Spring Framework
Module 4 – DAO, JDBC

Evgeniy Krivosheev
Vyacheslav Yakovenko
Last update: Feb, 2012

Contents

- DAO Design Pattern
- JDBC Support in Spring Framework
- javax.sql.DataSource
- Configuring javax.sql.DataSource
- JdbcTemplate
- RowMapper
- JdbcDaoSupport
- Parameterized SQL queries
- JdbcTemplate :: Insert / Update / Delete
- JdbcTemplate :: Other SQL queries

Spring :: DAO Design Pattern

LUXOFT TRAINING

- Spring DAO is a module aimed at data handling;
- Makes it easy to work with such technologies as JDBC, Hibernate, JDO, etc.;
- Allows to switch between technologies fairly easy;
- Facilitates handling specific exceptions;

Spring :: DAO Design Pattern

- Spring translates all technology-specific exceptions such as SQLException to its own exception class hierarchy with the DataAccessException as the root exception;
- Spring can also wrap checked exceptions specific to Hibernate, JDO, and JPA, and convert them to runtime exceptions;

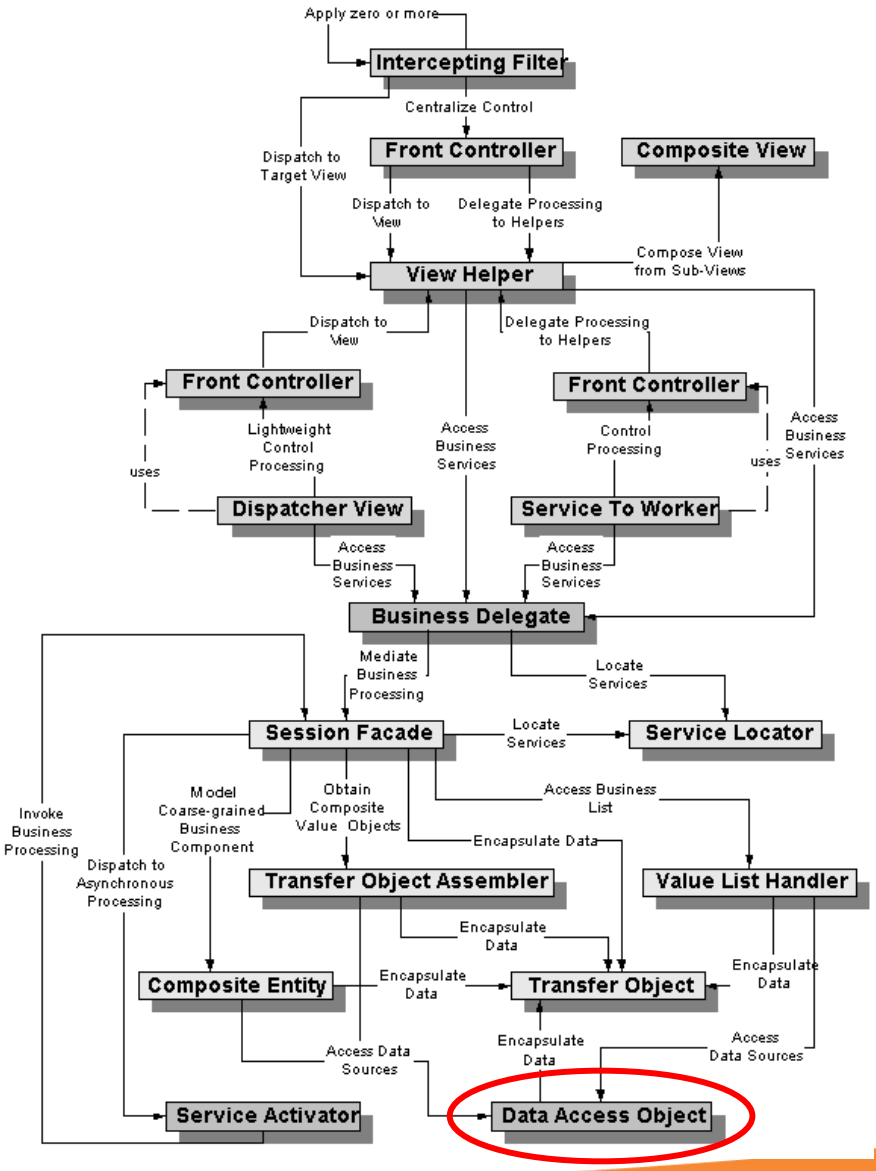
Spring :: DAO Design Pattern

LUXOFT TRAINING

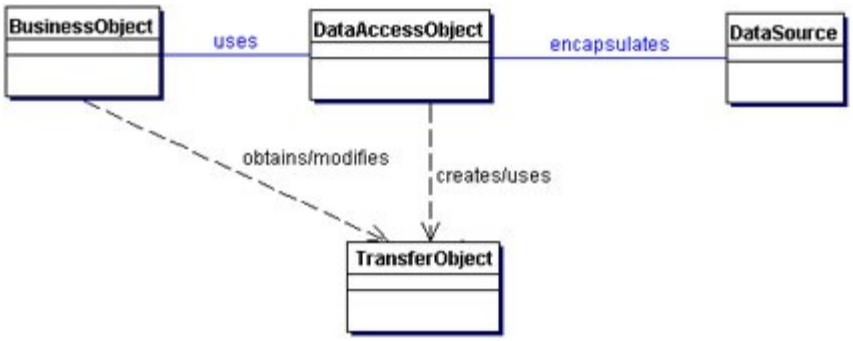
- Data structure design is abstracted from specific database;
- The code is simplified and business objects are explicit;
- Shipping from one DB (ORM, etc.) to another one is made easier;
- Data access mechanism is accumulated at separate level;

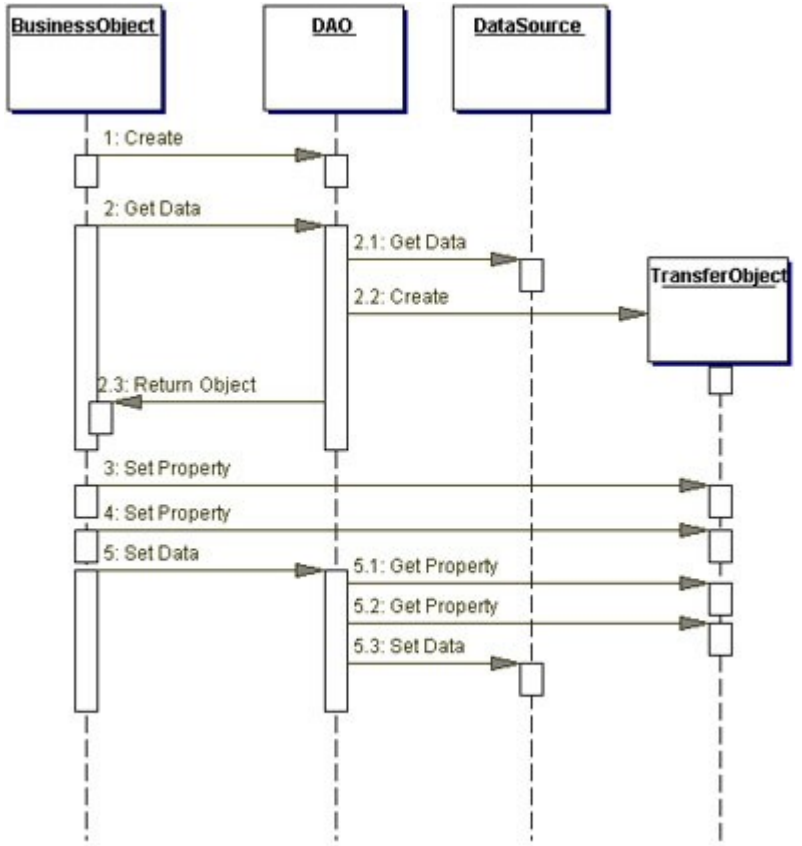
Spring :: DAO Design Pattern

- DAO Design Pattern can be found in catalog of JEE patterns: http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html. This is one of the most crucial patterns that is used to create application Persistence Layer;

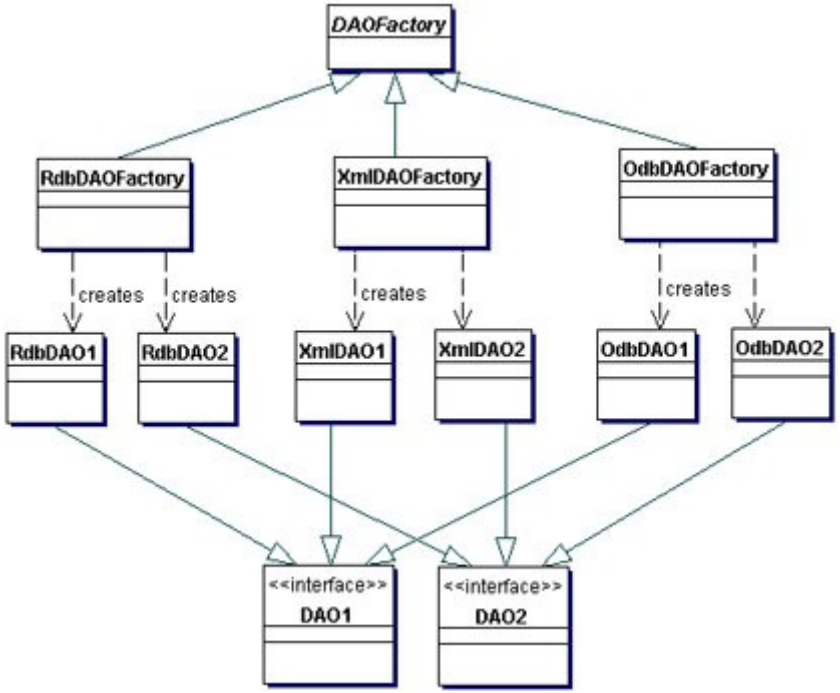


Spring :: DAO Design Pattern





Spring :: DAO Design Pattern



Spring :: JDBC Support

LUXOFT TRAINING

Without Spring:

- Define connection parameters;
- Open the connection;
- Specify the statement;
- Prepare and execute the statement;
- Iteration through the results;
- Do the work for each iteration;
- Process any exception;
- Handle transactions;
- Close the connection;

With Spring support:

- Specify the statement;
- Do the work for each iteration;

Spring :: JDBC Support

- As of Spring v.3.1 Java SE 7 features are supported as well as JDBC 4.1 features (try-with-resources);
- More detail:
  - http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html
  - http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/jdbc_41.html

Spring :: JDBC Support

Core classes for work with JDBC in Spring:

- **javax.sql.DataSource:** controls database connections;
- **JdbcTemplate** is a central class that control queries execution;
- **RowMapper**: controls mapping of each query row;
- **JdbcDaoSupport**: facilitates configuring and transferring parameters;

Spring :: javax.sql.DataSource

- **DataSource Interface** is a part of the **JDBC** specification that can be seen as connection factory;
- **Spring** connects the database via **DataSource;**
- **DataSource** allows to hide connection pooling and transaction management;

Spring :: Retrieving javax.sql.DataSource

LUXOFT TRAINING

- Configure its own:
  - Makes unit testing easier;
  - Web container is not required;
- Via JNDI;
- DataSource implementations:
  - Apache DBCP;
  - c3p0 is the most useful implementation;

Spring :: Configuring javax.sql.DataSource

```
<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/mydb"/>
    <property name="username" value="root"/>
    <property name="password" value="masterkaoli"/>
</bean>
```

OR

```
<jee:jndi-lookup id="dataSource"
        jndi-name="java:comp/env/jdbc/datasource"/>
```

Spring :: Configuring javax.sql.DataSource

In prototyping and testing, Spring allows moving embedded database up in context (HSQLDB / H2 / Derby). HSQLDB is used by default.

```
<jdbc:embedded-database id="dataSource" />
```

или

```
<jdbc:embedded-database id="dataSource">
    <jdbc:script location="schema.sql"/>
    <jdbc:script location="test-data.sql"/>
</jdbc:embedded-database>
```

Spring :: JdbcTemplate

JdbcTemplate is the central class in the package **org.springframework.jdbc.core**:

- Executes SQL queries;
- Iterates over results;
- Catches JDBC exceptions;

Parameters necessary when executing SQL query:

- **DataSource**;
- **RowMapper**;
- SQL query row;

Spring :: JdbcTemplate

- Instance of **JdbcTemplate** class is threadsafe;
- Can be configured only once and then be used in various DAO;
- **DataSource** is needed to create **JdbcTemplate**;
- Generally, **DataSource** is transferred to DAO and then to **JdbcTemplate**;

Spring :: RowMapper

- Interface from **org.springframework.jdbc.core**;
- It is implemented through **ResultSet** mapping in specific objects;
- Describes operations for each **ResultSet** row;
- Used in **query()** method from **JdbcTemplate** or for results of stored procedure;

Spring :: JdbcTemplate Example

- Create tables and business objects;
- Configure DataSource;
- Create DAO class;
- Transfer DataSource to DAO;
- Implement RowMapper;
- Create the JdbcTemplate instance;
- Transfer DataSource there;
- Invoke query() method;
- Parameters: SQL query and RowMapper;

Spring :: JdbcTemplate Example

LUXOFT TRAINING

```
<bean id="countryDao" class="jdbc.CountryDao">
    <constructor-arg ref="dataSource"/>
</bean>
```

```
public class CountryDao {
    private DataSource dataSource;
    public CountryDao(DataSource dataSource) {
        this.dataSource = dataSource;
    }
    public List getCountryList() {
        JdbcTemplate jdbcTemplate =
                        new JdbcTemplate(dataSource);
        return jdbcTemplate.query(
                        "select * from country",
                        new CountryRowMapper());
    }
}
```

Spring :: JdbcTemplate Example

```
public class CountryRowMapper implements RowMapper {
    public Object mapRow(ResultSet resultSet, int i)
            throws SQLException {
        Country country = new Country();
        country.setId(resultSet.getInt("id"));
        country.setName(resultSet.getString("name"));
        return country;
    }
}
```

Spring :: JdbcTemplate Example

- If invoking **countryDao.getCountryList()** method, a list of **Country** type object is obtained.

Spring :: JdbcDaoSupport

- DAO classes can inherit from **JdbcDaoSupport**;
- In this case **setDataSource(..)** method will be already implemented;
- **JdbcDaoSupport** facilitates working with **DataSource** and hides how JdbcTemplate is created;

Spring :: JdbcDaoSupport

```
<bean id="countryDao" class="dao.CountryDao">
    <property name="dataSource" ref="dataSource"/>
</bean>
```

```
public class CountryDao extends JdbcDaoSupport {
    public List getCountryList() {
        JdbcTemplate jdbcTemplate
                = getJdbcTemplate();
        return jdbcTemplate.query(
                "select * from country",
                new CountryRowMapper());
}
```

Spring :: Parameterized SQL queries

LUXOFT TRAINING

- Created using **NamedParameterJdbcTemplate**;
- Configured exactly as **JdbcTemplate;**
- **Note!** SimpleJdbcTemplate in Spring 3.1 is deprecated!

Spring :: NamedParameterJdbcTemplate

```
NamedParameterJdbcTemplate namedParameterJdbcTemplate =
        new NamedParameterJdbcTemplate(dataSource);

String sql = "select * from country" +
        " where name = :country_name";
Map namedParameters =
        Collections.singletonMap(
                "country_name", "Australia");
namedParameterJdbcTemplate.
        queryForInt(sql, namedParameters);
```

Spring :: ParameterizedRowMapper

```
ParameterizedRowMapper<Country> mapper =
        new ParameterizedRowMapper<Country>() {
            public Country mapRow(ResultSet rs, int rowNum)
                    throws SQLException {
                Country country = new Country();
                country.setId(rs.getInt("id"));
                country.setName(rs.getString("name"));
                return country;
            }
        };
return jdbcTemplate.queryForObject(
        "select * from country where id = ?",
        mapper, id);
```

Spring :: JdbcTemplate :: Insert

- Insert, Update and Delete are executed in the same way;
- The only difference is SQL query;

```
jdbcTemplate.update(
        "insert into country (id, name) values (?, ?)",
        new Object[]{12, "Watling"});
```

Spring :: JdbcTemplate :: Other SQL queries

- *Execute* method from JdbcTemplate can be used when executing any SQL query:

```
jdbcTemplate.execute(
        "create table mytable (" +
                "id integer, " +
                "name varchar(100))");
```

Exercises

LUXOFT TRAINING

№:6 : Using JDBC in Spring when handling data

- 45 min for practice;
- 15 min for discussion;

Any questions!?