

# Spring Framework

## - 데이터베이스 연동

## ■ CONTENTS

- JDBC 프로그래밍의 단점을 보완하는 스프링
- DataSource 설정
- JdbcTemplate을 이용한 쿼리 실행
- 트랜잭션 처리
- 스프링의 익셉션

## ■ 스프링의 데이터베이스 연동 지원

기존의 JDBC → 개선된 형태

→ JPA

→ mybatis

- 스프링은 JDBC, 하이버네이트, iBATIS 등의 다양한 기술을 이용해서 손쉽게 DAO 클래스를 구현할 수 있도록 지원  
↳ DB framework. 실증/구현
- Spring은 JDBC를 비롯하여 ORM 프레임워크를 직접적으로 지원하고 있기 때문에 simple하게 JDBC뿐만 아니라 ORM 프레임워크들과의 연동도 매우 쉬움
- Spring은 JDBC, ORM 프레임워크 등의 다양한 기술을 이용해서 손쉽게 DAO클래스를 구현할 수 있도록 지원

### Spring 제공

- 템플릿 클래스를 통한 데이터 접근 지원
- 의미 있는 예외 클래스 제공
- 트랜잭션 처리

## ■ 스프링의 데이터베이스 연동 지원

- 데이터 베이스 연동을 위한 템플릿 클래스

- 데이터에 접근하는 코드는 Connection 생성, PreparedStatement, ResultSet 등 거의 동일한 코드 구성을 갖는다.
- 스프링은 데이터베이스 연동을 위한 템플릿 클래스를 제공함으로써 개발자가 중복된 코드를 입력 해야 하는 성가신 작업을 줄일 수 있도록 돕는다.
- JDBC 뿐 아니라, iBATIS, JMS와 같은 다양한 기술에 대해 템플릿 클래스를 지원하고 있다.
- **Template 클래스들**
  - **JDBC : JdbcTemplate**
  - **iBatis : SqlMapClientTemplate**
  - **Hibernate : HibernateTemplate**

## ■ 스프링의 데이터베이스 연동 지원

### • 스프링의 예외지원

- 스프링은 데이터베이스 처리과정에서 발생하는 예외가 왜 발생을 했는지 좀더 구체적으로 확인 할 수 있도록 하기 위해, 데이터 베이스처리와 관련된 예외클래스를 지원하고 있다.
- JdbcTemplate 클래스는 처리과정에서 SQLException이 발생하면 스프링이 제공하는 예외 클래스 중 알맞은 예외클래스로 변환해서 발생 시킨다.
- JdbcTemplate 뿐만 아니라 SqlMapClientTempate과 같이 스프링이 제공하는 템플릿 클래스는 내부적으로 발생하는 예외클래스를 스프링이 제공하는 예외클래스로 알맞게 변환해서 예외를 발생 시킨다.
- 스프링이 제공하는 템플릿 클래스를 사용하면 데이터베이스 연동을 위해 사용하는 기술에 상관 없이 동일한 방식으로 예외를 처리 할 수 있다.
- Spring의 모든 예외 클래스들은 DataAccessException을 상속  
ex) BadSqlGrammerException, DataRetrievalFailureException

# ■ JDBC 프로그래밍의 단점을 보완하는 스프링

- JDBC 프로그래밍

: 항상 반복적으로 사용하는 코드가 존재

✂ 반복코드를 줄이기 위해 템플릿 메서드 패턴과 전략 패턴을 사용해서 JdbcTemplate 클래스를 제공.

- 간단한 트랜잭션 처리

트랜잭션의 경우 필요한 메서드 위에 @Transactional 애노테이션 사용  
커밋과 롤백을 자동으로 처리하기 때문에, 핵심코드에만 집중가능.

- JdbcTemplate 클래스

- SQL을 실행 하기 위한 메서드를 제공
- 데이터 조회, 삽입, 수정, 삭제를 위한 SQL 쿼리를 실행 할 수 있다.
- try~catch~finally 블록 및 커넥션 관리를 위한 중복 코드 삭제로 인한 코드량 감소 및 개발용이

## ■ DB 생성

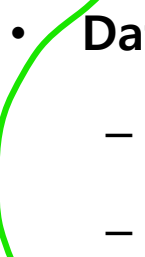
- Mysql 에서 스키마 생성 : board
- member 테이블 DDL

```
create table board.MEMBER (  
    ID int auto_increment primary key,  
    EMAIL varchar(255),  
    PASSWORD varchar(100),  
    NAME varchar(100),  
    REGDATE datetime,  
    unique key (EMAIL)  
)
```

## ■ DataSource 설정 데이터베이스 연결 구현한 설정

- 스프링이 제공하는 DB 연동 기능들은 DataSource를 사용해서 DB Connection을 구하도록 구현되어 있다. 

Ex) DB연동에 사용할 DataSource를 스프링 빈으로 등록하고, DB 연동 기능을 구현한 빈 객체(DAO)는 DataSource를 주입 받아 사용.

- DataSource 를 제공하는 모듈은 아래 두 가지가 있다.
    - c3p0
    - dbcp
- 



## ■ DataSource 설정

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  
    <dependency>  
        <groupId>mysql</groupId>  
        <artifactId>mysql-connector-java</artifactId>  
        <version>5.1.44</version>  
    </dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-pool2 -->  
    <dependency>  
        <groupId>org.apache.commons</groupId>  
        <artifactId>commons-pool2</artifactId>  
        <version>2.4.2</version>  
    </dependency>
```

```
<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->  
    <dependency>  
        <groupId>org.apache.commons</groupId>  
        <artifactId>commons-dbcp2</artifactId>  
        <version>2.1.1</version>  
    </dependency>
```

## ■ DataSource 설정

<bean

id="dataSource"

class="org.apache.commons.dbcp2.BasicDataSource"

p:driverClassName="com.mysql.jdbc.Driver"

✓ p:url="jdbc:mysql://localhost:3306/**board**?characterEncoding=utf8"

p:username="bit"

p:password="bit" />

## ■ DataSource 설정

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.44</version>
</dependency>

<dependency>
    <groupId>com.mchange</groupId>
    <artifactId>c3p0</artifactId>
    <version>0.9.2.1</version>
</dependency>
```

## ■ DataSource 설정

```
<bean
    id="dataSource"
    class="com.mchange.v2.c3p0.ComboPooledDataSource">
    <property name="driverClass" value="com.mysql.jdbc.Driver" />
    <property
        name="jdbcUrl"
        value="jdbc:mysql://localhost/board?characterEncoding=utf8" />
    <property name="user" value="bit" />
    <property name="password" value="bit" />
</bean>
```

## ■ JdbcTemplate을 이용한 쿼리 실행

- JdbcTemplate 템플릿 생성 1

```
public class MemberDao {  
  
    private JdbcTemplate jdbcTemplate;  
  
    public MemberDao(DataSource dataSource) {  
        this.jdbcTemplate = new JdbcTemplate(dataSource);  
    }  
  
}
```

```
<bean id="memberDao" class="spring.MemberDao">  
    <constructor-arg ref="dataSource" />  
</bean>
```

## ■ JdbcTemplate을 이용한 쿼리 실행

- JdbcTemplate 템플릿 생성 2

```
public class MemberDao {  
    private JdbcTemplate jdbcTemplate;  
    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {  
        this.jdbcTemplate = jdbcTemplate;  
    }  
}
```

```
<bean  
    id="jdbcTemplate"  
    class="org.springframework.jdbc.core.JdbcTemplate"  
    p:dataSource-ref="dataSource" />  
  
<bean  
    id="dao"  
    class="com.bitcamp.memberboard.member.dao.MemberDao" />
```

## ■ JdbcTemplate을 이용한 쿼리 실행

- JdbcTemplate 템플릿 생성 3

```
public class MemberDao {  
    @Autowired  
    private JdbcTemplate jdbcTemplate;  
  
}
```

```
<bean  
    id="jdbcTemplate"  
    class="org.springframework.jdbc.core.JdbcTemplate"  
    p:dataSource-ref="dataSource" />  
  
<bean  
    id="dao"  
    class="com.bitcamp.memberboard.member.dao.MemberDao" />
```

## ■ JdbcTemplate을 이용한 쿼리 실행

- JdbcTemplate 템플릿을 이용한 조회 쿼리 실행

### query()

- List<T> query(String sql, RowMapper<T> rowMapper)
- List<T> query(String sql, Object[] args, RowMapper<T> rowMapper)
- List<T> query(String sql, RowMapper<T> rowMapper, Object... args)

객체 변환 매핑

- sql 파라미터로 전달받은 쿼리를 실행하고, RowMapper를 이용해서 ResultSet의 결과를 자바 객체로 변환한다.
- sql 파라미터가 인덱스 기반 파라미터(PreparedStatement의 물음표)를 가진 쿼리인 경우, args 파라미터를 이용해서 각 인덱스 파라미터 값을 지정.



## ■ JdbcTemplate을 이용한 쿼리 실행

- RowMapper<T> 인터페이스

```
package org.springframework.jdbc.core
```

```
Public interface RowMapper<T>{
```

```
    T mapRow(ResultSet rs, int rowNum) throws SQLException;
```

```
}
```

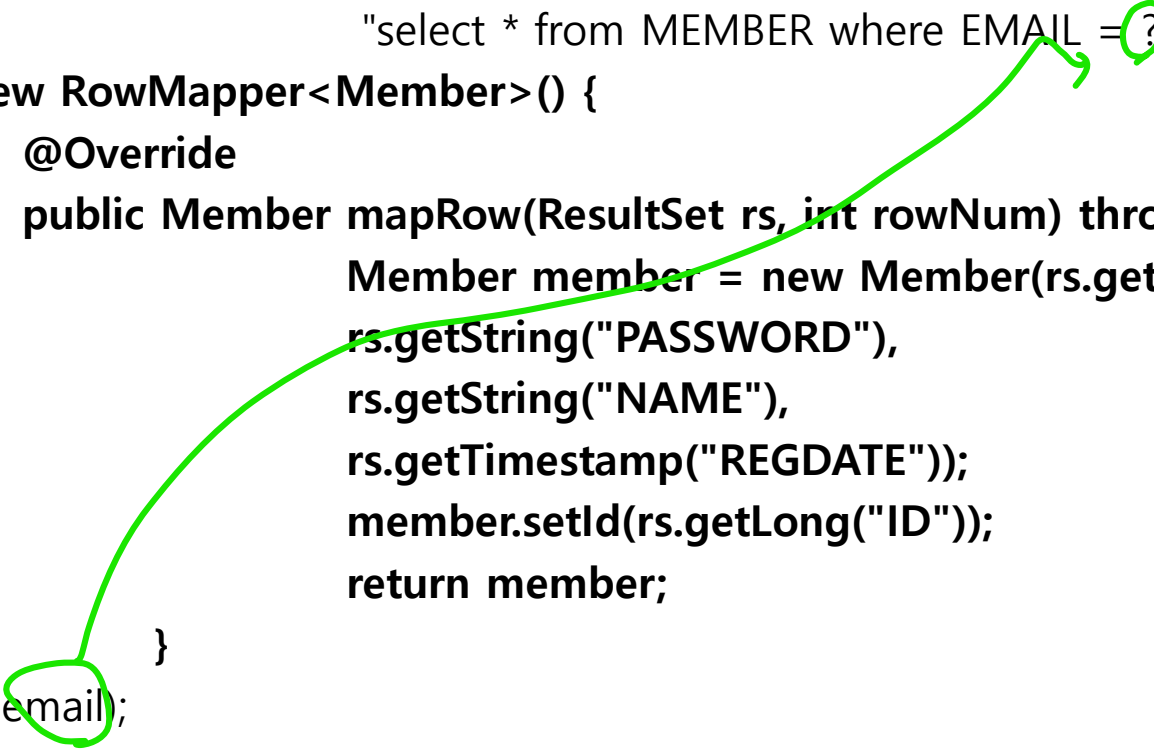
## ■ JdbcTemplate을 이용한 쿼리 실행

- RowMapper<T> 인터페이스 구현 클래스

```
new RowMapper<Member>() {  
    @Override  
    public Member mapRow(ResultSet rs, int rowNum) throws SQLException {  
        Member member = new Member(rs.getString("EMAIL"),  
            rs.getString("PASSWORD"),  
            rs.getString("NAME"),  
            rs.getTimestamp("REGDATE"));  
        member.setId(rs.getLong("ID"));  
        return member;  
    }  
}
```

## ■ Dao 구현

```
public class MemberDao {  
    private JdbcTemplate jdbcTemplate;  
    public MemberDao(DataSource dataSource) {  
        this.jdbcTemplate = new JdbcTemplate(dataSource);  
    }  
    public Member selectByEmail(String email) {  
        List<Member> results = jdbcTemplate.query(  
            "select * from MEMBER where EMAIL = ?",  
            new RowMapper<Member>() {  
                @Override  
                public Member mapRow(ResultSet rs, int rowNum) throws SQLException {  
                    Member member = new Member(rs.getString("EMAIL"),  
                    rs.getString("PASSWORD"),  
                    rs.getString("NAME"),  
                    rs.getTimestamp("REGDATE"));  
                    member.setId(rs.getLong("ID"));  
                    return member;  
                }  
            }, email);  
        return results.isEmpty() ? null : results.get(0);  
    }  
}
```



```
package spring;
```

```
import java.sql.ResultSet;
```

```
import java.sql.SQLException;
```

```
import org.springframework.jdbc.core.RowMapper;
```

```
public class MemberRowMapper implements RowMapper<Member> {
```

```
    @Override
```

```
    public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
```

```
        Member member = new Member(rs.getString("EMAIL"),
```

```
                                    rs.getString("PASSWORD"),
```

```
                                    rs.getString("NAME"),
```

```
                                    rs.getTimestamp("REGDATE"));
```

```
        member.setId(rs.getLong("ID"));
```

```
        return member;
```

```
    }
```

```
}
```

## ■ Dao 구현

```
public class MemberDao {  
    private JdbcTemplate jdbcTemplate;  
    public MemberDao(DataSource dataSource) {  
        this.jdbcTemplate = new JdbcTemplate(dataSource);  
    }  
    public Member selectByEmail(String email) {  
        List<Member> results = jdbcTemplate.query(  
            "select * from MEMBER where EMAIL = ?",  
            new MemberRowMapper() , email);  
        return results.isEmpty() ? null : results.get(0);  
    }  
}
```

## ■ JdbcTemplate을 이용한 쿼리 실행

- 결과가 1생인 경우의 조회 메서드
  - queryForObject(쿼리, 결과 타입 또는 로우맵퍼객체, 매개변수,...)
  - Dao 구현

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

```
public int count() {  
    String sql = "select count(*) from MEMBER";  
    Integer count = jdbcTemplate.queryForObject(sql, new MemberRowMapper());  
    return count;  
}
```

## ■ JdbcTemplate을 이용한 변경 쿼리 실행

- Insert, Update, Delete 쿼리를 실행할 때는 아래 메서드를 사용.
  - `int update(String sql)`
  - `int update(String sql, Object ... args)`

```
public void update(Member member) {  
    String sql="update MEMBER set NAME = ?, PASSWORD = ? where EMAIL = ?";  
    jdbcTemplate.update(sql, member.getName(), member.getPassword(),  
                        member.getEmail());  
}
```

## ■ JdbcTemplate을 이용한 변경 쿼리 실행

- **List<T> query(PreparedStatementCreator psc, RowMapper<T> rowMapper)**
- **int update(PreparedStatementCreator psc)**
- **int update(PreparedStatementCreator psc, KeyHolder generatedKeyHolder)**
  - aut\_increment로 설정한 컬럼은 행이 추가되면 자동으로 값이 할당되는 컬럼.
  - 쿼리 실행 후 자동 생성된 키 값을 알기 위해 KeyHolder 사용



## ■ JdbcTemplate을 이용한 변경 쿼리 실행

- PreparedStatementCreator를 이용한 쿼리
  - 직접 인덱스 파라미터를 설정해야 할 경우 사용

```
public interface PreparedStatementCreator
```

```
//이 인터페이스는 다음과 같은 하나의 메소드를 제공한다.
```

```
public PreparedStatement createPreparedStatement(Connection con)  
throws SQLException;
```

## ■ JdbcTemplate을 이용한 변경 쿼리 실행

```
jdbcTemplate.update( new PreparedStatementCreator() {  
    @Override  
    public PreparedStatement createPreparedStatement(Connection con)  
        throws SQLException {  
        // 파라미터로 전달 받은 Connection을 이용해서 PreparedStatement  
        PreparedStatement pstmt = con.prepareStatement(  
            "insert into MEMBER (EMAIL, PASSWORD, NAME, REGDATE) "+  
                "values (?, ?, ?, ?)", new String[] {"ID"});  
        pstmt.setString(1, member.getEmail());  
        pstmt.setString(2, member.getPassword());  
        pstmt.setString(3, member.getName());  
        pstmt.setTimestamp(4, new Timestamp(member.getRegisterDate().getTime()));  
        return pstmt;  
    }  
});
```

## ■ JdbcTemplate을 이용한 변경 쿼리 실행

- `List<T> query(PreparedStatementCreator psc, RowMapper<T> rowMapper)`
- `int update(PreparedStatementCreator psc)`
- `int update(PreparedStatementCreator psc, KeyHolder generatedKeyHolder)`
  - `auto_increment`로 설정한 컬럼은 행이 추가되면 자동으로 값이 할당되는 컬럼.
  - 쿼리 실행 후 자동 생성된 키 값을 알기 위해 `KeyHolder` 사용

## ■ JdbcTemplate을 이용한 변경 쿼리 실행

```
public void insert(final Member member) {  
    KeyHolder keyHolder = new GeneratedKeyHolder();  
    jdbcTemplate.update(new PreparedStatementCreator() {  
        @Override  
        public PreparedStatement createPreparedStatement(Connection con)  
            throws SQLException {  
            PreparedStatement pstmt = con.prepareStatement(  
                "insert into MEMBER (EMAIL, PASSWORD, NAME, REGDATE) "+  
                "values (?, ?, ?, ?)", new String[] {"ID"});  
            pstmt.setString(1, member.getEmail());  
            pstmt.setString(2, member.getPassword());  
            pstmt.setString(3, member.getName());  
            pstmt.setTimestamp(4, new Timestamp(member.getRegisterDate().getTime()));  
            return pstmt;  
        }  
    }, keyHolder);  
    Number keyValue = keyHolder.getKey();  
    member.setId(keyValue.longValue());  
}
```

## ■ DAO 완성

```
import java.sql.*;
import java.util.List;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.support.GeneratedKeyHolder;
import org.springframework.jdbc.support.KeyHolder;

public class MemberDao {

    private JdbcTemplate jdbcTemplate;

    public MemberDao(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }
}
```

## ■ DAO 완성

```
public Member selectByEmail(String email) {
    List<Member> results = jdbcTemplate.query(
        "select * from MEMBER where EMAIL = ?",
        new RowMapper<Member>() {
            @Override
            public Member mapRow(ResultSet rs, int rowNum)
                throws SQLException {
                Member member = new Member(
                    rs.getString("EMAIL"),
                    rs.getString("PASSWORD"),
                    rs.getString("NAME"),
                    rs.getTimestamp("REGDATE"));
                member.setId(rs.getLong("ID"));
            }
            return member;
        }, email);
    return results.isEmpty() ? null : results.get(0);
}
```

## ■ DAO 완성

```
public void insert(final Member member) {
    KeyHolder keyHolder = new GeneratedKeyHolder();
    jdbcTemplate.update(new PreparedStatementCreator() {
        @Override
        public PreparedStatement createPreparedStatement(Connection con)
            throws SQLException {
            PreparedStatement pstmt = con.prepareStatement(
                "insert into MEMBER (EMAIL, PASSWORD, NAME, REGDATE) "+
                "values (?, ?, ?, ?)", new String[] {"ID"});
            pstmt.setString(1, member.getEmail());
            pstmt.setString(2, member.getPassword());
            pstmt.setString(3, member.getName());
            pstmt.setTimestamp(4, new
                Timestamp(member.getRegisterDate().getTime()));
            return pstmt;
        }
    }, keyHolder);
    Number keyValue = keyHolder.getKey();
    member.setId(keyValue.longValue());
}
```

## ■ DAO 완성

```
public void update(Member member) {
    jdbcTemplate.update(
        "update MEMBER set NAME = ?, PASSWORD = ? where EMAIL = ?",
        member.getName(), member.getPassword(), member.getEmail());
}

public List<Member> selectAll() {
    List<Member> results = jdbcTemplate.query("select * from MEMBER",
        new RowMapper<Member>() {
            @Override
            public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
                Member member = new Member(rs.getString("EMAIL"),
                    rs.getString("PASSWORD"),
                    rs.getString("NAME"),
                    rs.getTimestamp("REGDATE"));
                member.setId(rs.getLong("ID"));
                return member;
            }
        });
    return results;
}
```



## ■ DAO 완성

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

## ■ @Transactional을 이용한 트랜잭션 처리

- 트랜잭션 범위에서 실행하고 싶은 메서드에 **@Transactional** 애노테이션만 붙이면 된다.
- 정상적인 처리를 위해서 아래 두 가지 설정을 해주어야 한다.
  - PlatformTransactionManager 빈 설정
  - @Transactional 애노테이션 활성화 설정

```
<!-- PlatformTransactionManager 빈 설정 -->
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- @Transactional 애노테이션 활성화 설정 -->
<tx:annotation-driven transaction-manager="transactionManager"/>
```

## ■ DB 연동 관련 예외

- ✓ Mysql 서버에 연결할 권한이 없는 경우
- ✓ DB가 실행 중이 아니거나, 방화벽에 막혀 있는 경우, DB 자체에 대한 네트워크 연결을 못할 경우
- ✓ Sql문법 오류, 공백 문자 누락

## ■ 로그 메시지

- **Log4j**

: 로그 메시지를 남기기 위해 사용되는 로깅 프레임워크

- **Pom.xml 에 의존 추가**

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
  <type>bundle</type>
</dependency>
```

- 로그 메시지를 어떤 형식으로 어디에 기록할지에 대한 정보를 설정 파일로 부터  
읽어옴.

DEBUG : 디버그  
INFO : 정보  
WARN : 경고  
ERROR : 에러  
FATAL : 심각한 오류

-----  
순서는 DEBUG > INFO > WARN > ERROR > FATAL 이며,

# ■ 로그 메시지

## Log4j.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/Log4j/">
  <appender name="console" class="org.apache.Log4j.ConsoleAppender">
    <layout class="org.apache.Log4j.PatternLayout">
      <param name="ConversionPattern"
value="[%t] [%d{yyyy-MM-dd HH:mm:ss}] %-5p %c:%M - %m%n" />
    </layout>
  </appender>

  <root>
    <priority value="INFO" />
    <appender-ref ref="console" />
  </root>

  <logger name="org.springframework.jdbc">
    <level value="DEBUG" />
  </logger>
</log4j:configuration>
```