Transactions in Web services:

Web Service Security

Web service Transactions (distributed Transactions)

Addressing

Trust

🡪Whenever the consumer is trying to access the provider then the consumer need to persist the data in the provider DB and Consumer DB as well which is called as Distributed transactions.

**🡪Transaction:**

A transaction is process either committing operation completely or roll-backing it, we should not partial commit or partial rollback if we can ensure to do that we are imposing the transactionality then we can say we are imposing the transactions as our application.

**🡪Types of transactions:**

1. Local Transactions

2. Global or XA or Distributed Transactions

**1. Local Transactions:**

Within the transactional boundary there exist only one transactional participating resource on whom we are trying to issuing commit or rollback then it is called as Local Transactions.

🡪Transactional resource means either we should commit or rollback.

Ex:

DB - Allows us to commit or rollback hence DB connections/resources are transactional participating resources.

File System - We don’t have any commit or rollback options on the file resources hence file system is not called as transactional participating resource.

1. Local Tx on Single DB

2. Local Tx on Multiple DB’s (2-Independent Local Individual Local Tx)

**2. Global or XA or Distributed Transactions:**

Within the transactional boundary if we have multiple global transactional participating resources on whom we are trying to issue a commit or rollback across all the global transactional participating resources then it is called as Global Transactions.

**A global transaction is a mechanism that allows a set of programming tasks, potentially using more than one resource manager and potentially executing on multiple servers, to be treated as one logical unit.**

Once a process is in transaction mode, any service requests made to servers may be processed on behalf of the current transaction. The services that are called and join the transaction are referred to as transaction participants. The value returned by a participant may affect the outcome of the transaction.

A global transaction may be composed of several local transactions, each accessing the same resource manager. The resource manager is responsible for performing concurrency control and atomicity of updates. A given local transaction may be either successful or unsuccessful in completing its access; it cannot be partially successful.

A maximum of 16 server groups can participate in a single transaction.

The Oracle Tuxedo system manages a global transaction in conjunction with the participating resource managers and treats it as a specific sequence of operations that is characterized by atomicity, consistency, isolation, and durability. In other words, a global transaction is a logical unit of work in which:

All portions either succeed or have no effect.

Operations are performed that correctly transform resources from one consistent state to another.

Intermediate results are not accessible to other transactions, although some processes in a transaction may access the data associated with another process.

Once a sequence is complete, its results cannot be altered by any kind of failure.

The Oracle Tuxedo system tracks the status of each global transaction and determines whether it should be committed or rolled back.

**The UserTransaction.begin method starts a global transaction and associates the transaction with the calling thread.**

**Difference between the Loacl and Global (Distributed) Tx:**

Basically the difference between a local transaction and a global transaction is resource bound. A global transaction will span multiple resources. A local transaction is limited to one resource/datasource.

When a managed data source is configured for global transactions, it returns connections that can participate in global transactions. A **Global transaction (also called a distributed transaction)** enlists more than one resource in the transaction.

Global Transactions provide the ability to work with multiple transactional resources (typically relational databases and message queues).

**The Distributed Transaction Process:**

The transaction manager is the primary component of the distributed transaction infrastructure; however, the JDBC driver and application server components should have the following characteristics:

The driver should implement the JDBC 2.0 API (including the Optional Package interfaces XADataSource and XAConnection) or higher and the JTA interface XAResource.

**The application server should provide a DataSource class that is implemented to interact with the distributed transaction infrastructure and a connection pooling module (for improved performance).**

**For more info see below link**

https://www.progress.com/jdbc/resources/tutorials/understanding-jta/distributed-transactions-and-the-transaction-manager

**Types of resources:**

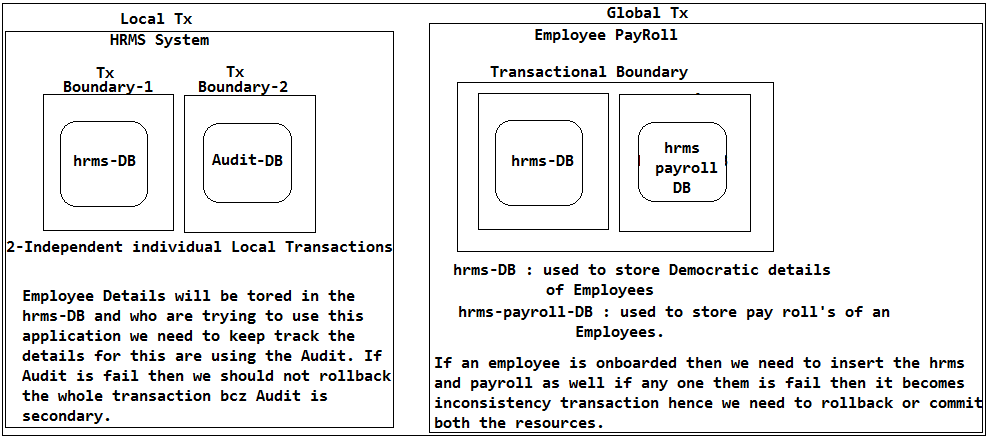
1. Transactional Resources

|- Local Transactional resources- can be DB’s or MOM-Que’s (JMS API)

|-Global transactional resources- can be DB’s or JMS-Que’s (JMS API)

2. Non-Transactional Resources

|-Non-Transactional Resources – File Systems

****

Global Tx Examples:

Pay-Pall transactions management across all the banks.

**🡪API’s available to work with Transactions**

Transactional Resources:

DB’s (allows us to commit or rollback)

MOM- Que’s (allows us to commit or rollback)

🡪In JMS also we have commit and rollback bcz if resource-1 put a message in Que it should be either commit or rollback so that other resources can see or use this Que. That means JMS-Server (MOM server) is which itself contains transactional semantics.

🡪Here Que and topic must and should be transactional Que and Topics then only these can participate in the Tx’s.

Sun Ms. has provided some sort of API’s to work with Transactions

DB- Local Tx: JDBC- API

MOM-Que’s: JMS (Java Messaging Service) API- Global Tx

**Ex:**

class EmployeeDao {

public void saveEmployee(EmployeeBo bo) {

Connection hrmsCon=null;

Connection payRollcon=null;

PreparedStatement pstmt1=null;

PreparedStatement pstmt2=null;

boolean isSuccess=false;

try {

hrmsCon=DM.getConnection(driverClass);

hrmsCon.setAutoComit(false);

// Saving Employee Democratic details

pstmt1=hrmsCon.prepareStatement(sql);

//substitute params

pstmt1.executeUpdate();

// Saving Employee Payroll details

payRollcon.setAutoComit(false);

pstmt2=payRollcon.prepareStatement(sql);

//substitute params

pstmt2.executeUpdate();

isSuccess=true;

} catch(SQLException sqe) {

throw new SQLException(sqe);

isSuccess=false;

} finally() {

if(pstmt1=!null) {

pstmt1.close();

}

if(pstmt2=!null) {

pstmt2.close();

}

if(hrmsCon!=null && payrollCon!=null) {

if(isSuccess) {

hrmsCon.commit();

payrollCon.commit();

} else {

hrmsCon.rollback();

payrollCon.rollback();

}

}

}

}

}

Case 1:

For example if pstmt1 is success and pstmt2 is fail the control comes to catch and marks flag isSuccess=false and within the finally it rollbacks the both the connections.

Case 2:

For example if pstmt1 is success and pstmt2 is also success and marks flag isSuccess=true with in the finally it commits the hrmsCon and while commiting the payrollCon it failed due to payroll-DB problem then hrmsCon will be committed but payroll becomes not persisted in the payroll-DB and hrmsCon will never rollback which is an inconsistency operation hence it is not an GLobal Transaction rather it is called as Local Tx on Multiple DB’s (2-Independent Local Individual Local Tx).

🡪In case of Local Tx the commit or rollback will be issued local to the resource that means hrms and payroll be will be issed individually, but not at a time which is called as Local Tx.

That means JDBC-API will not suports for Global Tx's.

🡪How to work with Tx in JDBC?

🡪It supports only local Tx's where commit or rollback will be issued local to the resource but not globally at a time.

🡪Jdbc works with only DB’s and it cannot work with any transactional resources to issue commit or rollback.

🡪Initially DB's will not have Global Tx's support hence in order to do the Global Tx's through the language API the DB's aslo need to support for the Global Tx's.

🡪But JBDC has been already developed which not supports to work with Global Tx's and JMS API which supports for MOM-server But we need an API which supports Global Tx's across the technologies, that's where Java has provided JTA (Java Transaction API) API.

🡪Every tech specific API has support for specific transaction resources which is called Local Transactions.

🡪**JTA can work with any java technologies resources like DB, JMS MOM-server which is called as Global Transactions.**

🡪Global Transactions is concept and each and every programming language has support for Global Transactions.

**🡪XA (eXtended Architecture):**

🡪In computing, the XA standard is a specification by The Open Group for distributed transaction processing (DTP). It describes the interface between the global transaction manager and the local resource manager.

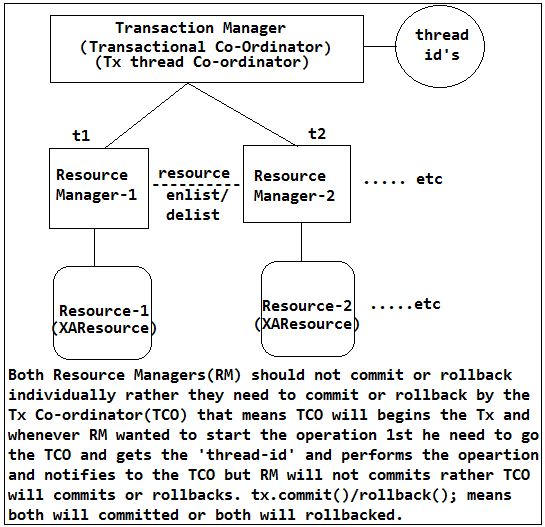
🡪The Java Transaction API (JTA), one of the Java Enterprise Edition (Java EE) APIs, enables distributed transactions to be done across multiple X/Open XA resources in a Java environment. JTA is a specification developed under the Java Community Process.

JTA provides for:

1. Demarcation of transaction boundaries.

2. X/Open XA API allowing resources to participate in transactions.

**X/Open XA architecture or JTA Architecture:**



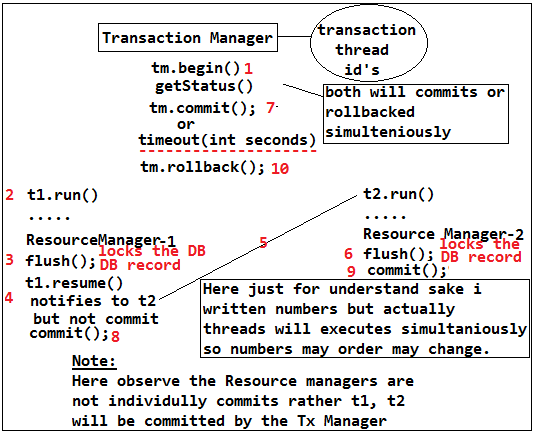
**Note:** In the above diagram

**Resource-1=ApplicationServer-1 representing dataSource = dataSource1**

**Resource-2= ApplicationServer-2 representing dataSource= dataSource2**

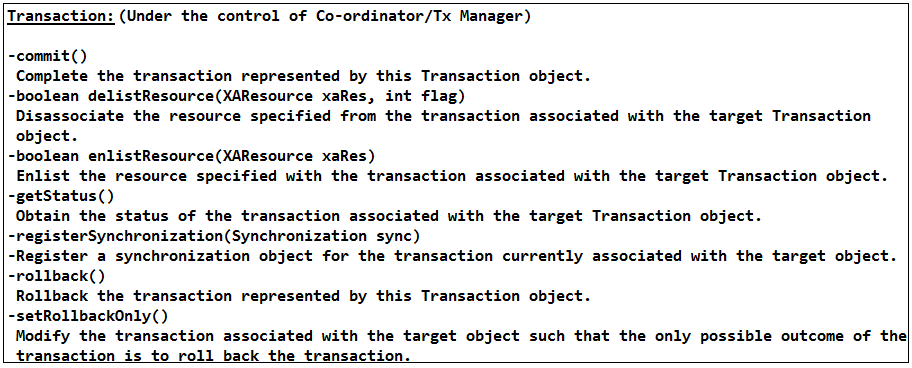
**Note that one Resource (Resource-1) can contains multiple connections which is called as Pool of connections.**

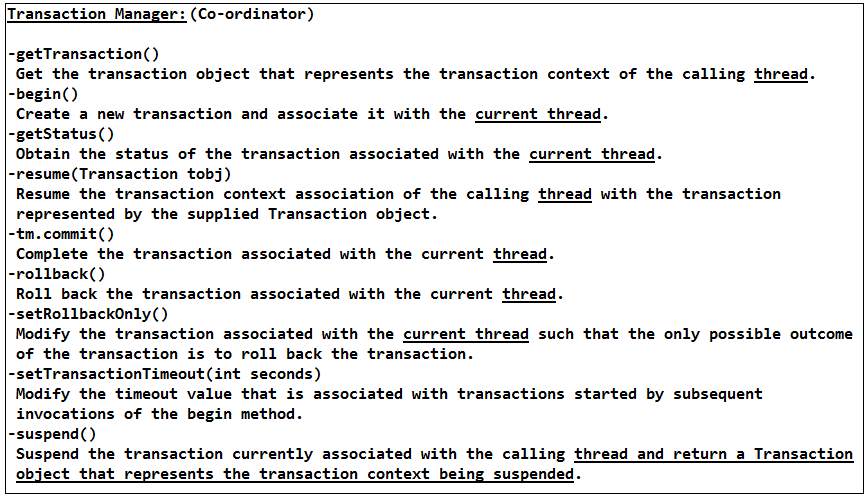
In the X/Open XA architecture, a transaction manager or transaction processing monitor (TP monitor) coordinates the transactions across multiple resources such as databases and message queues. Each resource has its own resource manager. The resource manager typically has its own API for manipulating the resource, for example the JDBC API to work with relational databases. In addition, the resource manager allows a TP monitor to coordinate a distributed transaction between its own and other resource managers. Finally, there is the application which communicates with the TP monitor to begin, commit or rollback the transactions. The application also communicates with the individual resources using their own API to modify the resource.



http://docs.oracle.com/cd/E17802\_01/products/products/jta/jta-1\_0\_1B-doc/

http://docs.oracle.com/cd/E13161\_01/tuxedo/docs10gr3/pgc/pgglob.html





Success Flow:

Whenever tx.commit() will happens then within this tx.commit() methods Tx manager will talks or sends token to the Resource Manager-1 whether the resource-1 is ready for the commit() or not then if Resource-1 is ready for commit then Resource Manager will locks the DB record for a while and writes the read log to file system then similarly Tx Manager will sends the token to the Resource Manager-2 prepare for commit and if both resource managers sends prepared/ready to commit token then Tx Manager will issues token back to both the Resource Managers to commit() then both will commits and both resource Managers will sends an acknowledgment to the Tx Manager tx has been success, so that Tx manager will marks this total Tx is success.

Failure Flow:

Even though the commit token issued by the Tx Manager if resource-1 is able to commit successfully and resource-2 is not able to commit successfully bcz of DB went down then Tx Managers will not gets any success token from the Resource Manager-2 so that Tx manager will do session has been timeout and then sends to the Resource Manager-1 to revert the commit that has happened on the resource-1 then Resource Manager will goes to the read locks then picks-up with what id he has inserted/committed then deletes that record. But now while reverting back to the resource-2 DB went down then Tx Manager will not gets any acknowledgement stating that resource-1 has been reverted/deleted the inserted record then Tx manager marks that Tx as Failure bcz changes are not reverted hence DBA will come and they will do DB-recovery and what Tx's has happened and what are failure based on the read-locks then they manually restores the DB by talking to the business peoples.

**This entire process is called as 2-phase commit.**

That means DB's also need to support for global Tx.

1) javax.transaction

Provides the API that defines the contract between the transaction manager and the various parties involved in a distributed transaction namely: resource manager, application, and application server.

Interfaces

|-Transaction

|-TransactionManager

|-UserTransaction

2) javax.transaction.xa

Provides the API that defines the contract between the transaction manager and the resource manager, which allows the transaction manager to enlist and delist resource objects (supplied by the resource manager driver) in JTA transactions.

Interfaces

|-XAResource

|-Xid

**🡪J2EE Transactions:**

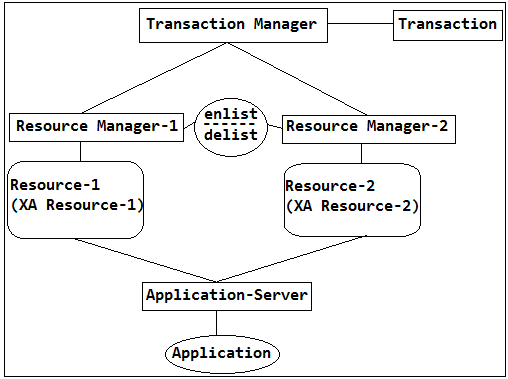
If we wanted to developed the global Tx then we need to write the TX-Manager (Tx Co-ordinator), Resource Manager and User Transaction classes by developer which is difficult mange that’s where the J2EE Application-server vendors has provided the Implementation for the JTA API to support the Global Transactions which s called as J2EE Transactions. The Global Tx is not applicable for J2SE applications they will not deployed on the J2EE container hence for J2SE applications we cannot impose Global Tx’s. That means without application server we cannot work with global Transactions.

🡪So web server (Tomcat) cannot support for Global Transactions but Application server (Web Logic, JBoss) can support for Global Tx’s.

🡪The Sun Ms’s will imposed to the Application servers vendors to support the global support during the JTA specification released hence all the application server will supports for the Global TX’s.

🡪So the developer no need to write the code rather application server will takes care of everything.

**🡪Working with Global Tx’s in J2EE:**



🡪In order to work with Global Tx’s in Application server we need to create Data Source and while creating the Data source we check **XA-Enable** then only it allows to work with Global Tx’s otherwise we cannot.

🡪In order to work with Global Tx’s 1st the DB also need support for the Global Tx’s hence we need to download Jdbc Driver class which supports for Global Tx’s that is the reason DB vendors will provides 2-types jrs example oracle ojdbc6.jar supports for local tx’s and **ojdbc\_xa.jar** which supports global Tx’s so we need to create Data source with OracleXADriver as driver class.

**Set up to work with Global Tx’s:**

Enable/check the XA

Ojdbcxa.jar

Configure the Data source with Oracle XA Driver class

In order to work with global we need to create data sources with 2-different DB’s or one DB-with 2-connections so that we can create 2-Data Sources then we work with Global Tx’s.

🡪In order to work with Global Tx’s we need **Data source which is an resource with in the Application server**. So in order to facilitate the Data source to the applications Application servers will provides JNDI-Registry where Application server will places resources to the external J2EE world applications so that we can access the Data sources that are there in the JNDI-Registry. We can get the Data Source from the Application server in 2-ways

1. Contextual Dependency

|- Contextual Dependency means upon some contract they will provide the Data source.

2. Dependency Look up

|-We need to write some look up logic to get the Data source

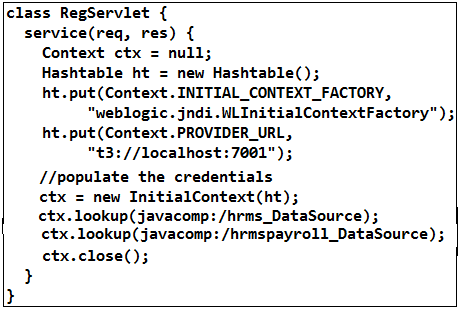
Purpose of JNDI Registry:

To expose the obj's/resources of the J2EE Application server to the J2EE applications that are deployed in it.

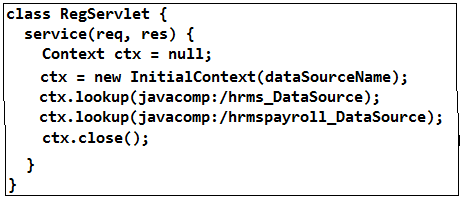
Getting the Data source:

1. Using Initial Context:

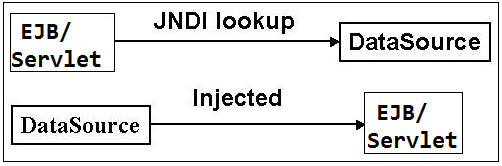
1. If our application component is not part of the same Application server then we need to populate the credentials, URL and Factory classes in Hashtable for accessing.

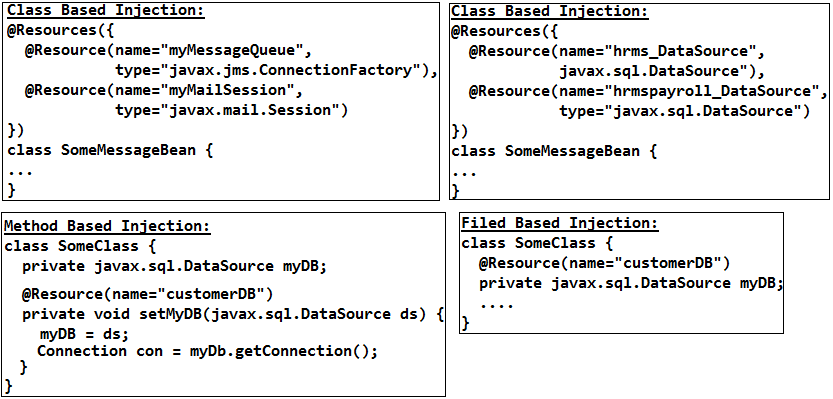


2. If we are deploying our application with in the same Application server then our component is a part of Application server then we no need to pass the credential info, URL, Factory classes rather we can directly create the InitialContext obj without any Hashtable.

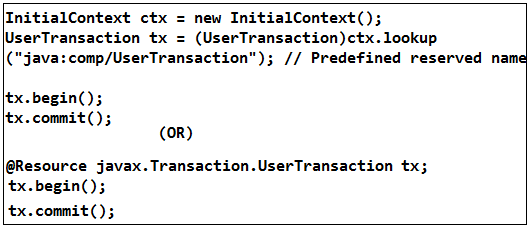


3. But in J2EE-5 onwards we no need to write the initial context rather we use annotation to get the JNDI Registry Data Source. That means Contextual Dependency Injection (CDI) is supported by every J2EE container from J2EE-5 onwards. CDI is supported by the GlasFish, Web Logic Applications servers and Web Sphere, JBoss latest Applications servers’ supports for this but tomcat will not supporters CDI.





🡪In order to work with Global Transactions along with Data source we need UserTransaction which there in the JNDI-Registry of the J2EE Application server with standard name.



The following table lists the types of classes in web and EJB modules that support dependency injection

|  |  |  |
| --- | --- | --- |
| **Container** | **Type of managed classes** | **Resource type** |
| Web | Servlet, listener classes, web services end-point, JAX-RPC handlers | DataSource, JMS, Mail, EJB, Environment entries, EntityManager, UserTransaction |
| EJB | Bean, interceptors, web services end-point | DataSource, JMS, Mail, Environment entries, EntityManager, EJB Context, UserTransaction, TimerService |

Ex:

class MySessionBean implements MySession {

@Resource javax.transaction.UserTransaction ut;

@Resource javax.sql.DataSource hrmsDataSource;

@Resource javax.sql.DataSource hrmsPayrollDataSource;

public void someMethod(...) {

java.sql.Connection con1;

java.sql.Connection con2;

java.sql.Statement stmt1;

java.sql.Statement stmt2;

// obtain con1 object and set it up for transactions

con1 = hrmsDataSource.getConnection();

stmt1 = con1.createStatement();

// obtain con2 object and set it up for transactions

con2 = hrmsPayrollDataSource.getConnection();

stmt2 = con2.createStatement();

// Now do a transaction that involves con1 and con2.

// start the transaction

ut.begin();

// Do some updates to both con1 and con2. The container

// automatically enlists con1 and con2 with the transaction.

stmt1.executeQuery(...);

stmt1.executeUpdate(...);

stmt2.executeQuery(...);

stmt2.executeUpdate(...);

stmt1.executeUpdate(...);

stmt2.executeUpdate(...);

// commit the transaction

ut.commit();

// release connections

stmt1.close();

stmt2.close();

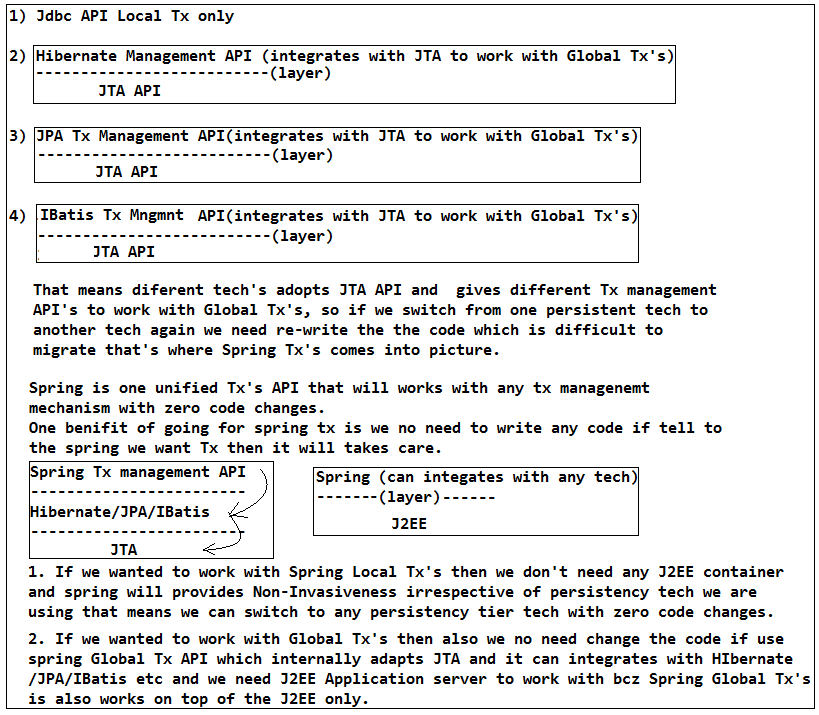
con1.close();

con2.close();

}

}

🡪If we wanted to work with Global Tx’s we need JTA API so for this we need J2EE application server which will provided the Implementation.



Note:

1. If we use spring without any changes in the code we can switch from local to Global Tx and Global to Local Tx’s.

2. Spring has even support for cloud (that means cloud enabled deployments can be done using spring)

3. Spring has integration with every build tools, SCM Repositories.