**Spring MVC in 5 minutes:**

In Details:

Typical JDBC Flow

* Load the driver.
  + Code will absolutely exact for different DAO.
* Get the connection.
  + Code will absolutely exact for different DAO.
* Create (prepare) statement.
  + Code will absolutely exact for different DAO.
* Execute queries
  + This changes as per query.
* Parsing through the result set to get the model object
  + This changes as per query result.
* Exception handling code
  + Code will absolutely exact for different DAO.

For ex class see

* SpringDB\src\A\_JDBC\dao\JDBCdaoImpl.java
  + DAO class - Databae related activities
* SpringDB\src\A\_JDBC\JDBC\_Demo.java
  + Main Class – Instatiation and use of DAO class
* SpringDB\src\A\_JDBC\model\Circle.java
  + Object representation of DB Table.

Issue with JDBC is because of boilerplate code

Lot of code is required just

Open a connection

Handle a data

Exception hadling for sql exception etc.

Even connection.close() inside finally need to catch exception.

How to use spring to make things simple?

How to convert typical JDBC project into simple spring enabled project?

* Class-path setup
  + Put the spring jars into classpath.
  + Put /SpringDB/src/XMLcontextFilesDB/ into classpath.
* Add spring-xml file
  + <context:annotation-config/>

Says spring project is annotation based

* + <context:component-scan base-package=*"B\_SpringDB.dao"* />
  + Package to look for bean(DAO) class.
* Tag the DAO class with component annotation.
  + This is same as registering(or entering) the class in xml file
* Use application context instead of direct instantiation

ApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.JDBC.xml");

JdbcDaoImpl daoImpl = context.getBean("jdbcDaoImpl", JdbcDaoImpl.**class**);

In this demo we are not using Spring features of DB but simply using JDBC approach and making daoImpl class as bean using spring.

In below examples we will replace the daoImpl with actual Spring features using different jars.

***DriverManagerDataSource***

org.springframework.jdbc.datasource.DriverManagerDataSource.class

Spring provides support class DriverManagerDataSource for DB connection

Steps to include DriverManagerDataSource in your project:

Step1: Configure DriverManagerDataSource in xml file.

<beans>

<context:annotation-config/>

<context:component-scan base-package=*"C\_SpringDataSource.dao"* />

<bean id = *"dataSource"* class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"url"* value=*"jdbc:mysql://localhost/SpringDB"*/>

<property name=*"username"* value=*"manish"*/>

<property name=*"password"* value=*"ruylopez"*/>

</bean>

</beans>

Step2: To make above *dataSource available in DAO class.*

* In DAO class create the dataSource member variable and it’s getter and setters.
* Mark the dataSource variable autowired.

@Autowired

**private** DataSource dataSource;

**public** DataSource getDataSource() {

**return** dataSource;

}

**public** **void** setDataSource(DataSource dataSource) {

**this**.dataSource = dataSource;

}

Step3: Finally, Obtain the connection string from dataSource.

connection = dataSource.getConnection();

Advantage:

* Connection string is specified at one location in spring-xml file instead of every method of DAO class.
* If you have 100 of DAO using same DB then just as above use autowired data source member variable in each DAO class.

DataSource is not very efficient implementation

* It doesn't implement connection-pooling
* the getConnection method every time creates a new connection

**dbcp.BasicDataSource**

DBCP is another implementation of DataSource which implements connection-pooling

org.apache.dbcp.BasicDataSource

Required jars

commons-dbcp-xxx.jar

commons-pool-xxx.jar

How to use DBCP data-source in your project?

Add the below jars in classpath:

Modify the spring-xml for DBCP class

<beans>

<context:annotation-config/>

<context:component-scan base-package=*"C\_SpringDataSource.dao"* />

<bean id = *"dataSource"* class=*"org.apache.commons.dbcp.BasicDataSource"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*/>

<property name=*"url"* value=*"jdbc:mysql://localhost/SpringDB"*/>

<property name=*"username"* value=*"manish"*/>

<property name=*"password"* value=*"\*\*\*\*\*\*"*/>

<property name=*"initialSize"* value=*"2"*/>

<property name=*"maxActive"* value=*"5"*/>

</bean>

</beans>

And all the other code (DAO and main) will remain same.

**JdbcTemplate**

org.springframework.jdbc.core.JdbcTemplate;

JDBC code for DB operations can be divided into three main components:

* Pre query execution
  + Loading driver manager
  + Getting connection
* During execution
  + Passing the query string
  + Getting the output from DB and working on that
* Post query execution
  + Closing the connection
  + Exception handling

jdbcTemplate class has code for pre execution and post-execution. So you need to focus only the stuffs which is related to “during execution”. pre and post execution is automatically taken care by jdbcTemplate class.

Using the jdbcTemplate:

Without Spring-xml :

In DAO class:

Use the jdbcTemplate as member variable and instantiate it.

**private** JdbcTemplate jdbcTemplate = **new** JdbcTemplate();

Provide it’s getter and setter

Set the datasource in to jdbcTemplate class.

This can be done in two ways.

jdbcTemplate.setDataSource(getDataSource());

getDataSource() provides the dataSource already configured in xml file for different class DriverManagerDataSource.

It’s always good to assign datasource while initializing JdbcTemplate as different methods within DAO can use it without code duplication.

This can be achieve by via

Moving the instantiation of JdbcTemplate class in setter of datasource and pass the datasource in constructor of JdbcTemplate.

Autowired the setter to make sure this setter is called during the initialization of spring container.

**public** **void** setDataSource(DataSource dataSource) {

**this**.dataSource = dataSource;

}

Creating bean definition for JdbcTemplate in xml file.

There is no change in main class.

Declaring in Spring-xml

With JdbcTemplate

* We don't need to create the connection
* We don't need to create the statement
* We don't need to close the connection/resultset/statement.
* We don't need to handle the exceptions.

Theses all taken care by JdbcTemplate class.

Only thing we do is

provide the datasource and execute the query using the methods provided by JdbcTemplate class.

Methods of JdbcTemplate

queryForInt

String sql = "SELECT count(\*) From CIRCLE";

**return** jdbcTemplate.~~queryForInt~~(sql);

Takes the sql as input and returns int.

queryForObject

String sql = "SELECT CIRCLE\_NAME From CIRCLE Where ID= ?";

The output of this query is Circle name which is string.

As JdbcTemplate doesn’t have queryForString so a general method queryForObject will be used. It has six overloaded methods will be discussed next. Here we need to use below version

jdbcTemplate.queryForObject(String sql, Object[] args, requiredType)

* sql : sql need to executed.
* Args : array of parameters for parameterized SQL (i.e. ?)
* requiredType : the object type need to return.

So the actual method call will be :

jdbcTemplate.queryForObject(sql, **new** Object[]{circleID}, String.**class**)

Rowmapper:

Why do we need this?

Returning Circle object itself:

String sql = "SELECT \* From CIRCLE Where ID= ?";

jdbcTemplate.queryForObject(sql, **new** Object[]{circleID}, String.**class**)

RowMapper :

UseCase : When object(complete row) or list of object need to be returned.

JdbcTemplate alone can't know which row is mapped with which resultset

We need to create object corresponding to table-row

We need to create customize row mapper which implements rowMapper interface.

In this customize rowMapper we need to map corresponding table-row to the object property.

queryForList