Spring Data access

Spring Connection management

Spring supports two existing ways to manage the Database connections.

DriverManager connections

JNDI DataSource connections

DriverManager connections:

When you want to use DriverManager connections, you need to configure the DriverManagerDataSource class in spring configuration Document as follows:

<bean id="dataSource"
Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName"value="com.mysql.jdbc.Driver"/>
<property name="url" value="jdbc:mysql://loaclhost/jtcindiadb"/>
<property name="username"value="root"/>
<property name="password" value="Som Prakash "/>
</bean>

Steps in My Eclipse:

1. Create the Java project with the project name Jtc40.
2. Add spring capabilities as follows:
   I. Select the project and Right click
   II. Select My Eclipse add spring capabilities
   III. Select the following:
      a) Spring version Spring3.0
      b) Spring 3.0 AOP Libraries
      c) Spring 3.0 core Libraries
      d) Spring 3.0 Persistence core libraries
      e) Spring 3.0 persistence JDBC Libraries

Iv. Click on Next.

v. provide spring configuration Document file name as jtcindia.xml

vi. click on Finish button.

3. Enable context namespace in the spring configuration Document.
4. Add the following two JAR files to project build path
   Mysql.jar
   Ojdbc14.jar
5. Configure two beans with the type driver Manager Data Source
   One bean for my Sql database
   Another bean for oracle Database

6. Create a package called com.jtcindia.spring.jdbc
7. writhe the testService class and inject above configured two data Source beans.
8. Write the client and run.

Jtc40: Files required

Jtc40.java	TestService.java
Jtcindia.xml	

Jtc 40.java

```
Package com.jtc india.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXml ApplicationContext;
/*
*@Author: Som Prakash  Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Jtc40{
Public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext("jtcindia.xml");
TestService ts=(TestService)ctx.getBean("ts");
Ts.showOraclelnfo();
Ts.showMySQLinfo();
```

TestService.java

```
Package com.jtcindia.spring.jdbc;
Import java.sql.*;
Import javax.annotation.*;
Import javax.sql.*;
/*
*@Author: Som Prakash  Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class TestService {

@Resource(name="oracleDS")
Private DataSource oracleDS;
@Resource(name="mysqlDS")
Private DataSource mysqlDS;

Public void show Oraclelnfo(){
Try{
Connection oracon = oracleDs.getConnection();
DatabaseMeteData
oradbmd=oracon.getMetaData();
System.out.println(oradbmd.getDatabaseProduct
Name{
System.out.println(oradbmd.getDefaultTransactionlsol
ation());
}catch (exception e) {
e.printStack Trace();
}
Public void showMySQLinfo(){
Try{
Connection mycon=mysqlDS.getConnection();
DatabaseMetaData mydbrnd = mycon.getMetaData();
System.out.println(mydbmd.getDatabaseProductName
()
}
System.out.println(mydbmd.getDefaultTransactionlsol
at ion());
}catch (Exception e) {
e.printStack Trace();
}}}
```

}	

Jtcindia.xml

```
<?xml version =”1.0” encoding=”UTF-8”?>
<beans xmlns=http://www.springgramework.org/schema/beans
      Xmlns:p=http://www.springframework.org/schema/p
      Xmlns:context=http://www.springframewok.org/schema/context
      Xsi:schemaLocation=”http://www.springframework.org/schema/beans
      Xmlns:xsi=”http://www.w3.org/2001/xmlschema-instance:
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org./schema/context
 http://www.springframework.org/schema/context/ spring context-3.0.xsd”>

<context:annotation-config/>
<bean id=”oracleDS” class=”org.springframework.jdbc.datasource.DriverManager DataSource”>
<property name=”driverClassname”value=”oracle.jdbc.driver.oracleDriver”/>
<property name=”url” value=”jdbc:oracle:thin:@localhoset:1521:XE”/>
<property name=”username”value=”system”/>
<property name=”password” value=”SomPrakash ”/>
</bean>
<bean id=”mysqlDS” class=”org.springframework.jdbc.datasource.DriverManager DataSource”>
<property name=”driverClassname”value=”oracle.jdbc.driver.oracleDriver”/>
<property name=”url” value=”jdbc:mysql://localhost/jtcindiadb”/>
<property name=”username”value=”root”/>
<property name=”password” value=”SomPrakash ”/>
</bean>
<bean id=”ts” class=”com.jtcindia.spring.jdbc.TestService”/>
</beans>
```

1. Configure Datasource in JBoss4.2 with Datasource JNDI name: JTCDataSourceJNDI.
2. Configure jndiTemplate in Spring Configuration Document file.

```
<bean id=”jndiTemp” class=”org.springframework.jndi.jndiTemplate”>
<property name=”environment”>
<props>
<prop key=”java.naming.factory.initial”>org.jnp.interfaces.NamingContextFactory</prop>
<prop key=”java.naming.provider.url”>localhost:1099</prop>
<prop key=”java.naming.factory.url.pkgs”>org.jboss.naming</prop>
</props></property>
</bean>
```

3. when you want to use JNDI dataSource connection you need to configure the following jndiObjectFactoryBean classs in spring configuration Document.

```
<bean id=”dataSource” class=”org.springframework.jndi.jndiobjectFactoryBean”>
<property name=”jndiName”value=”JTCDataSourceJNDI”/>
```

<property name="jndiTEmplate" ref="jndiTemp"/>
</bean>

Note: Example will be covered along with EJB

Spring DAO support
Spring supports DAO's in 3 ways:
1. provides various DAO classes for various persistent implementations as follows:
    a. jdbcDAoSupport
    b. HibernamteDaoSupport
    c. JpaDaoSupport
2. provides new Exception Hierarchy with DataAccess Exception as root exception for DAO Exception .Data AccessException is a runtime Exception so all the spring DAO Exceptions are runtime Exceptions.

Org.springframework.dao.DataAccessException
Java.lang.object
  Java.lang.Throwable
    Java.lang.Exception
      Java.lang.RuntimeException
        Java.springframework.core.NestedRuntimeException
          Org.springframework.dao.DataAccessException

3. Provides various Exceptions classes for various problems coming with Database interaction.

a) Abstract class DataAccessException extends NestedRuntimeException.
b) Class cleanupFailureDataAccessException extends Data Access Exception
c) Class dataAdccessResourceFailureException extends DataAccessException
d) Class dataRetrieval Railure Exception extends Data AccessException
e) Class deadlock loser data access exception extends Pessimistic locking failure exception
f) Class incorrect update semantics Data Access Exception extends invalid data access Resource usage Exception
g) Class invgalid data access api usage exception extends data AccessException
h) Class invalida Data Access resource usage exception extends data Access Exception
i) Class pessimistic Locking failure Exception extends concurrency failure Except
j) Class Type mismatch Data access exception extends invalid data access resource usage exception
k) Abstract class uncategorized data access Exception extends Data access Except

Spring data access with JDBC

When you want to perform any persistent operation then you need to write the JDBC code with the following steps:

Try{ //1

```
Take connection          //2
Create statement         //3
Prepare SQL              //4
Submit the SQL           //5
Process results          //6
}catch () {}
Finally{
Cleanup                  //7
}
```

You can visit the following problems with above code.

a) All the above statements other than 4 and 6 are common for all the persistent operations which give you code duplication problem.
b) All the methods in JDBC API are throwing one common exception called java.sql.SQLException which is checked exception .Because of checked exception, you need to write try and catch blocks for every program.
c) There is no clear categorization of exceptions in JDDC.

Above problems are solved as follows:

1. Jdbc Template is provided which centralizes the JDBC code.

Usage:

```
String sql=”insert into customers values(?????)”;
Object arg[]={c-101,Som,Som@,1234}
jdbcTemp.update(sql.args);

string sql=”update customers set email=? Where cid=?;
object args[]={Som@jtc”c-101”}
jdbcTemp.update(sql.args);

string sql=”dalete form customers where cid=?”;
object args[]={c-101}
jdbcTemp.update(sql,args);
```

2. In spring Date Access, there is one root exception called Data Access Exception which is unchecked or runtime exception. Because of unchecked exception, you no need to write try and catch blocks for every program.
3. In spring data access, there is clear caregorization of exceptions.

Important methods of Jdbc Template

1. Int update(spl) — Delete
2. Int update(sql,args) — Insert,Update,Delete
3. Int update(sql,args,argTypes)
4. Object query For Object(sql,row mapper)
5. Object query for object (sql,args,row Mapper)
6. Object query for object (sql,args,are types, row Mapper)
7. Object query for object (sql,args,class)

8. List query(sql,rowMapper)
9. List query (sql,args,row Mapper)
10. List query (sql,args,areTypes,rowMapper)
11. Int queryForlnt(sql);
12. Int query ForInt(sql,args);
13. Long queryForLong(sql);
14. Long queryForLong(sql,args);
15. Call(CSC,list)***
16. List query(PSC,rowMapper)
17. Int update(PSC)

Steps in my Eclipse: Spring data access with JDBC

1. Create the java project with the project name Jtc41.
2. Add Spring capabilities as follows:
   - Select the project and Right click
   - Select My Eclipse Add spring capabilities
   - Select the following
     - Spring version spring3.0
     - Spring 3.0 AOP Libraries
     - Spring 3.0 core Libraries
     - Spring 3.0 persistence core Libraries
     - Spring 3.0 Persistence IDBC Libraries
   - Click on Next.
   - Provide spring configuration Document file name as jtcindia.xml
   - Click on Finish button.
3. Enable context namespace in the spring Configuration Document.
4. Add myspl.jar file to project build path.
5. Database Steps:

```
CREATE DATABASE jtcindiadb;
USE jtcindia;
CREATE TABLE customers (
        cid int PRIMARY KEY, cname CHAR(10). Email CHAR (20)
        phone LONG, city CHAR (20));
```

6. Configure Driver Manager Data Source with the following for properties;

```
<bean id="data source"
Class="org.springframework.jdbc.datasource.DriverManagerDateSource">
<property name="driverClassName" value="com.mysql.jdbc.Driver"/>
<property name="url"value="jdbc:mysql://locallhost/jtcindiadb"/>
<property name="username" value="root"/>
<property name="password"value="Som Prakash "/>
</bean>
```

7. Configure jdbc Template class by injecting Data source through constructor injection.

```
<bean     id="jdbcTemp"
          Class="org.springframework.jdbc.core.jdbcTemplate"
```

Autowire=”constructor”/>

8. Create the package called com.jtcindia.spring.jdbc
9. Write the customerDAO interface with the required database operations.
10. Write the sub class for customer DAO interface called jdbc Customer DAO and do the following .
    - Inject jdbc Template into jdbc Customer DAO.
    - Override the customer DAO methods injdbc Customer DAO lusing the oppropriate methods of Jdbc Template.
11. Write the client and run it.

.

Jtc41: files required

Jtc41.java	customerDAO.java
jdbcCustomerDAO.java	Customer TO.java
Jtcindia.xml	

Jtc41.java

```
Package com.jtc india.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXml ApplicationContext;
Public class Jtc41{
Public static void main(String[] args) {
ApplicationContext ctx=lnew classPathXmlApplicationContext(
“jtcindia.xml”);
Customer DAO cdao=(customerDAO) ctx.getBean(“cdao”);
//1.addCustomer
Customer To cto=new Customer TO(203, Somp, Som1@jtc, 3333, noida)
Cdao.addcustomer(cto);
//2.updateCustomer
Coustomer TO cto1=new customerTO (106, som@jtc, 8888, Noida”);
Cdao. updateCustomer (cto1);
//3. deleteCustomer
Cdao.updateCustomer(107);
System.out.println(“check your database”);
```

CustomerDAO.java

```
Package com.jtcindia.spring.jdbc;
Public interface customer DAO {
Public void addCustomer (customer TO cto;
Updlic void updateCustomer (customer TO cto);
Publc void daloeteCustomer(int cid);
}
```

CustomerTO.java

```
Package com.jtcindia.spring.jdbc;
/*
*@Author: Som Prakash  Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class CustomerTO{
```

jdbcCuistoemrDAO.java

```
Package com.jtcindia.spring.jdbc;
Inport org.springframework. beans factory.
Annotation.autowired;
Import
org.springframework.jdbc.core.jdbcTemplate;
/*
*@Author: Som Prakash  Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class jdbcCustomer DAO lmplements
custoemrDAO{

@Autowired
Jdbc Template jdbcTemp;
Public void addcustomer(CustomerTO cto) {
String sql=”insert into customers values(?????)
Object
args[]={cto.getcid(),cto,getcname(),cto.getEmail(),
cto. getPhone().cto.getCity()};
Jdbc temp. update (sql. Args);
}
Public void deleteCustomer(int cid) {
Syring sql=”delete from customers where cid=?”;
Object args[]={cid};
jdbcTemp.update(sql,args);
}
Public void updateCustoemr(customerTO cto){
String sql=”update customer set
Cname=?,email=?,phone=?,cty=?,where cid=?”;
Object
args[]={cto.getCname(),cto.getEmail(),cto.getPhon
e(), cto.getcity(),cto, getcid()};
Jdbc Temp.update(sql,args);
}
}
```

```
Private int cid;              private string cname;
Private string email;     Private long phone;
Private string city;

Public customer TO () {}
Public customer TO (int cid, string cname, string
email, long phone,string city) {
This.cid=cid;          this.cname=cname;
This.email=email;  this.phone=phone;
This.city=city;
}

//setters and Getters
Public string to string () {
Return””+cid+”\t”+cname+”\t”+email+”\t”+
phone+”\t”+city;
} }
```

Jtcindia.xml

```
<beans….>
<context:annotation-config/>
<bean id=”dataSource”
class=”org.springframework.jdbc.datasource.Driver
Man ager Date Source”>
<property name=”driverClassName” value=”com.
Mysql.jdbc.Driver”/>
<property  name=”url”
value=”jdbc:mysql://localhost/jtcindiabd”/>
<property name=”username”value=”root/>
<property name=”password” value=”Som Prakash
”/>
</bean>
<bean id=”jdbcTemp” class=”org.spring
framework.jdbc.core.jdbcTemplate”
autowire=”construstor”/>
<ban id=”cdao”
class=”com.jtcindia.spring.jdbc.jdbcCustomerDAO
”/>
</beans>
```

Jtc42: files required

Jtc42.java	customerDAO.java
jdbcCustomerDAO.java	Customer TO.java
customerRowMapper.java	Jtcindia.xml

Jtc42.java

```
Package com.jtcindia.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXmlApplicationContext;
/*
*@Author: Som Prakash  Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Jtc42{
Public static void main (string[] args) {
ApplicationContext ctx=new classpathXmlApplicationContext("jtcindia.xml");
customerDAO cdao=(customerDAO) ctx.getBean("cdao");
//1.getCustomerByCid
System.out.println(getCustomerByCid");
customerTo cto-cdao.getCustomerByCid(106);
system.out.println(cto);
//2.getAllCustomers
System.out.println(getAllCustomers");
List<customerTO> list=cdao.getAllCustomers();
For(customerTO ct: list)
System.out.println(ct);
//3.getCustomersByEmail
System.out.println("getCustomersByEmail");
Cto=cdao.getCustomerByEmail("Som@jtc");
System.out.println(cto);
//4.getCustomersByCity
System.out.println("getCustomersByCity");
List=cdao.getCustomerByCity("Noida");
For(customerTO ct:list){
System.out.println(ct);
}
//5.getCustomerCount
System.out.println("getCustomerCount");
Int count=cdao.getCustomers Count();
System.out.println(No of cust: +count);
//6.getCustomerCityByEmail
System.out.println("getCustomerCityByEmail");
String ci=cdao.getCustomerCityByEmail("Som@jtc");
System.out.println(ci);
//7.getCustomerPhone ByEmail
System.out.println("getCustomerPhoneByEmail");
Long ph=cdao.getCustomerPhoneByEmail("Som@jtc");
System.out.println(ph);
} }
```

CustomerDAO.java

```
Package com.jtcindia.spring.jdbc
Import java.util.*;
Public interface CustomerDAO {
Public list<customerTO> getAllCustomers();
Public customerTo getCustomerBy Cid(int cid);
Public customerTO getCustomerByEmail(String email);
Public List<customerTO> getCustomersByCity(string city);
Public int getCustomersCount();
Public string getCustomerCityByEmail(Stringemail);
Public Long getCustomerPhoneByEmail(String email);
}
```

3. JdbcCustomerDAO.java

```
Package com.jtcindia.spring.jdbc;
Import java.util.List;
Import org.springframework.beans.factory. annotation.
Autowired;
Import org.springframework.beans.factory.
annotation.Autowired.
Import org.springframework.jdbc.core.jdbcTemplate;
Public class jdbcCustomerDAO implements
CustomerDAO{

@Autowired
JdbcTemplate jdbcTemp:
Public list<customerTO> getAllCustomers() {
String sql=”select* from customers”;
List list=jdbcTemp.query(sql, new
CustomerRowMapper());
Return list;
}
Public int getCustomersCount() {
String sql=”select count(*) from customers”;
Return jdbcTexp.queryForlnt(sql);
}
Public CustomerTO getCustomerbyCid(int cid){
String sql=”select*from customers where cid=?;
Object args[]={cid};
customerTO
cto=(CustomerTO)jdbcTemp.queryForobject(sql, args, new
customerRowMapper());
return cto;
}
Public list<CustomerTO> getCustomerByCity(String city){
String sql=”select*from customers where city=?”;
Object args[]={city};
List list=jdbcTexp.query(sql,args,new
customerRlowMapper());
return list;
}
Public customer TO getCustomerByEmail(Stringemail) {
String sql=”select*from customers where email=?;
Object args[]={email};
customerTO
cto=(customerTO)jdbcTemp.queryForObject(sql, args, new
CustomerRowMapper());
return cto;
}
Public string getCustomerCityByEmail(string email) {
String sql=”select city from customers where email=?;
Object args[]={email};
String
city=(string)jdbcTemp.queryForObject(Sql,args,string.class)
return city;
}
Public long getCustomer phone ByEmail(String email){
String sql=select phone form customers where email=”;
Object args[]={email};
Long
ph=(long)dbcTemp.queryForObject(sql,args,Long.class);
Return ph;
} }
```

CustomerRowMapper.java

```
Package com.jtcindia.spring.jdbc;
Import java.sql.ResultSet;
Import java.sql.SQLEception;
Import org.springframewok.jdbc.core.RowMapper;
/*
*@Author: Som Prakash  Rai
```

```
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class customerRowMapper implements RowMapper<customerTO> {
@override
Public customerTO mapRow(ResultSet rs, int rn) throws SQLException{
customerTO cto=new customerTO();
cto.getCid(rs.getint(1));
cto.setCname(rs.getString(2));
cto.getEmail(rs.getString(3));
cto.getphone(rs.getLong(4));
cto.getcity(rs.getString(5));
return cto;
}
}
```

Working with RowMapper:
When you execute any select statement then data will be stored in the ResultSet object. Now you have to write the code for collecting data form ResultSet Object and string That data in your TO's.
Ex:

```
List list=new ArrayList();
While(rs.next()){
customerTO cto=new Customer TO();
cto.setCid(rs.getInt(1)):
cto.setCname(rs.getString(2));
…..
…..
List.add(cto);
}
```

You may get the requirement to write the same code at different places whenever you execute select statement this gives you the code duplication problem.

You can centralize this knid of code with the help of Row mapper.

Steps:

- Write you own class by implementing Row Mapper interface.
- Override the following method.
  public customerTO mapRow(resultSet rs, int rn) throws SQLException
- Implement the code inside the map Row() to move the date from ResultSet to TO's.

Working with simpleJdbcTemplate
SimplejdbcTemplate is using JDK1.5 var args and generics to simplify the jdbcTemplate functionality.

With jdbcTemplate

String sql=”delete from customers where cid=?”;
Object arg[]={cid};
Int x=simpljdbcTemp.update(spl,args);

With simpljdbc Template;
String sql=”delete from customers where cid=?
Int X=simplejdbcTemp.update(sql,cid);

With jdbcTemplate:
String sql=”select*from customers where cname=? And email=? And phone=?”;
Object args[]={cname,email,phone};
RowMapper<customerTO> crm=newCustomerRowMapper();
List=jdbcTemp.querv(sql.args crm);

With simplejdbcTemplate:
String sql=”select*from customers where cname=? And phone=?;
RowMapper<CustomerTO> crm=new customerRowMapper();
List=simplejdbcTemp.query(sql,crm,cname,email,phone);

Jtc43: files required

Jtc43.java	customerDAO.java
jdbcCustomerDAO.java	Customer TO.java
Jtcindia.xml	

jdbcCustomerDAO.java

```
package com.jtcindia.spring.jdbc;
limport org.springframewok.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.simple.simplejdbcTemplate;
public class JdbcCustomerDAO implements CustomerDAO (
@Autowired
simplejdbcTemplate simplejdbc Temp;
publice void addCustomer(CustomerTO cto) {
string sql=”insert into customers values(?????);
simplejdbcTemp.update(sql,cto.getCid(),cto.getCname()l
cto.getEmail(),cto.getPhone(),cto.getCity());
}
Public void delecteCustomer(int cid) {
String sql=delect from customers where cid=?”;
simplJdbcTemp.update(Sql,cid);
}
Public void updateCustomer(CustomerTO cto) {
String sql=”update customers set cname=? Email=? Phone=? City=? Where cid=?”;
Simplejdbc Temp.update(sql,cto.getCname().cto.getEmail(),cto.getPhone(),cto.getCity(),cto.getCid());
}  }
```

Jtcindia.xml

```
<beans….>
<context:annotation config/>
<bean id=”dataSource” class=”org.springframework.jdbc datasource.Driver Manager DateSource”>
<property name=”driverClassName”value=”com.mysql.jdbc.driver”/>
<property name=”url” value=”jdbc:mysql://localhost/jtcindiadb”/>
<property name=”username” value=”root”/>
<property name=”password” value=”Som Prakash ”/>
</bean>
<bean id=”simplejdbcTemp” class=”org.springframework.jdbc.core.simple.simpleJdbcTemplate”
autowire=”constructor”/>
<bean id=”cdao” class=”com.jtcindia.spring.jdbc.jdbcCustomerDAO”/>
</beans>
```

Jtc44: files required

Jtc44.java	customerDAO.java
jdbcCustomerDAO.java	Customer TO.java
customerRowMapper.java	Jtcindia.xml

JdbcCustomerDAO.java

```
package com.jtcindia.spring.jdbc;
import java.util.List;
limport org.springframewok.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.simple.simplejdbcTemplate;
Public class jdbcCustomerDAO implements CustomerDAO {
@Autowired
simplejdbcTemplate simplejdbcTemp;
public list<customerTO> getAllCustomers() {
string sql=”select*from customers”;
List<customerTO>list=simplejdbcTemp.query(sql,new CustomerRowMapper());
Return list;
}
public Customer To getCustomerByCid(int cid) {
string sql=”select*from customers where cid=?”;
customerTO cto=(customerTO) simplejdbcTemp.QueryForObject(sql,new CustomerRowMapper(),cid);
return cto;
}
Public list<customerTO> getCustomersByCity(String city) {
String sql=”select*from customers where city=?;
List<customerTO> list=simplejdbcTemp.query(sql,new customer RewMapper(),city);
Return list;
}
Public customer to  getCustomerByEmail(String email) {
String sql=”select*from customers where email=?”;
customerTo cto=(customerTO)simplejdbcTemp.query ForObject(sql,new CustomerRowMapper(),email);
```

```
return cto;
}
Public string getCustomersByCity Email(String Email) {
String sql=”select city from customers where Email =?;
String city=(string) simplejdbcTemp.queryForObject(sql,string.class,email);
Return city;
}
Public int getCustomers Count(){
String sql=”select count(*) from customers”;
Return simplejdbcTemp.queryForlnt(sql);
}
Public Long getCustomerPhoneByEmail(String email){
String sql=”select phone from customers where email=?”;
Long ph=(Long)simplejdbcTemp.query ForObject(sql,long.class,email);
Return ph;
} }
```

Working with Jdbc DaoSupport

```
Public class jdbc DaoSupport extends DaoSupport implements InitializingBean{
jdbcTemplate jdbcTemplate;
public jdbc Template getJdbcTemplate(){
        return jdbc Template;
}
Public void setjdbcTemplate(jdbcTemplate jdbcTemplate){
        This.jdbcTemplate=jdbcTemplate;
}
….
Public void afgerPropertiesSet(){
If(jdbcTemplate==null){
        Throw some Exception.
}}
Class jdbcCustomerDAO extends jdbcDaoSupport imp CustomerDAO{
        Public int and customer (customerTO cto) {
        ……
                getjdbcTemplate().update(sql,args);
        ……
        }
}
```

Spring Configuration Document

```
        <bean id=”cdao” class=”….jdbcCustomerDAO” autowire=”byType”/>
```

Working with prepared StatementCreator

When you want to invoke any specific methods on connection then you have to take connection into your control. For this, you can use preparedStatementCreator.

```
Public int addCustomer (final customerTO cto) {
Final string sql=”insert into customers values(?????);
preparedStatementCreator Psc=new preparedStatementCreator(){
public preparedStatement createPreparedStatement(Connection con){
con.setTransactionlsolation(4);
preparedStatement ps=con.prepareStatement(sql);
ps.setInt(1,cto.getCid());
….
Return ps;
}
};
jdbcTemp.update(psc);
}
}
```

What update (preparedStatementCreator) is doing:

```
Void update(PreparedStatementCreator psc){
    Connection con=ds.getConnection();
    Ps=psc.createPreparedStatement(con);
    Ps.executeUpdate();
}
```

What update (sql,args) is doing:

```
Void update(sql,args){
    Connection con=ds.getConnection();
    Ps=con.prepareStatement(sql);
    Ps.setInt(1…);
    …
    Ps.executeUpdate();
}
```

Jtc45: files required

Jtc45.java	customerDAO.java
jdbcCustomerDAO.java	Customer TO.java
customerRowMapper.java	Jtcindia.xml

Jtc45.java

```
package com.jtcindia.spring.jdbc;
import java.util.List;
limport org.springframewok.context.ApplicationContext;
```

```
import org.springframework.context.support.classPathXmlApplicationContext;
Public class Jtc45{
Public static void main (string[] args) {
ApplicationContext ctx=new classpathXmlApplicationContext("jtcindia.xml");
customerDAO cdao=(customerDAO) ctx.getBean("cdao");
//1.Add Customer
customerTO cto=new customerTO(205, Somp, Som@jtc, 3333, noida");
cdao.addCustomer(cto);
//2.Get All Customers
System.out.println("getAllCustomers");
List<customerTO> list=cdao.getAllCustomers();
For (customerTO ct: list) {
System.out.println(ct);
} } }
```

CustomerDAO.java

```
Package com.jtcindia.spring.jdbc;
Import java.util.*;
Public interface customerDAO {
Public void addCustomer(CustomerTO cto);
Public List<customerTO> getAllCustomers();
}
```

jdbcCustomer DAO.java

```
package com .jtciondia.spring.jdbc;
import java.util.List;
import
org.springframework.jdbc.cope.preparedStatement
Creator;
import
org.springframework.jdbc.core.supporet.jdbcDao
dupport;
import java.spl.*;
Public class jdbcCustomerDAO extends
jdbcDaoSupport
Implements CustomerDAO {
Public void addCustomer(final CustomerTo cto){
preparedStatementCreator psc=new
pdreparedStatementCreator(){
Ps.setlnt(1,cto.getCid());
Ps.setString(2,cto.getCname());
Ps.setString(3,cto.getEmail());
Ps.setLong(4,cto.getPhone());
Ps.setString(5,cto.getCity());
Return ps;
}
};
getjdbcTemplate().update(psc);
}
Public List<customerTO> getAllCustomers(){
preparedStatementCreator psc=new
preparedStatementCreator(){
public preparedStatement
createPreparedStatement(Connection con)
throws SQlException{
preparedStatement ps=con.prepareStatement
("select* from customers");
return ps;
}
};
List list=getjdbcTemplate().query(psc,new
CustomerRowMapper());
Return list;
```

public preparedstatement createPreparedStatiement(Connection con) throws SQLException{ preparedStatement ps=con.prepareStatement(“insert into customers values(?????)	} }

Jtcindia.xml

```
<?xml version =”1.0” encoding=”UTF-8”?>



.
<property name=”driverClassName”value=”com.mysql.jdbc.Driver”/>
<property name=”url”value=”jdbc:mysql://localhost/jtcindiadb”/>
<property name=”username”value=”root”/>
<property name=”password” value=”Som Prakash ”/>
</bane>
<bean id=”jdbcTemp”class=”org.springframework.jdbc.core.jdbcTemplate”autowire=”constructor”/>
<bean id=”cdao” class=”com.jtcindia.spring.jdbc.jdbcCustomerDAO” autowire=”byType”/>
</beans>
```

Working with callableStatementCreator

CallableStatementCreator is used to invoke the Stored procedure running in the database.

Procedure in MySQL Database:

```
Delimiter $
Create procedure addCustomer(cid int, cnm varchar 12, em varchar 25 phn long city varchar 15
Begin
Insert into customers values (cid,cnm,eml,phn,city);
End;
$
Delimiter;
```

Java Statement:

```
Public void add customer (final customer TO cto){
callableStatementCreator csc=new callableStatementCreator() {
public calJtcle Statement create CalJtcleState ment(Connection con) throws SQLException {
callableStatement cs=con.prepareCall(“call addCustomer(?????);
cs.setInt(1.cto.getCid()); cs.setString(2,cto.getCname()); cs.setString(3,cto.getEmail());
cs.setLong(4,cto.getPhone());          cs.setString(5,cto.getCity());
return cs;
}}
sqlParameter charParam=new sqlParameter(Types.VARCHAR);
```

```
sqlParameter charParam=new sqlParameter(Types.LONGVARCHAR);
List<sqlParameter> plist=new ArrayList<sqlParameter>();
Plist.add(charParam);
Plist.add(charParam);
Plist.add(charParam);
Plist.add(longParam);
Plist.add(charParam);
GetjdbcTemplate(). Call(csc, plist);
}
```

Jtc46: files required

Jtc46.java	customerDAO.java
jdbcCustomerDAO.java	Customer TO.java
Jtcindia.xml	

Jtc 46.java

```
Package com.jtc india.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXmlApplicationContext;
/*
*@Author: Som Prakash  Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Jtc34{
Public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext("jtcindia.xml");
customerDAO cdao=(CustomerDAO) ctx.getBean("cdao");
//1.Add customer
customerTO cto=new customerTO(206,"som" Som@jtc 3333 Noida")
cdao.addCustomer(cto);
system.out.println("call completed");
}
}
```

CustomerDAO.java

```
Package com.jtcindia.spring.jdbc;
Public interface customerDAO {
Public void addCustomer(CustomerTO cto);
}
```

Jtcindia.xml

```
<beans….>
<context:annotation config/>
<bean id=”dataSource”
Class=”org.springframework.jdbc.datasource.DrivorManagerDatasource”>
<property name=”driverClassName” value=”com.mysql.jdbc.Driver”/>
<property name=”url”value=”jdbc:myql://localhost/jtcindiadb”/>
<property name=”username” value=”root”/>
<property name=”passwlord” value=”Som Prakash ”/>
</bean>
<bean id=”jdbcTemp”class=”org.springframework.jdbc.core.jdbcTemplate”autowire=”constructor”/>
<bean id=”cdao” class=”com.jtcindia.spring.jdbc.jdbcCustomerDAO” autowire=”byType”/>
</beans>
```

jdbcCustomerDAO.java

```
package com.jtcindia.spring.jdbc;
import java.util.ArrayList;
import java.utit.List;
import org.springframework.jdbc.core.callableStatementCreator;
import org.springframework.jdbc.core.sqlParameter;
import org.springframework.jdbc.core.support.jdbcDaoSupport;
import java.sql.*;
```

```
Public class jdbcCustomerDAO extends jdbcDaoSupport implements CustomerDAO {
Public void addCustomer(final CustomerTO cto) {
CalJtcle Statementcreator csc=new callableStatementCreator() {
Public callableStatement createCallableStatement(Connection con)
Throws SQLException{
callableStatement cs=con.prepareCall(“call addCustomer(?????);
cs.setint(1,cto.getCid());
cs.getString(2,cto.getCname());
cs.getString(3,cto.getEmail());
cs.getString(4.cto.getPhone());
cs.getString(5,cto.getCity());
return cs;
}
};
sqlParameter charParam=new sqlParameter(Types.VARCHAR);
sqlParameter charParam=new sqlParameter(Types.LONGVARCHAR);
List<sqlParameter> plist=new ArrayList<sqlParameter>();
Plist.add(charParam);
Plist.add(charParam);
Plist.add(charParam);
Plist.add(longParam);
Plist.add(charParam);
```

```
getjdbcTemplate().call(csc,plist);
}}
```

Working with Named Parameter Jdbc template
With jdbc Template

String sql=”delete from customers where cid=?”;
Object arg[]={cid};
Int x=simpljdbcTemp.update(spl,args);

With simpljdbc Template;
String sql=”delete from customers where cid=?
Int X=simplejdbcTemp.update(sql,cid);

With named parameterjdbcTemplate:
String sql=”delete form customers where cid=CID;
Map<string, object> parameters = new HashMap<string, object>();
Parameters.put(CID,cid);
nameParameterJdbcTemp.update(spl,parameters);

Jtc47: files required

Jtc47.java	customerDAO.java
jdbcCustomerDAO.java	Customer TO.java
customerRowMapper.java	Jtcindia.xml

Jtc 47.java

```
Package com.jtc india spring jdbc;
Import java.util.List;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXml ApplicationContext;
/*
*@Author: Som Prakash  Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class Jtc47{
Public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext(“jtcindia.xml”);
customerDAO=(CustomerDAO)ctx.getBean(“cdao”);
//1. addCustomer
customerTo cto=new customerTo(301, manish” manish@jtc” 3333” noida”);
//2.updateCustomer
customerTo cto1=new customerTO(203, som, som@jtc” 8888, Noida”);
cdao.updateCustomer(cto1);
```

```
//3.deleteCustomer
Cdao.deleteCustomer(205);
//4. getCustomerByCid
System.out.println("getCustomerByCid");
customerTo cto2=cdao.getCustomerByCid(106);
system.out.printin(cto2);
//5.getAllCustomers
System.out.println("getAllCustomers");
List<customerTO> list =cdao.getAllCustomers();
For(CustomerTO ct:list){
System.out.println(ct);
} } }
```

CustomerDAO.java

```
Package com.jtcindia.spring.jdbc;
Lmport java.util.list;
Public interface customerDAO {
Public void addCustomer(CustomerTO cto);
Public void update Customer(customerTO cto);
Public void deleteCustomer(int cid);
Public customerTO getCustomerByCid(int cid);
Public List<CustomerTO> getAllCustomers();
}
```

Jtcindia.xml;

```
<?xml version="1.0" encoding ="UTF-8"?>
<beans….>
<context:annotation-config/>
<bean id=dataSource"
Class="org.springframework.jdbc.datasource.DriverManagerDataSource">
<property name="driverClassName" value="com.mysql.jdbc.Driver"/>
<property name='url"value="jdbc:mysql://localhost/jtcindiadb"/>
<property name="username" value="root"/>
<property name="password" value="Som Prakash "/>
</bean>
<bean id="nameParameterjdbcTemp"
Class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate"
autowire="constructor"/>
<bean id="cdao"class="com.jtcindia.spring.jdbc.jdbcCustomerDAO"/>
</beans>
```

jdbcCustomerDAO.java

```
package com.jtcindia.spring.jdbc;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
Public void update Customer(CustomerTo cto){
String sql="update customers set cname=:cname,email="email,phone=:phone, city=city where cid="cid;
```

```
import org.springframework.beans.factory.annotation
Autowired;
import
org.springframework.jdbc.core.namedparam.MapSqlpa
rameterSource;
import org.springframework.jdbc.core.namedparam.
MapSqlpa rameterSource;
import org.springframework.jdbc.core.namedparam.
NamedPar ameterjdbcTemplate;
import
org.springframework.jdbc.core.namedparam.SqlParameter
Source;
/*
*@Author: Som Prakash  Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class JdbcCustomerDAO implements
CustomerDAO{
@Autowired
NamedParameterjdbcTemplate
nameParameterjdbcTemp;
```

```
Map<string,object> parameters = new
HashMap<string,object>();

Parameters.put("cname",cto.getCname());
Parameters.put("email",cto.getCname());
Parameters.put("phone",cto.getCname());
Parameters.put("city",cto.getCname());
Parameters.put("cid",cto.getCname());

nameParametirjdbcTemp.update(sql,param
eter);

Public customerTO getCustomerByCid(int
cid) {
String sql="select* from customers where
cid=cid";
sqlparameterSource parameters=new
mapSqlParameterSource("cid",cid);
customerTO cto=(customerTO)
nameParameterJDBCTemp.queryForObjec
t(sql,
parameters, new CustomerRewMapper());
return cto;
}
```