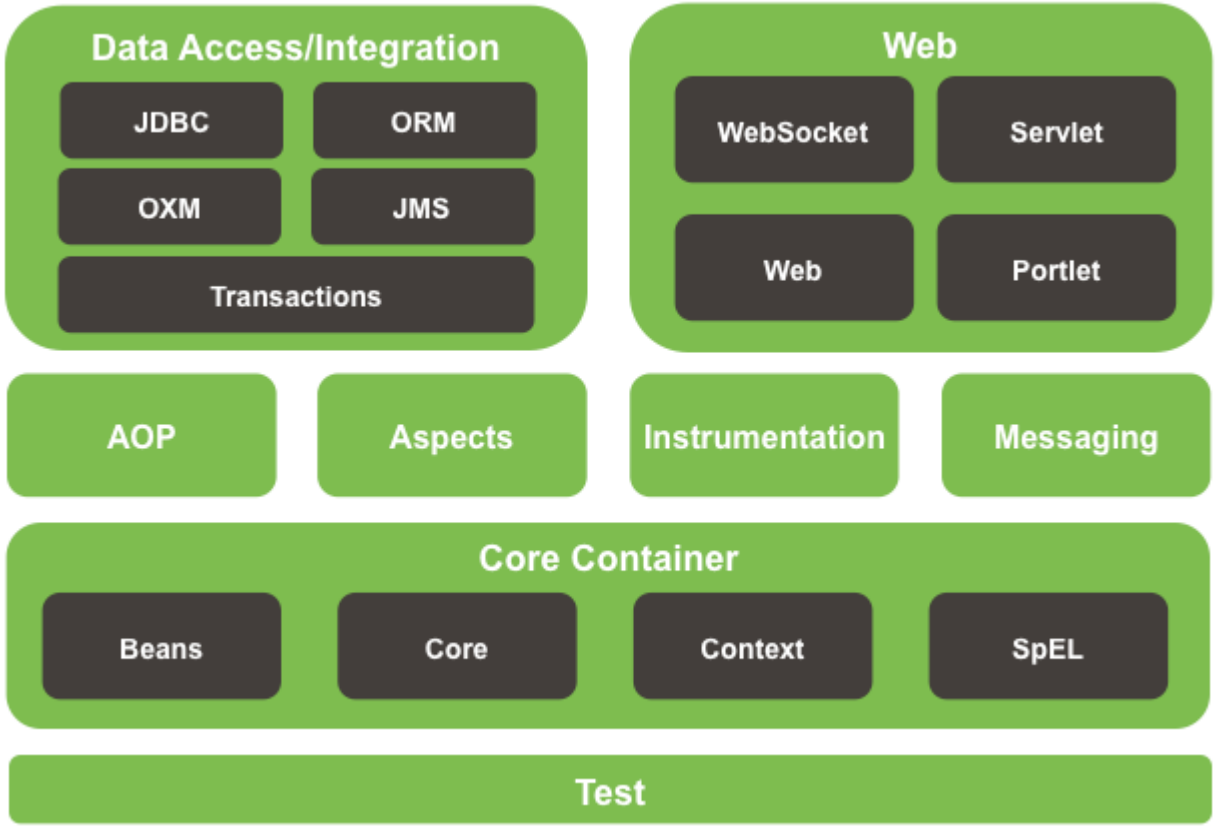
Spring JDBC

dasturlash.uz

Spring Modules

- There are more than 21 modules



- dasturlash.uz

Problems of JDBC API

- The problems of JDBC API are as follows:
  - We need to write a lot of code before and after executing the query, such as creating connection, statement, closing resultset, connection etc.
  - We need to perform exception handling code on the database logic.
  - We need to handle transaction.
  - Repetition of all these codes from one to another database logic is a time consuming task.
- JDBC API da quyidagi muommolari bor:
  - Biz query ni bajarishdan oldin va keyin juda ko'p kod yozishimiz kerak, masalan, ulanish, Statement , close ResultSet, Connection va hokazo.
  - Exception larni ushlash codlarini yozishimiz kerak.
  - Transaction larni ushlashimiz kerak
  - Shu ishlarni hargal bazada query jo’natish jarayonida takror va takro bo’ladi.

Spring JDBC Framework

- Spring JDBC Framework takes care of all the low-level details starting from opening the connection, preparing and executing the SQL statement, processing exceptions, handling transactions, and finally closing the connection.
- Spring JDBC Framework pastgi qatlamdagi Connection yaratish, SQL querylarni execute qilish, Exception larni ushlash, Transaction yaratish va Connection ni close qilish kabi past qatlamdagi kodlarni o’z zimmasiga oladi.
-

Spring JDBC Framework Dependency

- Spring-jdbc

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>5.3.20</version>
</dependency>
```

Spring JDBC Framework Dependency for Spring boot

- Spring JDBC dependency:
- **JDBC API**

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
```

  - Add Spring Boot dependence with name JDBC API

- Add dependency for database type. PostgreSql Example:

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```

Spring JDBC Framework

- Spring JDBC provides several approaches and correspondingly different classes to interface with the database.
- Speing JDBC database bilan ishlashning birnechta turlarini (class larni) taqdim etadi.

Spring JDBC Framework

- Spring framework provides following approaches for JDBC database access:
  - JdbcTemplate
  - NamedParameterJdbcTemplate
  - SimpleJdbcTemplate
  - SimpleJdbcInsert and SimpleJdbcCall
- Spring JDBC framework baza bila ulanishni quyidagi turlarini (classlarini) tagqim etadi.
  - JdbcTemplate
  - NamedParameterJdbcTemplate
  - SimpleJdbcTemplate
  - SimpleJdbcInsert and SimpleJdbcCall

JdbcTemplate

- Spring **JdbcTemplate** is a powerful mechanism to connect to the database and execute SQL queries.
- It internally uses JDBC api, but eliminates a lot of problems of JDBC API.

- Spring JdbcTemplate bu ma’lumotlar bazasiga ulanish va SQL so‘rovlarini bajarish uchun kuchli mexanizmdir.
- U ichki JDBC api-dan foydalanadi, lekin JDBC API-ning ko'plab muammolarini bartaraf qiladi.

JdbcTemplate class 1

- It is the central class in the Spring JDBC support classes. It takes care of creation and release of resources such as creating and closing of connection object etc. So it will not lead to any problem if you forget to close the connection.
- It handles the exception and provides the informative exception messages by the help of excepion classes defined in the **org.springframework.dao** package.
- We can perform all the database operations by the help of JdbcTemplate class such as insertion, updation, deletion and retrieval of the data from the database.
- JdbcTemplate bu Spring JDBC dagi eng muxim va markaziy class dir. U database bilan connection yaratish va uni close qilish ishlarini zimmasiga oladi.
- Sodir bo'ldigan Exceptionlarni o'zi ushlaydi.
- JdbcTemplate classi orqali Insert,Update,Delete,Select querylarni bajarsak bo'ladi.

JdbcTemplate class 2

- Instances of the JdbcTemplate class are threadsafe once configured. So, you can configure a single instance of a JDBC Template and then safely inject this shared reference into multiple DAOs.
- A common practice when using the JDBC Template class is to configure a DataSource in your Spring configuration file, and then dependency-inject that shared DataSource bean into your DAO classes. The JDBC Template is created in the setter for the DataSource.
- JdbcTemplate classi Threadsave bo'dir. Biz loyixada JdbcTemplate dan bitta ob'ekt yaratib uni xoxlagan DAO/Repository classlarimizda ishlatsak bo'ladi.
- Biz loyixada DataSource classini to'g'ri konfiguratsiya qilamiz va JdbcTemplate classi Databace xaqidagi malumotlarni DataSource classidan o'zi o'qib oladi.

JdbcTemplate methods

- public int update(String query)
  - is used to insert, update and delete records.
- public int update(String query,Object... args)
  - is used to insert, update and delete records using PreparedStatement using given arguments.
- public void execute(String query)
  - is used to execute DDL query.
- public T execute(String sql, PreparedStatementCallback action)
  - executes the query by using PreparedStatement callback.
- public T query(String sql, ResultSetExtractor rse)
  - is used to fetch records using ResultSetExtractor.
- public List query(String sql, RowMapper rse)
  - is used to fetch records using RowMapper.

JdbcTemplate methods 2

- public … queryForObject()
  - to query a single row record from database, and convert the row into an object via row mapper. Mapping can be done by SingleColumnRowMapper or Cutome rowMapper
- queryForList ()
  - it works, but not recommend, the mapping in Map may not same as the object, need casting.
- Yuqoridagi aytilgan metodlarni turli ko’rinishlari mavjut. Barchasini ko’rmaymiz. Eng ko’p ishlatiladiganini ko’ramiz.

Configuration

Spring JDBC Configure

- We can configure Spring JDBC in several ways.
  - Annotation based configuration
  - XML based configuration
  - application.properties file based configuration (only in Spring Boot)
-

Spring JDBC Annotation Configure 1

- PostgreSql

```
@Configuration
public class ApplicationConfig {
    @Bean
    public DataSource postgreSqlDataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("org.postgresql.Driver");
        dataSource.setUrl("jdbc:postgresql://localhost:5432/javaDb");
        dataSource.setUsername("javaDbUser");
        dataSource.setPassword("root");

        return dataSource;
    }
}
```

Spring JDBC Annotation Configure 2

- MySQL

```
@Configuration
public class ApplicationConfig {
    @Bean
    public DataSource mysqlDataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setDriverClassName("com.mysql.jdbc.Driver");
        dataSource.setUrl("jdbc:mysql://localhost:3306/springjdbc");
        dataSource.setUsername("guest_user");
        dataSource.setPassword("guest_password");

        return dataSource;
    }
}
```

Spring JDBC Xml configuration

- Postgrsql.

```
<bean id="myDataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="org.postgresql.Driver" />
      <property name="url" value="jdbc:postgresql://localhost:5432/dbname" />
      <property name="username" value="userName" />
      <property name="password" value="userPassword" />
   </bean>
```

Spring JDBC property file configuration

- PostgreSql

```
spring.datasource.url=jdbc:postgresql://localhost:5432/javaDb
spring.datasource.username=javaDbUser
spring.datasource.password=root
```

- MySQL

```
spring.datasource.url=jdbc:mysql://192.168.1.4:3306/test
spring.datasource.username=mkyong
spring.datasource.password=password
```

- Oracle

```
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:orcl
spring.datasource.username=system
spring.datasource.password=Password123
```

- Property based configuration exists only in Spring Boot projects

Spring JDBC Xml configuration

- MySql.

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
  destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/springjdbc"/>
    <property name="username" value="guest_user"/>
    <property name="password" value="guest_password"/>
</bean>
```

JdbcTemplate xml based configuration

```
<bean name="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>
```

- dataSource bean should present

JdbcTemplate annotation based configuration

- DataSource should present

```
@Bean
  public JdbcTemplate getTemplate() {
      JdbcTemplate jdbcTemplate = new JdbcTemplate();
      jdbcTemplate.setDataSource(dataSource());
      return jdbcTemplate;
  }
```

Misolchalar ko’ramiz

Prepare table

- .

```
create table if not exists student(
        id serial primary key,
        name varchar not null,
        surname varchar not null,
        phone varchar,
)
```

JdbcTemplate Autowiring

- Spring Automatically injects JdbcTemplate instance.
- Inside any component class (DAO/Repository).

```
@Autowired
  private JdbcTemplate jdbcTemplate;
```

JdbcTemplate Insert example

- Insert query

```
String sql = "INSERT INTO student (name,surname,phone) values('%s','%s','%s')";
 sql = String.format(sql, student.getName(), student.getSurname(), student.getPhone());

jdbcTemplate.update(sql);
```

JdbcTemplate Update example

- Update student

```
String sql = "Update student set name='%s', surname='%s', phone='%s' where id = %d";

sql = String.format(sql, student.getName(), student.getSurname(), student.getPhone(), id);

jdbcTemplate.update(sql);
```

JdbcTemplate QueryForObject : Single value

- Select single value (one column)

```
public int count() {
	return jdbcTemplate
		.queryForObject("select count(*) from student", Integer.class);
  }
```

JdbcTemplate Select example 1

- Get value

```
public Student get_1(Integer id) {
      String sql = "SELECT * FROM Student Where id =" + id;
      Student s = (Student) jdbcTemplate.queryForObject(sql,
            new BeanPropertyRowMapper(Student.class));
      return s;
  }
```

QueryForList

```
String sql = "Select title from book";
        return jdbcTemplate.queryForList(sql, String.class);
```

JdbcTemplate Select example 2

- Get List

```
String sql = "SELECT * FROM Student";
      List<Student> studentList = jdbcTemplate.query(sql,
                  new BeanPropertyRowMapper(Student.class));
```

RowMapper

- We can use RowMapper interface to fetch the records from the database using **query()** method of **JdbcTemplate** class.
- In the execute of we need to pass the instance of RowMapper now.
- **RowMapper** interface allows to map a row of the relations with the instance of user-defined class.
- It iterates the ResultSet internally and adds it into the collection. So we don't need to write a lot of code to fetch the records as ResultSetExtracto.
- RowMapper interface si har bitta row ni user belgilagan class tipiga o’g’irish uchun ishlatiladi.

RowMapper 1

- Advantage of RowMapper over ResultSetExtractor
  - RowMapper saves a lot of code becuase it internally adds the data of ResultSet into the collection.
- Method of RowMapper interface
  - It defines only one method mapRow that accepts ResultSet instance and int as the parameter list. Syntax of the method is given below:
  - **public** T mapRow(ResultSet rs, **int** rowNumber)**throws** SQLException

RowMapper Interface Example 1

```
public class StudentRowMapper implements RowMapper<Student> {
    public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
        Student student = new Student();
        student.setId(rs.getInt("id"));
        student.setName(rs.getString("name"));
        student.setSurname(rs.getString("surname"));
        student.setPhone(rs.getString("phone"));

        return student;
    }
}
```

RowMapper Interface Example 2

```
public Student get(Integer id) {
        String sql = "SELECT * FROM Student Where id =" + id;
        Student s = jdbcTemplate.queryForObject(sql, new StudentRowMapper());
        return s;
    }
```

RowMapper Interface Example 3

```
String sql = "SELECT * FROM Student";
        List<Student> studentList = jdbcTemplate.query(sql, new StudentRowMapper());
        return studentList;
```

ResultSetExtractor

- O’zingiz o’qib olding. Bu Mapper uchun edi.

PreparedStatement

- We can execute parameterized query using Spring JdbcTemplate by the help of **execute()** method of JdbcTemplate class.
- To use parameterized query, we pass the instance of **PreparedStatementCallback** in the execute method.
- Syntax of execute method to use parameterized query
  - **public** T execute(String sql,PreparedStatementCallback<T>);

PreparedStatementCallback interface

- It processes the input parameters and output results. In such case, you don't need to care about single and double quotes.

PreparedStatementCallback example 1

```
String sql = "INSERT INTO student (name,surname,phone) values(?,?,?)";
      jdbcTemplate.update(sql, new PreparedStatementCallback<Boolean>() {
         @Override
         public Boolean doInPreparedStatement(PreparedStatement ps) throws SQLException,
DataAccessException {
            ps.setString(1, student.getName());
            ps.setString(2, student.getSurname());
            ps.setString(3, student.getPhone());
            return ps.execute();
         }
      });
```

PreparedStatementCallback example 2

```
public Student getStudent(final Integer id) {
    final String SQL = "select * from Student where id = ? ";
    List <Student> students = jdbcTemplateObject.query(
      SQL, new PreparedStatementSetter() {

      public void setValues(PreparedStatement preparedStatement) throws SQLException {
        preparedStatement.setInt(1, id);
      }
    },
    new StudentMapper());
    return students.get(0);
  }
```

ResultSetExtractor

- **ResultSetExtractor** interface can be used to fetch records from the database. It accepts a ResultSet and returns the list.

- Method of ResultSetExtractor interface
  - It defines only one method extractData that accepts ResultSet instance as a parameter. Syntax of the method is given below:
  - **public** T extractData(ResultSet rs)**throws** SQLException,DataAccessException
-

ResultSetExtractor Example

```
String sql = "SELECT * FROM Student ";
      List<Student> s = jdbcTemplate.query(sql, new ResultSetExtractor<List<Student>>() {
          @Override
          public List<Student> extractData(ResultSet rs) throws SQLException, DataAccessException {
              List<Student> list = new ArrayList<Student>();
              while (rs.next()) {
                  Student e = new Student();
                  e.setId(rs.getInt(1));
                  e.setName(rs.getString(2));
                  e.setSurname(rs.getString(3));
                  e.setPhone(rs.getString(4));
                  list.add(e);
              }
              return list;
          }
      });
```

NamedParameterJdbcTemplate

NamedParameterJdbcTemplate

- Spring provides another way to insert data by named parameter.
- In such way, we use names instead of ?(question mark). So it is better to remember the data for the column.
- Simple example of named parameter query
  - insert into employee values (:id,:name,:salary)
  - Befor (insert into employee values (?,?,?))

- Method of NamedParameterJdbcTemplate class
  - In this example,we are going to call only the execute method of NamedParameterJdbcTemplate class. Syntax of the method is as follows:
  - pubic T execute(String sql,Map map,PreparedStatementCallback psc)

NamedParameterJdbcTemplate

```
public void create_named(Student student) {
        String sql = "INSERT INTO student (name,surname,phone) values(:name,:surname,:phone)";
        jdbcTemplate.update(sql);

        Map<String, Object> in = new HashMap();
        in.put("name", student.getName());
        in.put("surname", student.getSurname());
        in.put("phone", student.getPhone());

        namedParameterJdbcTemplate.update(sql, in);
    }
```

What else

- https://www.javatpoint.com/spring-JdbcTemplate-tutorial
- https://www.tutorialspoint.com/springjdbc/index.htm
- https://mkyong.com/spring-boot/spring-boot-jdbc-examples/