三种类型的锁

1. 线程锁：主要给方法、代码块加锁，synchronized是共享对象头，显示锁lock是共享某个变量，常用的同步工具都是线程锁
2. 进程锁：为了控制同一个操作系统中多个进程访问某个共享资源
3. 分布式锁：多个进程不再同一个系统中，用分布式锁控制多个进程对资源的访问

分布式锁使用的场景

1、高并发下争夺共享资源，比如秒杀对于库存这种共享资源就需要使用分布式锁

2、当后端服务是集群形式存在的时候，需要分布式锁

3、集群与分布式不同，集群是指多个服务器实现了同样的请求，分布式是指多台服务器有各自不同的功能点，多台功能的整合对外是一个完整的服务

注意：

微服务，分布式，集群的区别

分布式是一个完整服务的不同模块分开部署在不同机器上，集群是指各个服务器是相同的服务，而微服务则跟集群类似，不同的是他的服务可能会部署在同一个服务器上，目的是为了确保一个服务的bug或升级不会影响现有的系统业务

分布式锁的功能

排他性：同一时间只有一个客户端能获取到锁，其他客户端获取不到

避免死锁：锁在一段时间内有效，超过这个时间就会被释放

高可用：获取或释放锁的机制必须高可用且性能佳

分布式锁的解决方案

1. 基于数据库实现分布式锁
2. 基于缓存（redis，memcached，tair）实现分布式锁
3. 基于zookeeper实现分布式锁

基于数据库的分布式锁

1、创建一张锁的表的，线程想要锁住某个方法或者资源时，就在该表中增加一条记录，释放的时候删除这条记录

2、这个锁的resource（资源唯一标识）字段，是唯一性的，插入时候如果表中已经有就会插入失败，表明加锁失败

他的缺陷

1. 这个锁没有失效时间，一旦释放锁的操作失败就会导致锁记录一直在数据库中，其他线程无法获得锁。那么可以做一个定时任务去定时清理就好了
2. 这种锁的可靠性依赖于数据库，建议设置备库，避免单点，进一步提高可靠性
3. 这个锁是非阻塞的，插入失败就会报错，想获得锁就需要再次操作，如果是阻塞的，可以写个循环，直到插入成功再返回

4. 他是非可重入的，因为同一个线程没有释放之前无法再次获得锁，想要实现可重入，可以在数据库中加入主机信息，线程信息字段，插入前先查询数据判断是否是当前主机的当前线程，是否是的话可以直接把锁分配给他

乐观锁：

为资源表添加版本字段，每次更新的时候先去查询数据及版本，更新的时候在where语句中加入版本条件同时让版本+1，如果版本与之前读出来的不一致那么就更新失败即操作失败

乐观锁的有点比较明显，他不依赖于数据库本身的锁机制，不会影响请求的性能。缺点是需要对表加入额外的版本字段，增加了数据的冗余，同时并发量高的时候，version会频繁的变化，则会导致大量的请求失败，影响系统可用性，同时大量的请求同时请求同一行数据会导致数据库压力过大。因此乐观锁适用于并发量不高的应用

悲观锁

即总是假设最坏的情况，策略是使用行锁(for update锁定一行，执行业务逻辑之后，commit操作释放锁)，一个线程在释放锁之前其他线程执行for update查询操作会被阻塞，直到线程A释放锁之后才能继续，如果长时间未释放锁，后来的线程就会报错

特别的：

当表没有主键，for update就会锁表，同时当主键不明确（where id>0 for update）或者数据比较少的表mysql认为全表扫描会效率更高，就会使用表锁而不是行锁

基于zookeeper实现的分布式锁

zookeeper是一个为分布式应用提供一致性服务的开源组件，内部是一个分层的文件系统目录树结构，规定一个目录下只能有一个唯一的文件名。

他会为每一个锁创建一个目录，其中存储申请这个锁的线程的临时节点，当一个线程申请锁的时候就获取其中所有的节点，然后获取比自己小的节点，如果不存在就说明当前线程的顺序号最小便获得锁，如果有则创建并监听比自己次小的节点，当监听到那个节点结束之后便再次去查询，当自己获取后并直接结束则删除自己的节点。

基于缓存实现分布式锁

如redis，memcached都有分布式锁的解决功能

缓存实现的分布式锁会有更好的性能，并且缓存大都是集群部署的，可以避免单点问题，同时也有对数据的过期自动删除功能

缺点是使用超时时间来控制的失效时间并不是十分靠谱

总结

从性能将缓存>zookepeer>数据库，可靠性上zookeeper最好其次是缓存，再是数据库

这三种实现都有三个问题需要克服

1. 失效时间问题，可以数据库和zookepeer的可以创建一个监听程序，redis有过期时间设置，
2. 非阻塞的：可以循环申请直至申请到为止
3. 非可重入：一个线程获取到之后存储当前的主机及线程信息，下次获取前判断自己是否是当前锁的