1. **HashMap链表长度大于8后一定会转化为红黑树吗**  
    不一定 ，

实际上转换红黑树有个大前提,就是当前hash table的长度也就是HashMap的capacity(不是size)不能小于64.小于64就只是做个扩容.在不小于64的情况下,标题中的说法还有个不严谨的地方就是,在链表的长度为8时(准确的说是长度为7并且在继续塞第8个时),转换成红黑树,而不是超过8.

2**.如场景是, HashMap初始数组长度为8,只有第一个元素中有值,且有八个都在一条链表上 ,其他数组其他七个位置都是空的,这时候为什么要转化为红黑树而不是存放到数组中呢?**

**3. ConcurrentMashMap分段锁之间加了什么锁,读写之间是互斥的吗?**

**4.三次握手后如果一直不连接会怎样**  
  
**5. volatile的实现原理是怎么样的,它是怎么实现从内存中读取的?**  
 volatile可以保证线程可见性且提供了一定的有序性，但是无法保证原子性。在JVM底层volatile是采用“内存屏障”来实现的。观察加入volatile关键字和没有加入volatile关键字时所生成的汇编代码发现，加入volatile关键字时，会多出一个lock前缀指令，lock前缀指令实际上相当于一个内存屏障（也成内存栅栏），内存屏障会提供3个功能：

I. 它确保指令重排序时不会把其后面的指令排到内存屏障之前的位置，也不会把前面的指令排到内

存屏障的后面；即在执行到内存屏障这句指令时，在它前面的操作已经全部完成；

II. 它会强制将对缓存的修改操作立即写入主存；

III. 如果是写操作，它会导致其他CPU中对应的缓存行无效。

详情见：<https://blog.csdn.net/u012723673/article/details/80682208>

**6. lock锁的使用,它如果是一个接口，它的实现类有哪些,区别是什么?**

Lock与synchronized有以下区别：

Lock是一个接口，而synchronized是关键字。

synchronized会自动释放锁，而Lock必须手动释放锁。

Lock可以让等待锁的线程响应中断，而synchronized不会，线程会一直等待下去。

通过Lock可以知道线程有没有拿到锁，而synchronized不能。

Lock能提高多个线程读操作的效率。

synchronized能锁住类、方法和代码块，而Lock是块范围内的

1. **InnoDB的隔离级别是什么,它能避免幻读吗?怎么避免的?**

<https://www.cnblogs.com/lingqin/p/10279415.html>

SQL标准中定义了4种隔离级别，分别是：

Read uncommitted: 未提交读，事务中的修改，即使没有提交，对其他事务也是可见的。存在脏读

Read committed: 提交读，大多数数据库系统的默认隔离级别(MySQL不是), 一个事务从开始到提交之前，所做的修改对其他事务不可见。解决脏读，存在幻读和不可重复读

repeatable read: 可重复读，该级别保证在同一事务中多次读取同样记录的结果是一致的。解决脏读和不可重复读，理论上存在幻读，但是在InnoDB引擎中解决了幻读

Serializable：可串行化，强制事务串行执行

Repeatable Read(可重复读) 是默认的事务隔离级别，同时该引擎的实现基于多版本的并发控制协议——MVCC (Multi-Version Concurrency Control)，解决了幻读问题，当然 脏读和不可重复读也是不存在的。

**8.索引的底层实现是什么,简单介绍下。**  
  
**9.类加载的双亲委派模型**

**1.介绍一下spring中的AOP**

**2.mybatis和hibernate的区别与优劣**

**3.redis有哪些数据类型**

**4.手写一个单例模式**

**5.用到的技术栈**

**6.如何理解内存泄漏问题?有哪些情况会导致内存泄露?如何解决?**

**7.怎么理解强一致性、单调一致性和最终一致性?**

**8.分布式锁有哪些解决方案?**

**9.如何解决Redis缓存穿透的问题?**

redis **缓存穿透**：直接对存储层操作，失去了缓存层的意义

查询一个数据库中不存在的数据，比如商品详情，查询一个不存在的ID，每次都会访问DB，如果有人恶意破坏，很可能直接对DB造成过大的压力。

解决方案：

1. 当通过某一个key去查询数据的时候，如果对应在数据库中的数据都不存在，将此key对应的value设置为一个默认的值，比如“NULL”，并设置一个缓存的失效时间，这时在缓存失效之前，所有通过此key的访问都被缓存挡住，后面如果此key对应的数据在DB中存在时，缓存失效之后，通过此key再去访问数据，就能拿到新的value了。
2. 常见的则是采用布隆过滤器(可以用很小的内存保留很多的数据)，将所有可能存在的数据哈希到一个足够大的bitmap中，一个一定不存在的数据会被这个bitmap拦截掉，从而避免了对底层存储系统的查询压力。(**布隆过滤器**：实际是一个很长的二进制向量和一系列随机映射函数。布隆过滤器**可以用于检索一个元素是否在一个集合中**。它的**优点**是空间效率和查询时间都远远超过一般的算法，**缺点**是有一定的识别率和删除困难)

**缓存雪崩(缓存失效)**

缓存同一时间大面积的失效，所以后面的请求都会落到数据库上，造成数据库短时间内承受大量请求而崩掉。

解决方案：

1. 将系统中key的缓存失效时间均匀的错开，防止同一时间点有大量的key对应的缓存失效；
2. 重新设计缓存的使用方式，当我们通过key去查询数据时，首先查询缓存，如果此时缓存中查询不到，就通过分布式锁进行加锁，取得锁的进程查DB并设置缓存，然后解锁，其他进程如果发现有锁就等待，然后等解锁后返回缓存数据或者再次查询DB。
3. 尽量保证整个redis集群的高可用性，发现机器宕机尽快补上。
4. 本地encache缓存+hystrix限流&降级，避免mysql崩掉

假如已经崩溃，可以利用redis的持久化机制将保存的数据尽快恢复到缓存里。

**10.Redis hash算法用的是什么?**

**11.Redis集群方案应该怎么做?都有哪些方案?**

**12.redis 常用用的数据结构、**

字符串(String)，散列/哈希(hash)，列表(list)，无序集合类型(Set),有序集合类型(sorted set)

**13.readis的hash实现**

redis散列/哈希是键值对的集合，redis散列/哈希是字符串字段和字符串值之间的映射，但字段值只能是字符串，不支持其他类型，因此，他们用于表示对象。应用场景：存储用户的基本信息，等。

**14.redis cluster**

redis 最开始使用主从模式做集群，若master宕机需要手动配置slave转化为master；后来未来高可用提出来哨兵模式，该模式下有一个哨兵监视master和slave，若master宕机可自动将slave转为master，但它也有一个问题，就是不能动态扩充；所以在3.x提出cluster集群模式。

redis-cluster采用无中心结构，每个节点保存数据和整个集群状态，每个节点都和其它所有节点连接

1. **什么是类的加载？**  
     
   JVM把通过类名获得类的二进制流之后，把类放入方法区，并创建入口对象的过程被称为类的加载。经过加载，类就被放到内存里了。  
     
   **2. 哪些情况会触发类的初始化？**  
     
   类在5种情况下会被初始化：  
     
   第一，假如这个类是入口类，他会被初始化。  
     
   第二，使用new创建对象，或者调用类的静态变量，类会被初始化。不过静态常量不算。  
     
   第三，通过反射获取类，类会被初始化  
     
   第四，如果子类被初始化，他的父类也会被初始化。  
     
   第五，使用jdk1.7的动态语言支持时，调用到静态句柄，也会被初始化。  
     
   **3. 讲一下JVM加载一个类的过程**  
     
   同问题1。不过这里也可以问下面试官是不是想问类的生命周期。如果是问类的生命周期，可以回答有"加载、连接、初始化、使用、卸载"五个阶段，连接又可以分为"校验、准备、解析"三个阶段。  
     
   **4. 什么时候会为变量分配内存？**  
     
   在准备阶段为静态变量分配内存。  
     
   **5. JVM的类加载机制是什么？**  
     
   双亲委派机制，类加载器会先让自己的父类来加载，父类无法加载的话，才会自己来加载。  
     
   **6. 双亲委派机制可以打破吗？为什么**  
     
   可以打破，比如JDBC使用线程上下文加载器打破了双亲委派机制。原因是JDBC只提供了接口，并没有**提供**实现。

**TCP相关**

**tcp和UDP区别：**

TCP是一个面向连接、可靠地、基于字节流的传输层协议。

UDP是一个面向无连接的传输层协议。

1.面向连接：指的是客户端和服务器的连接，在双方互相通信之前，TCP需要三次握手建立连接，在双方互相通信之前，TCP需要三次握手建立连接，而UDP没有相应建立连接的过程。

2.可靠性：TCP花了非常多的功夫保证了连接的可靠，这个可靠性体现在有状态和可控性。

3.面向字节流，UDP的数据传输是基于数据报的，这是因为仅仅只是继承了IP层的特性，而TCP为了维护状态，将一个个IP包变成了字节流。

**mysql索引失效的情况：**

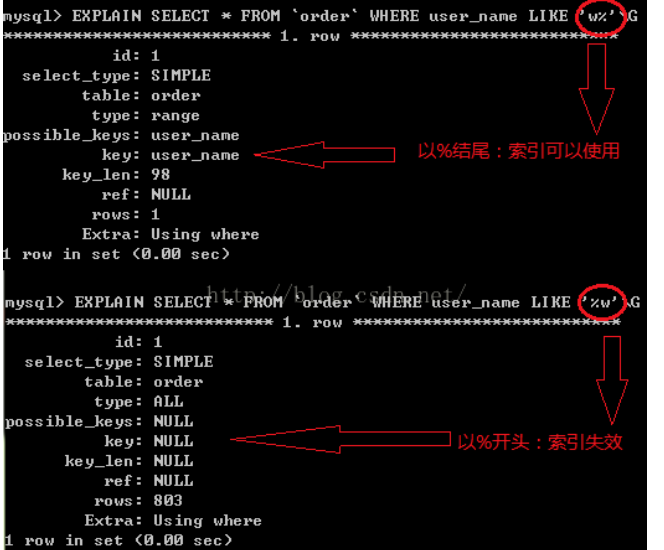
**详情见链接（**<https://www.cnblogs.com/applelife/p/10511455.html>**）**

**(**<https://www.jianshu.com/p/3ccca0444432>**)**

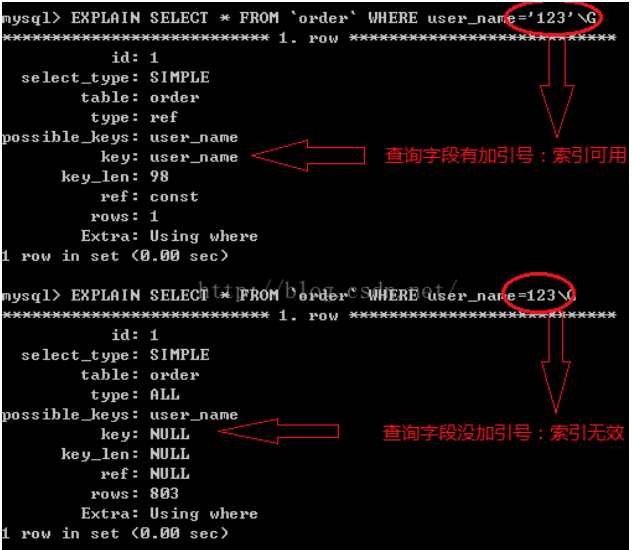
1.如果条件中有or，即使其中有条件带索引也不会使用（这也是为什么尽量少用or的原因）

注意：要想使用or，又想让索引生效，只能将or条件中的每个列都加上索引

1. 对于多列索引，不是使用的第一部分（第一个），则不会使用索引
2. like查询是以%开头，以%结尾的可以使用索引

例：

1. 如果列类型是字符串，那一定要在条件中将数据使用引号引起来，否则不使用索引



1. 如果mysql估计使用全表扫描要比使用索引快，则不使用索引

查看索引的使用情况： show status like ‘Handler\_read%';

大家可以注意：  
handler\_read\_key:这个值越高越好，越高表示使用索引查询到的次数  
handler\_read\_rnd\_next:这个值越高，说明查询低效。

其他：

1) 没有查询条件，或者查询条件没有建立索引

2) 在查询条件上没有使用引导列

3) 查询的数量是大表的大部分，应该是30％以上。

4) 索引本身失效

5) 查询条件使用函数在索引列上，或者对索引列进行运算，运算包括(+，-，\*，/，! 等) 错误的例子：select \* from test where id-1=9; 正确的例子：select \* from test where id=10;

6) 对小表查询

7) 提示不使用索引

8) 统计数据不真实

9) CBO计算走索引花费过大的情况。其实也包含了上面的情况，这里指的是表占有的block要比索引小。

10)隐式转换导致索引失效.这一点应当引起重视.也是开发中经常会犯的错误. 由于表的字段tu\_mdn定义为varchar2(20),但在查询时把该字段作为number类型以where条件传给Oracle,这样会导致索引失效. 错误的例子：select \* from test where tu\_mdn=13333333333; 正确的例子：select \* from test where tu\_mdn='13333333333';

12) 1,<> 2,单独的>,<,(有时会用到，有时不会)

13,like "%\_" 百分号在前.

4,表没分析.

15,单独引用复合索引里非第一位置的索引列.

16,字符型字段为数字时在where条件里不添加引号.

17,对索引列进行运算.需要建立函数索引.

18,not in ,not exist.

19,当变量采用的是times变量，而表的字段采用的是date变量时.或相反情况。

20,B-tree索引 is null不会走,is not null会走,位图索引 is null,is not null 都会走

21,联合索引 is not null 只要在建立的索引列（不分先后）都会走, in null时 必须要和建立索引第一列一起使用,当建立索引第一位置条件是is null 时,其他建立索引的列可以是is null（但必须在所有列 都满足is null的时候）,或者=一个值； 当建立索引的第一位置是=一个值时,其他索引列可以是任何情况（包括is null =一个值）,以上两种情况索引都会走。其他情况不会走