**JTA事务**

Java 提供 JTA用于分布式事务处理，常用的分布式事务标准为XA二阶段确认提交.

下面我们演示原生态XA分布式事务接口

代码如下 :

**class** MyXid **implements** Xid{

**int** formatId;

**byte** globalTransactionId[];

**byte** branchQualifier[];

**public** MyXid(){

}

**public** MyXid(**int** formatId,**byte**[] globalTransactionId,**byte**[] branchQualifier){

**this**.formatId = formatId;

**this**.globalTransactionId = globalTransactionId;

**this**.branchQualifier = branchQualifier;

}

**public** **int** getFormatId() {

**return** **this**.formatId;

}

**public** **void** setFormatId(**int** formatId){

**this**.formatId = formatId;

}

**public** **byte**[] getGlobalTransactionId() {

**return** **this**.globalTransactionId;

}

**public** **void** setGlobalTransactionId(**byte**[] globalTransactionId){

**this**.globalTransactionId = globalTransactionId;

}

**public** **byte**[] getBranchQualifier() {

**return** **this**.branchQualifier;

}

**public** **void** setBranchQualifier(**byte**[] branchQualifier){

**this**.branchQualifier = branchQualifier;

}

}

**public** **class** XaDemo {

**public** **static** **void** main(String[] args) {

System.***out***.println(**new** Date());

**for** (**int** i = 0; i < 1000; i++) {

Connection conn1=**null**;

Connection conn2=**null**;

MysqlXADataSource ds1=**null**;

MysqlXADataSource ds2=**null**;

XAConnection xaCon1=**null**;

XAConnection xaCon2=**null**;

XAResource res1=**null**;

XAResource res2=**null**;

Xid xid1=**null**;

Xid xid2=**null**;

Statement statement1=**null**;

Statement statement2=**null**;

**try** {

ds1=**new** MysqlXADataSource();

ds2=**new** MysqlXADataSource();

ds1.setURL("jdbc:mysql://localhost:3306/mytest?serverTimezone=UTC");

ds2.setURL("jdbc:mysql://localhost:3306/mytest?serverTimezone=UTC");

xaCon1=ds1.getXAConnection("root", "root");

xaCon2=ds2.getXAConnection("root", "root");

conn1=xaCon1.getConnection();

conn2=xaCon2.getConnection();

statement1=conn1.createStatement();

statement2=conn2.createStatement();

res1=xaCon1.getXAResource();

res2=xaCon2.getXAResource();

xid1=**new** MyXid(0, "1".getBytes(), "ab".getBytes());

xid2=**new** MyXid(0, "1".getBytes(), "bc".getBytes());

res1.start(xid1, XAResource.***TMNOFLAGS***);

statement1.execute("insert into t1 values(1)");

res1.end(xid1, XAResource.***TMSUCCESS***);

res2.start(xid2, XAResource.***TMNOFLAGS***);

statement2.execute("insert into t2 values(1)");

res2.end(xid2, XAResource.***TMSUCCESS***);

**int** result1=res1.prepare(xid1);

**int** result2=res2.prepare(xid2);

**if**(result1==res1.***XA\_OK***&&result2==res2.***XA\_OK***){

res1.commit(xid1, **false**);

res2.commit(xid2, **false**);

}**else**{

res1.rollback(xid1);

res2.rollback(xid2);

}

} **catch** (Exception e) {

e.printStackTrace();

**try** {

res1.rollback(xid1);

res2.rollback(xid2);

} **catch** (XAException e1) {

e1.printStackTrace();

}

}**finally**{

**try** {

statement1.close();

statement2.close();

xaCon1.close();

xaCon2.close();

conn1.close();

conn2.close();

} **catch** (Exception e2) {

e2.printStackTrace();

}

}

}

System.***out***.println(**new** Date());

}

}

上面是原生态代码，但是用起来很不方便，例如我还需要手动的通过**if**(result1==res1.***XA\_OK***&&result2==res2.***XA\_OK***)这个条件来决定是否需要提交还是回滚

怎么实现连接多个数据库时不要那么复杂，使得连接时像一个数据库一样开启事务关闭事务呢,

我们就需要引入JTA实现框架Atomikos

Atomikos框架的JTA实现支持JMS和数据库XA,我们要使用它首先要引入它的Jar包

我们演示使用JTA操作Mysql和ActiveMq

Pom.xml

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>nettyTest</groupId>

<artifactId>nettyTest</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>nettyTest</name>

<url>http://maven.aliyun.com/nexus/content/groups/public/</url>

<repositories>

<repository>

<id>central</id>

<name>central</name> <url>http://maven.aliyun.com/nexus/content/groups/public/</url>

</repository>

</repositories>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

<dependency>

<groupId>org.apache.activemq</groupId>

<artifactId>activemq-all</artifactId>

<version>5.14.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.transaction/jta -->

<dependency>

<groupId>javax.transaction</groupId>

<artifactId>jta</artifactId>

<version>1.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.atteo.moonshine/atomikos -->

<dependency>

<groupId>org.atteo.moonshine</groupId>

<artifactId>atomikos</artifactId>

<version>1.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>6.0.3</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

**public** **class** Texst {

//ActiveMq 的默认用户名

**private** **static** **final** String ***USERNAME*** = ActiveMQConnection.***DEFAULT\_USER***;

//ActiveMq 的默认登录密码

**private** **static** **final** String ***PASSWORD*** = ActiveMQConnection.***DEFAULT\_PASSWORD***;

//ActiveMQ 的链接地址

**private** **static** **final** String ***BROKEN\_URL*** ="failover://"+"tcp://" + "192.168.37.135" + ":" + 61616+"";

**public** **static** **void** main(String[] args) {

//mysql数据库1

AtomikosDataSourceBean ds1=**new** AtomikosDataSourceBean();

ds1.setUniqueResourceName("mysql1");

ds1.setXaDataSourceClassName("com.mysql.cj.jdbc.MysqlXADataSource");

Properties p1=**new** Properties();

p1.setProperty("user", "root");

p1.setProperty("password", "root");

p1.setProperty("URL", "jdbc:mysql://localhost:3306/mytest?serverTimezone=UTC");

ds1.setXaProperties(p1);

ds1.setPoolSize(5);

//mysql数据库2

AtomikosDataSourceBean ds2=**new** AtomikosDataSourceBean();

ds2.setUniqueResourceName("mysql2");

ds2.setXaDataSourceClassName("com.mysql.cj.jdbc.MysqlXADataSource");

Properties p2=**new** Properties();

p2.setProperty("user", "root");

p2.setProperty("password", "root");

p2.setProperty("URL", "jdbc:mysql://localhost:3306/mytest?serverTimezone=UTC");

ds2.setXaProperties(p2);

ds2.setPoolSize(5);

//消息队列activemq

XAConnectionFactory xacf=**new** ActiveMQXAConnectionFactory(***USERNAME***, ***PASSWORD***, ***BROKEN\_URL***);

AtomikosConnectionFactoryBean cf = **new** AtomikosConnectionFactoryBean();

cf.setUniqueResourceName("test");

cf.setXaConnectionFactory(xacf);

cf.setPoolSize(1);

Session session =**null**;

UserTransactionManager ut=**new** UserTransactionManager();

**try** {

javax.jms.Connection c\_jms = cf.createConnection();

ut.setTransactionTimeout(1000);

ut.begin();

java.sql.Connection con1=ds1.getConnection();

java.sql.Statement st1=con1.createStatement();

st1.execute("insert into tt values (1)");

java.sql.Connection con2=ds2.getConnection();

java.sql.Statement st2=con2.createStatement();

st2.execute("insert into tt values (1)");

session = c\_jms.createSession(**true**, Session.***SESSION\_TRANSACTED***);

Queue queue=**new** ActiveMQQueue("test");

TextMessage textMessage=session.createTextMessage();

textMessage.setText("hello");

MessageProducer producer=session.createProducer(queue);

producer.send(textMessage);

ut.commit();

System.***out***.println("......commited....");

}

**catch** (Exception e) {

**try** {

ut.rollback();

System.***out***.println("....rolledback........");

} **catch** (Exception e1) {

e1.printStackTrace();

}

e.printStackTrace();

}**finally**{

**if** (session!=**null**) {

**try** {

session.close();

} **catch** (JMSException e) {

e.printStackTrace();

}

}

cf.close();

ds1.close();

ds2.close();

}

System.***out***.println("here ");

}

}