



45、线程，数据库连接，事务之间的关系

(1) 参考

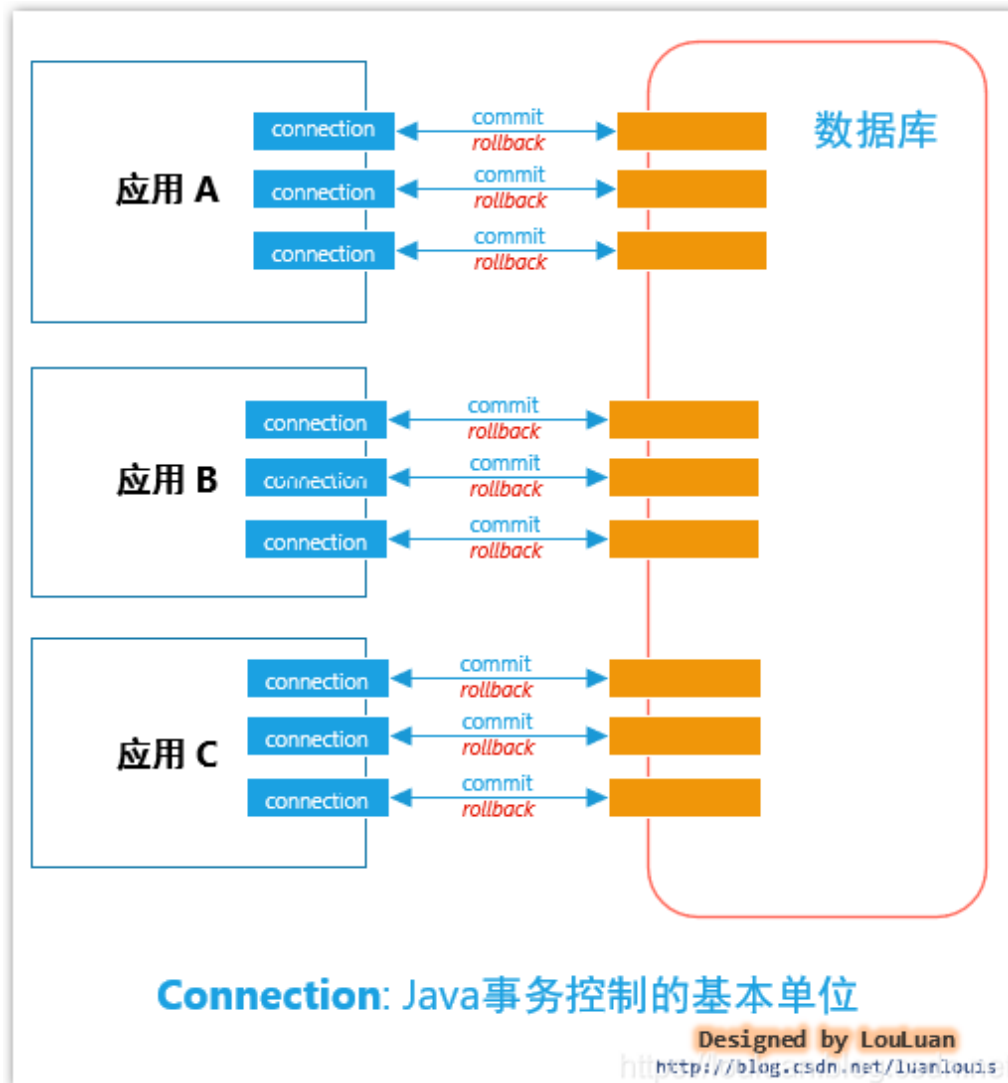
- [\(79条消息\) 事务篇-数据库连接和Java线程的关系_无双.蜗牛的博客-CSDN博客_连接和线程的关系](#)
- [\(82条消息\) 《Spring设计思想-事务篇》1数据库连接和Java线程的关系_亦山的博客-CSDN博客_java 多线程 数据库连接](#)

(2) Java中事务控制的基本单位：Connection

- 在java中API中，使用java.sql.Connection的实例，表示和数据库的一个连接，底层是采用TCP/IP方式进行连接。
- 连接后，可以通过操作此Connection实例，来进行事务控制。

思考：既然Connection实例能进行事务控制，那我们直接代码中创建并使用它可以吗？

然而事实上，我们并不能这么做，因为Connection和数据库有非常紧密的关系，数据库所在的系统资源是非常有限的。



(3) 有限的系统资源-----决定java.sql.Connection连接个数

- 应用程序和数据库之间建立连接，则数据库所在机器也会为此连接分配一定线程资源维护这个连接（客户端登录建立连接成功，数据库一方会保存此次会话连接信息默认保留8小时），如果连接数越多，消耗数据库所在机器的线程资源就越多。



- 不同的连接，可能会操作相同的表数据，出现高并发操作，所以数据库为了对ACID特性的支持（加锁：行锁，表锁），又会牺牲更多的资源。
- 综上，建立一个连接，会消耗数据库所在机器的一些资源，而这些资源是系统比较宝贵的有限资源，所以这些资源，就限制了数据库所能创建出的连接数和处理性能，当应用程序有数据库连接需求非常非常大时，很容易会达到数据库的连接并发瓶颈。

资源	说明
线程数	线程越多，线程的上下文切换会越频繁，会影响其处理能力
创建Connection的开销	由于Connection负责和数据库之间的通信，在创建环节会做大量的初始化，创建过程所需时间和内存资源上都有一定的开销
内存资源	为了维护Connection对象会消耗一定的内存
锁占用	在高并发模式下，不同的Connection可能会操作相同的表数据，就会存在锁的情况，数据库为了维护这种锁会有不少的内存开销

（4）数据库设置的最多支持多少个Connection连接？

```
-- 查看当前数据库最多支持多少数据库连接
show variables like '%max_connections%';
-- 设置当前运行时mysql的最大连接数，服务重启连接数将还原
set GLOBAL max_connections = 200;
-- 修改 my.ini 或者my.cnf 配置文件
max_connections = 200;
```

思考：数据库连接数设置越大越好吗？

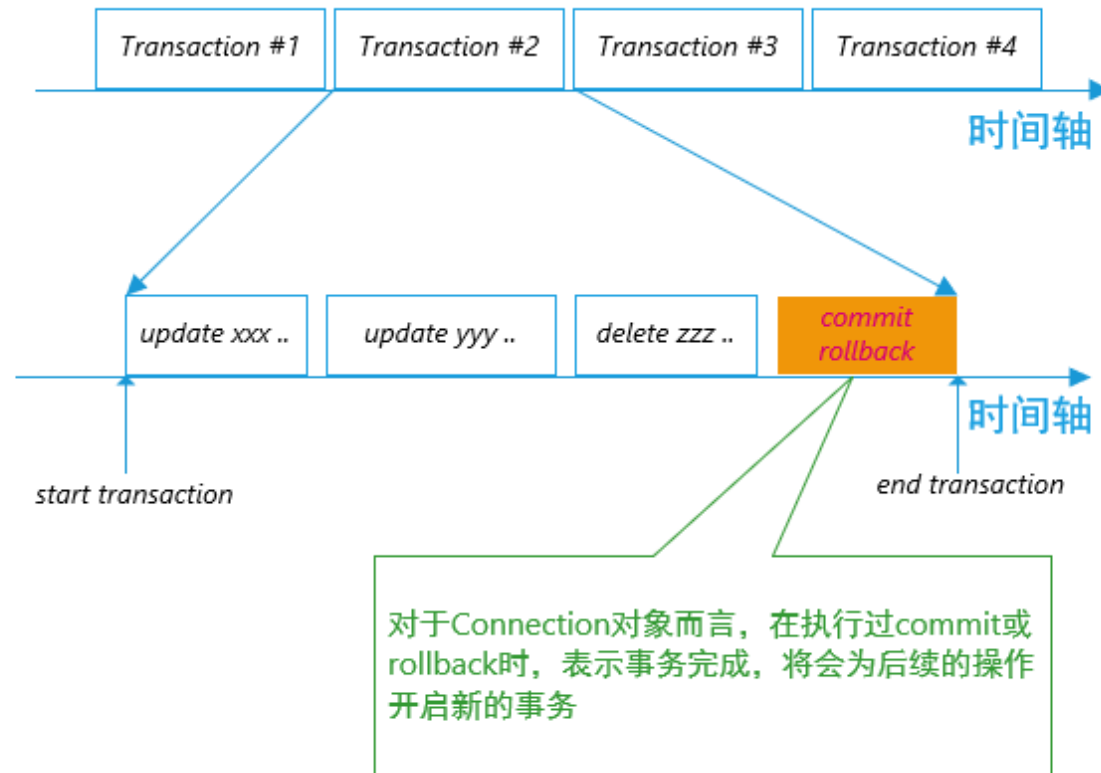
不是的，连接数越大，数据库所在操作系统需使用大量线程维护，而且会伴随着线程上下文的切换，并且连接数越多，操作同一张表的同一条数据的概率更大，反而导致整体数据库性能下降，所以具体连接数的大小设置，根据具体业务场景来调优。



(5) java.sql.Connection实例的特性

- 一个Connection实例中，在操作的时序上，事务和事务之间的执行是线性排开依次**串行执行**的。
 - 说白了，一个连接上的多个事务必须是线性的（序列化的事务操作模式），一个新事务的开启，必须在上一个事务完成之后(如果存在的话)。
- 而一个事务内是可以不限次数的SQL语句。
- Connection实例是基于TCP/IP协议的，所以在初始化建立连接之后，手动关闭之前，是会和数据库保持心跳存活连接，所以可以使用Connection实例执行不限次数的SQL语句请求，包括事务请求。
- 数据库连接池就是基于此特性建立的。

Connection对象事务控制操作时序



Designed by LouLuan

<http://blog.csdn.net/luanlouis>

(6) Java中如何实现java.sql.Connection实例中事务是线性操作。

(6-1) 一个Java线程的整个生命周期，可以独占一个Connection实例，去操作数据库吗？

- 可以独占，因为这样的话，一个线程内的所有操作都是同步的（线程互斥）和线性的（事务有序）。也就是说在事务进行期间，有一个重要的特性，Connection连接对象可以吸附在线程上，我把这种特性称之为事务对象

的线程吸附性 这种特性。但实际项目中并不会这么做，而是会根据这种特性做了一些改变，在Spring实现上，是使用了基于线程的ThreadLocal来表示这种线程依附行为。（后续会讲到）

（6-2）为什么不设计一个Java线程独占一个Connection连接？

- Java线程数量可能远超数据库连接数量，会出现僧多粥少的情况。
 - 上述例子查看连接数上限是200，而大型项目中，光HTTP请求线程数不止200个，如果独占，就会出现部分线程无法获取到数据库连接，进而无法完成业务操作。
- Java线程在工作过程中，真正使用连接操作数据库所占用的时间比例很短，则连接实际利用率低。
 - Java线程接收到HTTP请求后，业务中有很多环节需要处理（权限验证，参数校验，数值计算，SQL持久化），只有SQL持久化环节需要使用连接操作数据库，其余环节连接都是空闲状态，所以，如果Java线程整个生命周期都独占此连接，那么连接的使用率很低。

（6-3）一个Java线程不独占一个Connection连接的解决方案

- 普遍的解决方案是，**当线程需要做数据库操作时，才会去获取一个Connection连接**，线程使用完连接之后，连接被回收（放到连接池，并没销毁），被回收的连接等待下次直接分配给其它线程或本线程使用，只在操作数据库的时候获取连接，提高了连接的利用率。

（6-4）根据上述解决方案，思考一下，如果多个线程并发获取**同一个**Connection连接还会有什么问题？

- 会导致事务错乱。
- Thread #1的请求 update xxx 被thread #2回退掉，导致语句丢失，thread #1的事务不完整。

（6-5）多个线程并发获取的是同一个Connection连接会导致此连接中的事务之间错乱的解决方案

- 解决事务不完整的问题，本质上而言，其实就是多线程互斥去访问连接（synchronized,Lock），同一时间内，只有一个线程去占用Connection对象，去操作数据库。



```
java.sql.Connection sharedConnection = <创建流程>
## thread #1 的业务伪代码:
synchronized(sharedConnection){
    `update xxx`;
    `update yyy`;
    `commit`;
}
## thread #2 的业务伪代码:
synchronized(sharedConnection){
    `delete zzz`;
    `insert ttt`;
    `rollback`;
}
```

(6-6) 互斥性保证了多线程互斥使用同一个Connection连接, 但如果一个线程执行一部分业务SQL后, 出现异常, 没有catch来做rollback, 则存在尚未commit或rollback的SQL语句; 异常后会释放掉Connection锁对象, 另一个线程抢到一个Connection锁对象后, 执行自己的业务SQL, 并做了commit事务, 导致第一个线程中部分SQL被提交。

- 假如thread #2在执行语句 delete zzz,insert ttt,rollback的过程中, 在insert ttt之前有一段业务代码抛出了异常, 导致语句只执行到了 delete zzz, 没有做异常后的rollback操作, 这会导致在connection对象上有一个尚未提交的delete zzz请求;
- 当thread #1拿到了此connection 对象的锁之后, 接着执行 update xxx; update yyy; commit;
- 即:在两个线程执行完了之后, 对connection的操作为delete zzz; update xxx; update yyy; commit;

(6-7) 解决方案

- 确保多个线程在使用同一个Connection对象时，最终要明确每个线程对Connection做commit 或者rollback

```
java.sql.Connection sharedConnection = <创建流程>
## thread #1 的业务伪代码：
synchronized(sharedConnection){
    try{
        `update xxx`;
        `update yyy`;
        `commit`;
    } catch(Exception e){
        `rollback`; //之所以rollback ,是确保在执行事务的过程中，在connection对象上，清空尚未提交的所有SQL语句
    }
}
## thread #2 的业务伪代码：
synchronized(sharedConnection){
    try{
        `delete zzz`;
        `insert ttt`;
        `rollback`;
    } catch(Exception e){
        `rollback`; //之所以rollback ,是确保在执行事务的过程中，在connection对象上，清空尚未提交的所有SQL语句
    }
}
```

(6-8) 多个线程访问同一个Connection连接对象时，遵循两个基本原则

- 以互斥的方式访问Connection对象；



- 在线程执行结束时，应当最终及时提交(commit)或回滚(rollback)对Connection的影响；不允许存在尚未被提交或者回滚的语句。

(7) 应用与数据库建立连接后，当开启一个事务后，做完多个SQL操作后，事务提交或回滚，连接实例有必要销毁吗？

完全没有必要，根据前面可知，创建一个Connection对象实例的代价比较大（至少0.1s级别），而建立连接

- 后，可以不限次数的开启事务，及每个事务中不限次数的SQL操作，也就是说，当此次事务结束后，我可以紧接着使用这个Connection对象开启下一个事务。
- 所以，每次操作数据库前，都要重新创建一个Connection对象的话，会严重影响当前服务的性能和吞吐量。
- 目前的做法是完成一个事务操作后，并不会销毁Connection对象，而是将其回收到连接池中。

(8) 什么是连接池？

- 是一个统一管理一批java.sql.Connection实例的容器，一般会设置容器中连接数的上限。
- 连接池会为每一个获取Connection实例的请求，做连接分配，如果连接不足，设置等待时间。

连接池会根据Connection实例的使用情况，为了提高Connection实例的利用率，动态调整Connection实例的

- 数量，如果实际利用的Connection实例较少，会自动销毁掉一些处于无用状态的Connection实例，当请求量很大，再动态创建Connection实例。
- 常用连接池：HikariCP, 阿里的Druid, apache的DBCP

(9) 为何Service层手动开启一个事务后，调用三个DAO，就可以共用一个连接操作？

spring是通过 ThreadLocal <Connection>来保证同一个线程在其生命周期中，当多次操作数据库的时候（很

- 多个dao），每次都可以获得同一个数据库连接，为什么要确保是同一个数据库连接？是因为数据库的事务是基于数据库连接的，如果这个线程操作了三次dao，每次连接都不一样，那么就没办法保证这三次操作被同一个事务所管理。