**Spring Expression Language**

Using Spring expression language supports parsing and executing expression using @Value annotation

@Value(“#{12+24}”)

@Value(“#{}”)

We can provide expression with @Value annotation but we can also parse expression using some classes.

(Note - >we can call static method, access static variable and object)

But if you will have to call/invoke static method then we will have to use following syntax

T(class).method(param)

To use variable

T(class).variable

**Bean Scope**

The Spring Framework supports the following five scopes, three of which are available only if you use a web-aware ApplicationContext.

|  |  |
| --- | --- |
| **Sr.No.** | **Scope & Description** |
| 1 | **singleton**  This scopes the bean definition to a single instance per Spring IoC container (default). |
| 2 | **prototype**  This scopes a single bean definition to have any number of object instances. |
| 3 | **request**  This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext. |
| 4 | **session**  This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |
| 5 | **global-session**  This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext. |

If you are declaring bean inside xml file then we will have to declare scope there in bean declaration but if you are using stereotype annotation then we will have to use

@Scope annotation

<bean class=”” name=”” scope=”prototype”>

@Component

@Scope(“prototype”)

Class abc{

}

**Spring JDBC 🡪**

In core java we had JDBC API.

For achieving we had to follow 5 steps.

* Register Driver
* Connection
* Statement
* Execute Queries
* Close connections

Database

Application

Internally this Spring JDBC module has been built on top of JDBC API.

Why we need this Spring JDBC?

1. In order to avoid repetitive work
2. Checked Exceptions were there, every time we have to handle those exception explicitly

SQLException class was creating checked exceptions for us and we had to handle those all by using either try catch or by throws.

Spring JDBC 🡪 Spring JDBC module provides us a class by the name JdbcTemplate, this class is well tuned with methods to perform all operations on Database.

DataSource

(Interface)

Our application will perform

All database related operations.

DriverManagerDataSource

(Class)

For specifying Data source we need

1. Driver class name
2. url
3. username
4. password

**JdbcTemplate**

* Insert, update, delete 🡪 update()
* Select 🡪 execute()
* If I want to fetch only one record from database

Public T queryForObject(String query, RowMapper<T> rowMapper, Object args)

* If I want to get all data in table
  + Public List<T> query(String query, RowMapper<T> rowMapper)
* RowMapper is an interface 🡪 Row mapper get data from result set and convert it into class object
* For creating object of RowMapper we will have to create our class by implementing RowMapper interface then we will be able to create object of RowMapper.

Steps for Spring JDBC application

(Create database and create a table in that database)

Step 1. Create Maven project

Step 2. Add dependencies in POM.xml

Step 3. Create a class Emp.java

Step 4. Create config.xml file and do configuration.

* Notes 🡪 RowMapper is an interface 🡪 Row mapper get data from result set and convert it into class object