# mysql死锁检测机制初探

## 死锁说明

### 原理简介

锁情况，圆圈代表事务，方框代表资源，实线代表拥有，虚线代表锁等待，红线代表准备上锁

2

3

4

1

从上图看，A->2->B->1->A，形成死锁环路。同样，由于存在事务的锁等待，A->2->C->1->，在B释放资源2后回造成新的锁等待。

### 代码说明

查找并解决死锁的伪代码如下

do

{

deadlock\_search(); //查找死锁，如果发现死锁返回需牺牲的事务id，否则0

if (同一资源上锁等待超过200)

{

归为死锁情况，回滚当前事务

}

else if (发现死锁 而且 被牺牲的事务为当前lock的拥有者)

{

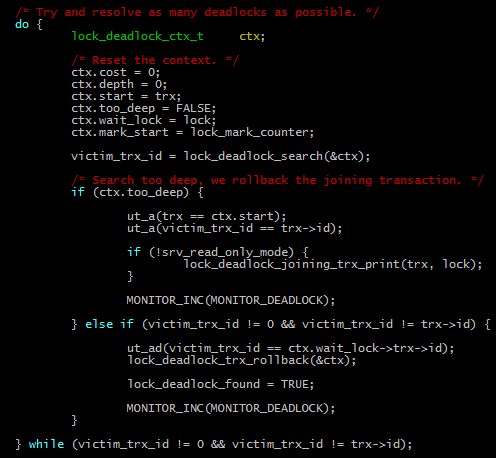
回滚被牺牲事务

}

} while (需要继续检测死锁);

需要继续检测死锁：如果找到死锁，且死锁中被牺牲的事务不是当前事务

外层代码



死锁检测代码

返回0表示未发现死锁，不为0表示发现死锁，被牺牲事务的id

伪代码如下

获取对象上的第一个锁

for (;;)

{

从堆栈中获取上一个

if (未找到锁) {

退出循环

}

else if (找到的锁是当前事务的锁)

{

//此时说明该锁上的分支已经完成遍历

}

else if(当前锁无冲突)

{

获取下一个锁对象

}

else if (形成环路)

{

//死锁发生，选择需要回滚的事务

}

else if (死锁搜索达到阀值)

{

当前事务作为被牺牲的事务，返回其id

}

else if (当前的锁处于锁等待状态) {

以当前锁作为分支起点，进行死锁检查

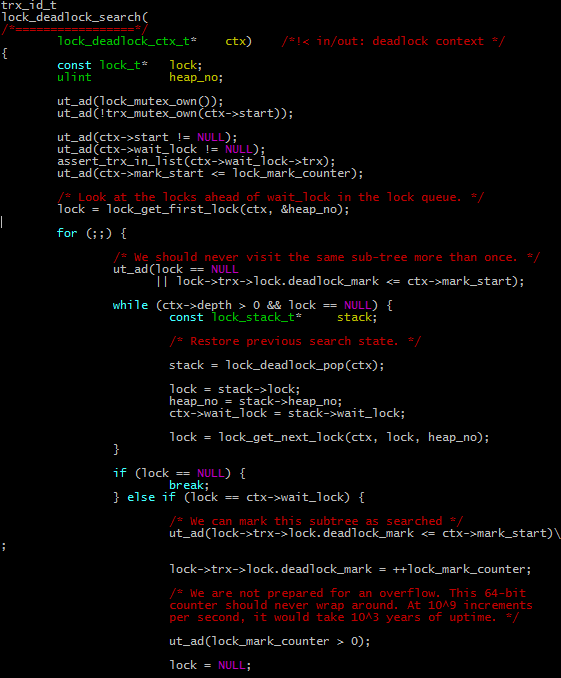
}

else {

获取下一个锁

}

}



## 实际例子

### 报错信息

------------------------

LATEST DETECTED DEADLOCK

------------------------

2015-09-02 11:13:21 7f8cb7cf3700TOO DEEP（A） OR LONG SEARCH IN THE LOCK TABLE WAITS-FOR GRAPH, WE WILL ROLL BACK FOLLOWING TRANSACTION

\*\*\* TRANSACTION:

TRANSACTION 7513201817, ACTIVE 0 sec setting auto-inc lock

mysql tables in use 1, locked 1

1 lock struct(s), heap size 360, 0 row lock(s)

MySQL thread id 19894045, OS thread handle 0x7f8cb7cf3700, query id 15242999467 10.8.92.22 ws\_secevt02 update

INSERT INTO event\_cp1001001\_v3( column04 , column06 , column19 , column20 , column13 , column03 , column01 , column07 , column02 , row\_key , column05 , column12 , column11 ) VALUES('221.204.184.112', '', null, '0', 'LOGIN\_APPLY', 'D242710250', '480523', '', 'D242710250', 'CP1001001\_2629716407820\_8b418b40-5120-11e5-9d81-4bd7dff89f6a', 'CRM', 'T', '2015-09-02 11:13:12.351')

\*\*\* WAITING FOR THIS LOCK TO BE GRANTED:

TABLE LOCK table `seceventshard02db`.`event\_cp1001001\_v3` trx id 7513201817 lock

\*\*\* WE ROLL BACK TRANSACTION (2) （B）

### 说明

A标识：一旦出现A标识出的红字，说明同一资源上上锁事务超过mysql的阀值200

B标识：一旦出现B标识出的红字，说明后发起的事务无法获得锁，后发起事务在上锁的过程中将自己回滚掉