扯谈下XA事务

标签： Java事务XA事务XA
2014-02-20 21:34 11125人阅读 评论(1) 收藏 举报

分类：

Java（64）

分布式

目录(?)[+]

普通事务

普通事务的实现是比较好理解的。以jdbm3为例，大概是这样的过程：

每个事务都新建一个事务文件，当commit时，先把修改过的数据块，写到事务文件里，然后再一次性地写到**数据库**文件里。

如果commit时挂掉了，那么重启之后，会再次从事务文件里把修改过的块写到数据库文件里。最后再删除事务文件。

https://github.com/jankotek/JDBM3

但是XA事务，即所谓的分布式事务却令人感到云里雾里。一是资料很少，网上的各种配置资料都是流于表面；二是可能实际应用的人也少。

最近研究了下，算是找到点门道了。

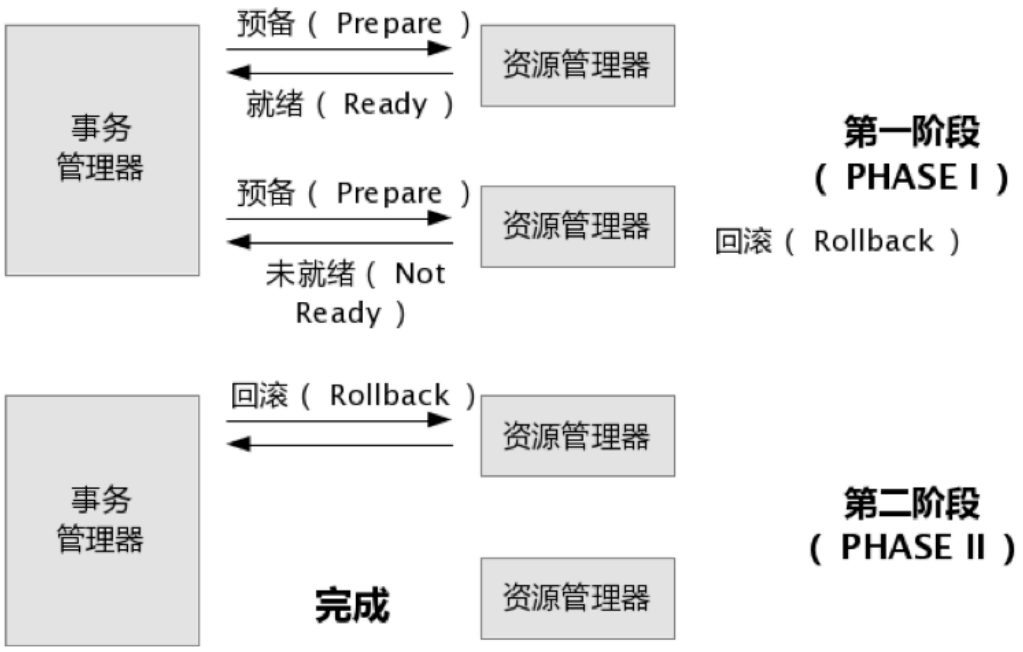
二阶段提交（Two-phase Commit）

首先，XA事务是基于二阶段提交（Two-phase Commit）实现的。二阶段提交本身并没有什么令人疑惑的地方。看wiki就可以知道是怎么回事了。

简而言之，有二种角色，事务管理者（DM, Transaction Manager），资源管理器（RM, Resource Manager），通常即数据库或者JMS服务器。

下面两个图片来自：http://www.infoq.com/cn/articles/xa-transactions-handle

出错回滚：



当然，还有各种中间出错时，要处理的情况，详细可以看infoq的原文。

令人疑惑的atomikos

二阶段提交协议是很容易理解的，但是真正令我疑惑的是**Java**实现的atomikos，一个分布式事务的Transaction Manager组件。

开始的时候，我以为事务管理器(TM)都是独立的一个服务，或者一个独立的进程，它和资源管理器(RM)之间通过网络通迅。

但是在网上看一些atomikos配置文章，都没有提到如何配置一个独立的Transaction Manager，只是简单地介绍了下如何配置atomikos，这些配置都是和应用在一起的。

而从配置里面也没法看出是如何保证在事务过程中，如果应用的进程挂掉后，是如何恢复的。

再把atomikos的例子代码下载下来，发现也没有提到是如何保证事务在失败后，如何协调的。

比如，在第二段提交时，当RM1 commit完成了，而RM2 commit还没有完成，而这时TM，即配置了atomikos的应用程序崩溃，那么这个事务并没有完成，还需要TM重启后协调，才能最终完成这个事务。但是没看到恢复部分的配置。

没办法，只能亲自跑一遍代码了。

跑了下atomikos的代码，在第二阶段提交时，把进程杀掉，发现的确是可以自动处理回滚事务，或者再次提交的。那么信息是保存在哪里的？也没有看到有什么配置文件。

最终，只能下XA的规范下载下来，再一点点慢慢看。

在The XA Specification里的2.3小节：Transaction Completion and Recovery 明确提到TM是要记录日志的：

> In Phase 2, the TM issues all RMs an actual request to commit or roll back the
> transaction branch, as the case may be. (**Before issuing requests to commit, the TM**
> **stably records the fact that it decided to commit, as well as a list of all involved RMs.**)
> All RMs commit or roll back changes to shared resources and then return status to the
> TM. The TM can then discard its knowledge of the global transaction.

TM是一定要把事务的信息，比如XID，哪个RM已经完成了等保存起来的。只有当全部的RM提交或者回滚完后，才能丢弃这些事务的信息。

于是再查看下atomikos例子运行目录，果然有一些文件日志文件：

127.0.1.1.tm13.epoch

tmlog13.log

tmlog.lck

tm.out

tm.out.lck

原来atomikos是通过在应用的目录下生成日志文件来保证，如果失败，在重启后可以通过日志来完成未完成的事务。

XA事务的假设条件

从XA的规范里找到了下面的说法：

The X/Open DTP model makes these assumptions:

TMs and RMs have access to stable storage　TM和RM都有牢靠的存储

TMs coordinate and control recovery　TM协调和控制恢复流程

RMs provide for their own restart and recovery of their own state. On request, an RM must give a TM a list of XIDs that the RM has prepared for commitment or has heuristically completed.　RM在得启和恢复时，得回应TM的请求，返回一系列的XID，是prepared的，或者是已经启发式地完成了的

也就是说，XA事务都假定了TM和RM都是有牢靠的存储的，所以也保证了TM重启后可以从日志里恢复还没处理完的事务。

TM可以向RM查询事务的状态，RM必须要返回一系列事务的XID，表明事务是prepared状态，还是已经commit的状态。

到这里，应该很明了了，XA事务是其限制的，而TM是XA事务的一个单点，TM必须要非常地牢靠。

从XA的接口函数，就可以大概看出协议是怎么工作的（来自XA规范文档）：

Name	Description	See
ax_reg	Register an RM with a TM.	Section 3.3.1 on page 16
ax_unreg	Unregister an RM with a TM.	Section 3.3.1 on page 16
xa_close	Terminate the AP's use of an RM.	Section 3.2 on page 13
xa_commit	Tell the RM to commit a transaction branch.	Section 3.4 on page 17
xa_complete	Test an asynchronous *xa_* operation for completion.	Section 3.5 on page 18
xa_end	Dissociate the thread from a transaction branch.	Section 3.3 on page 14
xa_forget	Permit the RM to discard its knowledge of a heuristically-completed transaction branch.	Section 3.4 on page 17
xa_open	Initialise an RM for use by an AP.	Section 3.2 on page 13
xa_prepare	Ask the RM to prepare to commit a transaction branch.	Section 3.4 on page 17
xa_recover	Get a list of XIDs the RM has prepared or heuristically completed.	Section 3.6 on page 18
xa_rollback	Tell the RM to roll back a transaction branch.	Section 3.4 on page 17
xa_start	Start or resume a transaction branch - associate an XID with future work that the thread requests of the RM.	Section 3.3 on page 14

如何避免XA事务

XA事务的明显问题是timeout问题，比如当一个RM出问题了，那么整个事务只能处于等待状态。这样可以会连锁反应，导致整个系统都很慢，最终不可用。

避免使用XA事务的方法通常是最终一致性。

举个例子，比如用户充值300元，为了减少DB的压力，先把这个放到消息队列里，然后后端再从消息队列里取出消息，更新DB。

那么如何保证，这条消息不会被重复消费？或者重复消费后，仍能保证结果是正确的?

1. 在消息里带上用户帐号在数据库里的版本，在更新时比较数据的版本，如果相同则加上300；
2. 比如用户本来有500元，那么消息是更新用户的钱数为800，而不是加上300；
3. 另外建一个消息是否被消费的表，记录消息ID，在事务里，先判断消息是否已经消息过，如果没有，则更新数据库，加上300,否则说明已经消费过了，丢弃。

前面两种方法都必须从流程上保证是单方向的，不能插入其它的东东。

其它的一些东东：

貌似一直有人想用zookeeper来实现2pc，或者类似的东东，因为zookeeper是比

较可靠的。但是感觉也没有办法解决timeout问题。

微软的XA事务恢复流程的文档：
http://msdn.microsoft.com/en-us/library/windows/desktop/ms681775(v=vs.85).aspx
There are two forms of XA transaction recovery, as follows:

- Cold recovery. Cold recovery performed if the transaction manager process fails while a connection to an XA resource manager is open. When the transaction manager restarts, it reads the transaction manager log file and re-establishes the connection to the XA resource manager by calling xa_open_entry. It then initiates XA recover by calling xa_recover_entry.
- Hot recovery. Hot recovery is performed if the transaction manager remains up while the connection between the transaction manager and the XA resource manager fails because the XA resource manager or the network fails. After the failure, the transaction manager periodically calls xa_open_entry to reconnect to the XA resource manager. When the connection is reestablished, the transaction manager initiates XA recovery by calling xa_recover_entry.

总结：

XA事务没有什么神秘的地方，二阶段提交也是一个人们很自然的一个处理方式。只不过，这个是规范，如果有多个资源之间要协调，而且都支持XA事务，那么会比较方便 。

参考：

The XA Specification 可以从这里下载到：
http://download.csdn.net/detail/hengyunabc/6940529
http://en.wikipedia.org/wiki/Two-phase_commit_protocol
http://www.infoq.com/cn/articles/xa-transactions-handle
http://java.sun.com/javaee/technologies/jta/index.jsp