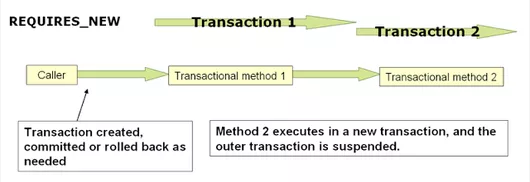
Global Transaction using JTA in Spring

**Global Transaction?**

Transaction Manager is an important aspect of any application. It is responsible for coordinating transaction related tasks across the application. Transactions can be related to single resource such as database operations or messaging system. But sometimes it may come over multiple resources also such as multiple database operations with different data sources or different resources as database and messaging system, which is called global transaction.

To manage transactions in any application, underlying framework on which application is build will provide manager to handle transaction related tasks. For example, Hibernate, Spring etc. If we are using single resource then easy to manage with given transaction manager such as HibernateTransactionManger or Spring’s TransactionManager. This we have already discussed in separate thread.

 If application required multiple resources which is independent of each other, then a separate manager is required to coordinate transactions among these. There are many api available to managing global transaction which is also termed as Distributed Transactions.

**Global Transactions in Spring?**

Spring given centralized transaction management PlatformTransactionManager which is further implemented to provide various transaction support. Mostly used implementations are:

* DataSourceTransactionManager
* JtaTransactionManager
* JpaTransactionManager

These are following component to provide basic idea of using JtaTransactionManager.

Click with ctrl to navigate to these topics.

1. Using JtaTransactionManager
2. **Using JtaTransactionManager**

In application, to use jta transaction manger, we need spring dependency for spring jdbc or orm with which transaction will be applied. Most importantly, we need global data source which will carry out db related operations. But global transactions are independent of local datasource, so we need container managed data source known as XA data source.

Spring Boot supports distributed JTA transactions across multiple XA resources using either an Atomikos or Bitronix embedded transaction manager. JTA transactions are also supported when deploying to a suitable Java EE Application Server. There are multiple others possible way to carry out distributed transactions.

To use Atomikos as transaction manager in spring, requires following dependency:

compile group: 'com.atomikos', name: 'transactions-jta', version: '4.0.3'

compile group: 'com.atomikos', name: 'transactions-jdbc', version: '4.0.3'

This will add atomikos transaction support for jta. Now, we will configure data sources which will be used by application in normal way.

**DataSource Configuration:**

AtomikosDataSourceBean : - The preferred class for using Atomikos connection pooling. Use an instance of this class if you want to use Atomikos JTA-enabled connection pooling. Using this bean, we can set name for this resource and xa data source also. For mysql connectivity, we need MysqlXADataSource datasource which will managed by Atomikos.

@Bean(initMethod = "init", destroyMethod = "close")

**public** DataSource xaLocalDataSource() {

MysqlXADataSource mysqlXaDataSource = **new** MysqlXADataSource();

mysqlXaDataSource.setUrl("jdbc:mysql://localhost:3306/test?useSSL=false&useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC");

//mysqlXaDataSource.setPinGlobalTxToPhysicalConnection(true);

mysqlXaDataSource.setPassword("root");

mysqlXaDataSource.setUser("root");

AtomikosDataSourceBean xaDataSource = **new** AtomikosDataSourceBean();

xaDataSource.setXaDataSource(mysqlXaDataSource);

xaDataSource.setUniqueResourceName("localds");

**return** xaDataSource;

}

@Bean

@DependsOn("xaCustomDataSource")

**public** JdbcTemplate xaCustomJdbc() {

**return** **new** JdbcTemplate(xaCustomDataSource());

}

Above sample code will create a datasource which will be used in JdbcTemplate. Using that we can carry out any db tasks. That is special datasource which be taken care by atomikos jta.

Main part of configuration is transaction manager which will be implementation of PlatformTransactionManager. JtaTransactionManager needs two object as parameter which are:

**UserTransaction TransactionManager**

Its implementation is given by atomikos only, so need complex implementation. Below will show sample configuration of transaction manager.

@Configuration

@EnableTransactionManagement

**public** **class** XADBConfiguration{

@Bean

**public** PlatformTransactionManager txManager () **throws** Throwable {

**return** **new** JtaTransactionManager(userTransaction(), atomikosTxManager());

}

@Bean

**public** UserTransaction userTransaction() **throws** SystemException {

UserTransactionImp userTran = **new** UserTransactionImp();

userTran.setTransactionTimeout(10000);

**return** userTran;

}

@Bean

**public** TransactionManager atomikosTxManager() {

UserTransactionManager userTxMangager = **new** UserTransactionManager();

userTransactionMangager.setForceShutdown(**true**);

**return** userTransactionMangager;

}

}

This code will add jta transaction manager support to the application. Now, we can carry out task using earlier configured JdbcTemplate. Similarly, we can configure multiple datasources with different configurations.

For testing global transaction, write simple service method which may invoke several dao carring different jdbctemplate. As we can see, we added @EnableTransactionManagement to enable it and place @Transactional above service method with required transaction properties as propagation or isolation.

Now, make a rest call using below details, as per the dao code after inserting data into resources, it will cause a RuntimeException which inturn must be rolled back properly from those two resouces.

Make rest call, and verify the same in those data bases that records are inserted or not. To check negative output, comment @Transactional of service method and do the same. This time even after application throws RuntimeException, records are inserted in both the datasources.

Rest uri: **POST** <http://localhost:8096/customer/save>

Json value : {

"customerName": "tran2",

"emailid": "tran2@zeta.com",

"address": "delhi"

}

Read more: <https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-jta.html>