<https://www.cnblogs.com/fanp/p/4633453.html>

    【为什么要锁】

    数据库是一个多用户使用的共享资源,比如一个用户表t\_user,两个浏览器前面的人登录了同个一个账号，把电话号码改了。当多个用户并发地存取数据时，在数据库中就会产生多个事务同时存取同一数据的情况。若对并发操作不加控制就可能会读取和存储不正确的数据，破坏数据库的一致性(脏读，不可重复读，幻读等)，可能产生死锁。为了解决这个问题，加锁是一个非常重要的技术，对实现数据库并发控制是一个好的方案。简单说，当一个执行sql语句的事务想要操作表记录之前，先向数据库发出请求，对你访问的记录集加锁，在这个事务释放这个锁之前，其他事务不能对这些数据进行更新操作。

    【有哪些锁】

     锁包括行级锁、表级锁、悲观锁、乐观锁

     行级锁:一种它锁，防止另外事务修改此行;在使用以下语句时，Oracle会自动应用行级锁：INSERT、UPDATE、DELETE、SELECT … FOR UPDATE [OF columns] [WAIT n | NOWAIT];SELECT … FOR UPDATE语句允许用户一次锁定多条记录进行更新.使用commit或者rollback释放锁。MySql的innodb存储引擎默认是行级锁。特点:开锁大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并发度也最高。适合于有大量按索引更新少量不同数据，同时又有并发查询的应用，如一些在线事务处理系统。

    表级锁:5种

   行共享 (ROW SHARE) – 禁止排他锁定表，与行排他类似，区别是别的事务还可以在此表上加任何排他锁。（除排他（exclusive）外）  
   行排他(ROW EXCLUSIVE) – 禁止使用排他锁和共享锁，其他事务依然可以并发地对相同数据表执行查询，插入，更新，删除操作，或对表内数据行加锁的操作，但不能有其他的排他锁（自身是可以的，没发现有什么用）  
   共享锁(SHARE) - 锁定表，对记录只读不写，多个用户可以同时在同一个表上应用此锁，在表没有被任何DML操作时，多个事务都可加锁，但只有在仅一个事务加锁的情况下只有此事务才能对表更新；当表已经被更新或者指定要更新时（select for update），任何事务都不能加此锁了。  
   共享行排他(SHARE ROW EXCLUSIVE) – 比共享锁更多的限制，禁止使用共享锁及更高的锁，在表没有被任何DML操作时，只有一个事务可以加锁，可以更新，书上说别的事务可以使用select for update锁定选中的数据行，可是实验后没被验证。  
   排他(EXCLUSIVE) – 限制最强的表锁，仅允许其他用户查询该表的行。禁止修改和锁定表

   行级锁和表级锁是根据锁的粒度来区分的，行记录，表都是资源，锁是作用在这些资源上的。如果粒度比较小(比如行级锁)，可以增加系统的并发量但需要较大的系统开销，会影响到性能，出现死锁，，因为粒度小则操作的锁的数量会增加;如果作用在表上，粒度大，开销小，维护的锁少，不会出现死锁，但是并发是相当昂贵的，因为锁定了整个表就限制了其它事务对这个表中其他记录的访问。

   悲观锁:

   Pessimistic Lock正如其名，它指的是对数据被外界（包括本系统当前的其他事务，以及来自外部系统的事务处理）修改持保守悲观态度，事务每次去操作数据的时候都假设有其他事务会修改需要访问的数据，所以在访问之前都要求上锁，行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁，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）。 一个典型的倚赖数据库的悲观锁调用： select \* from account where name=”Erica” for update 这条sql 语句锁定了account 表中所有符合检索条件（name=”Erica”）的记录。 本次事务提交之前（事务提交时会释放事务过程中的锁），外界无法修改这些记录。

    Hibernate悲欢锁实现:基于数据库锁机制

         Query q=Session.createQuery("select  \* from t\_profit where amount>10000");

          q.setLockMode("Profit",LockMode.UPGRADE);//Profit是Profit类的别名

         List<Profit> ps=q.list();

       执行的sql:select ....from t\_profit where amount>10000 for update.hibernate的悲观锁通过数据库的for update实现。

     LockMode.NONE：无锁机制;

     LockMode.WRITE:insert,update记录时自动获取悲观锁;

     LockMode.READ在读取时自动获取悲观锁;

     LockMode.UPGRADE:利用数据库的for update子句加锁;

     LockMode.UPGRADE\_NOWAIT:oracle特定实现，用oracle的for update nowait子句加锁

  乐观锁:

   Optimistic Lock,和悲欢锁相反，事务每次去操作数据之前，都假设其他事务不会修改这些需要访问的数据 ，所以 在访问之前不要求上锁，只是在进行更新修改操作的时候判断一下在访问的期间有没有其他人修改数据 了。它适用于多读的应用类型，冲突真的发生比较少的时候就比较好，这样省去了开销的开销，可以提高吞吐量;但如果是真的经常要发生冲突的，那每次还要去判断进行retry,反倒降低的性能，这个时候悲欢锁比较好。数据库如果提供类似于write\_condition机制的其实都是提供的乐观锁。

          它的实现大多是基于数据版本versin记录机制。举个例子:

       1.利润表t\_profit中有一个 version字段，当前值为1；而总资产余额字段（balance）为$10000

       2.操作员A读出version=1,从总资产减除2000，10000-2000=8000.

       3.A还没操作结束，此时操作员B也读出version=1,总资产减除5000，10000-5000=5000.

       4.A操作完成，把version加1，修改为2，把总资产减2000后提交更新数据库，更新成功

       5.B操作了，也加version加1，修改为2，把总资产减5000后提交更新数据库，此时发现version已经为2了，如B修改后加1的version一样，不满足乐观锁策略:"提交的版本必有大于记录当前的版本才能执行"。因此B的操作请求被驳回，这样就避免了B就version=1的旧数据修改的结果覆盖了A操作的结果的可能。如没有乐观锁，那A减去2000后剩余8000，但B操作的时候是用10000-5000剩余5000的，如果B的提交成功，总资产余额就是5000，但实际情况应该是8000-5000=3000的。出现总资产表记录和实际支出不一致。

    Hibernate对乐观锁的实现:

      <hibernate-mapping>

        <class name="com.f.TProfit" table="t\_profit" optimistic-lock="version"></class>

      </hibernate-mapping>