关于分布式事务、两阶段提交、一阶段提交、Best Efforts 1PC模式和事务补偿机制的研究

标签： managersharding数据库通讯java网络
2012-05-29 15:05 67811人阅读 评论(12) 收藏 举报

1.XA

XA是由X/Open组织提出的分布式事务的规范。XA规范主要定义了(全局)事务管理器(Transaction Manager)和(局部)资源管理器(Resource Manager)之间的接口。XA接口是双向的系统接口，在事务管理器（Transaction Manager）以及一个或多个资源管理器（Resource Manager）之间形成通信桥梁。XA之所以需要引入事务管理器是因为，在分布式系统中，从理论上讲（参考Fischer等的论文），两台机器理论上无法达到一致的状态，需要引入一个单点进行协调。事务管理器控制着全局事务，管理事务生命周期，并协调资源。资源管理器负责控制和管理实际资源（如数据库或JMS队列）。下图说明了事务管理器、资源管理器，与应用程序之间的关系：

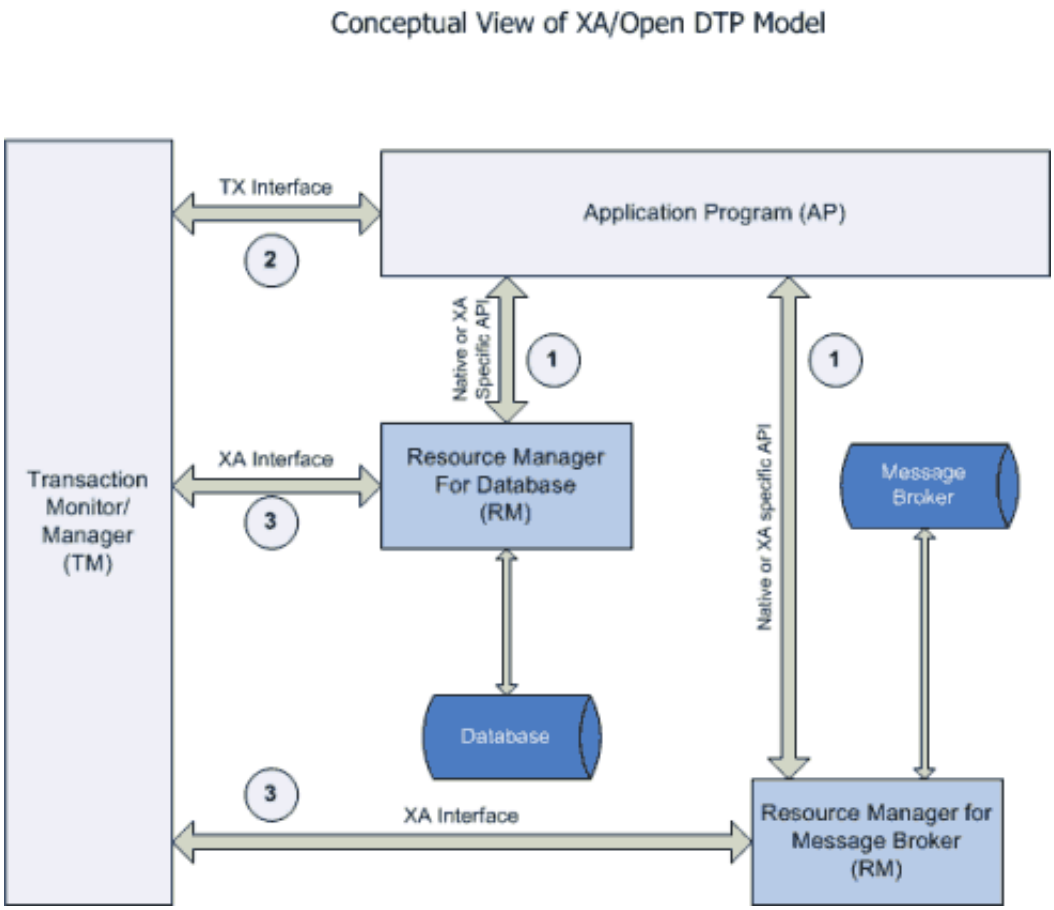

图1.XA规范下的分布式事务各类参与者之间的关系

2.JTA

作为**Java**平台上事务规范JTA（Java Transaction API）也定义了对XA事务的支持，实际上，JTA是基于XA**架构**上建模的，在JTA 中，事务管理器抽象为javax.transaction.TransactionManager接口，并通过底层事务服务（即JTS）实现。像很多其他的java规范一样，JTA仅仅定义了接口，具体的实现则是由供应商(如J2EE厂商)负责提供，目前JTA的实现主要由以下几种：

1.J2EE容器所提供的JTA实现(JBoss)

2.独立的JTA实现:如JOTM，Atomikos.这些实现可以应用在那些不使用J2EE应用服务器的环境里用以提供分布事事务保证。如Tomcat,Jetty以及普通的java应用。

3.两阶段提交

所有关于分布式事务的介绍中都必然会讲到两阶段提交，因为它是实现XA分布式事务的关键(确切地说：两阶段提交主要保证了分布式事务的原子性：即所有结点要么全做要么全不做)。所谓的两个阶段是指：第一阶段：准备阶段和第二阶段：提交阶段。

图2.两阶段提交示意图（摘自info发布的《java事务设计策略》一文）

1.准备阶段：事务协调者(事务管理器)给每个参与者(资源管理器)发送Prepare消息，每个参与者要么直接返回失败(如权限验证失败)，要么在本地执行事务，写本地的redo和undo日志，但不提交，到达一种“万事俱备，只欠东风”的状态。(关于每一个参与者在准备阶段具体做了什么目前我还没有参考到确切的资料，但是有一点非常确定：参与者在准备阶段完成了几乎所有正式提交的动作，有的材料上说是进行了“试探性的提交”，只保留了最后一步耗时非常短暂的正式提交操作给第二阶段执行。)

2.提交阶段：如果协调者收到了参与者的失败消息或者超时，直接给每个参与者发送回滚(Rollback)消息；否则，发送提交(Commit)消息；参与者根据协调者的指令执行提交或者回滚操作，释放所有事务处理过程中使用的锁资源。(注意:必须在最后阶段释放锁资源)

将提交分成两阶段进行的目的很明确，就是尽可能晚地提交事务，让事务在提交前尽可能地完成所有能完成的工作，这样，最后的提交阶段将是一个耗时极短的微小操作，这种操作在一个分布式系统中失败的概率是非常小的，也就是所谓的“网络通讯危险期”非常的短暂，这是两阶段提交确保分布式事务原子性的关键所在。（唯一理论上两阶段提交出现问题的情况是当协调者发出提交指令后当机并出现磁盘故障等永久性错误，导致事务不可追踪和恢复）

从两阶段提交的工作方式来看，很显然，在提交事务的过程中需要在多个

节点之间进行协调，而各节点对锁资源的释放必须等到事务最终提交时，这样，比起一阶段提交，两阶段提交在执行同样的事务时会消耗更多时间。事务执行时间的延长意味着锁资源发生冲突的概率增加，当事务的并发量达到一定数量的时候，就会出现大量事务积压甚至出现死锁，系统性能就会严重下滑。这就是使用XA事务

4.一阶段提交(Best Efforts 1PC模式)

不像两阶段提交那样复杂，一阶段提交非常直白，就是从应用程序向数据库发出提交请求到数据库完成提交或回滚之后将结果返回给应用程序的过程。一阶段提交不需要“协调者”角色，各结点之间不存在协调操作，因此其事务执行时间比两阶段提交要短，但是提交的“危险期”是每一个事务的实际提交时间，相比于两阶段提交，一阶段提交出现在“不一致”的概率就变大了。但是我们必须注意到：只有当基础设施出现问题的时候(如网络中断，当机等)，一阶段提交才可能会出现“不一致”的情况，相比它的性能优势，很多团队都会选择这一方案。关于在**spring**环境下如何实现一阶段提交,有一篇非常优秀的文章值得参考：http://www.javaworld.com/javaworld/jw-01-2009/jw-01-spring-transactions.html?page=5

5.事务补偿机制

像best efforts 1PC这种模式，前提是应用程序能获取所有的数据源，然后

使用同一个事务管理器(这里指是的spring的事务管理器)管理事务。这种模式最典型的应用场景非数据库sharding莫属。但是对于那些基于web service/rpc/jms等构建的高度自治(autonomy)的分布式系统接口，best efforts 1PC模式是无能为力的，此类场景下，还有最后一种方法可以帮助我们实现“最终一致性”，那就是事务补偿机制。关于事务补偿机制是一个大话题，本文只简单提及，以后会作专门的研究和介绍。

6.在基于两阶段提交的标准分布式事务和Best Efforts 1PC两者之间如何选择

一般而言，需要交互的子系统数量较少，并且整个系统在未来不会或很少引入新的子系统且负载长期保持稳定，即无伸缩要求的话，考虑到开发复杂度和工作量，可以选择使用分布式事务。对于时间需求不是很紧，对性能要求很高的系统，应考虑使用Best Efforts 1PC或事务补偿机制。对于那些需要进行sharding改造的系统，基本上不应再考虑分布式事务，因为sharding打开了数据库水平伸缩的窗口，使用分布式事务看起来好像是为新打开的窗口又加上了一把枷锁。

补充：关于网络通讯的危险期

由于网络通讯故障随时可能发生，任何发出请求后等待回应的程序都会有失去联系的危险。这种危险发生在发出请求之后，服务器返回应答之前，如果在这个期间网 络通讯发生故障，发出请求一方无法收到回应，于是无

法判断服务器是否已经成功地处理请求，因为收不到回应可能是请求没有成功地发送到服务器，也可能是服务 器处理完成后的回应无法传回请求方。这段时间称为网络通讯的危险期(In-doubt Time)。很显然，网络通讯的危险期是分布式系统除单点可靠性之外需要考虑的另一个可靠性问题。