一. 如果线上某台虚机 CPU Load 过高，该如何快速排查原因？只介绍思路和涉及的Linux命令即可。

造成cpu load过高的原因

Full GC 次数的增大

代码中存在Bug（例如死循环、正则的不恰当使用等）

排查方法

1. jps -v：查看 Java 进程号。

2. top -Hp [Java进程号]：查看当前进程下最耗费 CPU 的线程。

3. printf "%x\n" [步骤 2 中的线程号]：得到线程的 16 进制表示。

4. jstack [java进程号] | grep -A100 [步骤 3 的结果]：查看线程堆栈，定位代码行。

参考：如何使用 JStack 分析线程状态。

二. 请简要描述 MySQL 数据库联合索引的命中规则，可举例说明。

（1）MySQL 联合索引遵循最左前缀匹配规则，即从联合索引的最左列开始向右匹配，直到遇到匹配终止条件。例如联合索引 (col1, col2, col3)，where 条件为 col1 = `a` AND col2 = `b` 可命中该联合索引的 (col1, col2) 前缀部分，where 条件为 col2 = `b` AND col3 = `c` 不符合最左前缀匹配，不能命中该联合索引。

（2）匹配终止条件为范围操作符（如 >、<、between、like 等）或函数等不能应用索引的情况。例如联合索引 (col1, col2, col3)，where 条件为 col1 = `a` AND col2 > 1 AND col3 = `c`，在 col2 列上为范围查询，匹配即终止，只会匹配到col1，不能匹配到 (col1, col2, col3)。

（3）where 条件中的顺序不影响索引命中。例如联合索引 (col1, col2, col3)，where 条件为 col3 = `c` AND col2 = b AND col1 = `a`，MySQL 优化器会自行进行优化，可命中联合索引 (col1, col2, col3)。

三. 什么是分布式事务，分布式事务产生的原因是什么？分布式事务的解决方案有哪些？分别有哪些优缺点？

1. 什么是分布式事务

分布式事务是指事务参与者、支持事务的服务器、资源服务器及事务管理器分别位于不同的分布式系统的不同节点上。即一次大操作由不同的小操作组成，这些小操作分布在不同的服务器上，且属于不同的应用，分布式事务需要保证这些小操作要么全部成功，要么全部失败。本质上，分布式事务就是为了保证不同数据库的数据一致性。

2. 分布式事务产生的原因

（1）Service 产生多个节点

随着互联网快速发展，微服务、SOA 等服务架构模式正在被大规模的使用。如：一个公司之内，用户的资产可能分为好多个部分，比如余额、积分、优惠券等。在公司内部有可能积分功能由一个微服务团队维护，优惠券又是另外的团队维护。这样的话就无法保证积分扣减了之后，优惠券能否扣减成功。

（2）Resource多个节点

同样的，互联网发展得太快了，我们的 MySQL 一般来说装千万级的数据就得进行分库分表。对于一个支付宝的转账业务来说，你给朋友转钱，有可能你的数据库是在北京，而你朋友的钱是存在上海，所以我们依然无法保证他们能同时成功。

3. 分布式事务的解决方案

（1）2PC

第一阶段：事务管理器要求每个涉及到事务的数据库预提交（precommit）此操作，并反映是否可以提交。

第二阶段：事务协调器要求每个数据库提交数据，或者回滚数据。

优点：尽量保证了数据的强一致，实现成本较低，在各大主流数据库都有自己实现，对于 MySQL 是从 5.5 开始支持。

缺点：

a. 单点问题：事务管理器在整个流程中扮演的角色很关键，如果其宕机，比如在第一阶段已经完成，在第二阶段正准备提交的时候事务管理器宕机，资源管理器就会一直阻塞，导致数据库无法使用。

b. 同步阻塞：在准备就绪之后，资源管理器中的资源一直处于阻塞，直到提交完成，释放资源。

c. 数据不一致：两阶段提交协议虽然为分布式数据强一致性所设计，但仍然存在数据不一致性的可能，比如在第二阶段中，假设协调者发出了事务 commit 的通知，但是因为网络问题该通知仅被一部分参与者所收到并执行了 commit 操作，其余的参与者则因为没有收到通知一直处于阻塞状态，这时候就产生了数据的不一致性。

总的来说，XA 协议比较简单，成本较低，但是其单点问题，以及不能支持高并发（由于同步阻塞）依然是其最大的弱点。

（2）TCC

第一阶段：Try 阶段，尝试执行，完成所有业务检查（一致性），预留必须业务资源（准隔离性）。

第二阶段：Confirm 阶段，确认执行真正执行业务，不作任何业务检查，只使用Try阶段预留的业务资源，Confirm 操作满足幂等性。要求具备幂等设计，Confirm 失败后需要进行重试。

第三阶段：Cancel 阶段，取消执行，释放 Try 阶段预留的业务资源，Cancel 操作满足幂等性，Cancel 阶段的异常和 Confirm 阶段异常处理方案基本上一致。

TCC 适合强隔离性、严格一致性要求、执行时间较短的业务。

（3）本地消息表

此方案的核心是将需要分布式处理的任务通过消息日志的方式来异步执行。消息日志可以存储到本地文本、数据库或消息队列，再通过业务规则自动或人工发起重试。人工重试更多的是应用于支付场景，通过对账系统对事后问题的处理。对于本地消息队列来说核心是把大事务转变为小事务。

（4）MQ 事务

在 RocketMQ 中实现了分布式事务，实际上其是对本地消息表的一个封装，将本地消息表移动到了 MQ 内部。

（5）Saga事务

核心思想是将长事务拆分为多个本地短事务，由 Saga 事务协调器协调，如果正常结束那就正常完成，如果某个步骤失败，则根据相反顺序一次调用补偿操作。

四. 请描述 https 的请求过程。

（1）客户端向服务器发起 https 请求，连接到服务器的443端口；

（2）服务器将自己的公钥（数字证书）发送给客户端。服务器端有一个密钥对，即公钥（数字证书）和私钥，是用来进行非对称加密使用的，服务器端保存着私钥，不能将其泄露，公钥可以发送给任何人；

（3）客户端收到服务器端的公钥之后，检查其合法性，如果发现发现公钥有问题，那么 https 传输就无法继续，如果公钥合格，则客户端会生成一个客户端密钥，然后用服务器的公钥对客户端密钥进行非对称加密成密文，至此，https 中的第一次 http 请求结束。

（4）客户端发起 https 中的第二个 http 请求，将加密之后的客户端密钥发送给服务器；

（5）服务器接收到客户端发来的密文之后，会用自己的私钥对其进行非对称解密，解密之后的明文就是客户端密钥，然后用客户端密钥对数据进行对称加密，这样数据就变成了密文，服务器将加密后的密文发送给客户端；

（6）客户端收到服务器发送来的密文，用客户端密钥对其进行对称解密，得到服务器发送的数据。这样 https 中的第二个 http 请求结束，整个 https 传输完成。

五. 什么是事务传播行为？你知道 Spring 事务中都有哪些传播类型吗？如何使用/指定传播类型？

IO设计中 Reactor 和 Proactor 区别。

以字符串的形式读入两个数字，再以字符串的形式输出两个数字的和。

给定一个字符串，你的任务是计算这个字符串中有多少个回文子串（回文串是一个正读和反读都一样的字符串）。具有不同开始位置或结束位置的回文串，即使是由相同的字符组成，也会被计为是不同的子串。

有 N 堆金币排成一排，第 i 堆中有 C[i] 块金币。每次合并都会将相邻的两堆金币合并为一堆，成本为这两堆金币块数之和。经过 N-1 次合并，最终将所有金币合并为一堆。请找出将金币合并为一堆的最低成本。其中，1 <= N <= 30，1 <= C[i] <= 100。

给定一组个字符串，为每个字符串找出能够唯一识别该字符串的最小前缀。