Atitit 提升稳定性 数据库死锁

目录

[1.1. 配置数据库死锁检测超时时间从默认50s到10s 1](#_Toc8336)

[1.2. 调整隔离级别到read commit 1](#_Toc3939)

[1.3. mysql数据库连接使用完毕立即释放掉。 1](#_Toc13225)

[1.4. 每个连接增加gc检测，超时强制销毁退出。 2](#_Toc1737)

[1.5. 分库，分为100个库，自然提升稳定性， 2](#_Toc22272)

[1.6. 使用小事务机制 只针对必要业务上大事务 2](#_Toc11594)

[2. Db开启死锁检测死锁日志 2](#_Toc10892)

[2.1. 分析是否有长事务 3](#_Toc29594)

[3. 框架优化 ----spring 3](#_Toc21090)

[3.1. 限制事务范围防止死锁 3](#_Toc21216)

[3.2. 调整spring 事务管理器 缩小事务范围 减少表锁 3](#_Toc23249)

[3.3. 大批量处理不要使用事务，或者只针对单个条目执行事务 4](#_Toc18160)

[4. 批量处理优化 提升性能 4](#_Toc22064)

[4.1. 使用原始接口，不要orm提升update性能 4](#_Toc20299)

[4.2. 稍微合并sql但要注意死锁问题 4](#_Toc19035)

[4.3. 增大连接数 4](#_Toc21812)

[4.4. 使用dml cache模式 4](#_Toc29436)

[4.5. 使用单语句事务模式 4](#_Toc1884)

[5. 常见死锁问题原因 4](#_Toc18178)

[5.1. 行锁变表锁导致死锁问题 4](#_Toc32751)

[5.2. 批量更新 导致对长事务 5](#_Toc25340)

[5.3. Ab ba死锁问题 5](#_Toc28681)

[5.3.1. .1 死锁案例一ab ba问题 5](#_Toc18230)

[5.3.2. 3.2 死锁案例二 gap锁 6](#_Toc9183)

[5.3.3. 3.3 死锁案例三 范围批量修改 7](#_Toc4037)

[6. 其它关于查看死锁的命令： 8](#_Toc3358)

## 配置数据库死锁检测超时时间从默认50s到10s

## 调整隔离级别到read commit

加快事务进程 与简单化 。。Oracle 和mssql的隔离级别就是这个read commit，业务准确性不会有问题

## mysql数据库连接使用完毕立即释放掉。

方法，因为mysql数据库对行锁问题，很容易升级为表锁。所以不建议使用数据库连接池，因为会造成连接迟迟得不到释放，导致事务一直进行中。。mysql数据库连接使用完毕立即释放掉。。

线程池提升的性能可以增加机器负载均衡解决即可。。稳定性优先与性能

## 每个连接增加gc检测，超时强制销毁退出。

。。相当于超时熔断器，一般一个事务执行7秒足够了

## 分库，分为100个库，自然提升稳定性，

锁表死锁概率自然减少到了百分之一

## 使用小事务机制 只针对必要业务上大事务

# Db开启死锁检测死锁日志

mysql 5.7.15 之后 增加了innodb\_deadlock\_detect 参数，控制是否打开死锁检测。

show global variables like '%timeout%';

set global lock\_wait\_timeout=20; -- dml lock

set global innodb\_lock\_wait\_timeout=20;

set global wait\_timeout=30;

- 调整隔离级别 和事务日志刷盘模式

set global transaction isolation level read committed;

show global variables like '%tx\_isolation%';

set global innodb\_flush\_log\_at\_trx\_commit=2;

show global variables like '%tx\_isolation%';

show global variables like '%trx%';

show global variables like '%dead%';

show global variables like '%lock%';

show global variables like '%conn%';

show global variables like '%lock%';

show global variables like '%timeout%';

show global variables like '%deadlock%';

## 分析是否有长事务

可以启动 查看trx表里面的事务。。

可能批量处理就少数事务那就是有问题类。

## 增加定时任务 死锁检测与解除

# 框架优化 ----spring

## 限制事务范围防止死锁

## 调整spring 事务管理器 缩小事务范围 减少表锁

尽量控制事务的大小，减少锁定的资源量和锁定时间长度；

## 大批量处理不要使用事务，或者只针对单个条目执行事务

# 批量处理优化 提升性能

## 使用原始接口，不要orm提升update性能

## 稍微合并sql但要注意死锁问题

## 增大连接数

## 使用dml cache模式

## 使用单语句事务模式

# 常见死锁问题原因

## 行锁变表锁导致死锁问题

START TRANSACTION;

update user set bals=111 where id=1;

commit;

如果此事务中途等待或长时间不提交，那么会锁表 锁住整个表

造成其他用户也不能修改自己的数据

START TRANSACTION;

update user set bals=22 where id=2;

commit;

Rc级别修改可解决此问题

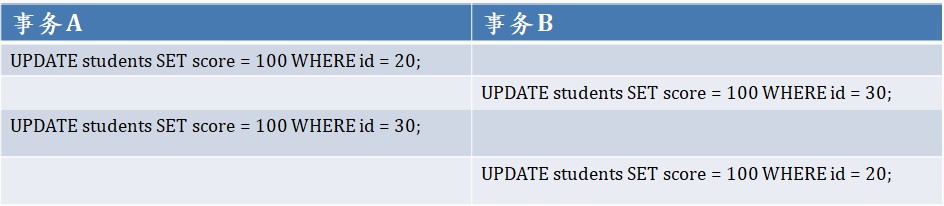
## 批量更新 导致对长事务

## Ab ba死锁问题

增加超时检测 和 及时销毁连接

小事务也可以解决此问题

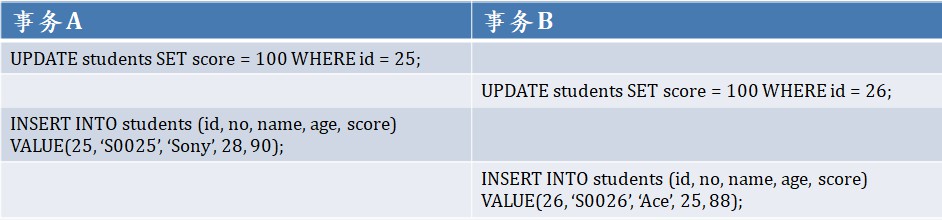
### .1 死锁案例一ab ba问题



死锁的根本原因是有两个或多个事务之间加锁顺序的不一致导致的，这个死锁案例其实是最经典的死锁场景。

首先，事务 A 获取 id = 20 的锁（lock\_mode X locks rec but not gap），事务 B 获取 id = 30 的锁；然后，事务 A 试图获取 id = 30 的锁，而该锁已经被事务 B 持有，所以事务 A 等待事务 B 释放该锁，然后事务 B 又试图获取 id = 20 的锁，这个锁被事务 A 占有，于是两个事务之间相互等待，导致死锁。

### 3.2 死锁案例二 gap锁



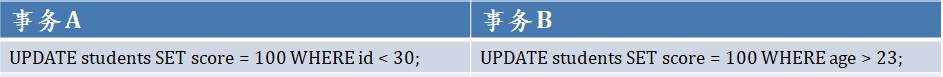
首先事务 A 和事务 B 执行了两条 UPDATE 语句，但是由于 id = 25 和 id = 26 记录都不存在，事务 A 和 事务 B 并没有更新任何记录，但是由于数据库隔离级别为 RR，所以会在 (20, 30) 之间加上间隙锁（lock\_mode X locks gap before rec），间隙锁和间隙锁并不冲突。之后事务 A 和事务 B 分别执行 INSERT 语句要插入记录 id = 25 和 id = 26，需要在 (20, 30) 之间加插入意向锁（lock\_mode X locks gap before rec insert intention），插入意向锁和间隙锁冲突，所以两个事务互相等待，最后形成死锁。

要解决这个死锁很简单，显然，前面两条 UPDATE 语句是无效的，将其删除即可。另外也可以将数据库隔离级别改成 RC，这样在 UPDATE 的时候就不会有间隙锁了。这个案例正是文章开头提到的死锁日志中的死锁场景，别看这个 UPDATE 语句是无效的，看起来很傻，但是确实是真实的场景，因为在真实的项目中代码会非常复杂，比如采用了 ORM 框架，应用层和数据层代码分离，一般开发人员写代码时都不知道会生成什么样的 SQL 语句，我也是从 DBA 那里拿到了 binlog，然后从里面找到了事务执行的所有 SQL 语句，发现其中竟然有一行无效的 UPDATE 语句，最后追本溯源，找到对应的应用代码，将其删除，从而修复了这个死锁。

Gap 锁往往是程序中导致死锁的真凶，由于默认情况下 MySQL 的隔离级别是 RR，所以如果能确定幻读和不可重复读对应用的影响不大，可以考虑将隔离级别改成 RC，可以避免 Gap 锁导致的死锁；

小事务也可以解决此问题

### 3.3 死锁案例三 范围批量修改



别看这个案例里每个事务都只有一条 SQL 语句，但是却实实在在可能会导致死锁问题，其实说起来，这个死锁和案例一并没有什么区别，只不过理解起来要更深入一点。要知道在范围查询时，加锁是一条记录一条记录挨个加锁的，所以虽然只有一条 SQL 语句，如果两条 SQL 语句的加锁顺序不一样，也会导致死锁。

在案例一中，事务 A 的加锁顺序为： id = 20 -> 30，事务 B 的加锁顺序为：id = 30 -> 20，正好相反，所以会导致死锁。这里的情景也是一样，事务 A 的范围条件为 id < 30，加锁顺序为：id = 15 -> 18 -> 20，事务 B 走的是二级索引 age，加锁顺序为：(age, id) = (24, 18) -> (24, 20) -> (25, 15) -> (25, 49)，其中，对 id 的加锁顺序为 id = 18 -> 20 -> 15 -> 49。可以看到事务 A 先锁 15，再锁 18，而事务 B 先锁 18，再锁 15，从而形成死锁

小事务解决

# 其它关于查看死锁的命令：

1：查看当前的事务  
SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_TRX;

2：查看当前锁定的事务

SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_LOCKS;

3：查看当前等锁的事务  
SELECT \* FROM INFORMATION\_SCHEMA.INNODB\_LOCK\_WAITS;