

# 分布式事务系列（2.1）分布式事务的概念

收藏

乒乓狂魔

• 发表于 2年前

• 阅读 2535

• 收藏 69

• 点赞 3

• 评论 7

独家直播！大数据应用场景全解析>>>

HOT

## 1 系列目录

- 分布式事务系列（开篇）提出疑问和研究过程
- 分布式事务系列（1.1）Spring事务管理器  
PlatformTransactionManager源码分析
- 分布式事务系列（1.2）Spring事务体系
- 分布式事务系列（2.1）分布式事务模型与接口定义
- 分布式事务系列（3.1）jotm的分布式案例
- 分布式事务系列（3.2）jotm分布式事务源码分析
- 分布式事务系列（4.1）Atomikos的分布式案例

## 2 X/Open DTP

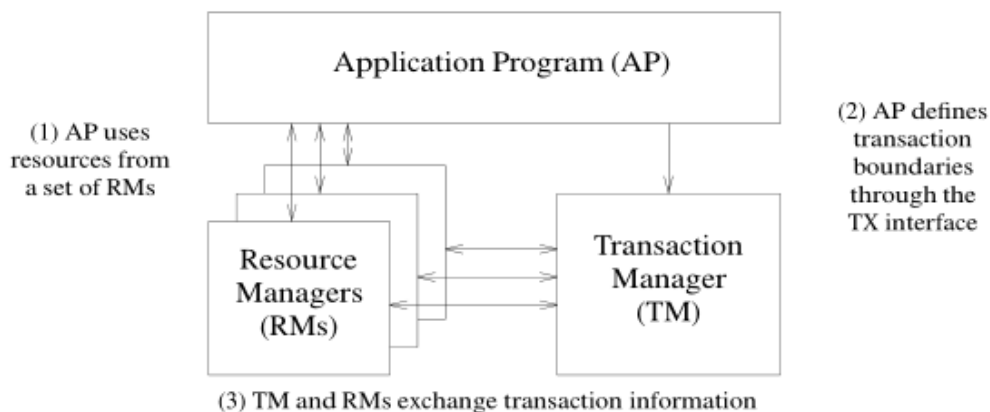
DTP全称是Distributed Transaction Process，即分布式事务模型。之前我们接触的事务都是针对单个数据库的操作，如果涉及多个数据库的操作，还想保证原子性，这就需要使用分布式事务了。而X/Open DTP就是一种分布式事务处理模型。

### 2.1 X/Open DTP模型

X/Open是一个组织，维基百科上这样说明：

X/Open是1984年由多个公司联合创建的一个用于定义和推进信息技术领域开放标准的公司

上述组织就针对分布式事务提出了一个模型，即DTP模型(Distributed Transaction Process),该模型如下所示：



上面主要涉及了三个对象：

- AP(Application Program)：应用程序
- TM(Transaction Manager)：事务管理器

负责协调和管理事务，它和之前说的Spring的事务管理器PlatformTransactionManager可不一样。

- RM(Resource Manager)：资源管理器

可以理解为数据库或者JMS。

他们三者的关系：

- AP通过TM来操作多个RM，AP也可以通过RM的本地事务接口来

操作单个RM

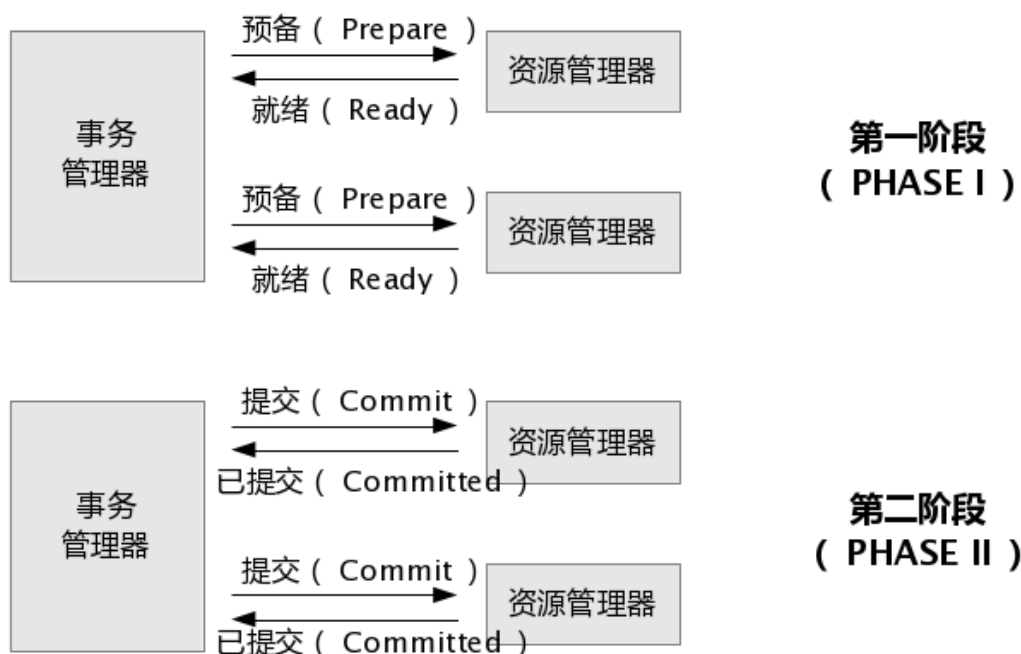
- TM和RM可以互相通信，他们之前的通信协议就是XA协议

更多的规范内容，可以参见[Distributed Transaction Processing:The XA Specification](#)

## 2.2 两阶段提交

全称是The two-phase commit protocol，简称2PC

当需要协调多个资源管理器进行分布式事务的时候，就需要使用到两阶段提交协议，如下图所示：



第一个阶段：事务管理器通过XA协议，向资源管理器发送prepare命令，询问他们预提交是否成功，每个资源管理器给出自己的响应，预提交成功或者失败

第二个阶段：根据第一个阶段的资源管理器的回复情况，如果全部表示预提交成功，则事务管理器向所有的资源管理器发送最终的提交命令。如果有一个资源管理器回复预提交失败，则事务管理器向所有的资源管理器发送回滚命令，全部进行回滚

通过这样的两阶段提交协议，即可完成了分布式事务的功能。这样的协议的正是基于了事务管理器和资源管理器的双向通信能力，而这在之前

的本地事务中，如jdbc事务中，我们只能通过Connection向数据库传递命令，不能从数据库获取事务的一些执行信息，这种情况是单向的。

然而还是存在一些问题的：

- 两阶段提交协议是阻塞式协议，所以事务管理器必须等待每一个资源管理器发送回复之后，才能进行下一步操作。一旦资源管理器挂掉，事务管理器则收不到响应，就会造成事务管理器一直等待。这时候就必须引入超时机制，一旦超过某个时间还没有回复，则认为失败，回滚所有操作，这些都是非常耗性能的。
- 一旦事务管理器挂掉，则资源管理器则一直等待事务管理器的命令。这时候可能就需要使用备份的事务管理器来接替原来的事务管理器的工作

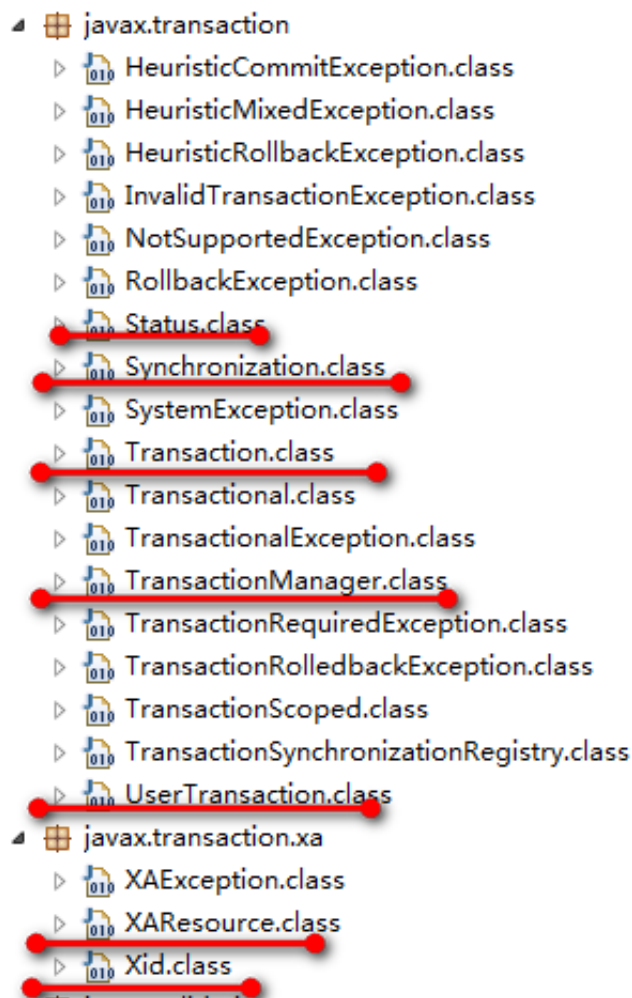
更多资料与问题，见如下：

- [InfoQ上的Java事务设计策略](#)
- [两阶段提交协议](#)

### 3 JTA接口定义

上述接口规范不是针对某种语言的，java是如何来落实上述规范的呢？

这就是JTA的内容了。先来预览下JTA的包结构：



主要2个大包的内容：

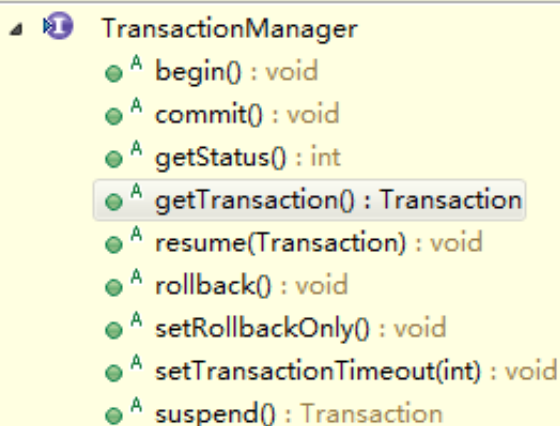
- javax.transaction
- javax.transaction.xa

### 3.1 AP、TM、RM三大对象

- AP：我们的应用程序
- TM：即javax.transaction.TransactionManager 事务管理器
- RM：即javax.transaction.xa.XAResource 我称之为与资源管理器的一个通信代表，我们通过XAResource接口方法和资源管理器进行通信

下面来详细说明：

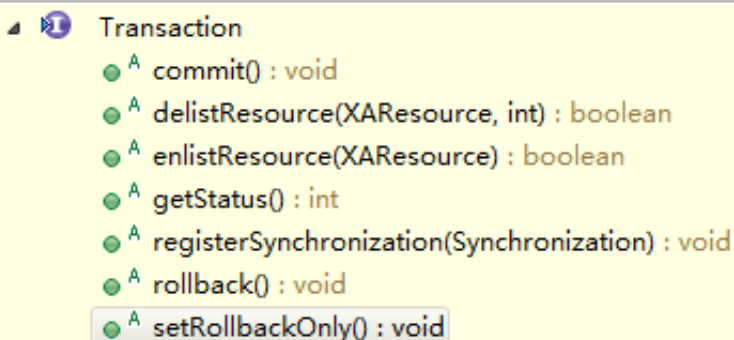
TM：即javax.transaction.TransactionManager，总体接口定义如下：



```
TransactionManager
  begin() : void
  commit() : void
  getStatus() : int
  getTransaction() : Transaction
  resume(Transaction) : void
  rollback() : void
  setRollbackOnly() : void
  setTransactionTimeout(int) : void
  suspend() : Transaction
```

- begin():创建一个新的事务并关联到当前线程
- Transaction getTransaction(): 获取与当前线程关联的事务
- commit():提交与当前线程关联的事务
- rollback(): 回滚与当前线程关联的事务
- 等等

定义了一些常用的事务操作，这里操作的事务的定义为 `javax.transaction.Transaction`，接口如下所示：

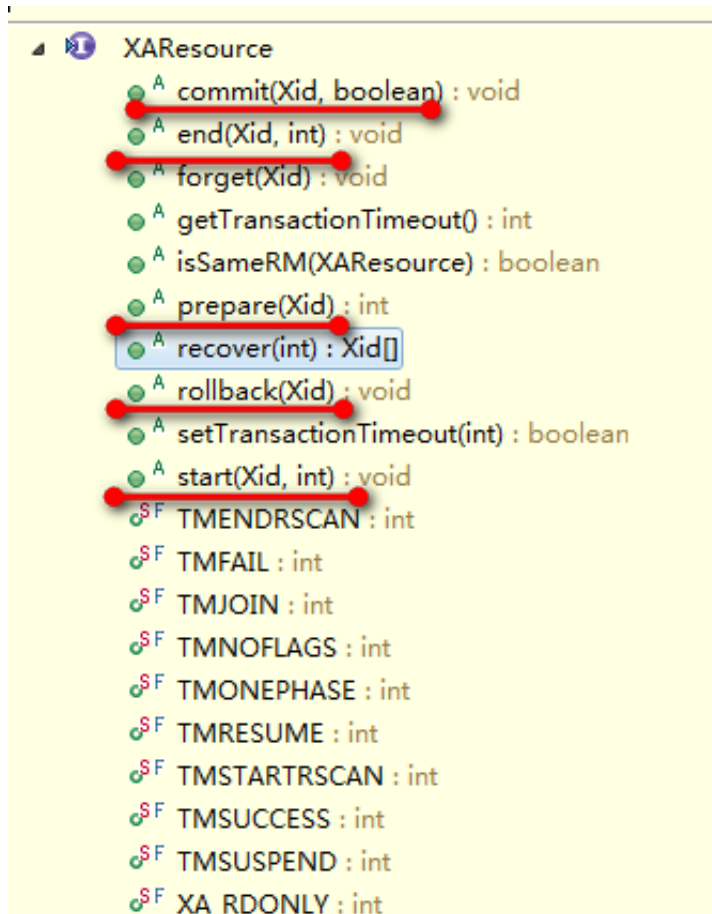


```
Transaction
  commit() : void
  delistResource(XAResource, int) : boolean
  enlistResource(XAResource) : boolean
  getStatus() : int
  registerSynchronization(Synchronization) : void
  rollback() : void
  setRollbackOnly() : void
```

- commit(): 提交事务
- rollback(): 回滚事务
- enlistResource(XAResource xaRes): 把给定的 XAResource（资源管理器的一个通信代表）加入当前事务中来，一个分布式事务会涉及与多个资源管理器交互，该操作就是把某个资源管理器的通信代表纳入当前事务中来

- delistResource(XAResource xaRes, int flag): 把给定的 XAResource (资源管理器的一个通信代表) 从当前事务中去除

RM:资源管理器, 与它通信的代表就是 javax.transaction.xa.XAResource, 资源管理器的一个通信代表, 我们通过XAResource与资源管理器来通信。所以XAResource的实现类实现了两阶段提交协议, 接口内容如下:



我们看到有一个Xid接口: X/Open组织规定了, 对于每个分布式事务, 都有一个对应的唯一标示。这个唯一标示在java中就是Xid接口, 该接口文档如下:

The Xid interface is a Java mapping of the X/Open transaction identifier XID structure The Xid interface is used by the transaction manager and the resource managers This interface is not visible to the application programs

接下来看下XAResource的其他方法

- `start(Xid xid, int flags)`和`end(Xid xid, int flags)`用于告知资源管理器的事务的边界，即在这两个方法之间的sql等操作才会纳入xid对应的分布式事务中
- `prepare(Xid xid)`：则就是上述图片中的第一个阶段，针对xid对应的分布式事务，向资源管理器发送预提交命令
- `rollback(Xid xid)`、`commit(Xid xid, boolean onePhase)` 如果所有资源管理器都回复OK,则向所有资源管理器发送commit提交命令，否则发送rollback回滚命令

下面看一个简单模拟两阶段提交的例子，加深对上面的方法的理解：

```
xaRes1.start(xid, XAResource.TMNOFLAGS);
stmt1.executeUpdate("insert into test_table1 values (100)");
xaRes1.end(xid, XAResource.TMSUCCESS);

stmt1.executeUpdate("insert into test_table1 values (99)");

xaRes2.start(xid, XAResource.TMNOFLAGS);
stmt2.executeUpdate("insert into test_table2 values (100)");
xaRes2.end(xid, XAResource.TMSUCCESS);

ret1 = xaRes1.prepare(xid);
ret2 = xaRes2.prepare(xid);
if (ret1 == XAResource.XA_OK && ret2 == XAResource.XA_OK) {
    xaRes1.commit(xid, false);
    xaRes2.commit(xid, false);
}else{
    xaRes1.rollback(xid);
    xaRes2.rollback(xid);
}
```

过程说明：

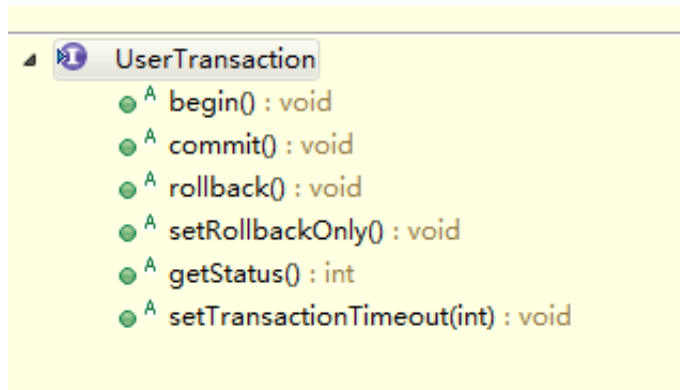
- 首先使用`start`、`end`来告知资源管理器分布式事务的边界，上述`stmt1.executeUpdate("insert into test_table1 values (99)");`没有`start`、`end`范围之内，则不会划入分布式事务的管辖中，而是作为了一般的本地事务。
- 然后遍历每个资源管理器的通信代表，使他们询问资源管理器预提交的情况
- 如果全部OK则全部提交，否则全部回滚



更多的例子，可以参考这里[JAVA分布式事务原理及应用](#)

## 3.2 UserTransaction接口

UserTransaction接口是给开发人员使用的事务接口，屏蔽了底层的实现细节，通过该接口就可以操作一个分布式事务。该接口的实现通常是委托给TM即事务管理器来完成。接口内容如下：



可以看到，UserTransaction的这些方法在TM即事务管理器中都有对应的方法。

## 3.3 JTA中涉及的角色

涉及到哪些角色呢？他们的职责与任务分别什么呢？

- 开发人员：只需通过使用UserTransaction接口来进行操作分布式事务
- TransactionManager的实现提供者：实现UserTransaction、TransactionManager、Transaction接口，通过与XAResource接口交互来实现分布式事务，TransactionManager的实现提供者如jotm、Atomikos，他们的上述接口分别如下
  - 1.1 jotm的UserTransaction实现是org.objectweb.jotm.Current
  - 1.2 jotm的TransactionManager实现仍然是org.objectweb.jotm.Current
  - 1.3 jotm的Transaction实现是org.objectweb.jotm.TransactionImpl

- 2.1 Atomikos的UserTransaction实现是  
`com.atomikos.icatch.jta.UserTransactionImp`
  - 2.2 Atomikos的TransactionManager实现是  
`com.atomikos.icatch.jta.UserTransactionManager`
  - 2.3 Atomikos的Transaction实现是  
`com.atomikos.icatch.jta.TransactionImp`
  - XAResource接口的实现者：需要由资源管理器者来实现，若资源管理器是数据库，则数据库需要提供XAResource接口的实现。如对于mysql来说，对应的实现为  
`com.mysql.jdbc.jdbc2.optional.MysqlXAConnection`。这种形式下的数据库驱动就是支持分布式事务的数据库驱动
- 更多的支持分布式的数据库驱动可以看下这里[XA Drivers](#)

#### XA Drivers

- [Configuring FirstSQL for XA](#)
- [Configuring Oracle](#)
- [Configuring Sybase ASE for XA](#)
- [Configuring Informix IDS for XA](#)
- [Configuring Microsoft SQL Server for XA](#)
- [Configuring MySQL for XA](#)
- [Configuring DB2](#)
- [Configuring PostgreSQL for XA](#)
- [Configuring Apache Derby for XA](#)

## 3.4 JTS

JTS是什么呢？它和JTA是什么关系呢？

这个我也是没弄清楚，可以参考这里的说法，如果觉得不对，自行去深入研究下,原文在这里[JTA、JTS各司其职，共职分布式事务。](#)

JTS也定义了一套规范，它约定了各个程序角色之间如何传递事务上下文，它源自CORBA 的OTS规范，基于IIOP（一种软件交互协议）。不要认为JTS是JTA的实现，JTA其实就定义了一个空架子，告诉JTA的实现者应该怎样做怎样做，但是具体到做的时候JTS就来插一手了。因为JTA约

定的这些角色要进行事务上下文的交互啊，JTS约定了应该怎样去进行交互

### 3.5 疑问解答

- 1 是不是数据库驱动必须要支持XA协议，即支持分布式事务？

不是，可以使用第三方框架如jotm、Atomikos来模拟XA协议

- 2 使用分布式事务，是不是一定要应用服务器的支持？

不是。

jboss是支持的，所以我们可以直接使用jboss自己实现的

UserTransaction来实现分布式事务（使用JNDI方式，这时候要求数据库驱动支持XA协议），可以不需要像jotm、Atomikos等第三方框架tomcat就是不支持的，但是我们可以使用第三方框架来实现，如jotm、Atomikos

### 4 结束语

这一篇这要了解了分布式事务的一些概念与JTA的接口定义，下一步就开始深入源码分析jotm、Atomikos是如何来实现分布式事务的，先从jotm说起

- jotm实现分布式事务的例子
- jotm的实现原理