Contents

[Spring JDBC Template 1](#_Toc171612953)

[XML 2](#_Toc171612954)

[Spring – ResultSetExtractor 3](#_Toc171612955)

[Sprng DataJPA 3](#_Toc171612956)

[Advantages of Using JPA 3](#_Toc171612957)

[Repository 5](#_Toc171612958)

[CrudRepository 5](#_Toc171612959)

[PagingAndSortingRepository 5](#_Toc171612960)

[Spring Boot 5](#_Toc171612961)

[Spring dispatcher 6](#_Toc171612962)

[Spring dispatcher with out web.xml 7](#_Toc171612963)

[Remote Invoker 7](#_Toc171612964)

[Spring – Expression Language(SpEL) 8](#_Toc171612965)

# **Spring JDBC Template**

**JdbcTemplate**is acentral class in the **JDBC**core package that simplifies the use of **JDBC**and helps to avoid common errors. It internally uses**JDBC API** and eliminates a lot of problems with **JDBC API**. It executes SQL queries or updates, initiating iteration over ResultSets and catching **JDBC**exceptions and translating them to the generic. It executes core **JDBC**workflow, leaving application code to provide SQL and extract results. It handles the exception and provides the informative exception messages with the help of exception classes defined in the **org.springframework.dao** package.

The common methods of spring JdbcTemplate class.

| **Methods** | **Description** |
| --- | --- |
| public int update(String query) | Used to insert, update and delete records. |
| public int update(String query, Object… args) | Used to insert, update and delete records using **PreparedStatement**using given arguments. |
| public T execute(String sql, PreparedStatementCallback action) | Executes the query by using**PreparedStatementCallback.** |
| public void execute(String query) | Used to execute **DDL**query. |
| public T query(String sql, ResultSetExtractor result) | Used to fetch records using **ResultSetExtractor**. |

## XML

<bean id="ds"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName"

value="com.mysql.jdbc.Driver" />

<property name="url"

value="jdbc:mysql://localhost:3306/JPA?characterEncoding=latin1" />

<property name="username" value="root" />

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

</bean>

<bean id="jdbcTemplate"

class="org.springframework.jdbc.core.JdbcTemplate">

<property name="dataSource" ref="ds"></property>

</bean>

<bean id="studentdao" class="com.dao.StudentDAOImpl">

<property name="jdbcTemplateObject" ref="jdbcTemplate"></property>

</bean>

DriverManagerDataSource class will take care of connection creation.

JdbcTemplate **Will take care of executing the quires**

The Java Database Connectivity (JDBC API) provides universal data access from any data sources(including relational databases, spreadsheets & flat files). The **JdbcTemplate** is the most basic and classic approach for data access. The **NamedParameterJdbcTemplate** wraps the JdbcTemplate and allows the use of named parameters instead of the traditional **JDBC** **‘?’** placeholder.

<https://www.geeksforgeeks.org/spring-using-sql-scripts-with-spring-jdbc-jpa-hsqldb/?ref=lbp>

## Spring – ResultSetExtractor

<https://www.geeksforgeeks.org/spring-resultsetextractor/?ref=lbp>

**check for exceptions in JDBC template**

<https://www.baeldung.com/spring-jdbc-jdbctemplate>

# Sprng DataJPA

JPA is a Java specification(Jakarta Persistence API) and it manages relational data in Java applications. To access and persist data between Java object(Plain Old Java object)/ class and relational database, we can use JPA. Upon  Object-Relation Mapping (ORM), it follows the mechanisms. It has the runtime EntityManager API and it is responsible for processing queries and transactions on the Java objects against the database. The main highlight is it uses JPQL (Java Persistent Query Language) which is platform-independent. JPA mainly covers persistence in terms of

* The Java Persistence API
* Object-Relational metadata
* Moreover under the persistence package API is defined.
* We cannot say that JPA is a framework, but It defines a concept and it can be implemented by any framework.

### **Advantages of Using JPA**

* No need to write DDL/DML queries, instead we can map by using XML/annotations.
* JPQL is used and since it is platform-independent, we no need to depend on any native SQL table. Complex expressions and filtering expressions are all handled via JPQL only.
* Entity can be partially stored in one database like MySQL and the rest can be in Graph database Management System.
* Dynamic generation of queries is possible.
* Integration with Spring framework is easier with a custom namespace.

**Units comprised in JPA are available under javax persistence package:**

|  |  |
| --- | --- |
| **Persistence** | It has static methods to obtain an EntityManagerFactory instance |
| **EntityManagerFactory** | Factory class for EntityManager and responsible for managing multiple instances of EntityManager |
| **EntityManager** | It is an interface that works for the Query instance |
| **Entity** | They are persistent objects and stored as records in the database |
| **Persistence Unit** | Set of all entity classes |
| **EntityTransaction** | It has one-to-one relationship with EntityManager. |
| **Query** | To get relation objects that meet the criteria. |

Many have a confusion between JPA and HIbernate. Let us see few key differences

| **JPA** | **Hibernate** |
| --- | --- |
| JPA : It is a Java **specification**for mapping relational data in Java application. It is not a framework | Hibernate is an **ORM framework** and in that way data persistence is possible. |
| In JPA, no implementation classes are provided. | In Hibernate, implementation classes are provided. |
| Main advantage is It uses JPQL (Java Persistence Query Language)  and it is platform-independent query language. | Here it is using HQL (Hibernate Query Language). |
| It is available under javax.persistence package. | It is available under org.hibernate package. |
| In Hibernate, EclipseLink, etc. we can see its implementation. | Hibernate is the provider of JPA. |
| Persistence of data is handled by EntityManager. | Persistence of data is handled by Session. |

## Repository

It is a market interface.

## CrudRepository

This contains all the crude operations.

## PagingAndSortingRepository

On top of the CrudRepository there is a PagingAndSortingRepository abstraction that adds additional methods to ease paginated access to entities:

PagingAndSortingRepository<User, Long> repository = *// … get access to a bean*

Page<User> users = repository.findAll(**new** PageRequest(1, 20));

There are different type of repositories and have implementations for those repositories.

# **Spring Boot**

@SpringBootApplication

@SpringBootApplication annotation can be used to enable those three features

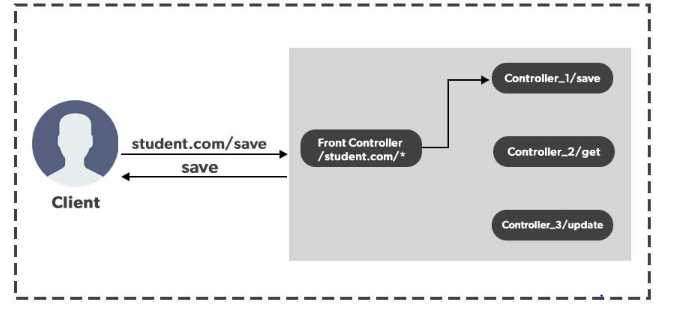
* @EnableAutoConfiguration: enable [Spring Boot’s auto-configuration mechanism](https://docs.spring.io/spring-boot/docs/2.0.x/reference/html/using-boot-auto-configuration.html)
* @ComponentScan: enable @Component scan on the package where the application is located
* @Configuration: allow to register extra beans in the context or import additional configuration classes

@Configuration annotation which indicates that the class has @Bean definition methods So Spring container can process the class and generate Spring Beans to be used in the application

In Spring beans are created in the bean.xml file.

# **Spring dispatcher**

**DispatcherServlet**acts as the Front Controller for Spring-based web applications. So now what is Front Controller? So it is pretty simple. Any request is going to come into our website the front controller is going to stand in front and is going to accept all the requests and once the front controller accepts that request then this is the job of the front controller that it will make a decision that who is the right controller to handle that request. For example, refer to the below image. Suppose we have a website called student.com and the client is making a request to save student data by hitting the following URL student.com/save, and its first comes to the front controller and once the front controller accepts that request it is going to assign to the Controller\_1 as this controller handle the request for /save operation. Then it is going to return back the response to the Client.



So now you might be thinking about how to create a front controller in a Spring MVC Application? But the good news is, the front controller is already created by the Spring Framework Developer, and the name of that particular controller is DispatcherServlet. You can use that front controller in your Spring MVC project. You really not required to create a front controller but you can reuse that front controller created by the Spring Framework Developer and they named it as DispatcherServlet.

[*DispatcherServlet*](https://www.geeksforgeeks.org/what-is-dispatcher-servlet-in-spring/)*handles an incoming****HttpRequest****, delegates the request, and processes that request according to the configured HandlerAdapter interfaces that have been implemented within the Spring application along with accompanying annotations specifying handlers, controller endpoints, and response objects.*

*Example :*

<https://www.geeksforgeeks.org/how-to-configure-dispatcher-servlet-in-web-xml-file/?ref=lbp>

# Spring dispatcher with out web.xml

<https://www.geeksforgeeks.org/spring-webapplicationinitializer-with-example/?ref=lbp>

# Remote Invoker

<https://www.geeksforgeeks.org/spring-remoting-by-http-invoker/?ref=lbp>

# Spring – Expression Language(SpEL)

https://www.geeksforgeeks.org/spring-expression-languagespel/?ref=lbp

# Spring Security

**Spring Security**is one of the projects by the Spring team that is built using the Spring framework in Java. This project aims to make it easy for developers to secure web applications against common exploits such as **Cross-Site Request Forgery** (CSRF) attacks. It contains code that can be customized or used as-is, depending on the use case. Its major function is to manage authentication and authorization at both the Web request and method invocation levels. Internally, the spring security framework contains a series of servlet filters that handle various aspects of security. Although it adheres to Spring’s set-up conventions, programmers may select between default provisions and modify them to their specific requirements. Spring security works on the following four core concepts

* **Authentication** – Is the user really who he claims to be?
* **Authorization** – Does the user have the appropriate role?
* **Password Storage** – How is the password stored? In Memory or a database.
* **Servlet Filters**– Are there any new filters that we need to add or just use the default ones provided by the spring team?