数据库锁的产生原因及解决办法

数据库和操作系统一样，是一个多用户使用的共享资源。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。加锁是实现数据库并 发控制的一个非常重要的技术。在实际应用中经常会遇到的与锁相关的异常情况，当两个事务需要一组有冲突的锁，而不能将事务继续下去的话，就会出现死锁，严 重影响应用的正常执行。

在数据库中有两种基本的锁类型：排它锁（Exclusive Locks，即 X 锁）和共享锁（Share Locks，即 S 锁）。当数据对象被加上排它锁时，其他的事务不能对它读取和修改。加了共享锁的数据对象可以被其他事务读取，但不能修改。数据库利用这两 种基本的锁类型来对数据库的事务进行并发控制。

死锁的第一种情况

一个用户 A 访问表 A(锁住了表 A),然后又访问表 B；另一个用户 B 访问表 B(锁住了表 B)，然后企图访问表 A；这时用户 A 由于用户 B 已经锁住表 B，它必须等待用户 B 释放表 B 才能继续，同样用户 B 要等用户 A 释放表 A 才能继续，这就死锁就产生了。

解决方法：

这种死锁比较常见，是由于程序的 BUG 产生的，除了调整的程序的逻辑没有其它的办法。仔细分析程序的逻辑，对于数据库的多表操作时，尽量按照相同的顺序进 行处理，尽量避免同时锁定两个资源，如操作 A 和 B 两张表时，总是按先 A 后 B 的顺序处理， 必须同时锁定两个资源时，要保证在任何时刻都应该按照相同的顺序来锁定资源。

死锁的第二种情况

用户 A 查询一条纪录，然后修改该条纪录；这时用户 B 修改该条纪录，这时用户 A 的事务里锁的性质由查询的共享锁企图上升到独占锁，而用户 B 里的独占锁由于 A 有共享锁存在所以必须等 A 释放掉共享锁，而 A 由于 B 的独占锁而无法上升的独占锁也就不可能释放共享锁，于是出现了死锁。这种死锁比较隐蔽，但在稍大点的项 目中经常发生。如在某项目中，页面上的按钮点击后，没有使按钮立刻失效，使得用户会多次快速点击同一按钮，这样同一段代码对数据库同一条记录进行多次操 作，很容易就出现这种死锁的情况。

解决方法：

1、对于按钮等控件，点击后使其立刻失效，不让用户重复点击，避免对同时对同一条记录操作。

2、使用乐观锁进行控制。乐观锁大多是基于数据版本（Version）记录机制实现。即为数据增加一个版本标识，在基于数据库表的版本解决方案中，一般是通过为数据库表增加一个“version”字段来实现。读取出数据时，将此版本号一同读出，之后更新时，对此版本号加一。此时，将提交数据的版本数据与数据库表对应记录的当前版本信息进行比对，如果提交的数据版本号大于数据库表当前版本号，则予以更新，否则认为是过期数据。乐观锁机制避免了长事务中的数据 库加锁开销（用户 A 和用户 B 操作过程中，都没有对数据库数据加锁），大大提升了大并发量下的系统整体性能表现。Hibernate 在其数据访问引擎中内置了乐观锁实现。需要注意的是，由于乐观锁机制是在我们的系统中实现，来自外部系统的用户更新操作不受我们系统的控制，因此可能会造 成脏数据被更新到数据库中。

3、使用悲观锁进行控制。悲观锁大多数情况下依靠数据库的锁机制实现，如 Oracle 的 Select … for update 语句，以保证操作最大程度的独占性。但随之而来的就是数据库性能的大量开销，特别是对长事务而言，这样的开销往往无法承受。如一个金融系统， 当某个操作员读取用户的数据，并在读出的用户数据的基础上进行修改时（如更改用户账户余额），如果采用悲观锁机制，也就意味着整个操作过程中（从操作员读 出数据、开始修改直至提交修改结果的全过程，甚至还包括操作员中途去煮咖啡的时间），数据库记录始终处于加锁状态，可以想见，如果面对成百上千个并发，这 样的情况将导致灾难性的后果。所以，采用悲观锁进行控制时一定要考虑清楚。

死锁的第三种情况

如果在事务中执行了一条不满足条件的 update 语句，则执行全表扫描，把行级锁上升为表级锁，多个这样的事务执行后，就很容易产生死锁和阻塞。类似的情 况还有当表中的数据量非常庞大而索引建的过少或不合适的时候，使得经常发生全表扫描，最终应用系统会越来越慢，最终发生阻塞或死锁。

解决方法：

SQL 语句中不要使用太复杂的关联多表的查询；使用“执行计划”对 SQL 语句进行分析，对于有全表扫描的 SQL 语句，建立相应的索引进行优化。

5．小结

总体上来说，产生内存溢出与锁表都是由于代码写的不好造成的，因此提高代码的质量是最根本的解决办法。有的人认为先把功能实现，有 BUG 时再在测试阶段进 行修正，这种想法是错误的。正如一件产品的质量是在生产制造的过程中决定的，而不是质量检测时决定的，软件的质量在设计与编码阶段就已经决定了，测试只是 对软件质量的一个验证，因为测试不可能找出软件中所有的 BUG。

相关文章:

ORACLE 里几种锁模式

遇到一个多事务并发的问题

mysql 数据库锁

推荐圈子: Pipboy

更多相关推荐 对锁机制的研究要具备两个条件：

1．数据量大

2．多个用户同时并发

如果缺少这两个条件，数据库不容易产生死锁问题。研究起来可能会事倍功半。如果这两个条件都有，但你还是按数据库缺省设置来处理数据，则会带来很多的问题，比如：

1）丢失更新

A,B 两个用户读同一数据并进行修改,其中一个用户的修改结果破坏了另一个修改的结果

2）脏读

A 用户修改了数据时,B 用户也在读该数据,但 A 用户因为某些原因取消了对数据的修改,数据恢复原值,此时 B 得到的数据就与数据库内的数据产生了不一致

3）不可重复读

B 用户读出该数据并修改,同时，A 用户也在读取数据，此时 A 用户再读取数据时发现前后两次的值不一致

SQL SERVER 作为多用户数据库系统，以事务为单位，使用锁来实现并发控制。SQLSERVER 使用“锁”确保事务完整性和数据一致性。

一、锁的概念

锁（LOCKING）是最常用的并发控制机构。是防止其他事务访问指定的资源控制、实现并发控制的一种主要手段。锁是事务对某个数据库中的资源（如表和记 录）存取前，先向系统提出请求，封锁该资源，事务获得锁后，即取得对数据的控制权，在事务释放它的锁之前，其他事务不能更新此数据。当事务撤消后，释放被 锁定的资源。
当一个用户锁住数据库中的某个对象时，其他用户就不能再访问该对象

二、锁的粒度
SQL Server 2000 具有多粒度锁定，允许一个事务锁定不同类型的的资源。为了使锁定的成本减至最少，SQL Server 自动将资源锁定在适合任务的级别。锁定在较小的粒度（例如行）可以增加并发但需要较大的开销，因为如果锁定了许多行，则需要控制更多的锁。锁定在较大的粒 度（例如表）就并发而言是相当昂贵的，因为锁定整个表限制了其它事务对表中任意部分进行访问，但要求的开销较低，因为需要维护的锁较少。SQL Server 可以锁定行、页、扩展盘区、表、库等资源。

资源 级别 描述
RID 行锁 表中的单个行
Key 行级锁 索引中的行
Page 页级锁 一个数据页或者索引页
Extent 页级锁 一组数据页或者索引页
Table 表级锁 整个表
Database 数据库级锁 整个数据库

选择多大的粒度，根据对数据的操作而定。如果是更新表中所有的行，则用表级锁;如果是更新表中的某一行，则用行级锁。
行级锁是一种最优锁，因为行级锁不可能出现数据既被占用又没有使用的浪费现象。但是，如果用户事务中频繁对某个表中的多条记录操作，将导致对该表的许多记 录行都加上了行级锁，数据库系统中锁的数目会急剧增加，这样就加重了系统负荷，影响系统性能。因此，在 SQL Server 中，还支持锁升级(lock escalation)。
所谓锁升级是指调整锁的粒度，将多个低粒度的锁替换成少数的更高粒度的锁，

以此来降低系统负荷。在 SQL Server 中当一个事务中的锁较多，达到锁升级门限时，系统自动将行级锁和页面锁升级为表级锁。
特别值得注意的是，在 SQL Server 中，锁的升级门限以及锁升级是由系统自动来确定的，不需要用户设置。

三、锁的模式
锁模式以及描述表

锁模式 描述
共享（S） 用于不更改或不更新数据（只读操作），如 SELECT 语句
更新（U） 用于可更新的资源中。防止当多个会话在读取、锁定以及随后可能进行的资源更新时发生常见形式的死锁。
排它（X） 用于数据修改操作，例如 INSERT、UPDATE 或 DELETE。确保不会同时对同一资源进行多重更新
意向 当 Microsoft SQL Server 数据库引擎获取低级别的锁时，它还将在包含更低级别对象的对象上放置意向锁.例如： 当锁定行或索引键范围时，数据库引擎将在包含行或键的页上放置意向锁。当锁定页时，数据库引擎将在包含页的更高级别的对象上放置意向锁。

意向锁的类型为：意向共享（IS）、意向排它（IX）以及意向排它共享（SIX）
架构 在执行依赖于表架构的操作时使用。架构锁的类型为：架构修改（Sch-M）和架构稳定（Sch-S）
大容量更新（BU） 向表中大容量复制数据并指定了 TABLOCK 提示时使用

四 SQL Server 中锁的设置
1 处理死锁和设置死锁优先级
死锁就是多个用户申请不同封锁,由于申请者均拥有一部分封锁权而又等待其他用户拥有的部分封锁而引起的无休止的等待
可以使用 SET DEADLOCK_PRIORITY 控制在发生死锁情况时会话的反应方式。
Syntax:

SET DEADLOCK_PRIORITY { LOW | NORMAL}

其中 LOW 说明该进程会话的优先级较低，在出现死锁时，可以首先中断该进程的事务。

2 处理超时和设置锁超时持续时间。

@@LOCK_TIMEOUT 返回当前会话的当前锁超时设置，单位为毫秒

SET LOCK_TIMEOUT 设置允许应用程序设置语句等待阻塞资源的最长时间。当语句等待的时间大于 LOCK_TIMEOUT 设置时，系统将自动取消阻塞的语句，并给应用程序返回"已超过了锁请求超时时段"的 1222 号错误信息

示例

1）将锁超时期限设置为 1,800 毫秒。

SET LOCK_TIMEOUT 1800

2) 配置索引的锁定粒度

可以使用 sp_indexoption 系统存储过程来设置用于索引的锁定粒度

3）设置事务隔离级别

SET TRANSACTION ISOLATION LEVEL

五 查看锁的信息

1 执行 EXEC SP_LOCK 报告有关锁的信息

2 查询分析器中按 Ctrl+2 可以看到锁的信息

六、奇怪的 sql 语句

Java 代码

```
begin tran
update titles set title_idid＝title_id   where 1＝2
if (selectavg(price)fromtitles)>
begin
update titles set price＝price*1.10
where price<(select avg(price)from titles)
end
commit tran

begin tran
```

```
update titles set title_idid=title_id   where 1=2
if (selectavg(price)fromtitles)>
begin
update titles set price=price*1.10
where price<(select avg(price)from titles)
end
commit tran
```

update titles set title_idid=title_id   where 1=2，这个条件是永远也不会成立的，如此写的含义是什么呢？

这里的 where 子句看起来很奇怪，尽管计算出的结果总是 false。当优化器处理此查询时，因为它找不到任何有效的 SARG，它的查询规划就会强制使用 一个独占锁定来进行表扫描。此事务执行时，where 子句立即得到一个 false 值，于是不会执行实际上的扫描，但此进程仍得到了一个独占的表锁定。

因为此进程现在已有一个独占的表锁，所以可以保证没有其他事务会修改任何数据行，能进行重复读，且避免了由于 holdlock 所引起的潜在性死锁。

但是，在使用表锁定来尽可能地减少死锁的同时，也增加了对表锁定的争用。因此，在实现这种方法之前，你需要权衡一下：避免死锁是否比允许并发地对表进行访问更重要。

所以，在这个事务中，没有其他进程修改表中任何行的 price。

七 如何避免死锁

1 使用事务时，尽量缩短事务的逻辑处理过程，及早提交或回滚事务；

2 设置死锁超时参数为合理范围，如：3 分钟-10 分种；超过时间，自动放弃本次操作，避免进程悬挂；

3 所有的 SP 都要有错误处理（通过@error）

4 一般不要修改 SQL SERVER 事务的默认级别。不推荐强行加锁

5 优化程序，检查并避免死锁现象出现；

1）合理安排表访问顺序

2）在事务中尽量避免用户干预，尽量使一个事务处理的任务少些。

3）采用脏读技术。脏读由于不对被访问的表加锁，而避免了锁冲突。在客户机/服务器应用环境中，有些事务往往不允许读脏数据，但在特定的条件下，我们

可以用脏读。

4）数据访问时域离散法。数据访问时域离散法是指在客户机/服务器结构中，采取各种控制手段控制对数据库或数据库中的对象访问时间段。主要通过以下方式实 现：合理安排后台事务的执行时间，采用工作流对后台事务进行统一管理。工作流在管理任务时，一方面限制同一类任务的线程数（往往限制为 1 个），防止资源过多占 用；另一方面合理安排不同任务执行时序、时间，尽量避免多个后台任务同时执行，另外，避免在前台交易高峰时间运行后台任务

5）数据存储空间离散法。数据存储空间离散法是指采取各种手段，将逻辑上在一个表中的数据分散到若干离散的空间上去，以便改善对表的访问性能。主要通过以下方法实现：第一，将大表按行或列分解为若干小表；第二，按不同的用户群分解。

6）使用尽可能低的隔离性级别。隔离性级别是指为保证数据库数据的完整性和一致性而使多用户事务隔离的程度，SQL92 定义了 4 种隔离性级别：未提交读、提交读、可重复读和可串行。如果选择过高的隔离性级别，如可串行，虽然系统可以因实现更好隔离性而更大程度上保证数据的完整性和一致性，但各事务间冲突而 死锁的机会大大增加，大大影响了系统性能。

7）使用 Bound Connections。Bound connections 允许两个或多个事务连接共享事务和锁，而且任何一个事务连接要申请锁如同另外一个事务要申请锁一样，因此可以允许这些事务共享数据而不会有加锁的冲突。

8）考虑使用乐观锁定或使事务首先获得一个独占锁定。

八如何对行、 表、数据库加锁

1 如何锁一个表的某一行

Java 代码

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SELECT * FROM table1 ROWLOCK WHERE A = 'a1'

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SELECT * FROM table1 ROWLOCK WHERE A = 'a1'
```

2 锁定数据库的一个表

```
select col1 from 表 (tablockx) where 1=1 ;
```

加锁后其它人不可操作，直到加锁用户解锁，用 commit 或 rollback 解锁

3.实例

建表

Java 代码

```
create table table1(A varchar(50)   not   null, B varchar(50) ,C
varchar(50));
create table table2(D varchar(50),E varchar(50))
insert table1 (A,B,C) values( ‘a1’ ,’ b1’ ,’ c1’ );
insert table1 (A,B,C) values( ‘a2’ ,’ b2’ ,’ c2’ );
insert table1 (A,B,C) values( ‘a3’ ,’ b3’ ,’ c3’ );
insert table2 (D,E) values( ‘d1’ ,’ e1’ );
insert table2 (D,E) values( ‘d2’ ,’ e2’ );

create table table1(A varchar(50)   not   null, B varchar(50) ,C
varchar(50));
create table table2(D varchar(50),E varchar(50))
insert table1 (A,B,C) values( ‘a1’ ,’ b1’ ,’ c1’ );
insert table1 (A,B,C) values( ‘a2’ ,’ b2’ ,’ c2’ );
insert table1 (A,B,C) values( ‘a3’ ,’ b3’ ,’ c3’ );
insert table2 (D,E) values( ‘d1’ ,’ e1’ );
insert table2 (D,E) values( ‘d2’ ,’ e2’ );
```


1）排它锁

Java 代码

```
-- A 事务先更新 table1 表，在更新时，对其他事务进行排他
begin tran
update table1 set A=’aa’ where B=’b2’;
waitfor delay ’00:00:30’;  --等待 30 秒
commit tran
-- A 事务先更新 table2 表
begin tran
select * from table1 where B=’b2’;
commit tran
```

-- A 事务先更新 table1 表，在更新时，对其他事务进行排他

```
begin tran
update table1 set A='aa' where B='b2';
waitfor delay '00:00:30';  --等待 30 秒
commit tran
```

-- A 事务先更新 table2 表

```
begin tran
select * from table1 where B='b2';
```

commit tran 若同时执行上述两个事务，则 select 查询必须等待 update 执行完毕才能执行即要等待 30 秒

2）共享锁

Java 代码

-- A 事务先查询 table1 表，在查询时，加共享锁，防止其他事务对该表进行修改操作

```
begin tran
select * from table1 holdlock where B='b2' ;
-holdlock 人为加锁
waitfor delay '00:00:30';--等待 30 秒
commit tran
```


-- A 事务先查询 table1 表，后更改 table1 表

```
begin tran
select A,C from table1 where B='b2';
update table1 set A='aa' where B='b2';
commit tran
```

-- A 事务先查询 table1 表，在查询时，加共享锁，防止其他事务对该表进行修改操作

```
begin tran
select * from table1 holdlock where B='b2' ;
-holdlock 人为加锁
waitfor delay '00:00:30';--等待 30 秒
commit tran
```

-- A 事务先查询 table1 表，后更改 table1 表

```
begin tran
select A,C from table1 where B='b2';
update table1 set A='aa' where B='b2';
```

commit tran 若并发执行上述两个事务，则 B 事务中的 select 查询可以执行，而 update 必须等待第一个事务释放共享锁转为排它锁后才能执行即要等待 30 秒

3）死锁

Java 代码

-- A 事务先更新 table1 表，然后延时 30 秒，再更新 table2 表；

```
begin tran
update table1 set A='aa' where B='b2';
--这将在 Table1 中生成排他行锁，直到事务完成后才会释放该锁。
waitfor delay '00:00:30';
--进入延时
update table2 set D='d5' where E='e1' ;
commit tran
```

-- B 事务先更新 table2 表，然后延时 10 秒，再更新 table1 表；

```
begin tran
update table2 set D='d5' where E='e1';
--这将在 Table2 中生成排他行锁，直到事务完成后才会释放该锁
waitfor delay '00:00:10'
--进入延时
update table1 set A='aa' where B='b2' ;
commit tran
```

-- A 事务先更新 table1 表，然后延时 30 秒，再更新 table2 表；

```
begin tran
update table1 set A='aa' where B='b2';
--这将在 Table1 中生成排他行锁，直到事务完成后才会释放该锁。
waitfor delay '00:00:30';
--进入延时
```

```
update table2 set D='d5' where E='e1' ;
commit tran
-- B 事务先更新 table2 表，然后延时 10 秒，再更新 table1 表；
begin tran
update table2 set D='d5' where E='e1';
--这将在 Table2 中生成排他行锁，直到事务完成后才会释放该锁
waitfor delay '00:00:10'
--进入延时
update table1 set A='aa' where B='b2' ;
commit tran
```

若并发执行上述两个事务，A,B 两事务都要等待对方释放排他锁，这样便形成了死锁。

九、sqlserver 提供的表级锁

sqlserver 所指定的表级锁定提示有如下几种

1. HOLDLOCK: 在该表上保持共享锁，直到整个事务结束，而不是在语句执行完立即释放所添加的锁。
2. NOLOCK：不添加共享锁和排它锁，当这个选项生效后，可能读到未提交读的数据或“脏数据”，这个选项仅仅应用于 SELECT 语句。
3. PAGLOCK：指定添加页锁（否则通常可能添加表锁）
4. READCOMMITTED 用与运行在提交读隔离级别的事务相同的锁语义执行扫描。默认情况下，SQL Server 2000 在此隔离级别上操作。
5. READPAST: 跳过已经加锁的数据行，这个选项将使事务读取数据时跳过那些已经被其他事务锁定的数据行，而不是阻塞直到其他事务释放锁，READPAST 仅仅应用于 READ COMMITTED 隔离性级别下事务操作中的 SELECT 语句操作
6. READUNCOMMITTED：等同于 NOLOCK。
7. REPEATABLEREAD：设置事务为可重复读隔离性级别。
8. ROWLOCK：使用行级锁，而不使用粒度更粗的页级锁和表级锁。
9. SERIALIZABLE：用与运行在可串行读隔离级别的事务相同的锁语义执行扫描。等同于 HOLDLOCK。
10. TABLOCK：指定使用表级锁，而不是使用行级或页面级的锁，SQL Server 在该语句执行完后释放这个锁，而如果同时指定了 HOLDLOCK，该锁一直保持到这个事务结束。

11. TABLOCKX：指定在表上使用排它锁，这个锁可以阻止其他事务读或更新这个表的数据，直到这个语句或整个事务结束。

12. UPDLOCK ：指定在读表中数据时设置更新锁（update lock）而不是设置共享锁，该锁一直保持到这个语句或整个事务结束，使用 UPDLOCK 的作用是允许用户先读取数据（而且不阻塞其他用户读数据），并且 保证在后来再更新数据时，这一段时间内这些数据没有被其他用户修改

SELECT * FROM table WITH (HOLDLOCK) 其他事务可以读取表，但不能更新删除

SELECT * FROM table WITH (TABLOCKX) 其他事务不能读取表,更新和删除

十、应用程序锁

应用程序锁就是客户端代码生成的锁，而不是 sql server 本身生成的锁处理应用程序锁的两个系统存储过程

sp_getapplock： 锁定应用程序资源

sp_releaseapplock： 为应用程序资源解锁

）使用尽可能低的隔离性级别。隔离性级别是指为保证数据库数据的完整性和一致性而使多用户事务隔离的程度，SQL92 定义了 4 种隔离性级别：未提交读、提 交读、可重复读和可串行。如果选择过高的隔离性级别，如可串行，虽然系统可以因实现更好隔离性而更大程度上保证数据的完整性和一致性，但各事务间冲突而死 锁的机会大大增加，大大影响了系统性能。

7）使用 Bound Connections。Bound connections 允许两个或多个事务连接共享事务和锁，而且任何一个事务连接要申请锁如同另外一个事务要申请锁一样，因此可以允许这些事务共享数据而不会有加锁的冲突。

8）考虑使用乐观锁定或使事务首先获得一个独占锁定。

八如何对行、 表、数据库加锁

1 如何锁一个表的某一行

Java 代码

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
SELECT * FROM table1 ROWLOCK WHERE A = 'a1'
```

如果能有更多的 隔离级别对应锁的详细对比说明,就更好了

==========

本文转自网络