**业务中使用分布式的场景**

解决java集群的session共享的问题:

1.客户端cookie加密

2.集群中,各个应用服务器提供了session复制的功能,tomcat和jboss都实现了这样的功能,性能随服务器的增加急剧下降,容易引起广播风暴,session数据需要序列化,影响性能.

3.session的持久化,将session写入数据库,每次读取带来性能开销,写入内存,提高性能,宕机数据会丢失.

4.使用共享存储来保存session,频繁的进行序列化和反序列化影响性能.

5.使用缓存保存session,存入到memcached的数据需要序列化,效率极低.

分布式锁的实现方案:

基于数据库实现分布式锁,基于缓存实现分布式锁,基于Zookeeper实现分布式锁.

**CAP:何一个分布式系统都无法同时,一致性,可用性,分区容错性.**

**一般情况下,牺牲强一致性,实现高可用,保证最终的是数据一致性**

**一个方法在同一时间只能被一台机器上的一个线程执行.**

**这把锁是一把可重入锁,(避免死锁)**

**最好是一把阻塞锁**

**有高可用的获取锁和释放锁的功能,**

**获取锁和释放锁的性能要好.**

**基于数据库表实现分布式锁,直接创建一张锁表,当要锁住某个方法或资源时,就在表中增加一条记录,释放锁就删除这条记录.**

**数据库只一个单点,一旦数据库挂掉,业务系统不可用.(设置两个数据库,一旦挂掉,快速切换数据库)**

**锁没有设置失效时间,一旦解锁操作失败,锁记录一直存在在数据库中,其他线程无法再获取锁.(做一个定时任务,每个一定时间将数据库中的超时数据清理一遍)**

**该锁只能是非阻塞的,一旦插入失败就会报错,想要再次获得锁,就需要再次出发锁操作.(写个死循环,直到insert成功,再返回成功)**

**锁是非重入的,同一线程在没有释放锁之前无法再次获得该锁. (增加字段,记录获得锁的机器的主机信息和线程信息,下次获取锁时,先查询数据库,如果存在,直接将锁分配给他.)**

**基于数据库排他锁,借助数据库的自带的锁实现分布式锁.**

**在查询语句后增加for update,数据库会在查询过程中给数据库表增加排它锁,当某条记录被加上排他锁,其他线程无法再改记录上增加排它锁.可以认为获得排它锁的线程即可获得分布式锁,当获取到锁后,可执行方法的业务逻辑,通过连接提交来释放锁.**

**基于缓存实现分布式锁,基于缓存实现,性能好,缓存可以集群部署,可以解决单点问题..**

**同样的问题:**

**失效时间:缓存的过期时间**

**非阻塞:while重复执行**

**非重入:保存信息,下次先读取,有的话直接获取锁.**

**通过超时时间控制锁的超时时间,并不是十分靠谱.**

**基于zookeeper实现分布式锁.**

**基于zookeeper临时有序节点,可以实现分布式锁,大致思想是每个客户端对某个方法加锁时,在zookeeper上的与该方法对应的指定节点的目录下,生成一个唯一的瞬时有序节点.判断是否获取所得方式很简单,只需判断有序节点中序号最小的一个,当释放锁的时候,只需要将瞬时节点删除即可.同时可以避免服务器宕机导致的锁释放,而产生的死锁问题.**

**问题:**锁无法释放,zookeeper在创建锁时,客户端会在zk中创建一个临时节点,一旦客户端获取锁之后突然挂掉(session连接断掉),临时节点就会临时删除,其他客户端就会再次获得锁.

非阻塞锁:使用zk可以实现阻塞的锁,客户端可以通过在zk中创建顺序节点,并在节点上绑上监听器,一旦节点有变化,zk或通知client,client会检查自己创建的节点是不是当前节点最小的,如果是,那么自己就获得锁,执行业务逻辑.

不可重入:别的直接对比,这里和最小节点的额数据对比,如果相同,获取锁,如果不一样,就再次创建一个临时的顺序节点,参与排队.

单点问题:ZK是集群部署的,只有有半数机器存活,就可以对外提供服务.

**ZK提供锁没有缓存的性能高.**

**负载均衡算法:随机算法.权重随机,在每个机器上设置权重数,**