什么是线程死锁？死锁如何产生？如何避免线程死锁？

死锁的介绍：

线程死锁是指由于两个或者多个线程互相持有对方所需要的资源，导致这些线程处于等待状态，无法前往执行。当线程进入对象的 synchronized 代码块时，便占有了资源，直到它退出该代码块或者调用 wait 方法，才释放资源，在此期间，其他线程将不能进入该代码块。当线程互相持有对方所需要的资源时，会互相等待对方释放资源，如果线程都不主动释放所占有的资源，将产生死锁。

死锁的产生的一些特定条件：

1、互斥条件：进程对于所分配到的资源具有排它性，即一个资源只能被一个进程占用，直到被该进程释放 。
2、请求和保持条件：一个进程因请求被占用资源而发生阻塞时，对已获得的资源保持不放。
3、不剥夺条件：任何一个资源在没被该进程释放之前，任何其他进程都无法对他剥夺占用。
4、循环等待条件：当发生死锁时，所等待的进程必定会形成一个环路（类似于死循环），造成永久阻塞。

如何避免：

1、加锁顺序：
当多个线程需要相同的一些锁，但是按照不同的顺序加锁，死锁就很容易发生。如果能确保所有的线程都是按照相同的顺序获得锁，那么死锁就不会发生。当然这种方式需要你事先知道所有可能会用到的锁，然而总有些时候是无法预知的。
2、加锁时限：
加上一个超时时间，若一个线程没有在给定的时限内成功获得所有需要的锁，则会进行回退并释放所有已经获得的锁，然后等待一段随机的时间再重试。
但是如果有非常多的线程同一时间去竞争同一批资源，就算有超时和回退机制，还是可能会导致这些线程重复地尝试但却始终得不到锁。
3、死锁检测：

死锁检测即每当一个线程获得了锁，会在线程和锁相关的数据结构中（map、graph 等等）将其记下。除此之外，每当有线程请求锁，也需要记录在这个数据结构中。死锁检测是一个更好的死锁预防机制，它主要是针对那些不可能实现按序加锁并且锁超时也不可行的场景。

notify 和 notifyAll 区别

他们的作用都是通知处于等待该对象的线程。
1、notifyAll 使所有原来在该对象上等待被 notify 的线程统统退出 wait 的状态，变成等待该对象上的锁，一旦该对象被解锁，他们就会去竞争。
2、notify 是通知其中一个线程，不会通知所有的线程。

谈一谈对 MySQL InnoDB 的认识

介绍：
InnoDB 引擎是 MySQL 数据库的一个重要的存储引擎,和其他存储引擎相比,InnoDB 引擎的优点是支持兼容 ACID 的事务(类似于 PostgreSQL),以及参数完整性(有外键)等。现在 Innobase 实行双认证授权.MySQL5.5.5 以后默认的存储引擎都是 InnoDB 引擎。

特点是：
1、具有较好的事务支持：支持 4 个事务隔离级别，支持多版本读
2、行级锁定：通过索引实现，全表扫描仍然会是表锁，注意间隙锁的影响
3、读写阻塞与事务隔离级别相关
4、具有非常高效的缓存特性：能缓存索引，也能缓存数据
5、整个表和主键以 Cluster 方式存储，组成一颗平衡树
6、所有 Secondary Index 都会保存主键信息

适用场景：
1、需要事务支持（具有较好的事务特性）
2、行级锁定对高并发有很好的适应能力，但需要确保查询是通过索引完成
3、数据更新较为频繁的场景
4、数据一致性要求较高
5、硬件设备内存较大，可以利用 InnoDB 较好的缓存能力来提高内存利用率，尽可能减少磁盘 IO

谈一谈数据库事务的隔离级别？

1、Read uncommitted（读未提交）就是一个事务可以读取另一个未提交事务的数据。
2、Read committed（读提交）就是一个事务要等另一个事务提交后才能读取数据。
3、Repeatable read（重复读）就是在开始读取数据（事务开启）时，不再允许修改操作。
4、Serializable（序列化）在该级别下，事务串行化顺序执行，可以避免脏读、不可重复读与幻读。是最高的事务隔离级别，但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用。
事务的作用就是保证数据的一致性、完整性。事务隔离级别越高，在并发下会产生的问题就越少，但同时付出的性能消耗也将越大，因此很多时候必须在并发性和性能之间做一个权衡。所以设立了几种事务隔离级别，以便让不同的项目可以根据自己项目的并发情况选择合适的事务隔离级别，对于在事务隔离级别之外会产生的并发问题，在代码中做补偿。

MySQL 主备同步的基本原理

MySQL 支持单向、异步复制，复制过程中一个服务器充当主服务器，而一个或多个其它服务器充当从服务器。

MySQL 复制是基于主服务器在二进制日志中跟踪所有对数据库的更改。因此，要进行复制，必须在主服务器上启用二进制日志。每个从服务器从主服务器接收主服务器已经记录到日志的数据。

当一个从服务器连接主服务器时，它通知主服务器从服务器在日志中读取的最后一次成功更新的位置。从服务器接收从那时起发生的任何更新，并在本机上执行相同的更新。然后封锁并等待主服务器通知新的更新。从服务器执行备份不会干扰主服务器，在备份过程中主服务器可以继续处理更新。