====2020/09/13====

synchronized volatile lock原理
ReentrantLock，ThreadLocal
线程的状态
java连接数据库

一、synchronized volatile lock

1. synchronized（关键字，jvm层面）：

a. synchronized用来锁住对象、方法、代码块，互斥、共享、可重入性；

b. 确保线程互斥访问同步代码，保证共享变量的修改能够及时可见。

c. synchronized的底层是通过一个monitor对象来完成的，notify/wait 这些方法都是依赖于monitor对象。每个对象有一个monitor监视器锁，当monitor被占用时就会处于锁定状态，线程执行monitorenter指令，尝试获取monitor的所有权；如果monitor的进入数为0，当前线程进入monitor，然后将进入数设置为1，该线程时monitor的所有者；如果线程已经占有了monitor，知识重新进入，则monitor的进入数加1；如果其他线程占有了monitor，这当前线程进入阻塞状态，直到monitor的进入数0，再尝试获取monitor的所有权；

d. 当monitor的所有者线程执行monitorexit时，monitor的进入数减1，如果减1后进入数为0，那么线程退出monitor，不在是这个monitor的所有者，其他线程可以尝试获取monitor的所有权。

e. ~~互斥：synchronized代码块中，只有一个线程能够获得锁，其他线程会阻塞；~~

f. ~~共享：类中的多个synchronized代码块，共享同一个对象锁；~~

g. ~~可重入性：同一线程可多次获得对象上的锁，通过计数器加锁加1，退出减1。~~

2. volatile（修饰符）：

a. 保证了可见性、有序性，不能保证原子性，用来修饰被不同线程访问的变量。

b. 原理：

i. 在对volatile变量进行写操作的时候，JVM会向处理器发送一条Lock前缀的指令，将这个变量所在工作内存的数据写回到主内存，在多处理器下，为了保证各个处理器的缓存是一致的，会实现缓存一致性协议，每个处理器通过嗅探找到主内存中的数据，来检测自己的数据是否过期，当处理器发现数据被修改，会将当前处理器的缓存行设置为无效状态。当前处理器要对这个数据进行修改操作的时候，会强制重新从系统内存里把数据读到处理器缓存里。

c. 原理：在volatile变量修饰的共享变量进行写操作时，会增加一个lock前缀的指令，lock前缀的指令会有2个操作，

i. 将当前工作内存的数据写到主内存中；

ii. 更新到主内存的操作会引起其他线程对应的工作内存设置为无效。

iii. lock前缀指令相当于一个内存屏蔽，volatile的底层就是通过内存屏蔽来实现的。

d. 可见性原理：

i. 线程对volatiel变量进行修改时，JVM会把该线程对应的工作内存中的共享变量值刷新到主内存中，刷新到主内存的操作会引起其他线程的工作内存设置成无效，JVM会把该线程对应的工作内存设置为无效，那么该线程只能从主内存中重新读取共享变量，保证变量值最新。

3. Lock（类），ReentrantLock实现了Lock接口，可以通过该类来实现线程同步。

a. 关键字：State状态值，双向链表，CAS+自旋；lock.lock()，lock.unlock()；

b. 存储结构：包括1个state状态值，是int类型的，和一个双向链表，state状态值用于表示锁的状态，双向链表用于存储等待中的线程；

c. 获得锁：线程通过CAS来对状态值修改，如果没有修改成功，会把线程放入双向链表中等待，尾进头出；当线程到头部时，尝试cas更新锁的state状态值，如果更新成功则表示获取锁成功，从链表头部移除。

d. 释放锁：修改状态值，调整等待链表。

二、可见性、原子性、有序性

1. 可见性：变量被操作之后，能够快速的写入内存，并提醒其他线程更新；
2. 原子性：过程属于原子操作；
3. 有序性：在Java内存模型中，允许处理器对指令进行重排序，重排序过程不会影响单线程程序的执行，却会影响到多线程并发执行的正确性。

三、synchronized和lock

1. 存在层面：synchronized是Java关键字，是jvm层的；lock是一个类；
2. 锁的获取：如果线程A获得锁，线程B会等待；
3. 锁的释放：synchronized中线程执行完或出现异常，线程释放锁；Lock中必须在finally中释放；
4. 锁的状态性能：sychronized中锁的状态是无法判断的，适用于少量同步；Lock锁的状态可以判断，适用于大量同步。

====2020/09/12====

一、锁

1. 作用：数据库使用锁是为了支持对共享数据进行并发访问，提供数据的完整性和一致性。
2. 分类：悲观锁、乐观锁、共享锁、排它锁、记录锁、间隙锁、临键锁。

二、锁的具体分类

1. 乐观锁和悲观锁：

a. 乐观锁：

i. 乐观锁是基于乐观的概念，每次去读数据的时候都认为其他事务不会对数据进行修改操作，但是在更新数据的时候，会先判断这个值是否被其他事务修改，如果发生冲突，就进行回滚，cas就是一种乐观锁；

ii. 优点：避免了加锁解锁的开销；

iii. 缺点：增加了冲突。

iv. 乐观锁是在提交的时候检验冲突，悲观锁是加锁避免冲突。资源竞争小，用乐观锁；资源竞争大，用悲观锁。

b. 悲观锁：

i. 认为数据随时会修改，所以在数据处理过程中需要对数据加锁，这样别的事务拿到这个数据就会block阻塞，直到它拿到锁。关系型数据库中的行锁、表锁、读锁、写锁都是悲观锁，在读之前先加上锁；悲观锁的实现方式，依赖于数据库；

ii. 优点：使用阻塞的方式，所以避免冲突；

iii. 缺点：并发性能不好，因为未获得锁会阻塞。

2. 共享锁和排它锁：（按锁的类型分类）

a. 共享锁：

i. 共享锁，S锁，也是读锁，事务A对数据加了共享锁，其他事务也只能对这个数据加S锁，多个用户可以同时读，但不允许有写操作，直到除了某个事务自身外，其他事务都放掉共享锁，这个事务才能获得排他锁。

b. 排它锁：

i. 排它锁，X锁，事务A对对象加X锁以后，只有事务A可以读写该对象，其他事务不能对该对象加任何锁，直到事务A释放X锁。

c. 意向锁：

i. 意向共享锁、意向排它锁；

ii. 意向锁是用来表示事务接下来想要获得的锁的类型；

iii. 意向排它锁和共享锁是不兼容的。

3. 一致性非锁定读和一致性锁定读

a. 非锁定读：

i. 通过MVCC实现，读取的是快照版本。

b. 锁定读：

i. 用法：s锁lock in share mode, x锁for update；

ii. 防止死锁。因为使用共享锁的时候，修改操作，需要先获得共享锁，读取数据，再升级为排它锁，然后进行修改操作。这样如果同时有多个事务对同一个数据对象申请共享锁，然后再同时升级为排它锁，这些事务都不会释放共享锁，而是等待其他事务释放，就造成了死锁。

iii. 因此，在数据修改前的select 语句中直接申请排它锁，其他数据就无法获取共享锁和排它锁，避免死锁。

MVCC

i. MVCC多版本并发控制，查询需要对资源加共享锁（s），修改需要对数据加排他锁（X）；

ii. 利用undo log回滚日志使读写不阻塞，实现了可重复读。当一个事务正在对一条数据进行修改的时候，该资源会被加上排它锁。在事务未提交时，对加锁资源进行的读操作，读操做无法读到被锁资源，会通过一些特殊的标识符去读undo log 中的数据，这样读到的是事务执行之前的数据。

4. 外键和锁

a. 在Innodb中，对于一个外键，如果没有显示的加索引，会自动地加上索引，避免表锁；

b. 对于外键的修改，需要对父表加S锁。这是，如果父表有X锁，则子表上加S锁的操作会阻塞；如果对父表加上S锁之后，则父表上不会加上X锁，保证了数据的一致性。

5. 行锁的三种算法（记录锁、间隙锁、临键锁）

a. 在可重复读的事务隔离级别下解决幻读的问题。幻读：可重复读的事务隔离级别下可能出现幻读，因为，可重复度保证了当前事务不会读取到其他事务已提交的update 操作。但同时，也会导致当前事务无法感知到其他事务的Insert或delete操作，这就产生了幻读。

b. 记录锁（Record lock）：

i. 锁住的是一条记录，它必须命中索引并且索引是唯一索引，否则会使用间隙锁；

c. 间隙锁（gap lock）：

i. 锁定的是一个范围，但不包含记录本身，多个事务可以同时持有间隙锁，但任何一个事务都不能在锁范围内进行插入修改操作。

ii. 间隙锁是基于非唯一索引的，它锁定一段范围内的索引记录。

iii. ~~非唯一索引的记录之间的间隙上加的锁，锁定的是尚未存在的记录，间隙锁是基于临键锁的~~；

d. 临键锁（Next-key lock）：

i. 临键锁是记录锁+间隙锁，也是用在非唯一索引的，锁定的是一个范围，并且锁定记录本身，是一个左开右闭的区间，只会出现在可重复读隔离级别，是为了防止幻读。

ii. ~~在根据非唯一索引对记录进行加锁修改操作时，会获得该记录的临键锁，和该记录下一个区间的间隙锁。~~

iii. ~~临键锁是记录锁+间隙锁，锁定的是一个左开右闭的索引区间。间隙锁和临键锁只会出现在可重复读隔离级别，可以避免幻读~~。