

Mon, Wednesday, Thursday => 6.30 to 9.00AM IST  
SpringDataJpa, SpringBootLogging(SL4J), SpringBootExceptionHandler(MVC)

SpringJDBC

```
|=> JdbcTemplate(C)
    |=> for Nonselect/DML sql queries
        a. public int update(String query)
    |=> for select query
        a. public xxxx queryXXXXX(,,)
```

Note:

While performing retrieval operation, to convert the data from ResultSet Object to Business Object, SpringJDBC

environment has provided a predefined interface in the form of "RowMapper".

```
public interface RowMapper<T>{
    public Object mapRow(ResultSet rs,int rowCount);
}
```

default implementation class is :: BeanPropertyRowMapper.

This class will take care of copying the record from ResultSet to B0.

Expectation : colnames in dbTable and B0 properties/fieldname should be same.

Working with ResultSetExtractor<T> callback interface

=====

=> if select query is executed which gives multiple records to process then we need to go for ResultSetExtractor/RowCallbackHandler

=> The best use case is getting List<B0> from RS after executing Select SQL query that gives multiple records.

```
public interface ResultSetExtractor<T> {
    T extractData(ResultSet rs) throws SQLException, DataAccessException;
}
```

default implementation class is :: RowMapperResultSetExtractor.

This class will take care of keeping the record into List<B0>.

default implementation class is :: BeanPropertyRowMapper.

This class will take care of copying the record from ResultSet to B0.

Expectation : colnames in dbTable and B0 properties/fieldname should be same.

```
eg::: return jdbcTemplate.query(GET_STUDENT_BY_CITY,
    new RowMapperResultSetExtractor<StudentB0>(new
BeanPropertyRowMapper<StudentB0>(StudentB0.class)),
    city1, city2, city3);
```

Working with RowCallbackHandler<T> callback interface

=====

@FunctionalInterface

```
public interface RowCallbackHandler {
    void processRow(ResultSet rs) throws SQLException;
}
```

=> A RowCallbackHandler object is typically stateful, it keeps the result state with in the object, to be available for

later inspection.

## ResultSetExtractor(I)

- => it is stateless in nature
- => extractData(,) call back method, but executes only once.
- => involves only one ResultSet
- => Good in performance
- => Support of Generics
- => Readymade implementation class is available.

## RowCallbackHandler(I)

- => it is stateful in nature
- => processRow(,) call back method, executes for multiple times.
- => Involves multiple ResultSet(n+1) in the entire process
- => Bad in performance
- => No support of generics
- => No ready made implementation class.

## NamedParameterJdbcTemplate

=====

It is similar to JdbcTemplate only, but it works with named parameter.

NamedParameterJdbcTemplate supports both positional(?) and named parameter(:)

eg: select empno,ename,job,sal from employee where empno>=? and empno<=?

(positional parameter)

select empno,ename,job,sal from employee where empno>=:no1 and empno<=:no2

(named parameter)

## Setting the value to NamedParameterJdbcTemplate

=====

a. using Map<String,Object> obj

b. Using SqlParameterSource(I) implementation

a. MapSqlParameterSource(c)

=> it uses addValue(,,) takes param name and value as the

arguments.

b. BeanPropertySqlParameterSource(c)

=> it allows to set javabean object values as the

namedparameter values

condition: propertyname and parameter name should

match.

## SimpleJdbcCall

=====

=> It is a multithreaded, reusable object representing a call to stored procedure.

=> It provides meta data for processing to simplify the code needed to access basic stored procedures.

=> All we need to do is provide the name of stored procedure and map containing the parameters when you execute the call.

=> The names of supplied parameters will be matched with IN and OUT parameters declared when the stored procedure is created.

## Storedprocedure

=====

```
CREATE DEFINER='root'@'localhost' PROCEDURE `P_GET_PRODUCT_BY_NAME`(IN name1  
VARCHAR(20), IN name2 VARCHAR(20))
```

```
BEGIN
```

```
SELECT pid,pname,price,qty FROM products WHERE pname IN (name1,name2);
```

```
END$$
```

```
DELIMITER ;
```

```

SimpleJdbcCall jdbc = new
SimpleJdbcCall(dataSource).withProcedureName("P_GET_PRODUCT_BY_NAME")
    .returningResultSet("products", new
BeanPropertyRowMapper<ProductBO>(ProductBO.class));
Map<String, Object> out = jdbc.execute(Map.of("name1", name1, "name2", name2));
List<ProductBO> listProducts = (List<ProductBO>) out.get("products");

```

Storedprocedure

=====

```

CREATE PROCEDURE `get_contact`(IN contact_id INTEGER,
    OUT _name varchar(45),
    OUT _email varchar(45),
    OUT _address varchar(45),
    OUT _phone varchar(45))

```

BEGIN

```

    SELECT name, email, address, telephone
    INTO _name, _email, _address, _phone
    FROM Contact WHERE id = contact_id;

```

END

```

SimpleJdbcCall actor = new
SimpleJdbcCall(dataSource).withProcedureName("get_contact");
SqlParameterSource inParams = new MapSqlParameterSource().addValue("contact_id",
contactId);
Map<String, Object> outParams = actor.execute(inParams);
String name = (String) outParams.get("_name");
String email = (String) outParams.get("_email");
String address = (String) outParams.get("_address");
String phone = (String) outParams.get("_phone");
System.out.println(name + ", " + email + ", " + address + ", " + phone);

```

OOPs

=====

ResultSet =====> DB table available

```
rs.getXXXX()
```

Map<K,V> =====> Result of StoredProcedur output params

```
m.get(Key)
```

