这样可以使网络中的路由器或链路不致过载。**拥塞控制所要做的都有一个**前提：网络能够承受现有的网络负荷。**拥塞控制是一个**全局性的过程**，涉及到所有的主机、路由器，以及与降低网络传输性能有关的所有因素。

    流量控制：指点对点通信量的控制，是端到端正的问题。流量控制所要做的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

    拥塞控制代价：需要获得网络内部流量分布的信息。在实施拥塞控制之前，还需要在结点之间交换信息和各种命令，以便选择控制的策略和实施控制。这样就产生了额外的开销。拥塞控制还需要将一些资源分配给各个用户单独使用，使得网络资源不能更好地实现共享。

2. 几种拥塞控制方法

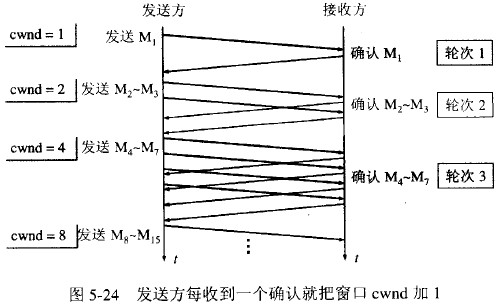
    慢开始( slow-start )、拥塞避免( congestion avoidance )、快重传( fast retransmit )和快恢复( fast recovery )。

2.1 慢开始和拥塞避免

    发送方维持一个拥塞窗口 cwnd ( congestion window )的状态变量。拥塞窗口的大小取决于网络的拥塞程度，并且动态地在变化。发送方让自己的发送窗口等于拥塞。

    发送方控制拥塞窗口的原则是：只要网络没有出现拥塞，拥塞窗口就再增大一些，以便把更多的分组发送出去。但只要网络出现拥塞，拥塞窗口就减小一些，以减少注入到网络中的分组数。

    慢开始算法：当主机开始发送数据时，如果立即所大量数据字节注入到网络，那么就有可能引起网络拥塞，因为现在并不清楚网络的负荷情况。因此，较好的方法是先探测一下，即由小到大逐渐增大发送窗口，也就是说，由小到大逐渐增大拥塞窗口数值。通常在刚刚开始发送报文段时，先把拥塞窗口 cwnd 设置为一个最大报文段MSS的数值。而在每收到一个对新的报文段的确认后，把拥塞窗口增加至多一个MSS的数值。用这样的方法逐步增大发送方的拥塞窗口 cwnd ，可以使分组注入到网络的速率更加合理。

****

每经过一个传输轮次，拥塞窗口 cwnd 就加倍。一个传输轮次所经历的时间其实就是往返时间RTT。不过“传输轮次”更加强调：把拥塞窗口cwnd所允许发送的报文段都连续发送出去，并收到了对已发送的最后一个字节的确认。

另，慢开始的“慢”并不是指cwnd的增长速率慢，而是指在TCP开始发送报文段时先设置cwnd=1，使得发送方在开始时只发送一个报文段（目的是试探一下网络的拥塞情况），然后再逐渐增大cwnd。

    为了防止拥塞窗口cwnd增长过大引起网络拥塞，还需要设置一个慢开始门限ssthresh状态变量（如何设置ssthresh）。慢开始门限ssthresh的用法如下：

    当 cwnd < ssthresh 时，使用上述的慢开始算法。

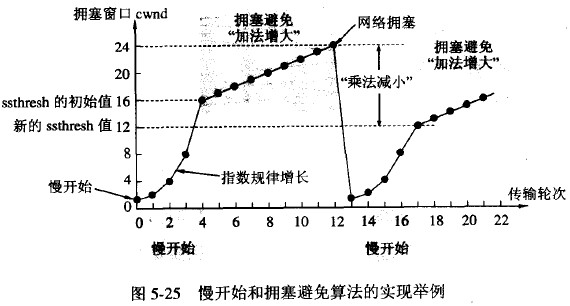
    当 cwnd > ssthresh 时，停止使用慢开始算法而改用拥塞避免算法。

    当 cwnd = ssthresh 时，既可使用慢开始算法，也可使用拥塞控制避免算法。

拥塞避免算法：让拥塞窗口cwnd缓慢地增大，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍。这样拥塞窗口cwnd按线性规律缓慢增长，比慢开始算法的拥塞窗口增长速率缓慢得多。

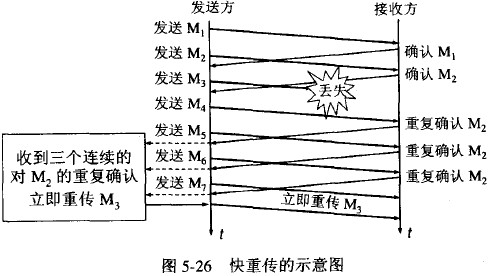
    无论在慢开始阶段还是在拥塞避免阶段，只要发送方判断网络出现拥塞（其根据就是没有收到确认），就要把慢开始门限ssthresh设置为出现拥塞时的发送方窗口值的一半（但不能小于2）。然后把拥塞窗口cwnd重新设置为1，执行慢开始算法。这样做的目的就是要迅速减少主机发送到网络中的分组数，使得发生拥塞的路由器有足够时间把队列中积压的分组处理完毕。

    如下图，用具体数值说明了上述拥塞控制的过程。现在发送窗口的大小和拥塞窗口一样大。



2.2 快重传和快恢复

 快重传算法首先要求接收方每收到一个失序的报文段后就立即发出重复确认（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时才进行捎带确认。



接收方收到了M1和M2后都分别发出了确认。现在假定接收方没有收到M3但接着收到了M4。显然，接收方不能确认M4，因为M4是收到的失序报文段。根据可靠传输原理，接收方可以什么都不做，也可以在适当时机发送一次对M2的确认。但按照快重传算法的规定，接收方应及时发送对M2的重复确认，这样做可以让发送方及早知道报文段M3没有到达接收方。发送方接着发送了M5和M6。接收方收到这两个报文后，也还要再次发出对M2的重复确认。这样，发送方共收到了接收方的四个对M2的确认，其中后三个都是重复确认。快重传算法还规定，发送方只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段M3，而不必继续等待M3设置的重传计时器到期。由于发送方尽早重传未被确认的报文段，因此采用快重传后可以使整个网络吞吐量提高约20%。

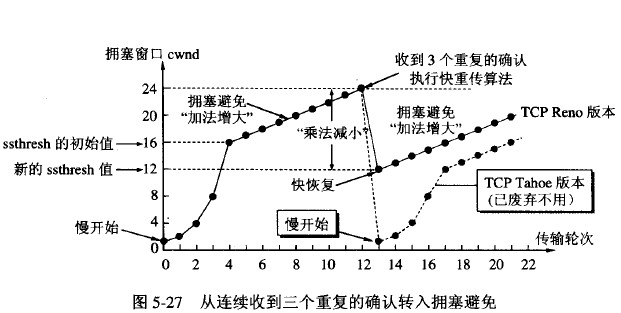
  与快重传配合使用的还有快恢复算法，其过程有以下两个要点：

    <1>. 当发送方连续收到三个重复确认，就执行“乘法减小”算法，把慢开始门限ssthresh减半。这是为了预防网络发生拥塞。请注意：接下去不执行慢开始算法。

    <2>. 由于发送方现在认为网络很可能没有发生拥塞，因此与慢开始不同之处是现在不执行慢开始算法（即拥塞窗口cwnd现在不设置为1），而是把cwnd值设置为慢开始门限ssthresh减半后的数值，然后开始执行拥塞避免算法（“加法增大”），使拥塞窗口缓慢地线性增大。

    下图给出了快重传和快恢复的示意图，并标明了“TCP Reno版本”。

      区别：新的 TCP Reno 版本在快重传之后采用快恢复算法而不是采用慢开始算法。



 也有的快重传实现是把开始时的拥塞窗口cwnd值再增大一点，即等于 ssthresh + 3 X MSS 。这样做的理由是：既然发送方收到三个重复的确认，就表明有三个分组已经离开了网络。这三个分组不再消耗网络 的资源而是停留在接收方的缓存中。可见现在网络中并不是堆积了分组而是减少了三个分组。因此可以适当把拥塞窗口扩大了些。

    在采用快恢复算法时，慢开始算法只是在TCP连接建立时和网络出现超时时才使用。

    采用这样的拥塞控制方法使得TCP的性能有明显的改进。

    接收方根据自己的接收能力设定了接收窗口rwnd，并把这个窗口值写入TCP首部中的窗口字段，传送给发送方。因此，接收窗口又称为通知窗口。因此，从接收方对发送方的流量控制的角度考虑，发送方的发送窗口一定不能超过对方给出的接收窗口rwnd 。

    发送方窗口的上限值 = Min [ rwnd, cwnd ]

    当rwnd < cwnd 时，是接收方的接收能力限制发送方窗口的最大值。

    当cwnd < rwnd 时，则是网络的拥塞限制发送方窗口的最大值。

 七十八、里氏变换

其严格表述如下：如果对每一个类型为S的对象o1，都有类型为T的对象o2，使得以T定义的所有程序P在所有的对象o1代换o2时，程序P的行为没有变化，那么类型S是类型T的子类型。这个定义比较拗口且难以理解，因此我们一般使用它的另一个通俗版定义：

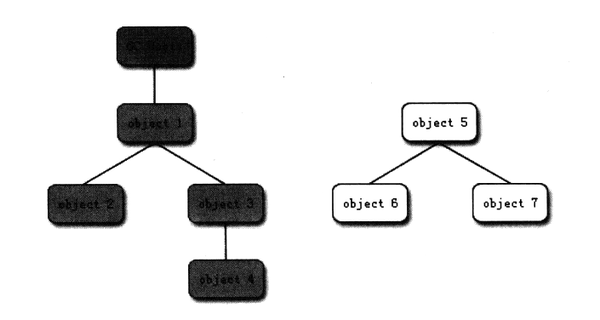
|  |
| --- |
| **里氏代换原则(Liskov Substitution Principle, LSP)：所有引用基类（父类）的地方必须能透明地使用其子类的对象。** |

  里氏代换原则告诉我们，**在软件中将一个基类对象替换成它的子类对象，程序将不会产生任何错误和异常，反过来则不成立，如果一个软件实体使用的是一个子类对象的话，那么它不一定能够使用基类对象。**

   里氏替换原则通俗的来讲就是：**子类可以扩展父类的功能，但不能改变父类原有的功能。**它包含以下4层含义：

* 子类可以实现父类的抽象方法，但不能覆盖父类的非抽象方法。
* 子类中可以增加自己特有的方法。
* 当子类的方法重载父类的方法时，方法的前置条件（即方法的形参）要比父类方法的输入参数更宽松。
* 当子类的方法实现父类的抽象方法时，方法的后置条件（即方法的返回值）要比父类更严格。

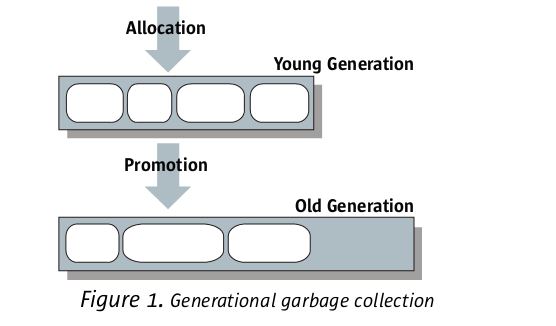
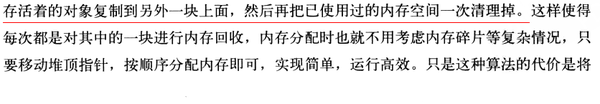
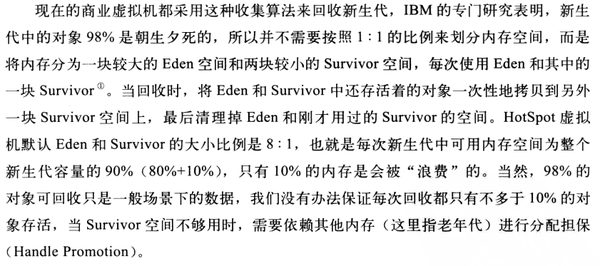
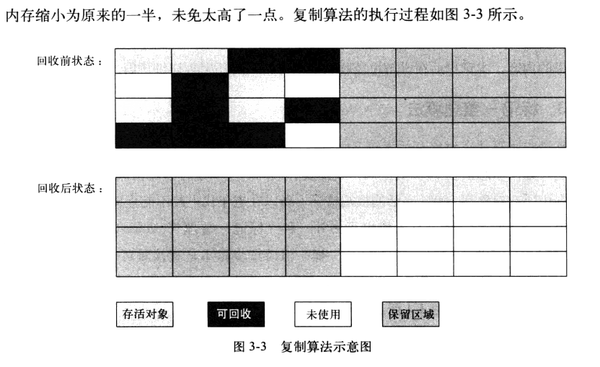
七十九、java立即回收机制

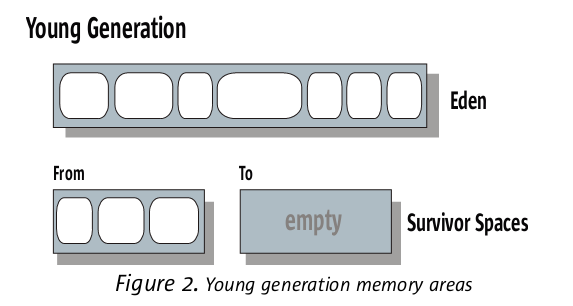
**有哪些方法可以判断一个对象已经可以被回收，JVM怎么判断一个对象已经消亡可以被回收？**  
**①引用计数算法**  
**给对象中添加一个引用计数器，每当有一个地方引用它时，计数器就加1；当引用失效时，计数器值就减1；任何时刻计数器都为0的对象就是不可能再被使用的。**  
**Java语言没有选用引用计数法来管理内存，因为引用计数法不能很好的解决循环引用的问题。**  
**②根搜索算法**  
**在主流的商用语言中，都是使用根搜索算法来判定对象是否存活的。**  
**GC Root Tracing 算法思路就是通过一系列的名为"GC Roots"的对象作为起始点，从这些节点开始向下搜索，搜索所走过的路径称为引用链（Reference Chain），当一个对象到GC Roots没有任何引用链相连，即从GC Roots到这个对象不可达，则证明此对象是不可用的。**

比如上图，左边的对象都是存活的，右边的都是可以回收的。

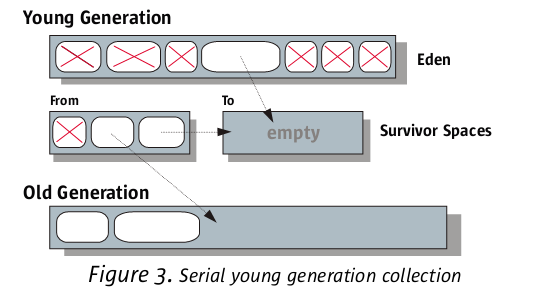
**（4）那些对象可以作为GC Roots？**  
虚拟机栈（栈帧中的本地变量表）中的引用的对象  
方法区中的类静态属性引用的对象  
方法区中的常量引用的对象  
本地方法栈中JNI（Native方法）的引用对象

HotSpot JVM （下简称JVM）的内存管理  
JVM将堆分成了 二个大区 Young 和 Old 如下图：

;而Young 区又分为 而Young 区又分为 **Eden、Servivor1、Servivor2**, 两个**Survivor 区相对地作为为From 和 To 逻辑区域, 当Servivor1作为 From 时 ， Servivor2 就作为 To, 反之亦然**  
关于为什么要这样区分Young（将Young区分为Eden、Servivor1、Servivor2以及相对的From和To ），这要牵涉到JVM的垃圾回收算法的讨论。  
1）因为引用计数法无法解决循环引用问题，JVM并没有采用这种算法来判断对象是否存活。  
2）JVM一般采用GCRoots的方法，只要从任何一个GCRoots的对象可达，就是不被回收的对象  
3）判断了对象生死，怎么进行内存的清理呢？  
4）标记-清除算法，先标记那些要被回收的对象，然后进行清理，简单可行，但是①标记清除效率低，因为要一个一个标记和清除②造成大量不连续的内存碎片，空间碎片太多可能会导致当程序在以后的运行过程中需要分配较大对象的时候无法找到足够的连续内存而不得不触发另一次垃圾收集动作。  
5）采用复制收集算法：将可用内存按照容量分为大小相等的两块，每次只是使用其中的一块。当这一块的内存用完了，就将可用内存中  
  
  
  
上面的过程就解释了为什么我们的Yong内存需要分为三块，Eden，Survivor1，Survivor2以及FROM TO相对使用的用法

FROM TO相对使用的用法  
因此当Eden区满的时候 GC执行，这时会将 Eden 区和 From 区中还被引用的对象会被移到 To区 ，个别大对象和部分From对象在To已满的情况下会被放到Old区,如下图：

因此当Eden区满的时候 GC执行，这时会将 Eden 区和 From 区中还被引用的对象会被移到 To区 ，个别大对象和部分From对象在To已满的情况下会被放到Old区,如下图：

HotSpot VM 内存堆的两个Servivor区

HotSpot VM 内存堆的两个Servivor区

## 八十、谈谈对jvm的理解

JVM是虚拟机，也是一种规范，他遵循着冯·诺依曼体系结构的设计原理。不同的[操作系统](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/jiaomingliang/article/details/_blank)指令集以及[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/jiaomingliang/article/details/_blank)都有着差异，而JVM通过在操作系统上建立虚拟机，自己定义出来的一套统一的数据结构和操作指令，把同一套语言翻译给各大主流的操作系统，实现了跨平台运行，可以说JVM是java的核心，是java可以一次编译到处运行的本质所在。

JVM的组成和运行原理 。

JVM的毕竟是个虚拟机，是一种规范，虽说符合冯诺依曼的计算机设计理念，但是他并不是实体计算机，所以他的组成也不是什么存储器，控制器，运算器，输入输出设备。在我看来，JVM运行在真实的操作系统中表现的更像应用或者说是进程

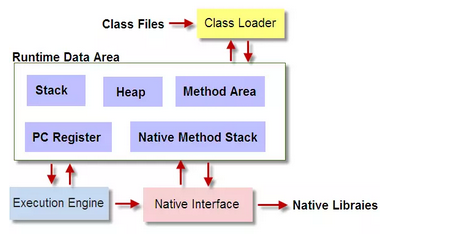
1.JVM在JDK中的位置。

JDK是java开发的必备工具箱，JDK其中有一部分是JRE，JRE是JAVA运行环境，JVM则是JRE最核心的部分。

2.JVM的组成

JVM由4大部分组成：ClassLoader，Runtime Data Area，Execution Engine，Native Interface。

我从CSDN找了一张描述JVM大致结构的图：



2.1. ClassLoader 是负责加载class文件，class文件在文件开头有特定的文件标示，并且ClassLoader只负责class文件的加载，至于它是否可以运行，则由Execution Engine决定。

2.2.Native Interface 是负责调用本地接口的。他的作用是调用不同语言的接口给JAVA用，他会在Native Method Stack中记录对应的本地方法，然后调用该方法时就通过Execution Engine加载对应的本地lib。原本多于用一些专业领域，如JAVA驱动，地图制作引擎等，现在关于这种本地方法接口的调用已经被类似于Socket通信，WebService等方式取代。

2.3.Execution Engine 是执行引擎，也叫Interpreter。Class文件被加载后，会把指令和数据信息放入内存中，Execution Engine则负责把这些命令解释给操作系统。

2.4.Runtime Data Area 则是存放数据的，分为五部分：Stack，Heap，Method Area，PC Register，Native Method Stack。几乎所有的关于java内存方面的问题，都是集中在这块。下图是javapapers.com上关于Run-time Data Areas的描述：

八十、volatile修饰之后就变得不一样了：

第一：使用volatile关键字会强制将修改的值立即写入主存；

第二：使用volatile关键字的话，当线程2进行修改时，会导致线程1的工作内存中缓存变量stop的缓存行无效（反映到硬件层的话，就是CPU的L1或者L2缓存中对应的缓存行无效）；

第三：由于线程1的工作内存中缓存变量stop的缓存行无效，所以线程1再次读取变量stop的值时会去主存读取。