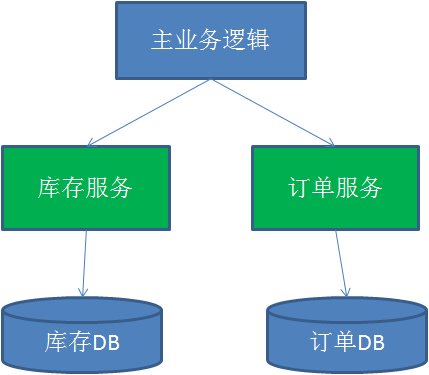
# 什么是分布式事务

分布式事务是指事务的参与者、支持事务的服务器、资源服务器以及事务管理器分别位于不同的分布式系统上的不同节点上。如下图



数据库怎么保持数据一致：三组文件(控制文件，日志文件，数据文件)

# 二、XA的2段提交方案

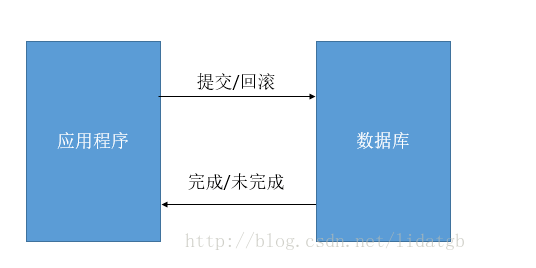
### 1. 什么是XA协议

XA协议由Oracle tuxdeo首先提出的，并交给X/OPEN组织，作为资源管理器(数据库)与实物管理器的接口标准。目前主流的各大数据库都提供对XA的支持，XA协议采用两段提交方式来管理分布式事务，XA接口提供资源管理器与事务管理之间进行通信的标准接口

XA就是X/open DTP定义的交易中间件与数据库之间的接口规范(即接口函数)，交易中间件用它通知数据库事务的开始、结束以及提交、回滚等，XA接口函数由数据库厂商提供

X/OPEN 组织（即现在的Open Group）定义了分布式事务处理模型。X/OpenDTP模型（1994）包括应用程序(AP)、事务管理器(TM)、资源管理器(RM)、通信资源管理器(CRM)四部分。一般，常见的事物管理器（TM）是交易中间件，常见的资源管理器(RM)是数据库，常见的通信资源管理器(CRM)是消息中间件

### 2． XA协议的一阶段提交

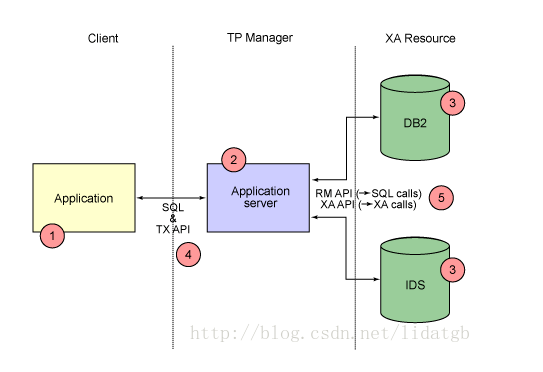


如果在程序中开启了事务，那么在应用程序发出提交/回滚请求后，数据库执行操作,而后将成功/失败返回给应用程序，程序继续执行

一阶段提交协议相对简单。优点也很直观。它不再与其它的对象交互，节省了判断步骤和时间。所以在性能上是在阶段提交协议种最好的。但缺点也很明显，数据库确认执行事务的时间较长，出问题的可能性就随之增大。如果有多个数据源，一阶段提交协议无法协调他们之间的关系

### 3.XA协议的二段提交、

在一阶段协议的基础上，有了二阶段协议，二阶段协议的好处是添加了一个管理者角色



很明显，二阶段协议通过将两层变为三层，增加了中间的管理者角色，从而协调多个数据源之间的关系，二阶段提交协议分为两个阶段



应用程序调用了事务管理器的提交方法，此后第一阶段分为两个步骤：

事务管理器通知参与该事务的各个资源管理器，通知他们开始准备事务。

资源管理器接收到消息后开始准备阶段，写好事务日志并执行事务，但不提交，然后将是否就绪的消息返回给事务管理器（此时已经将事务的大部分事情做完，以后的内容耗时极小）。



第二阶段也分为两个步骤：

事务管理器在接受各个消息后，开始分析，如果有任意其一失败，则发送回滚命令，否则发送提交命令。

各个资源管理器接收到命令后，执行（耗时很少），并将提交消息返回给事务管理器。

    事务管理器接受消息后，事务结束，应用程序继续执行。

    为什么要分两步执行？一是因为分两步，就有了事务管理器统一管理的机会；二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耗时极短，耗时极短意味着操作失败的可能性也就降低。

    同时，二阶段提交协议为了保证事务的一致性，不管是事务管理器还是各个资源管理器，每执行一步操作，都会记录日志，为出现故障后的恢复准备依据。

    二阶段提交协议的存在的弊端是阻塞，因为事务管理器要收集各个资源管理器的响应消息，如果其中一个或多个一直不返回消息，则事务管理器一直等待，应用程序也被阻塞，甚至可能永久阻塞。

# 三、TCC解决方案

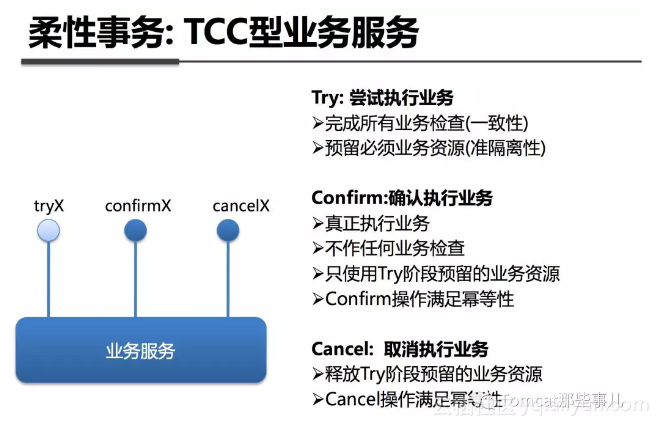
### 1.TCC介绍

TCC是由支付宝架构师提供的一种柔性解决分布式事务解决方案，主要包括三个步骤

Try：预留业务资源/数据效验

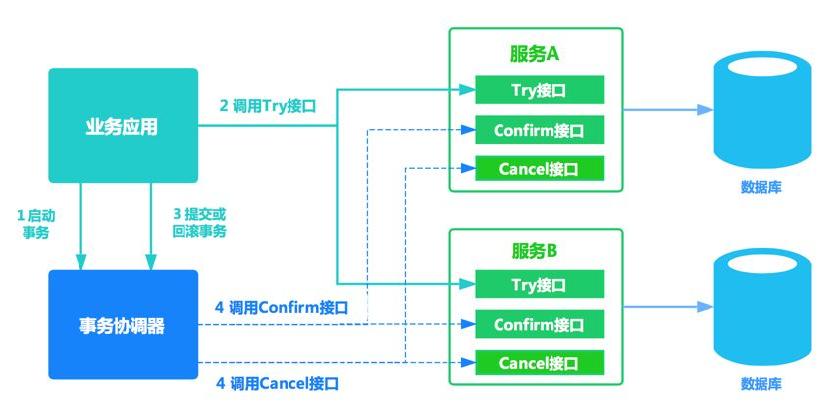
Confirm：确认执行业务操作

Cancel：取消执行业务操作



### 2.TCC原理

TCC方案在电商、金融领域落地较多。TCC方案其实是两阶段提交的一种改进。其将整个业务逻辑的每个分支显示的分成了Try、Confirm、Cancel三个操作。Try部分完成业务的准备工作，confirm部分完成业务的提交。Cancel部分完成事务的回滚。基本原理如下图所示



事务开始时，业务应用会向事务协调起注册启动事务。之后业务应用会调用所有服务的Try接口，完成一阶段准备。之后事务协调起会根据Try接口返回情况，决定调用confirm接口或者cancel接口。如果接口调用失败，会进行重试。

微服务倡导服务的轻量化、易部署，而TCC方案种很多事务的处理逻辑需要应用自己编码实现。复杂且开发量大

### 3.TCC优缺点

#### 3.1. TCC优点

让应用自己定义数据库操作的粒度，使得降低锁冲突、提高吞吐量成为可能。

#### 3.2.不足之处

对应用的侵入性强，业务逻辑的每个分支都需要实现try、confirm、cancel三个操作，应用侵入性较强。改造成本高

实现难度较大。需要按照网络状态、系统故障等不同的失败原因实现不同的回滚策略。为了满足一致性的要求，confirm和cancel接口必须实现幂等。

# 四、什么是LCN框架

### 1. LCN框架的由来

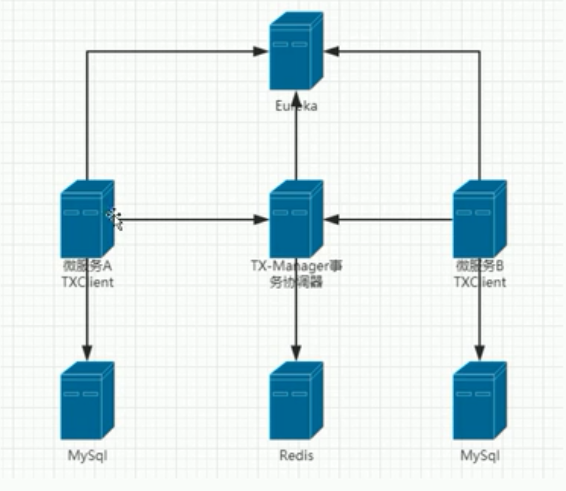
**LCN并不生产事务，LCN只是本地事务的协调工**

在设计框架之初的1.0~2.0的版本时，框架设计的步骤是如下的，名字取其首字母得来的命名。

锁定事务单元（lock）、确认事务模块状态（confirm）、通知事务（notify）

# 五、LCN框架原理及执行步骤

### 1.LCN执行原理



在上图种，微服务A，微服务B，TxMnaager事务协调起，都需要去Euraka中注册服务。Eureka是用户Txmanager与其他服务之间的相互服务发现。Redis是用于存放我们事务组的信息以及补偿的信息。然后微服务A与微服务B他们都需要去配置上我们Txclient的包架构(代码的包架构)，来支持我们的LCN框架，以及他们的数据库

### 2.LCN执行步骤

#### 2.1创建事务组：

事务组是指的我们在整个事务过程中把各个节点(微服务)单元的事务信息存储在一个固定单元里。但这个信息并不是代表是事务信息，而是只是作为一个模块的标识信息。

创建事务组是指在事务发起方开始执行业务代码之前先调用TxManager创建事务组对象，然后拿到事务标识GroupId的过程

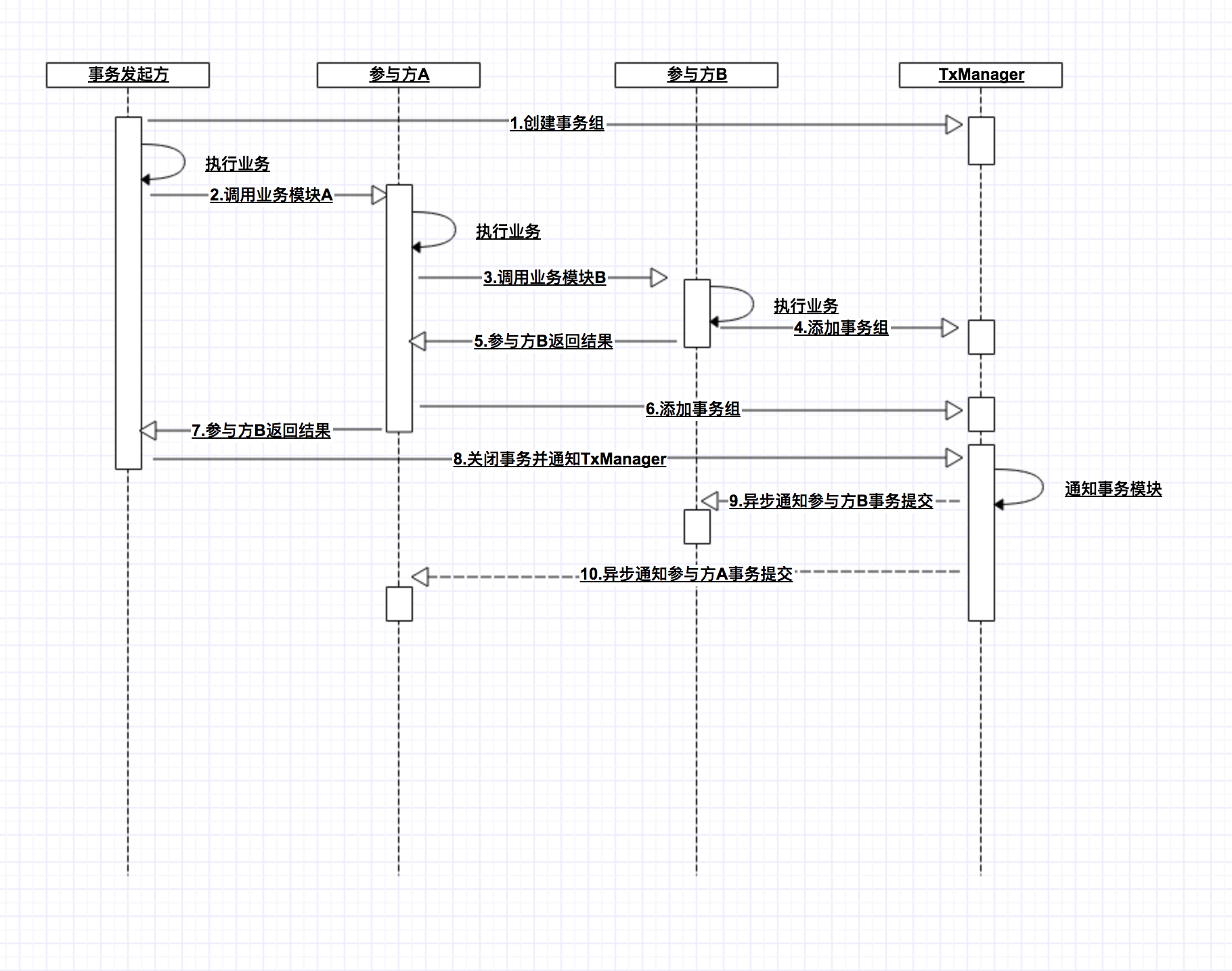
#### 2.2添加事务组

添加事务组是指参与方在执行完业务方法以后，将该模块的事务信息添加通知给TxManager的操作

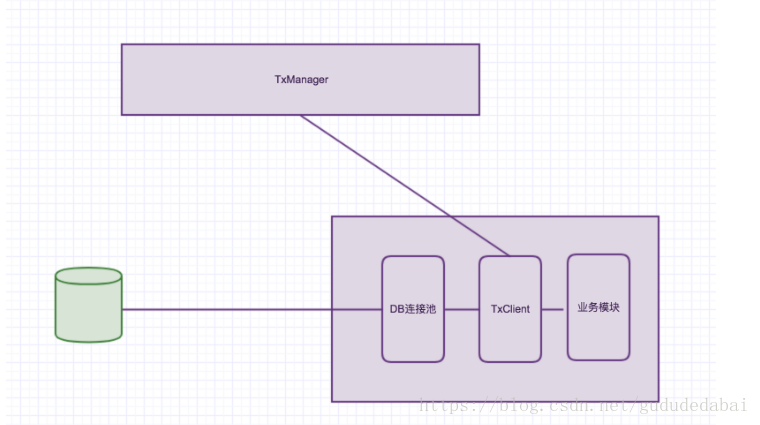
#### 2.3关闭事务组

是指在发起方执行完业务代码以后，将发起方执行结果状态通知给TxManager的动作。当执行完关闭事务组方法以后，TxManager将根据事务组信息来通知相应的参与模块提交或回滚事务

2.4业务执行流程图



# 六、什么是LCN的事务协调机制



如图：假设服务已经执行到关闭事务组的过程，那么接下来作为一个模块执行通知给TxManager。然后告诉他本次事务已经完成。那么如图中TxManager下一个动作就是通过事务组的id，获取到本次事务组的事务信息，然后查看一下对应有哪几个模块参与，然后如果是有A/B/C三个模块，那么对应的三个模块做通知、提交、回滚

那么提交的时候是提交给谁呢？

是提交给了我们的Txclient模块。然后Txclient模块下有一个连接池。就是框架自定义的一个连接池(如图的DB连接池)；这个连接池其实就是在没有通知事务之前一直占有着这次事务的连接资源，就是没有释放。但是他在切面里面执行了close方法。在执行close的时候。如果需要（TxManager）分布式事务框架的链接。他被叫做“假关闭”，也就是没有关闭，只是在执行了一个关闭方法。实际的资源是没有释放的。这个资源是掌握在LCN的连接池里的

然后当TxManager通知提交或事务回滚的时候呢?

TxManager会通知我们的TxClient端。然后TxClient回去执行相应的提交或回滚。提交或回滚之后再去关闭链接。这就是LCN的事务协调机制。说白了就是代理DataSource的机制。相当于是拦截了一下连接池，控制了连接池的事务提交

# 七、什么是LCN的事务补偿机制

### 1.什么是补偿事务机制？

LCN的补偿事务原理是模拟上次失败事务的请求，然后传递给TxClient模块然后再次执行该次请求事务。

简单的说，lcn事务补偿是指在服务器挂机和网络抖动情况下TxManager无法通知事务单元时。（通知不到也就两种原因服务器挂了和网络出问题）在这种情况下TxManger会做一个标示，然后返回给发起方。告诉他本次事务有存在没有通知到的情况。

那么如果是接收到这个信息之后呢，发起方就会做一个标示，标示本次事务是需要补偿事务的。这就是事务补偿机制

### 2.为什么需要实物补偿

事务补偿是指在执行某个业务方法时，本应该执行成功的操作缺因为服务器挂机或者网络抖动等问题导致事务没有正常提交。此种场景就需要通过补偿来完成事务，从而达到事务的一致性。

### 3.补偿机制的触发条件?

当执行关闭事务组步骤时，若发起方接收到失败的状态后将会把该次事务识别为待补偿事务，然后发起方将该次事务数据异步通知给TxManager。TxManager接收到补偿事务以后先通知补偿回调地址。然后再根据是否开启自动补偿事务状态来补偿或保存该次切面事务数据

# 八、如何使用

### 1.TM的准备环境

1.TM启动依赖于redis、eureka服务，所有需要先启动redis、eureka服务

2.修改TM配置文件。主要修改redis跟eureka相关配置

### 2.TC的环境准备

1.TC引入POM依赖

<dependency>

<groupId>com.codingapi.txlcn</groupId>

<artifactId>txlcn-tc</artifactId>

<version>5.0.2.RELEASE</version>

</dependency>

<dependency>

<groupId>com.codingapi.txlcn</groupId>

<artifactId>txlcn-txmsg-netty</artifactId>

<version>5.0.2.RELEASE</version>

</dependency>

2.修改TC配置文件，配置TM地址

txmanger.url=http://TM的ip:port/tx/manager/

3.在TC端注入一个bean实现TxManagerTxUrlService接口获取配置的TM地址

@Service

**public** **class** TxManagerTxUrlServiceImpl **implements** TxManagerTxUrlService{

@Value("${txmanger.url}")

**private** String url;

**public** String getTxUrl() {

// **TODO** Auto-generated method stub

**return** url;

}

}

[4.在需要分布式事务的方法上加@TxTransaction](mailto:4.在需要分布式事务的方法上加@TxTransaction)注解，事务发起发需要设置isStart = true。

@TxTransaction(isStart = **true**)

**public** **void** testLoopCall(**int** i) {

// **TODO** Auto-generated method stub

Orders orders = **new** Orders();

orders.setPrice(2000);

orders.setItemid(i);

Inventory inventory = **new** Inventory();

inventory.setItemnum(88);

inventory.setItemid(i);

ordersServcie.addOrder(orders);

inventorService.addItemRandomErr(inventory);

}

# 九、TC集群

1.在TC配置下加入以下配置

#微服务集群且用到 LCN事务模式时，为保证性能请开启TX-LCN重写的负载策略

tx-lcn.springcloud.loadbalance.enabled=true

# 关闭Ribbon的重试机制

ribbon.MaxAutoRetriesNextServer=0

# 十、使用当中的问题

1.跟Hystric一起使用有问题

2.当TC端事务调用超过TM端配置tm.compensate.maxWaitTime

的值时，会执行回滚

3.必须确保spring容器中有datasource的bean对象