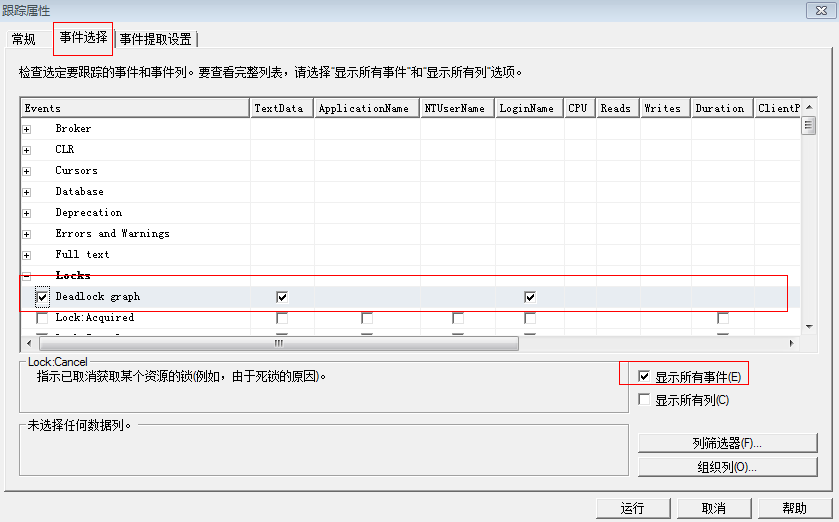
使用SQL Profiler抓取死锁事件

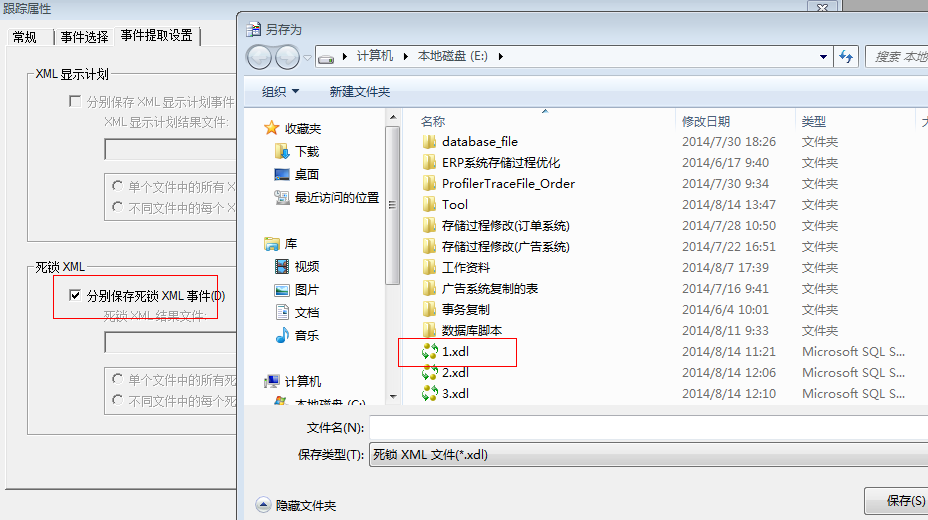
1、打开SQL Profiler



2、选择选项卡：事件选择，勾选右下角的显示所有事件，找到Locks事件，勾选上Deadlock graph。

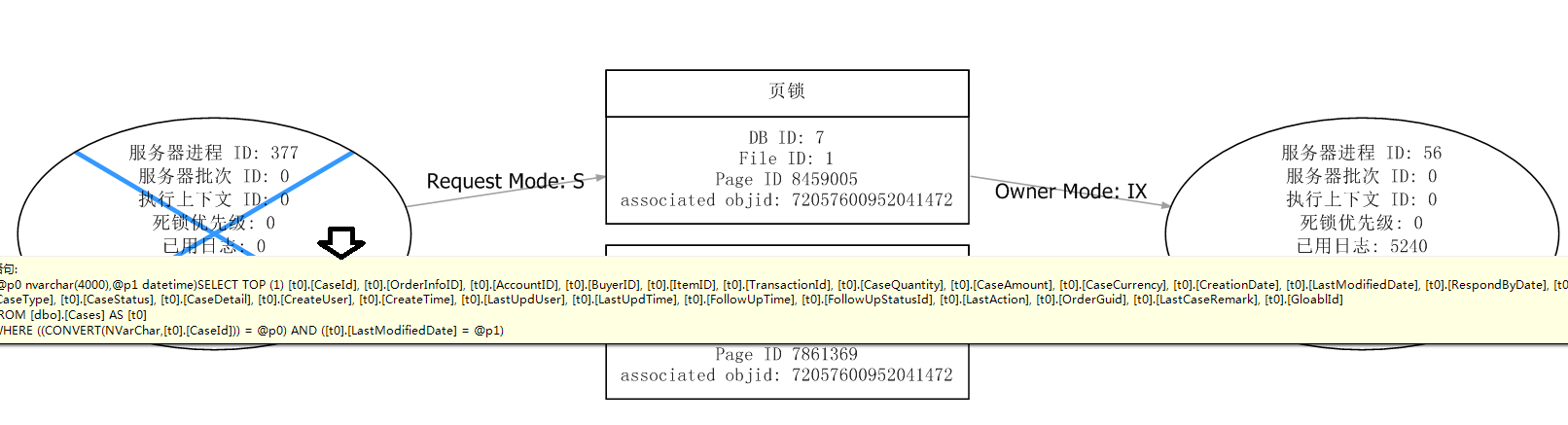


3、再选择第三个选项卡：事件提取设置，勾上死锁XML，弹出文件保存路径，输入文件名即可。

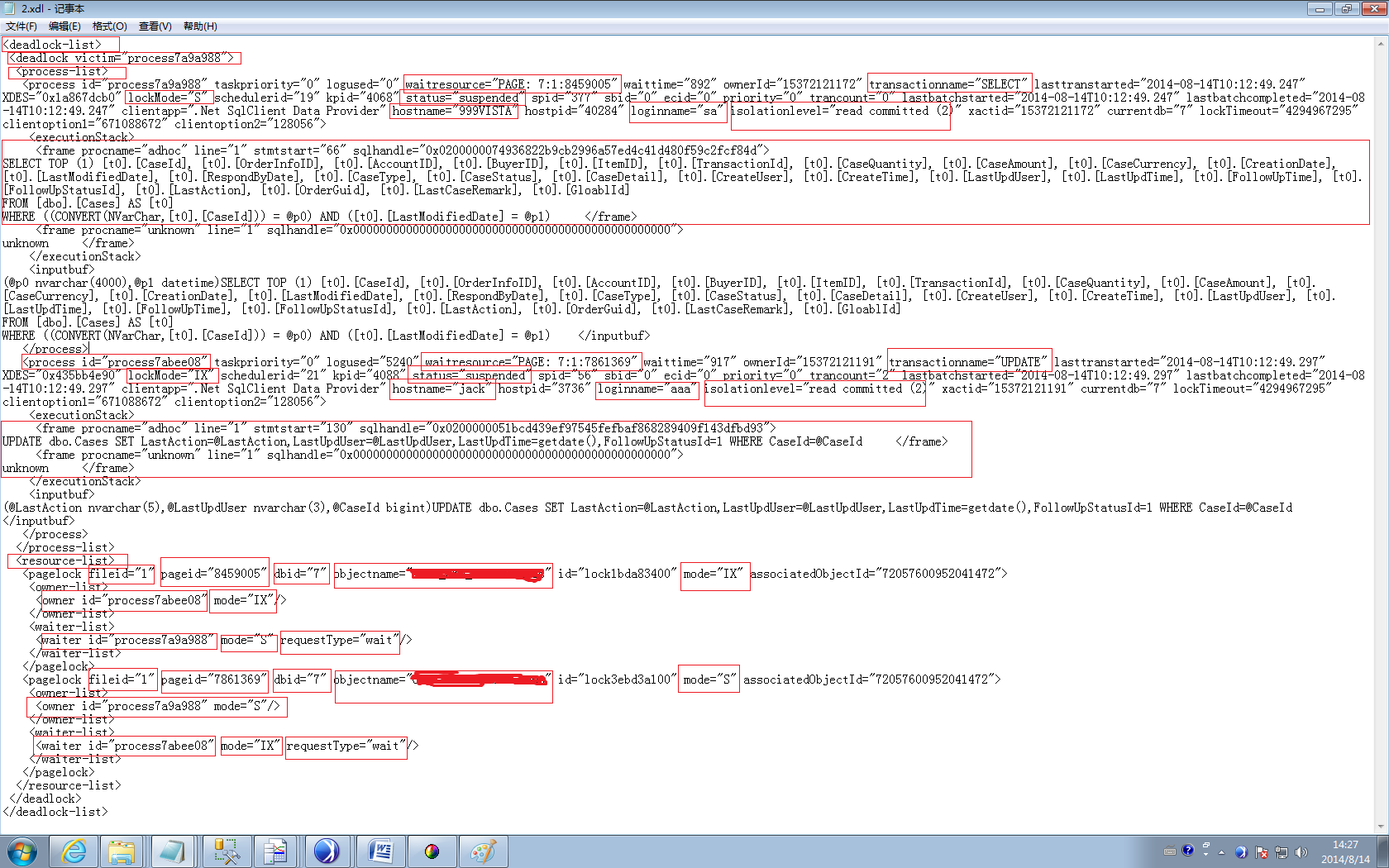


4、以上设置，出现死锁时会保存到死锁XML文件中，经操作发现只保存了最后一个死锁图，如果死锁比较多，需要使用SQL Profiler逐个保存。

5、死锁XML文件默认后缀为.xdl，双击直接打开就会使用SSMS打开，仅仅只能看到很基础的信息。如下：



这种图示并不能详细的说明问题。我们可以采取另外一种办法，左键选择死锁XML文件，右键选择打开方式：文本编辑器。整个死锁的内容就一目了然了，如下：



红色的框框，是我标注出来的，我主要对黑色框框里面的做详细说明，其实有很多了解死锁的朋友看了之后基本已经明白了。

1. 每一个死锁XML文件有一个根节点：deadlock-list。
2. 在根节点下，有一个节点deadlock victim="**process59bcbc8**"，其中victim值为死锁的编号，如果打开的一个SQL Profiler下出现多个死锁，那么它们的编号是一致的，只有不同的SQL Profiler追踪的死锁编号才会有不同。此外该值为牺牲的进程ID号，与别的ID号区分，这个在之后会详细介绍。
3. 在deadlock victim节点下有两个节点，process-list，resource-list。process-list节点主要解释死锁的一些相关信息。resource-list节点解释锁的授予情况。
4. process-list节点，一般有两个process节点，每个节点代表着一个死锁中的一个进程，如果多个进程造成的死锁，那就会出现多个process节点。我们先看第一个process节点，这个节点根据之前的图片可以看出，是被牺牲的进程。

1）属性id="process7a9a988"，这个是被牺牲的进程ID号，在文件中我们可以看到两个process节点的ID号不同。另一个节点的id属性为id="process7abee08"

1. 属性waitresource=”PAGE: 7:1:8459005”，Page代表这是一个页锁，这个页面出现在数据库ID=7,文件ID=1，页面ID=8459005的位置。另一个Procee节点下的waitresource="PAGE: 7:1:7861369"。
2. 属性transactionname ="SELECT"，表示事务类型，SELECT表示这是一个查询事务，可以跟另一个Procee节点下的transactionname属性进行比较，transactionname="UPDATE"
3. 属性lockMode="S"，锁的模式为共享锁。另一个process节点的锁模式lockMode="IX"为意向排他锁。
4. 属性status="suspended"，锁的状态。另一个process节点的status="suspended"。
5. 属性hostname="999VISTA"指操作这个事务的电脑名。另一个process节点的hostname="jack"
6. 属性loginname="sa"指登录数据库的SQL账户。另一个process节点的loginname="aaa"。
7. 属性isolationlevel="read committed (2)"指事务隔离级别。
8. 节点executionStack，在它之下有两个frame节点，第一个节点中间的文本是执行语句。第二个节点，没弄明白什么意思。
9. 节点inputbuf，这个节点中间的文本也是执行语句，与executionStack节点下第一个frame节点的文本相同，推测executionStack节点下第二个节点是不是在一个执行进程中其他的SQL语句？？
10. resource-list节点。这个节点下有两个pagelock节点，分别代表两个进程各自持有的页锁。如果有多个页锁，是不是会有多个pagelock节点？或者是表锁那会是什么节点？？
11. 第一个pagelock节点，fileid="1" pageid="8459005" dbid="7"，这个跟process-list节点下第一个process节点的属性waitresource=”PAGE: 7:1:8459005”意思相同。objectname=” 表名"，这个是表名，具体格式：db\_name.schema\_name.Table\_name。
12. pagelock节点下有多个owner-list节点，这个是每一个进程就有一个节点，该示例中由于死锁是由两个进程造成，只有两个节点。
13. owner-list节点有一个owner，这个代表着锁的授予情况，推测：如果持有多个锁，可能有多个节点。
14. owner节点：id="process7abee08" mode="IX"，id表示进程ID，mode表示锁类型。
15. pagelock节点下有多个waiter-list节点，这个节点是与owner-list节点一一对应。
16. waiter-list节点有一个waiter节点，这个代表着锁的授予情况，推测：如果持有多个锁，可能有多个节点。
17. waiter节点：id="process7a9a988" mode="S" requestType="wait"，id表示进程ID，mode表示锁类型，requestType表示等待授予中。
18. 通过以上的介绍，我们可以了解到：

进程process7a9a988先拥有了Page:7:1:7861369的共享锁S，这个时候，它请求获取Page:7:1:8459005的共享锁S，与此同时，process7abee08先拥有了Page:7:1:8459005的意向排他锁IX，又请求获取Page:7:1:7861369的意向排他锁，这样，两个进程相互阻塞造成了死锁的原因。我对此结合数据库表结构分析，发现该表的每行字段都有6880个字符，也就是说，一行就占用了一个页面，同时，进程process7a9a988使用的是LINQ查询，由于LINQ导致这个查询居然不走聚集索引，先走AccountID字段的索引然后在嵌套循环主键索引CaseID，是死锁的原因之一。

以上就是本人对死锁的见解，如有异议，欢迎提出讨论。