**悲观锁**

每次拿数据都以为别人会修改，所以每次拿数据时都会上锁。  
实现：开启事务，启用锁机制

**乐观锁**

每次拿数据时候都认为别人不会修改，所以不会上锁，但是在更新数据时候会判断在此期间是否有人更新过。  
实现：1.使用版本号2.使用时间戳

一、概念上区别

乐观锁（\*\* Optimistic Locking\*\*）：顾名思义，对加锁持有一种乐观的态度，即先进行业务操作，不到最后一步不进行加锁，"乐观"的认为加锁一定会成功的，在最后一步更新数据的时候再进行加锁。

悲观锁\*\*（Pessimistic Lock）\*\*：正如其名字一样，悲观锁对数据加锁持有一种悲观的态度。因此，在整个数据处理过程中，将数据处于锁定状态。悲观锁的实现，往往依靠数据库提供的锁机制（也只有数据库层提供的锁机制才能真正保证数据访问的排他性，否则，即使在本系统中实现了加锁机制，也无法保证外部系统不会修改数据）。

二、实现方式：

乐观锁：

* version方式：一般是在数据表中加上一个数据版本号version字段，表示数据被修改的次数，当数据被修改时，version值会加一。当线程A要更新数据值时，在读取数据的同时也会读取version值，在提交更新时，若刚才读取到的version值为当前数据库中的version值相等时才更新，否则重试更新操作，直到更新成功。

sql实现代码：

**update** **table** **set** x=x+1, **version**=**version**+1 **where** **id**=#{**id**} **and** **version**=#{**version**};

* CAS（定义见后）操作方式：即compare and swap 或者 compare and set，涉及到三个操作数，数据所在的内存值，预期值，新值。当需要更新时，判断当前内存值与之前取到的值是否相等，若相等，则用新值更新，若失败则重试，一般情况下是一个自旋操作，即不断的重试。（这种方式作者也是最近刚知道，惭愧惭愧）

悲观锁：是由数据库自己实现了的，要用的时候，我们直接调用数据库的相关语句就可以了（原理：共享资源每次只给一个线程使用，其它线程阻塞，用完后再把资源转让给其它线程），如行锁、读锁和写锁等，都是在操作之前加锁，在Java中，synchronized的思想也是悲观锁。

三、使用场景

* 乐观锁：比较适合读取操作比较频繁的场景，如果出现大量的写入操作，数据发生冲突的可能性就会增大，为了保证数据的一致性，应用层需要不断的重新获取数据，这样会增加大量的查询操作，降低了系统的吞吐量。
* 悲观锁：比较适合写入操作比较频繁的场景，如果出现大量的读取操作，每次读取的时候都会进行加锁，这样会增加大量的锁的开销，降低了系统的吞吐量。

四、特点

* 乐观锁：乐观锁的特点先进行业务操作，不到万不得已不去拿锁。即“乐观”的认为拿锁多半是会成功的，因此在进行完业务操作需要实际更新数据的最后一步再去拿一下锁就好。
* 悲观锁：悲观锁的特点是先获取锁，再进行业务操作，即“悲观”的认为获取锁是非常有可能失败的，因此要先确保获取锁成功再进行业务操作。通常所说的“一锁二查三更新”即指的是使用悲观锁。

五、sql实现代码和案例

因为本文主要是讲解的是面试的时候怎样回答，所以就不将具体sql代码和案例贴出来了，感兴趣的同学可以自行去看一下，如果以后有时间，我也会单独写一篇详细的实现案例出来。

总结：以上，基本回答完前四点后，这道题已经能在面试官心里达到八九十分了。但是我们的人生不仅仅是为了面试，希望同学们一定要弄懂其中的原理和区别，悲观锁和乐观锁都有自己的优缺点，**简而言之记得一句话：读取频繁使用乐观锁，写入频繁使用悲观锁。乐观锁不能解决脏读的问题。**