Spring Framework
Module 4 – DAO, JDBC

Evgeniy Krivosheev
Vyacheslav Yakovenko
Last update: Feb, 2012

Содержание

- Шаблон проектирования DAO
- Поддержка JDBC в Spring Framework
- javax.sql.DataSource
- Конфигурирование javax.sql.DataSource
- JdbcTemplate
- RowMapper
- JdbcDaoSupport
- Параметризированные SQL- запросы
- JdbcTemplate :: Insert / Update / Delete
- JdbcTemplate :: Другие SQL запросы

Spring :: Шаблон проектирования DAO

- Spring DAO – модуль для работы с данными;
- Упрощает использование таких технологий как JDBC, Hibernate, JDO, etc.;
- Позволяет относительно просто переключаться с одной технологии на другую;
- Упрощает обработку специфичных исключений;

Spring :: Шаблон проектирования DAO

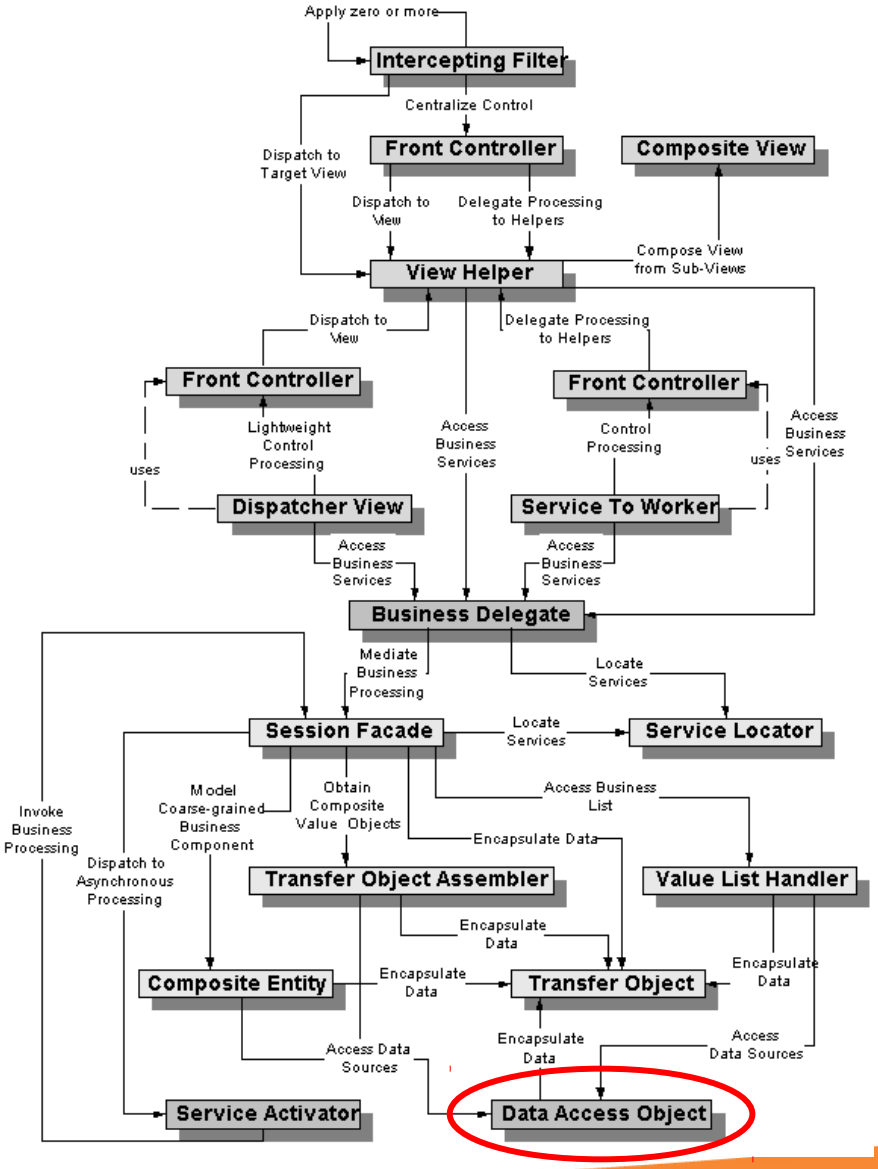
- Spring оборачивает все исключения, специфичные для конкретной технологии (например, SQLException), в свою иерархию исключений, где корнем является DataAccessException;
- Также для Hibernate, JDO и JPA, Spring оборачивает проверяемые исключение и делает из них исключения времени выполнения;

Spring :: Шаблон проектирования DAO

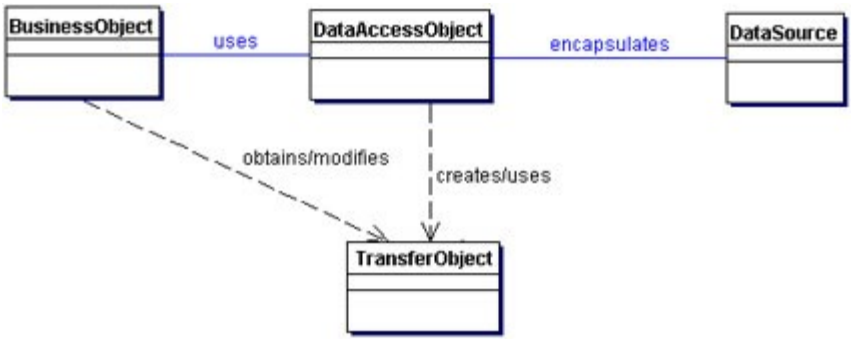
- Абстракция для разработки структуры данных не зависимо от конкретной БД;
- Упрощает код и обеспечивает прозрачность бизнес - объектов;
- Облегчает переносимость с одной БД (ORM, etc.) на другую;
- Концентрирует механизм доступа к данным на отдельном уровне;

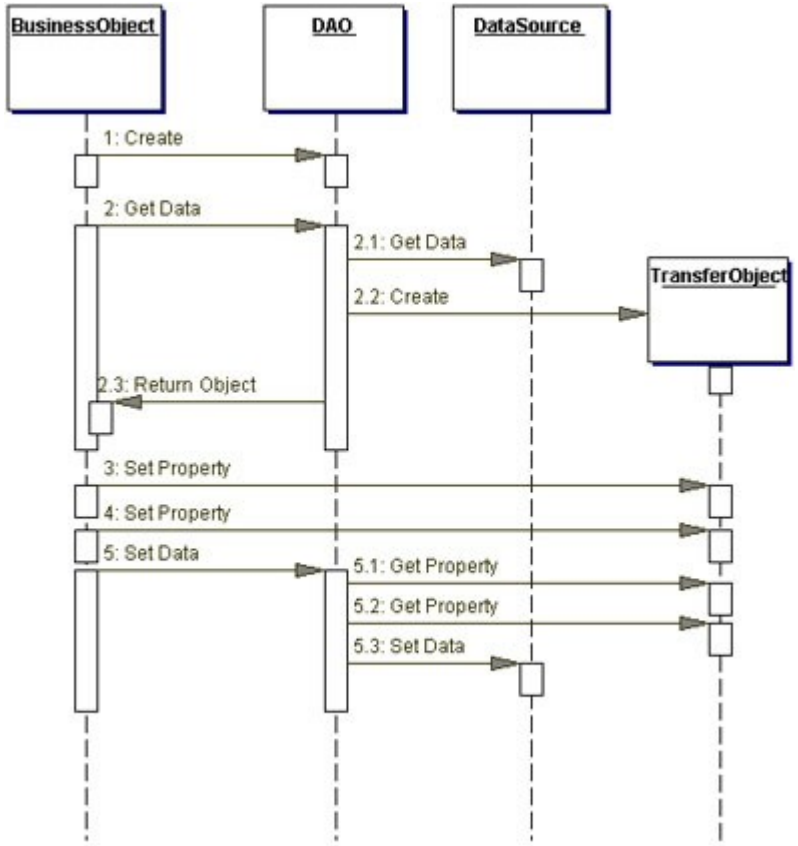
Spring :: Шаблон проектирования DAO

- Сам шаблон проектирования DAO представлен в каталоге JEE – шаблонов: http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html и является одним из ключевых шаблонов, позволяющих организовать “Persistence Layer” приложения;

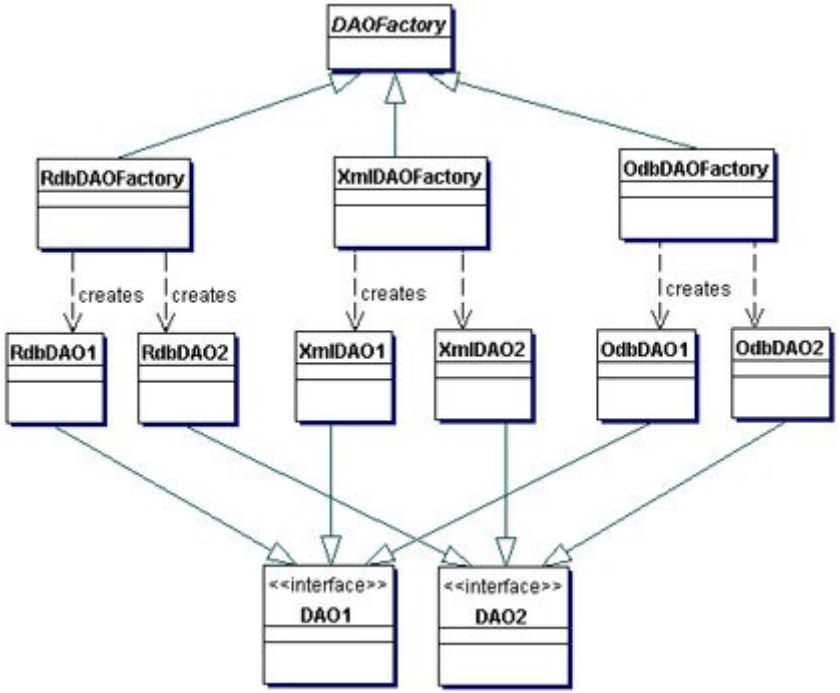


Spring :: Шаблон проектирования DAO





Spring :: Шаблон проектирования DAO



Spring :: Поддержка JDBC

LUXOFT TRAINING

Без Spring:

- Задание параметров соединения;
- Открытие соединения;
- Определение запроса;
- Подготовка и выполнение запроса;
- Итерация по результатам;
- Действие для каждой итерации;
- Обработка исключений;
- Обработка транзакции;
- Закрытие соединения;

При поддержке Spring:

- Определение запроса;
- Выполнение работы для каждой итерации;

Spring :: Поддержка JDBC

- Начиная с v.3.1 Spring, поддерживает возможности Java SE 7, а вместе с ними и возможности JDBC 4.1 (try-with-resources);
- Подробнее:
  - http://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html
  - http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/jdbc_41.html

Spring :: Поддержка JDBC

Основные классы для работы с JDBC в Spring:

- **javax.sql.DataSource** - Отвечает за соединение с БД;
- **JdbcTemplate** - Основной класс, отвечает за весь процесс выполнения запросов;
- **RowMapper** - Отвечает за mapping каждой строки запроса;
- **JdbcDaoSupport** - Упрощает конфигурацию и передачу параметров;

Spring :: javax.sql.DataSource

LUXOFT TRAINING

- Интерфейс **DataSource** – часть **JDBC** спецификации – фабрика соединений;
- **Spring** получает соединение с базой через **DataSource;**
- **DataSource** позволяет спрятать получение соединений и управление транзакциями;

Spring :: Получение javax.sql.DataSource

LUXOFT TRAINING

- Сконфигурировать свой:
  - Это упрощает unit-тестирование;
  - Не нужен веб-контейнер;
- Через JNDI;
- Реализации DataSource:
  - Apache DBCP;
  - c3p0 – наиболее удобная реализация;

Spring :: Конфигурирование 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>
```

ИЛИ

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

Spring :: Конфигурирование javax.sql.DataSource

Для прототипирования и тестирования Spring предоставляет возможность поднятия в контексте встроенных БД (HSQLDB / H2 / Derby). По умолчанию используется HSQLDB.

```
<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 - основной класс из пакета **org.springframework.jdbc.core**:

- Выполняет SQL-запросы;
- Итерируется по результатам;
- Отлавливает JDBC исключения;

Параметры, необходимые для выполнения SQL-запроса:

- **DataSource**;
- **RowMapper**;
- Строка SQL-запроса;

Spring :: JdbcTemplate

LUXOFT TRAINING

- Экземпляр класса **JdbcTemplate** является потокобезопасным;
- Можно конфигурировать его один раз, а потом использовать во многих DAO;
- Для создания **JdbcTemplate** необходим **DataSource**;
- Обычно **DataSource** передается в DAO, а оттуда в **JdbcTemplate**;

Spring :: RowMapper

- Интерфейс из **org.springframework.jdbc.core**;
- Его реализации выполняют mapping **ResultSet**'а в конкретные объекты;
- Описывает операции для каждой строки **ResultSet**'а;
- Используется в методе **query()** из **JdbcTemplate** или для результатов хранимых процедур;

Spring :: JdbcTemplate - пример

- Создаем таблицы и бизнес-объекты;
- Конфигурируем DataSource;
- Создаем класс DAO;
- Передаем DataSource в DAO;
- Реализуем RowMapper;
- Создаем экземпляр JdbcTemplate;
- Передаем ему DataSource;
- Вызываем метод query();
- Параметры – SQL-запрос и RowMapper;

Spring :: JdbcTemplate - пример

```
<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 - пример

```
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 - пример

- При вызове метода **countryDao.getCountryList()** получаем список объектов типа **Country**.

Spring :: JdbcDaoSupport

- Можно DAO классы делать наследниками от **JdbcDaoSupport**;
- В этом случае метод **setDataSource(..)** уже будет реализован;
- **JdbcDaoSupport** упрощает работу с **DataSource** и «прячет» создание JdbcTemplate;

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 :: Параметризированные SQL-запросы

- Создаются с использованием **NamedParameterJdbcTemplate**;
- Конфигурация такая же как для **JdbcTemplate;**
- **Внимание!** SimpleJdbcTemplate, в Spring 3.1 - 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

LUXOFT TRAINING

```
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 и Delete выполняются аналогично;
- Отличия только в SQL-запросе;

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

Spring :: JdbcTemplate :: Другие SQL запросы

- Для выполнения любого SQL-запроса может быть использован метод execute из JdbcTemplate:

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

Упражнения

LUXOFT TRAINING

№:6 : «Использование JDBC в Spring при работе с данными»

- 45 мин – самостоятельная работа;
- 15 мин – обсуждение;

LUXOFT TRAINING

Вопросы!?