

Java内存可以分为程序计数器、Java虚拟机栈、本地方法栈、Java堆、方法区这几个部分。其中程序计数器、Java虚拟机栈、本地方法栈属于线程私有，而Java堆和方法区属于线程共享区域。对于Java的堆内存可以划分为新生代和老年代。在新生代中划分为一个Eden区和两个Survivor区。创建一个Java对象时，会通过指针碰撞或空闲列表法为对象分配内存。而当我们访问一个对象的时候可以通过句柄或者直接地址的方式进行对象的访问。

垃圾收集算法：

标记-清除；复制；标记-整理；分代整理

内存优化：减少对象的创建，尽量使用局部变量，尽量在finally块中释放资源，没有必要时请不用使用静态变量，尽量不要使用finalize方法，尽量避免非常大的内存分配

Java内存泄露：

(1). 诸如 HashMap、Vector 等集合类的静态使用最容易出现内存泄露，因为这些静态变量的生命周期和应用程序一致，所有的对象Object也不能被释放，因为他们也将一直被Vector等应用着。

(2). 各种资源连接包括数据库连接、网络连接、IO连接等没有显式调用close关闭，不被GC回收导致内存泄露。

(3). 监听器的使用，在释放对象的同时没有相应删除监听器的时候也可能导致内存泄露。

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；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；
5. 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。这是因为，修改性能和检索性能是互相矛盾的。当增加索引时，会提高检索性能，但是会降低修改性能。当减少索引时，会提高修改性能，降低检索性能。因此，当修改性能远远大于检索性能时，不应该创建索引。

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 容器控制了对象，主要控制了外部资源获取（不只是对象包括比如文件等）。

由容器帮我们查找及注入依赖对象，对象只是被动的接受依赖对象，所以是反转；哪些方面反转了？依赖对象的获取被反转了。

DI即“依赖注入”：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。

DI是通过反射来实现的，反射允许程序通过某个类的名字或者对象来得到类本身。spring就是通过反射来完成注入的。

●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）。

Spring中AOP代理由Spring的IOC容器负责生成、管理，其依赖关系也由IOC容器负责管理。因此，AOP代理可以直接使用容器中的其它bean实例作为目标，这种关系可由IOC容器的依赖注入提供。Spring创建代理的规则为：

1、默认使用Java动态代理来创建AOP代理，这样就可以为任何接口实例创建代理了

2、当需要代理的类不是代理接口的时候，Spring会切换为使用CGLIB代理，也可强制使用CGLIB

  Spring AOP采用的是动态代理，在运行期间对业务方法进行增强，所以不会生成新类，对于动态代理技术，Spring AOP提供了对JDK动态代理的支持以及CGLib的支持。

       JDK动态代理只能为接口创建动态代理实例，而不能对类创建动态代理。需要获得被目标类的接口信息（应用Java的反射技术），生成一个实现了代理接口的动态代理类（字节码），再通过反射机制获得动态代理类的构造函数，利用构造函数生成动态代理类的实例对象，在调用具体方法前调用invokeHandler方法来处理。

inverse和cascade这两个的作用就已经讲解完了

1、inverse的权限在cascade之上，意思就是cascade是否有用，还得看inverse这个属性

2、inverse的作用：在映射关系中，让其中一方去维护关系，好处就是能提高性能，不用重复维护。维护两种关系，看下

　2.1 控制级联关系是否有效

　cascade是否有效，就得看inserve的值，如果是自己方来维护关系，那么cascade就有效，反之无效

　2.2 控制外键关系

这个就得通过让自己拥有对方的实例引用(可能是set，也可能就是单个存储对象的变量)，这样才具备控制外键关系的能力，然后看inserve的值，

3、inverse只能在一的一方设置，并且默认值是true，也就是说，不设置inverse时，默认是让多的一方去维护关系，这种一般是在双向、外键关系中才设置inverse的值，如果是单向的，就只有一方有维护关系的权利。

TCP与UDP区别总结：

1、TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接

2、TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付

Tcp通过校验和，重传控制，序号标识，滑动窗口、确认应答实现可靠传输。如丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。

3、UDP具有较好的实时性，工作效率比TCP高，适用于对高速传输和实时性有较高的通信或广播通信。

4.每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信

5、TCP对系统资源要求较多，UDP对系统资源要求较少。

6. TCP是面向字节流的，UDP是面向报文的；

为什么UDP有时比TCP更有优势?

UDP以其简单、传输快的优势，在越来越多场景下取代了TCP,如实时游戏。

（1）网速的提升给UDP的稳定性提供可靠网络保障，丢包率很低，如果使用应用层重传，能够确保传输的可靠性。

（2）TCP为了实现网络通信的可靠性，使用了复杂的拥塞控制算法，建立了繁琐的握手过程，由于TCP内置的系统协议栈中，极难对其进行改进。

采用TCP，一旦发生丢包，TCP会将后续的包缓存起来，等前面的包重传并接收到后再继续发送，延时会越来越大，基于UDP对实时性要求较为严格的情况下，采用自定义重传机制，能够把丢包产生的延迟降到最低，尽量减少网络问题对游戏性造成影响。

Get与POST的区别

从功能上讲，GET一般用来从服务器上获取资源，POST一般用来更新服务器上的资源；

从请求参数形式上看，GET请求的数据会附在URL之后，即将请求数据放置在HTTP报文的 请求头 中，以?分割URL和传输数据，参数之间以&相连。而POST请求会把提交的数据则放置在是HTTP请求报文的 请求体 中。

就安全性而言，POST的安全性要比GET的安全性高，因为GET请求提交的数据将明文出现在URL上，而且POST请求参数则被包装到请求体中，相对更安全。

从请求的大小看，GET请求的长度受限于浏览器或服务器对URL长度的限制，允许发送的数据量比较小，而POST请求则是没有大小限制的。

从输入网址到获得页面的过程

　　(1). 浏览器查询 DNS，获取域名对应的IP地址:具体过程包括浏览器搜索自身的DNS缓存、搜索操作系统的DNS缓存、读取本地的Host文件和向本地DNS服务器进行查询等。对于向本地DNS服务器进行查询，如果要查询的域名包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析(此解析具有权威性)；如果要查询的域名不由本地DNS服务器区域解析，但该服务器已缓存了此网址映射关系，则调用这个IP地址映射，完成域名解析（此解析不具有权威性）。如果本地域名服务器并未缓存该网址映射关系，那么将根据其设置发起递归查询或者迭代查询；

　　(2). 浏览器获得域名对应的IP地址以后，浏览器向服务器请求建立链接，发起三次握手；

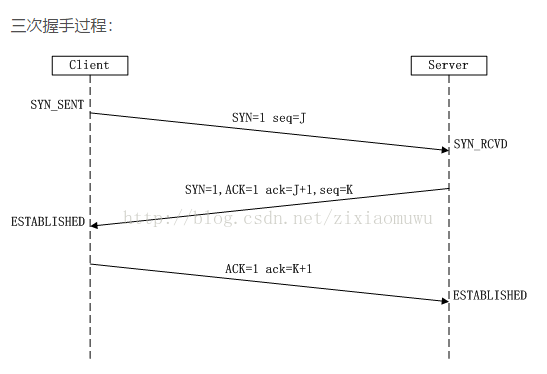
　　(3). TCP/IP链接建立起来后，浏览器向服务器发送HTTP请求；

　　(4). 服务器接收到这个请求，并根据路径参数映射到特定的请求处理器进行处理，并将处理结果及相应的视图返回给浏览器；

　　(5). 浏览器解析并渲染视图，若遇到对js文件、css文件及图片等静态资源的引用，则重复上述步骤并向服务器请求这些资源；

　　(6). 浏览器根据其请求到的资源、数据渲染页面，最终向用户呈现一个完整的页面。

TCP三次握手建立连接，四次挥手断开连接



问题1： 为什么要三次握手（我要和你建立链接，你真的要和我建立链接么，我真的要和你建立链接）？

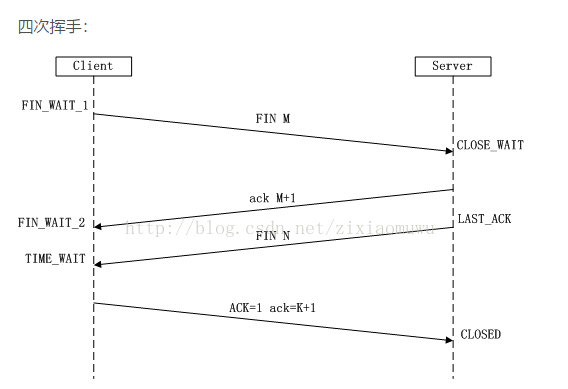
答：三次握手的目的是建立可靠的通信信道，说到通讯，简单来说就是数据的发送与接收，而三次握手最主要的目的就是双方确认自己与对方的发送与接收机能正常。

        第一次握手：Client什么都不能确认；Server确认了对方发送正常

        第二次握手：Client确认了：自己发送、接收正常，对方发送、接收正常；Server确认了：自己接收正常，对方发送正常

        第三次握手：Client确认了：自己发送、接收正常，对方发送、接收正常；Server确认了：自己发送、接收正常，对方发送接收正常

所以三次握手就能确认双发收发功能都正常，缺一不可。



为什么连接的时候是三次握手，关闭的时候却是四次握手(我要和你断开链接；好的，断吧。我也要和你断开链接；好的，断吧)？

答：因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步握手。

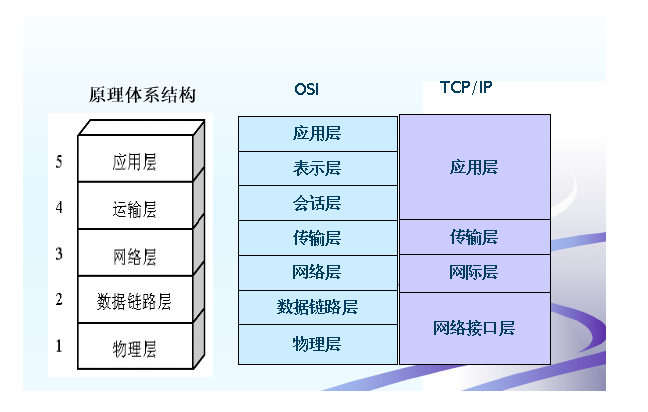
Session 与 Cookie 的对比

实现机制：Session的实现常常依赖于Cookie机制，通过Cookie机制回传SessionID；

大小限制：Cookie有大小限制并且浏览器对每个站点也有cookie的个数限制，Session没有大小限制，理论上只与服务器的内存大小有关；

安全性：Cookie存在安全隐患，通过拦截或本地文件找得到cookie后可以进行攻击，而Session由于保存在服务器端，相对更加安全；

服务器资源消耗：Session是保存在服务器端上会存在一段时间才会消失，如果session过多会增加服务器的压力。



corePollSize：核心线程数。在创建了线程池后，线程中没有任何线程，等到有任务到来时才创建线程去执行任务。默认情况下，在创建了线程池后，线程池中的线程数为0，当有任务来之后，就会创建一个线程去执行任务，当线程池中的线程数目达到corePoolSize后，就会把到达的任务放到缓存队列当中。

maximumPoolSize：最大线程数。表明线程中最多能够创建的线程数量。

keepAliveTime：空闲的线程保留的时间。

TimeUnit：空闲线程的保留时间单位。

BlockingQueue<Runnable>：阻塞队列，存储等待执行的任务。参数有ArrayBlockingQueue、LinkedBlockingQueue、SynchronousQueue可选。

ThreadFactory：线程工厂，用来创建线程

RejectedExecutionHandler：队列已满，而且任务量大于最大线程的异常处理策略