###### 有哪些方法保证HashMap的线程安全？

* 使用HashTable

原因：其主要方法均实现同步锁

* 使用ConcurrentHashMap

原因：采用Lock，保证线程安全，另外由于增加segment层，因此实现了片段锁，在效率上有了较大的提升

* 使用synchronizedMap

原因：和HashTable一样，采用同步锁保证线程安全

* 采用锁机制进行同步

原因：同步锁可以保证在线程安全

* 重写HashMap

重复造轮子，这个方法基本无用。

###### ConcurrentHashMap

相对于HashMap，ConcurrentHashMap增加Segment一层，Segment层是一个数组，且长度固定16，不可扩容。每个Segment初始大小为2，由于存在负载因子0.75，所以向Segment插入第二个元素的时候，便会扩容。

真是因为Segment的存在，所以ConcurrentHashMap可以实现片段锁（每次只锁对应的Segment），在效率上高于HashTable等。

使用Lock锁，依赖CPU，显示存在，需要手动解锁。可以实现公平锁和非公平锁。

###### Happens Before 原则

###### Volatile

是Java提供的一种同步机制，由于对Volatile变量的操作不用加锁，因此线程也不会阻塞，比Synchronized更加轻量。

使用Volatile后，变量将不被CPU Cache缓存也不会缓存到其他的寄存器或处理器无法访问的地方，JVM保证每次都从内存中读取。

使用Volatile后，具有两个特性：

1. 对所有线程的可见性，即变量将变为共享的
2. 禁止重排序优化

读取的效率和普通变量相差无几，但是写入会更加耗时，因为需要更多的内存屏障保证安全。

###### Integer.valueOf(100)和Integer.valueOf(1000)的区别

如果Integer.valueOf(X)，X的值在-128~127之间，则返回缓存实例，否则返回新的对象；

因此Integre.valueOf(100) == Integre.valueOf(100)，得到的结果是true, Integre.valueOf(1000) == Integre.valueOf(1000) 得到的结果是false

###### 线程池及构造参数

Jdk提供了4中线程池：

newSingleThreadPool：一个核心线程，没有缓冲线程

newFixedThreadPool：固定数量的核心线程，没有缓冲线程

newCachedThreadPool：没有缓冲线程，全是缓冲线程

newScheduledThreadPool：具有常规线程的功能，也可以配置定时任务，代替Timer的有效方案

官方不建议开发人员直接使用ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockQueue<Runnable> workQueue, ThreadFactory threadFactory, RejectedExecutionHandler handler)；

corePoolSize： 核心线程数量

maximumPoolSize： 最大缓冲线程数量

keepAliveTime： 存活时间

unit： 时间单位

workQueue： 任务队列，ArrayBlockingQueue（基于数组，需指定大小），LinkedBlockingQueue（基于链表，无须指定大小，默认值32），SynchronousQueue（newCachedThreadPool使用，无需排队，直接执行）

threadFactory： 线程工厂

handler： 拒绝策略， ThreadPoolExecutor.AbortPolicy 丢弃任务，并抛出 RejectedExecutionException 异常。ThreadPoolExecutor.CallerRunsPolicy：该任务被线程池拒绝，由调用 execute方法的线程执行该任务。 ThreadPoolExecutor.DiscardOldestPolicy ： 抛弃队列最前面的任务，然后重新尝试执行任务。ThreadPoolExecutor.DiscardPolicy，丢弃任务，不过也不抛出异常。

###### ThreadLocal原理及如何保证安全性、在什么情况下会有问题

ThreadLocal内部通过为每个使用的线程创建一个“副本”（本质上是一个ThreadLocalMap），用于隔离各个线程，从而达到线程安全。

ThreadLocal默认是不使用的，因为ThreadLocal会创建副本，对资源有格外的消耗。

ThreadLocal本质是ThreadLocalMap，以Entry为单位存放于该Map，然而Entry是实现弱引用接口，但是entry下的value又是强引用，这会导致key已经被回收，但是value还存在，即内存泄漏。但是ThreadLocal的每一个方法内都实现了对key == null 的数据清除，因此在每次使用完成ThreadLocal后最好使用一个其中的方法，达到减少内存泄漏的可能。而且在get之前务必先使用set方法，否则会抛出空指针异常

###### 悲观锁与乐观锁

在宏观上，锁被分为悲观锁和乐观锁。

悲观锁：总是以最坏的情况考虑数据被访问，认为每次的数据访问都会让数据发生更改，所以每次数据操作都会上锁。

乐观锁：总是认为数据被访问不会发生修改，所以不加锁，当数据执行写入的时候，才会判断是否数据已被修改。

###### Mysql的索引类型

分为5种类型：普通索引、唯一索引、主键索引、组合索引、全文索引。

普通索引：

最基础、最简单、最常用、没有限制条件；

唯一索引：

不允许重复值、允许空值；

主键索引：

不允许重复值、不允许空值；

组合索引：

不允许空值、遵循最左前缀原则；

全文索引：

fulltext索引配合match against操作使用；

注意：在数据量较大时候，现将数据放入一个没有全局索引的表中，然后再用CREATE index创建fulltext索引，要比先为一张表建立fulltext然后再将数据写入的速度快很多。

其他：

索引可以提升查询速度，但是会降低更新速度，因为不仅要更新数据，还要更新索引文件，这也是不要使用过多的索引的原因。

建立索引应避免空值；

建立索引应尽量使用短字段；

如果需要对多个字段排序，则最好创建复合索引；

计算列无法使用索引；

多个机器同时对条数据进行读写如何保证数据的一致性

此问题实际上问的是分布式数据一致性问题；

|  |  |  |
| --- | --- | --- |
| 分布式 | 锁 | 基于数据库：使用数据表或者使用数据库锁，实现简单，受限于数据库性能 基于缓存： 基于Zookeeper： |
| 事务 | 回滚接口：耦合度高，适合简单场景 本地消息表：基本避免分布式事务，实现了最终一致性，但是数据库吞吐量和性能方面存在瓶颈。 非事务消息：适合大多数场景，但上线前需进行充分测试。整体优于本地消息表的方式 事务消息：实现难度大，须修改MQ事务消息部分代码或自行实现 补偿机制：通过定时任务扫描日志进行补偿并通过邮件通知相关人员 + 人工补偿 |

###### HashMap的loadFactory大了小了有什么影响

loadFactory，可变因子。Hashmap扩容与它有直接的关系。

HashMap默认大小为1 << 4 = 16，计算得到16\*0.75 = 12。当HashMap存入数据量大于12的时候，HashMap便会扩容。

如果loadFactory过小，便会导致HashMap频繁的扩容，且浪费内存

如果loadFactory过大，便会导致可能超过Hash Map自身的容量，从而出现异常。