## https://www.jianshu.com/p/ce1fb8497883

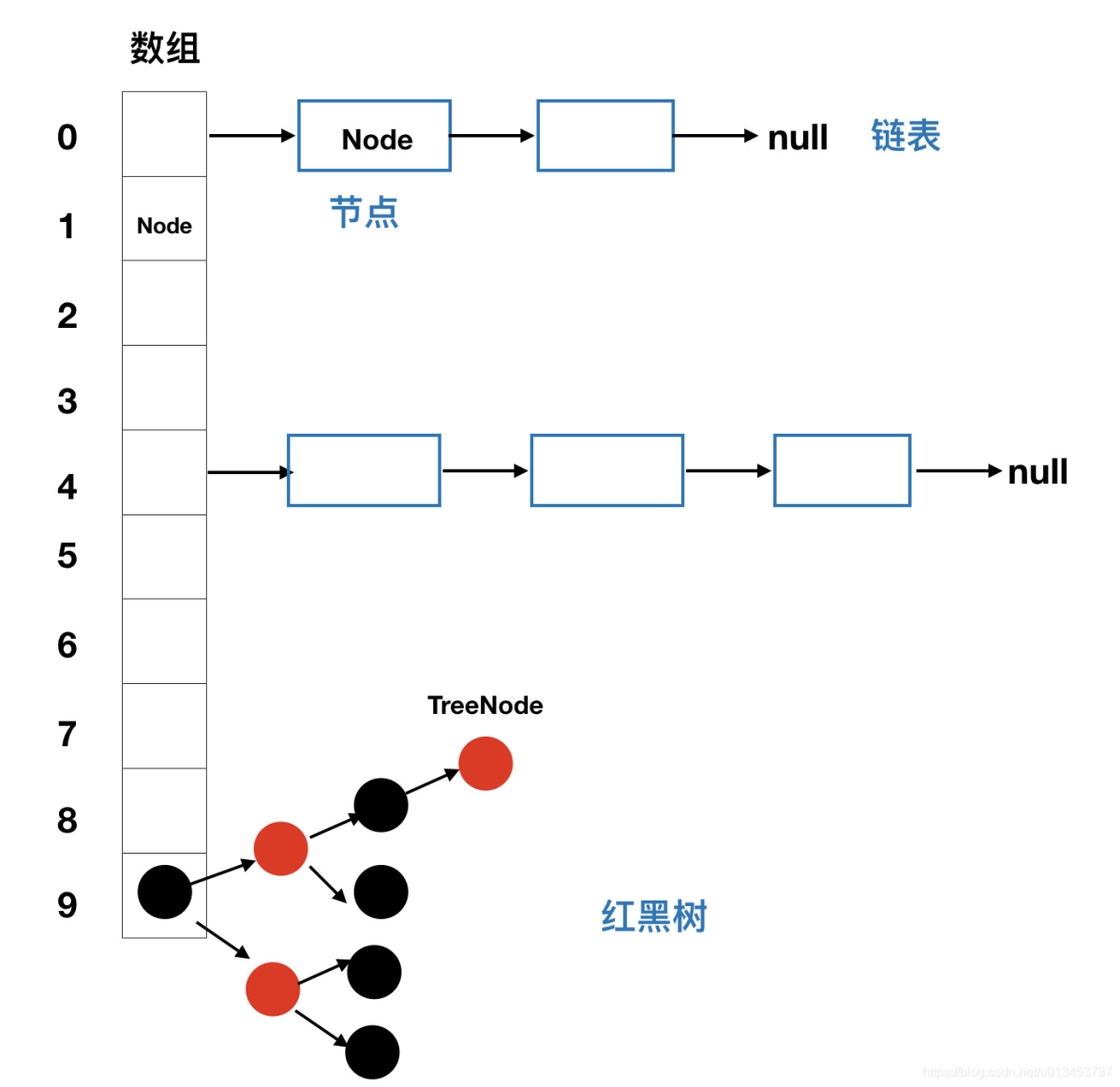
# Java基础

## 1.Collections.sort排序内部原理

在Java 6中Arrays.sort()和Collections.sort()使用的是MergeSort，而在Java 7中，内部实现换成了TimSort(归并排序的改良版)，其对对象间比较的实现要求更加严格

## 2.hashMap原理，java8做的改变

从结构实现来讲，HashMap是数组+链表+红黑树（JDK1.8增加了红黑树部分）实现的。HashMap最多只允许一条记录的键为null，允许多条记录的值为null。HashMap非线程安全。ConcurrentHashMap线程安全。解决碰撞：当出现冲突时，运用拉链法，将关键词为同义词的结点链接在一个单链表中，散列表长m，则定义一个由m个头指针组成的指针数组T，地址为i的结点插入以T(i)为头指针的单链表中。Java8中，冲突的元素超过限制（8），用红黑树替换链表。



## 3.String 和 StringBuilder 的区别

## 4.Vector 与 Array 的区别

1）ArrayList在内存不够时默认是扩展50% + 1个，Vector是默认扩展1倍。

2）Vector属于线程安全级别的，但是大多数情况下不使用Vector，因为线程安全需要更大的系统开销。

## 5.HashMap 与 Hashtable 的区别

1） 历史原因: Hashtable继承Dictonary类, HashMap继承自abstractMap

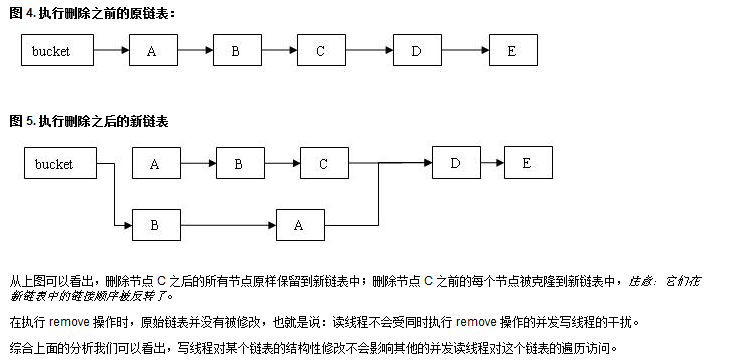
2） HashMap允许空的键值对, 但最多只有一个空对象，而HashTable不允许。

3） HashTable同步，而HashMap非同步，效率上比HashTable要高

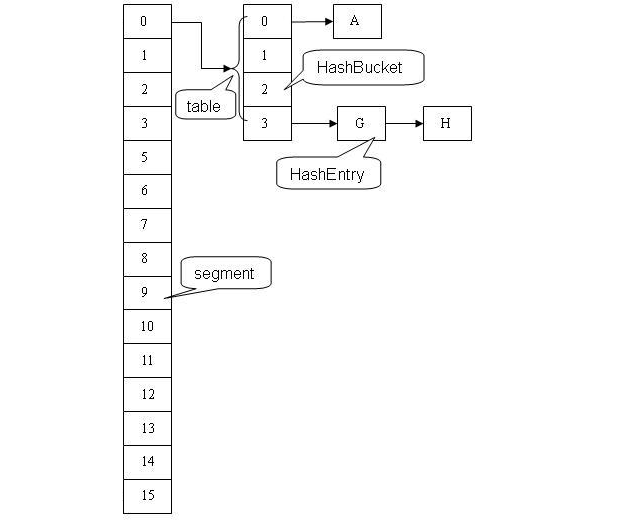
## 6.ConncurrentHashMap和hashtable比较（两个线程并发访问map中同一条链，一个线程在尾部删除，一个线程在前面遍历查找，问为什么前面的线程还能正确的查找到后面被另一个线程删除的节点）

ConcurrentHashMap融合了hashtable和hashmap二者的优势。hashtable是做了同步的，即线程安全，hashmap未考虑同步。所以hashmap在单线程情况下效率较高。hashtable在的多线程情况下，同步操作能保证程序执行的正确性。但是hashtable是阻塞的，每次同步执行的时候都要锁住整个结构，ConcurrentHashMap正是为了解决这个问题而诞生的，

ConcurrentHashMap允许多个修改操作并发进行，其关键在于使用了锁分离技术（一个Array保存多个Object，使用这些对象的锁作为分离锁，get/put时随机使用任意一个）。它使用了多个锁来控制对hash表的不同部分进行的修改。在JDK 1.6中，有HashEntry结构存在，每次插入将新添加节点作为链的头节点（同HashMap实现），而且每次删除一个节点时，会将删除节点之前的所有节点拷贝一份组成一个新的链，而将当前节点的上一个节点的next指向当前节点的下一个节点，从而在删除以后有两条链存 在，因而可以保证即使在同一条链中，有一个线程在删除，而另一个线程在遍历，它们都能工作良好，因为遍历的线程能继续使用原有的链。



Java8中，采用volatile HashEntry保存数据，table元素作为锁；从table数组+单向链表加上了红黑树。红黑树是一种特别的二叉查找树，特性为：1.节点为红或者黑 2.根节点为黑 3.叶节点为黑 4.一节点为红，则叶节点为黑 5.一节点到其子孙节点所有路径上的黑节点数目相同。

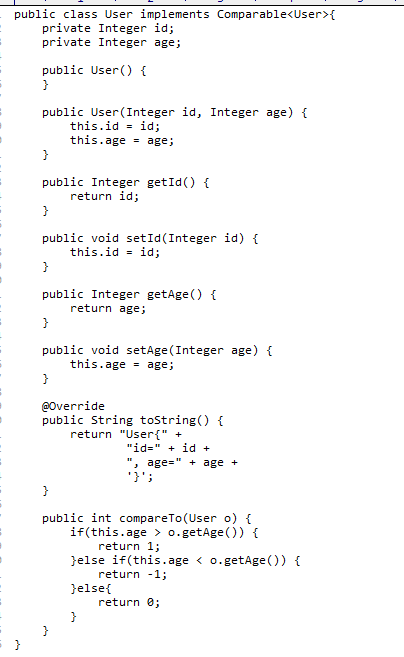
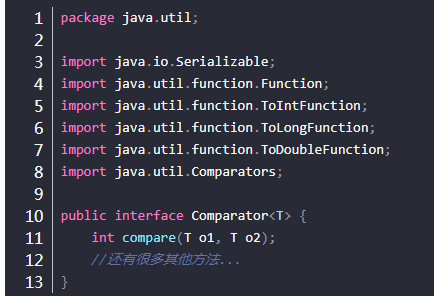


ConcurrentHashMap 类中包含两个静态内部类 HashEntry 和 Segment。HashEntry 用来封装映射表的键 / 值对；Segment 用来充当锁的角色，每个 Segment 对象守护整个散列映射表的若干个桶

## 7.ArrayList与 LinkedList 的区别？

## 8.Java 中，Comparator 与Comparable 有什么不同？

Comparable 接口用于定义对象的自然顺序，是排序接口，而 comparator 通常用于定义用户定制的顺序，是比较接口。我们如果需要控制某个类的次序，而该类本身不支持排序(即没有实现Comparable接口)，那么我们就可以建立一个“该类的比较器”来进行排序。Comparable 总是只有一个，但是可以有多个 comparator 来定义对象的顺序。

## 9.抽象类是什么？它与接口有什么区别？你为什么要使用过抽象类？

## 10.描述 Java 中的重载和重写？

重载和重写都允许你用相同的名称来实现不同的功能，但是重载是编译时活动，而重写是运行时活动。你可以在同一个类中重载方法，但是只能在子类中重写方法。重写必须要有继承

重写：1、在子类中可以根据需要对从基类中继承来的方法进行重写。2、重写的方法和被重写的方法必须具有相同方法名称、参数列表和返回类型。3、重写方法不能使用比被重写的方法更严格的访问权限。

重载的时候，方法名要一样，但是参数类型和个数不一样，返回值类型可以相同也可以不相同。无法以返回型别作为重载函数的区分标准。

## 11.Collection与Collections的区别是什么？

Collection是Java集合框架中的基本接口；

Collections是Java集合框架提供的一个工具类，其中包含了大量用于操作或返回集合的静态方法。

## 12.Java中多态的实现原理

所谓多态，指的就是父类引用指向子类对象，调用方法时会调用子类的实现而不是父类的实现。多态的实现的关键在于“动态绑定”。

## 13.object中定义了哪些方法？

clone(), equals(), hashCode(), toString(),

notify(), notifyAll(),wait(),

finalize(), getClass()

## 15.说出 5 个 JDK 1.8 引入的新特性？

Java 8 在 Java 历史上是一个开创新的版本，下面 JDK 8 中 5 个主要的特性：

Lambda 表达式；允许像对象一样传递匿名函数 Stream API，充分利用现代多核 CPU，可以写出很简洁的代码 ；Date 与 Time API，最终，有一个稳定、简单的日期和时间库可供你使用 扩展方法，现在，接口中可以有静态、默认方法； 重复注解，现在你可以将相同的注解在同一类型上使用多次。

## 16.java中public，private，protected以及默认关键字的访问范围：

Protected可在包内及包外子类访问，default只能同一包内访问，prvate只能同一类

## 19. 匿名内部类是什么？如何访问在其外面定义的变量？

匿名内部类也就是没有名字的内部类，匿名内部类只能使用一次，它通常用来简化代码编写。

匿名内部类只能访问外部类的Final变量. Java 8更加智能：如果局部变量被匿名内部类访问，那么该局部变量相当于自动使用了final修饰。

## 20. 如何创建单例模式？说了双重检查，他说不是线程安全的。如何高效的创建一个线程安全的单例？

参考：https://www.cnblogs.com/yoga21/p/9224557.html

一种是通过枚举，一种是通过静态内部类。

## 21.poll() 方法和 remove() 方法的区别？

poll() 和

remove() 都是从队列中取出一个元素，但是 poll() 在获取元素失败的时候会返回空，但是 remove() 失败的时候会抛出异常。

## 22.写一段代码在遍历 ArrayList 时移除一个元素

# JVM

## 1.JVM如何加载一个类的过程，双亲委派模型中有哪些方法

参考：https://blog.csdn.net/djokermax/article/details/81539639

类加载过程：加载、验证（验证阶段作用是保证Class文件的字节流包含的信息符合JVM规范，不会给JVM造成危害）、准备（准备阶段为变量分配内存并设置类变量的初始化）、解析（解析过程是将常量池内的符号引用替换成直接引用）、初始化。

双亲委派模型中方法：双亲委派是指如果一个类收到了类加载的请求，不会自己先尝试加载，先找父类加载器去完成。当顶层启动类加载器表示无法加载这个类的时候，子类才会尝试自己去加载。当回到最开的发起者加载器还无法加载时，并不会向下找，而是抛出ClassNotFound异常。

方法：启动（Bootstrap）类加载器，标准扩展（Extension）类加载器，应用程序类加载器（Application ），上下文类加载器（context class loader）。意义是防止内存中出现多份同样的字节码 。

## 2.GC算法（什么样的对象算是可回收对象，可达性分析），CMS收集器

jvm是如何判断一个对象已经变成了可回收的“垃圾”，一般是两个方法：引用记数法和根搜索算法。引用记数法没办法解决循环引用的问题，所以用根搜索。从一系列的”GC Roots“对象开始向下搜索，搜索走过的路径称为引用链。当一个对象到”GC Roots“之间没有引用链时，被称为引用不可达。引用不可到的对象被认为是可回收的对象。

几种垃圾收集器：1，Serial New/Serial Old(串行)，2，Parrallel New (并行)，3，Parrallel Scavenge，4，Parrallel Old，5，CMS（CMS收集器是一个以获得最短回收停顿时间为目标的收集器，它是一种并发收集器，采用的是Mark-sweep算法。），6，G1（是一款并行与并发收集器，并且可建立可预测的停顿时间模型，整体上是基于标记清理，局部采用复制）

## 3.JVM分为哪些区，每一个区干吗的？

1）方法区(method)：被所有的线程共享。方法区包含所有的类信息和静态变量。

2）堆(heap)：被所有的线程共享，存放对象实例以及数组，Java堆是GC的主要区域。

3）栈(stack)：每个线程包含一个栈区，栈中保存一些局部变量等。

4）程序计数器：是当前线程执行的字节码的行指示器。

## 4.JVM新生代，老年代，持久代，都存储哪些东西？

持久代主要存放的是Java类的类信息，与垃圾收集要收集的Java对象关系不大。所有新生成的对象首先都是放在年轻代的，年老代中存放的都是一些生命周期较长的对象。

## 5.内存溢出和内存泄漏：

内存溢出：程序申请内存时，没有足够的内存，out of memory；内存泄漏值垃圾对象无法回收，可以使用memory analyzer工具查看泄漏。

## 6.进程与线程：

进程值运行中的程序（独立性，动态性，并发性），线程指进程中的顺序执行流。

进程是资源分配的最小单位,线程是程序执行的最小单位

区别是：1.进程间不共享内存 2.创建进程进行资源分配的代价要大得多，所以多线程在高并发环境中效率高。

## 7.序列化与反序列化：

序列化指将java对象转化为字节序列，反序列化相反。主要是为了java线程间通讯，实现对象传递。只有实现了Serializable或Externalizable接口类对象才可被序列化。

## 8.64 位 JVM 中，int 的长度是多数？

Java 中，int 类型变量的长度是一个固定值，与平台无关，都是 32 位。意思就是说，在 32 位 和 64 位 的Java 虚拟机中，int 类型的长度是相同的。

## 9.Java 中 WeakReference 与 SoftReference的区别？

Java中一共有四种类型的引用。StrongReference、 SoftReference、 WeakReference 以及 PhantomReference。

StrongReference 是 Java 的默认引用实现, 它会尽可能长时间的存活于 JVM 内，当没有任何对象指向它时将会被GC回收

WeakReference，顾名思义, 是一个弱引用, 当所引用的对象在

JVM 内不再有强引用时, 将被GC回收

虽然 WeakReference 与 SoftReference 都有利于提高 GC 和 内存的效率，但是 WeakReference ，一旦失去最后一个强引用，就会被 GC 回收，而 SoftReference 会尽可能长的保留引用直到 JVM 内存不足时才会被回收(虚拟机保证), 这一特性使得

SoftReference 非常适合缓存应用

## 10.解释 Java 堆空间及 GC？

当通过 Java 命令启动

Java 进程的时候，会为它分配内存。内存的一部分用于创建堆空间，当程序中创建对象的时候，就从对空间中分配内存。GC 是 JVM 内部的一个进程，回收无效对象的内存用于将来的分配。

## 11.Java 中堆和栈有什么区别？

JVM 中堆和栈属于不同的内存区域，使用目的也不同。栈常用于保存方法帧和局部变量，而对象总是在堆上分配。栈通常都比堆小，也不会在多个线程之间共享，而堆被整个 JVM 的所有线程共享。

# 并发，锁

## 1.volatile关键字， Lock

并发编程中：原子性问题，可见性问题，有序性问题。

volatile关键字能保证可见性，字能禁止指令重排序，但是不能保证原子性。可见性只能保证每次读取的是最新的值，但是volatile没办法保证对变量的操作的原子性。在生成的会变语句中加入Lock关键字和内存屏障。

（volatile关键字的作用：保证了变量的可见性（visibility）。被volatile关键字修饰的变量，如果值发生了变更，其他线程立马可见，避免出现脏读的现象。

参考：https://www.cnblogs.com/YLsY/p/11295732.html

Java内存模型规定所有的变量都是存在主存当中，每个线程都有自己的工作内存。线程对变量的所有操作都必须在工作内存中进行，而不能直接对主存进行操作。并且每个线程不能访问其他线程的工作内存。变量的值何时从线程的工作内存写回主存，无法确定。）

Lock 实现提供了比使用synchronized 方法和语句可获得的更广泛的锁定操作，它能以更优雅的方式处理线程同步问题。用sychronized修饰的方法或者语句块在代码执行完之后锁自动释放，而用Lock需要我们手动释放锁

## 2.MYSQL常用优化（sql优化，表结构优化等）

SQL优化、表机构优化、索引优化、缓存参数优化

## SQL优化

**1.减少 IO 次数**

　　IO永远是数据库最容易瓶颈的地方，这是由数据库的职责所决定的，大部分数据库操作中超过90%的时间都是 IO 操作所占用的，减少 IO 次数是 SQL 优化中需要第一优先考虑，当然，也是收效最明显的优化手段。

**2.降低 CPU 计算**

　　除了 IO 瓶颈之外，SQL优化中需要考虑的就是 CPU 运算量的优化了。order by, group by,distinct … 都是消耗 CPU 的大户(这些操作基本上都是 CPU 处理内存中的数据比较运算)。当我们的 IO 优化做到一定阶段之后，降低 CPU 计算也就成为了我们 SQL 优化的重要目标

**优化方法：**

改变 SQL 执行计划

### 表结构优化

https://www.cnblogs.com/eeds-wangwei/p/8135771.html

一、**数据类型选择**

数据库操作中最为耗时的操作就是 IO 处理，大部分数据库操作 90% 以上的时间都花在了 IO 读写上面。所以尽可能减少 IO 读写量，可以在很大程度上提高数据库操作的性能。

我们无法改变数据库中需要存储的数据，但是我们可以在这些数据的存储方式方面花一些心思。下面的这些关于字段类型的优化建议主要适用于记录条数较多，数据量较大的场景，因为精细化的数据类型设置可能带来维护成本的提高，过度优化也可能会带来其他的问题：

　　1.数字类型：非万不得已不要使用DOUBLE，不仅仅只是存储长度的问题，同时还会存在精确性的问题。同样，固定精度的小数，也不建议使用DECIMAL，建议乘以固定倍数转换成整数存储，可以大大节省存储空间，且不会带来任何附加维护成本。对于整数的存储，在数据量较大的情况下，建议区分开 TINYINT / INT / BIGINT 的选择，因为三者所占用的存储空间也有很大的差别，能确定不会使用负数的字段，建议添加unsigned定义。当然，如果数据量较小的数据库，也可以不用严格区分三个整数类型。

　　2.字符类型：非万不得已不要使用 TEXT 数据类型，其处理方式决定了他的性能要低于char或者是varchar类型的处理。定长字段，建议使用 CHAR 类型，不定长字段尽量使用 VARCHAR，且仅仅设定适当的最大长度，而不是非常随意的给一个很大的最大长度限定，因为不同的长度范围，MySQL也会有不一样的存储处理。

　　3.时间类型：尽量使用TIMESTAMP类型，因为其存储空间只需要 DATETIME 类型的一半。对于只需要精确到某一天的数据类型，建议使用DATE类型，因为他的存储空间只需要3个字节，比TIMESTAMP还少。不建议通过INT类型类存储一个unix timestamp 的值，因为这太不直观，会给维护带来不必要的麻烦，同时还不会带来任何好处。

　　4.ENUM & SET：对于状态字段，可以尝试使用 ENUM 来存放，因为可以极大的降低存储空间，而且即使需要增加新的类型，只要增加于末尾，修改结构也不需要重建表数据。如果是存放可预先定义的属性数据呢?可以尝试使用SET类型，即使存在多种属性，同样可以游刃有余，同时还可以节省不小的存储空间。

　　5.LOB类型：强烈反对在数据库中存放 LOB 类型数据，虽然数据库提供了这样的功能，但这不是他所擅长的，我们更应该让合适的工具做他擅长的事情，才能将其发挥到极致。在数据库中存储 LOB 数据就像让一个多年前在学校学过一点Java的营销专业人员来写 Java 代码一样。

二、**字符编码**

字符集直接决定了数据在MySQL中的存储编码方式，由于同样的内容使用不同字符集表示所占用的空间大小会有较大的差异，所以通过使用合适的字符集，可以帮助我们尽可能减少数据量，进而减少IO操作次数。

　　1.纯拉丁字符能表示的内容，没必要选择 latin1 之外的其他字符编码，因为这会节省大量的存储空间。

　　2.如果我们可以确定不需要存放多种语言，就没必要非得使用UTF8或者其他UNICODE字符类型，这回造成大量的存储空间浪费。

　　3.MySQL的数据类型可以精确到字段，所以当我们需要大型数据库中存放多字节数据的时候，可以通过对不同表不同字段使用不同的数据类型来较大程度减小数据存储量，进而降低 IO 操作次数并提高缓存命中率。

三、**适当拆分**

有些时候，我们可能会希望将一个完整的对象对应于一张数据库表，这对于应用程序开发来说是很有好的，但是有些时候可能会在性能上带来较大的问题。

　　当我们的表中存在类似于 TEXT 或者是很大的 VARCHAR类型的大字段的时候，如果我们大部分访问这张表的时候都不需要这个字段，我们就该义无反顾的将其拆分到另外的独立表中，以减少常用数据所占用的存储空间。这样做的一个明显好处就是每个数据块中可以存储的数据条数可以大大增加，既减少物理 IO 次数，也能大大提高内存中的缓存命中率。

　　上面几点的优化都是为了减少每条记录的存储空间大小，让每个数据库中能够存储更多的记录条数，以达到减少 IO 操作次数，提高缓存命中率。下面这个优化建议可能很多开发人员都会觉得不太理解，因为这是典型的反范式设计，而且也和上面的几点优化建议的目标相违背。

四、**适度冗余**

为什么我们要冗余?这不是增加了每条数据的大小，减少了每个数据块可存放记录条数吗?

　　确实，这样做是会增大每条记录的大小，降低每条记录中可存放数据的条数，但是在有些场景下我们仍然还是不得不这样做：

　　1.被频繁引用且只能通过 Join 2张(或者更多)大表的方式才能得到的独立小字段。

　　2.这样的场景由于每次Join仅仅只是为了取得某个小字段的值，Join到的记录又大，会造成大量不必要的 IO，完全可以通过空间换取时间的方式来优化。不过，冗余的同时需要确保数据的一致性不会遭到破坏，确保更新的同时冗余字段也被更新。

五、**尽量使用 NOT NULL**

NULL 类型比较特殊，SQL 难优化。虽然 MySQL NULL类型和 Oracle 的NULL 有差异，会进入索引中，但如果是一个组合索引，那么这个NULL 类型的字段会极大影响整个索引的效率。此外，NULL 在索引中的处理也是特殊的，也会占用额外的存放空间。

　　很多人觉得 NULL 会节省一些空间，所以尽量让NULL来达到节省IO的目的，但是大部分时候这会适得其反，虽然空间上可能确实有一定节省，倒是带来了很多其他的优化问题，不但没有将IO量省下来，反而加大了SQL的IO量。所以尽量确保 DEFAULT 值不是 NULL，也是一个很好的表结构设计优化习惯。

**3.java每改一点都需要重新编译打包部署，有没有更好的方法**

可以使用热加载

**4.进程间通信有哪几种方式？**

1）管道（Pipe），2）命名管道（named pipe），3）信号（Signal），4）消息（Message）队列，5）共享内存，6）内存映射（mapped memory），7）信号量（semaphore），8）套接口（Socket）

https://blog.csdn.net/truong/article/details/39228249/

PipedOutputStream pos =new PipedOutputStream();

PipedInputStream pis =new PipedInputStream();

方法一 通过访问共享变量的方式(注:需要处理同步问题)  
方法二 通过管道流

**5.Sychronized修饰静态方法，锁定类本身而不是实例，非静态方法锁定实例。**

**6. 操作系统什么情况下会死锁？**

所谓死锁：是指多个进程在运行过程中因争夺资源而造成的一种僵局。产生的原因：竞争资源：当系统中多个进程使用共享资源，并且资源不足以满足需要，会引起进程对资源的竞争而产生死锁。进程间推进的顺序非法：请求和释放资源的顺序不当，也同样会导致产生进程死锁

**7.产生死锁的四个条件：**

1.互斥条件（进程独占资源）2.请求与保持（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（若干进程之间形成一种头尾相接的循环等待资源关系）

**8. 如何理解分布式锁？**

由于在平时的工作中，线上服务器是分布式多台部署的，经常会面临解决分布式场景下数据一致性的问题，那么就要利用分布式锁来解决这些问题。

**9. 线程同步与阻塞的关系？同步一定阻塞吗？阻塞一定同步吗？**

线程同步与否 跟 阻塞非阻塞没关系，同步是个过程，阻塞是线程的一种状态。多个线程操作共享变量时可能会出现竞争。这时需要同步来防止两个以上的线程同时进入临界区内，在这个过程中后进入临界区的线程将阻塞，等待先进入的线程走出临界区。

**10. 同步和异步有什么区别？**

同步和异步最大的区别就在于。一个需要等待，一个不需要等待。同步可以避免出现死锁，读脏数据的发生，一般共享某一资源的时候用，如果每个人都有修改权限，同时修改一个文件，有可能使一个人读取另一个人已经删除的内容，就会出错，同步就会按顺序来修改。

**11. 线程池**

根据系统自身的环境情况，有效的限制执行线程的数量，使得运行效果达到最佳。线程主要是通过控制执行的线程的数量，超出数量的线程排队等候，等待有任务执行完毕，再从队列最前面取出任务执行

**12. 如何调用 wait（）方法？使用 if 块还是循环？为什么？**

wait() 方法应该在循环调用，因为当线程获取到 CPU 开始执行的时候，其他条件可能还没有满足，所以在处理前，循环检测条件是否满足会更好。

wait()，notify（）和notifyall（）方法是java.lang.Object类为线程提供的用于实现线程间通信的同步控制方法。等待或者唤醒

**13. 实现线程的几种方法**

(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

**14. 什么是多线程环境下的伪共享（false sharing）？**

伪共享是多线程系统（每个处理器有自己的局部缓存）中一个众所周知的性能问题。缓存系统中是以缓存行（cache line）为单位存储的。缓存行是2的整数幂个连续字节，一般为32-256个字节。最常见的缓存行大小是64个字节。当多线程修改互相独立的变量时，如果这些变量共享同一个缓存行，就会无意中影响彼此的性能，这就是伪共享。