Session-37

Spring with JDBC

Spring with JDBC: (Introduction)

-> Some Problems occurred when a java programmer directly work with JDBC in application development. They are

i) JDBC technology exceptions are checked, so we must use try, catch blocks in the code at various places which increases the complexity of the application

ii) In JDBC if we open the connection with database, we only responsible to close that connection. If not we may get some connection issues

iii) If you see JDBC, it will throws error codes of the database, whenever an exception is raised. These error codes are different from one database to another database. So our application becomes database dependent.

-> In order to overcome the above problems by using JDBC directly, Spring framework has provided one abstraction layer on top of existing JDBC technology.

-> We used to call this layer as Spring-JDBC.

-> So spring-JDBC layer will take cares about connection management and error managements.

-> It means spring framework has provided an exception translator and it translates the checked exceptions obtained using JDBC to un-checked exceptions of spring type and finally the un-checked exceptions are thrown to java programmer

-> And no need to open and close the database connection and it will be taken care by the spring framework.

-> A java application can get connection with database using following 2 ways

1. By using java.sql.DriverManager [Class]
2. By using javax.sql.DataSource [Interface]

-> Spring framework uses DataSource interface to obtain the connection with database internally.

-> It means we will use any one of the following 2 implementation classes of DataSource interface.

1. org.springframework.jdbc.datasource.DriverManagerDataSource [class]
2. org.apache.commons.dbcp.BasicDataSource [class]

Note 1:- The above 2 classes are suitable when our spring application is at developing stage.

Note 2:- In real time programmers uses connection pooling service provided by the application server.

-> In above 2 classes DriverManagerDataSource is given by spring framework and it is equal to DriverManager class

-> It means spring framework internally opens a new connection and closes the connection for each

operation done on the database.

-> In spring config we need to configure the following 4 properties to obtain connection with database.
syntax:-

```
<bean id="id1" class="org.springframework.datasource.DriverManagerDataSource">
                                    [ or ]
<bean id="id1" class="org.apache.commons.dbcp.BasicDataSource">
   <property name="driverClassName" value="" />
   <property name="url" value="" />
   <property name="username" value="" />
   <property name="password" value="" />
</bean>
```

JdbcTemplate Class In Spring-JDBC:

-> JdbcTemplate class is given in org.springframework.jdbc.core.* package and this class will provides methods for executing the SQL commands on a database
-> JdbcTemplate class provided the following 3 type of methods to execute SQL operations on the database:

1. execute()
2. update()
3. query()

Note: execute and update methods are for non-select operations on the database, and query method is for select operations on the database.

-> JdbcTemplate class depends on DataSource object only, as it will opens database connection internally with DataSource.
-> So we must give this DataSource object to JdbcTemplate by using either setter approach or constructor approach.

Spring config file if we insert DriverManagerDataSource object into JdbcTemplate class with setter injection:

syntax:

```
<bean id=”id1” class=”org.springframework.datasource.DriverManagerDataSource”>
   <property name=”driverClassName” value=”” />
   <property name=”url” value=”” />
   <property name=”username” value=”” />
   <property name=”password” value=”” />
</bean>
<bean id="id2" class="org.springframework.jdbc.core.JdbcTemplate">
   <property name="dataSource" ref="id1" />
<bean>
```