Spring Framework

11. Spring JDBC 개요

Tacademy

CONTENTS

학습 목표

- 데이터 액세스 공통 개념에 대하여 이해할 수 있습니다.
- Spring JDBC 개요에 대하여 이해할 수 있습니다.
- Spring JDBC의 JdbcTemplate 클래스에 대하여 이해할 수 있습니다.

1. 데이터 액세스 공통 개념

DAO(Data Access Object) 패턴

- 데이터 액세스 계층은 DAO 패턴을 적용하여 **비즈니스 로직과 데이터 액세스 로직을 분리**하는 것이 원칙이다.
- 비즈니스 로직이 없거나 단순하면 DAO와 서비스 계층을 통합 할 수도 있지만 의미 있는 비즈니스 로직을 가진 엔터프라이즈 애플리케이션이라면 데이터 액세스 계층을 DAO 패턴으로 분리해야 한다.
- DAO패턴은 서비스계층에 영향을 주지 않고 **데이터 액세스 기술을 변경**할 수 있는 장점을 가지고 있다.

컨넥션 풀링을 지원하는 DataSource

컨넥션 풀링은 미리 정해진 개수만큼의 **DB 컨넥션을 풀(Pool)에 준비**해두고, 애플리케이션이 **요청할 때마다 Pool에서 꺼내서 하나씩 할당**해주고 다시 **돌려받아서 Pool에 넣는 식의 기법**이다.

- 다중 사용자를 갖는 엔터프라이즈 시스템에서라면 반드시 DB 컨넥션 풀링 기능을 지원하는 DataSource를 사용해야 한다.
- Spring에서는 DataSource를 공유 가능한 Spring Bean으로 등록해 주어 사용할 수 있도록 해준다.

DataSource 구현 클래스 종류

❖테스트환경을 위한 DataSource

SimpleDriverDataSource

- Spring이 제공하는 **가장 단순한 DataSource 구현 클래스**이다.
- getConnection()을 호출할 때마다 매번 DB 컨넥션을 새로 만들고 따로 풀(pool)을 관리하지 않으므로 단순한 테스트용으로만 사용해야 한다.

SingleConnectionDriverDataSource

- **순차적으로 진행되는 통합 테스트에서는 사용** 가능하다.
- 매번 DB 커넥션을 생성하지 않기 때문에 SimpleDriverDataSource 보다 빠르게 동작한다.

DataSource 종류

❖오픈소스 DataSource

Apache Commons DBCP

- 가장 유명한 오픈소스 DB 커넥션 풀(pool) 라이브러리이다.
- Apache의 Commons 프로젝트(http://commons.apache.org/dbcp/)

c3p0 JDBC/DataSource Resource Pool

- c3p0는 JDBC 3.0 스펙을 준수하는 Connection과 Statement 풀(pool)을 제공하는 라이브러리이다.
- c3p0 웹 사이트(http://www.mchange.com/projects/c3p0/)

* 두 가지 모두 수정자(setter) 메서드를 제공하므로 Spring Bean으로 등록해서 사용하기 편리하다.

2. Spring JDBC 개요

JDBC란?

JDBC는 모든 자바의 데이터 액세스 기술의 근간이 된다.
엔티티 클래스와 애노테이션을 이용하는 최신 ORM 기술도
내부적으로는 DB와의 연동을 위해 JDBC를 이용한다.

- 안정적이고 유연한 기술이지만, 로우 레벨 기술로 인식되고 있다.
- 간단한 SQL을 실행하는 데도 중복된 코드가 반복적으로 사용되며, DB에 따라 일관성 없는 정보를 가진 채로 Checked Exception으로 처리한다.

장점	단점
대부분의 개발자가 잘 알고 있는 친숙한 데이터 액세스 기술로 별도의 학습 없이 개발이 가능하다.	Connection과 같은 공유 리소스를 제대로 릴리즈 해주지 않으면 시스템의 자원이 바닥나는 버그를 발생시킨다.

Spring JDBC란?

JDBC의 장점과 단순성을 그대로 유지하면서도 기존 JDBC의 단점을 극복할 수 있게 해주고, 간결한 형태의 API 사용법을 제공하며, JDBC API에서 지원되지 않는 편리한 기능을 제공한다.

- Spring JDBC는 반복적으로 해야 하는 많은 작업들을 대신 해준다.
- Spring JDBC를 사용할 때는 실행할 SQL과 바인딩 할 파라미터를 넘겨 주거나, 쿼리의 실행 결과를 어떤 객체에 넘겨 받을지를 지정하는 것만 하면 된다.
- Spring JDBC를 사용하려면 먼저, DB 컨넥션을 가져오는 DataSource를 Bean으로 등록해야 한다.

Spring JDBC가 해주는 작업(1)

Connection 열기와 닫기

- Connection과 관련된 모든 작업을 Spring JDBC가 필요한 시점에서 알아서 진행한다.
- 진행 중에 예외가 발생했을 때도 열린 모든 Connection 객체를 닫아준다.

Statement 준비와 닫기

- SQL 정보가 담긴 Statement 또는 PreparedStatement를 생성하고 필요한 준비 작업을 해주는 것도 Spring JDBC가 한다.
- Statement도 Connection과 마찬가지로 사용이 끝나고 나면 Spring JDBC가 알아서 객체를 닫아준다.

Spring JDBC가 해주는 작업(2)

Statement 실행

- SQL 담긴 Statement를 실행하는 것도 Spring JDBC가 해준다.
- Statement의 실행결과는 다양한 형태로 가져올 수 있다.

ResultSet Loop 처리

- ResultSet에 담긴 쿼리 실행 결과가 한 건 이상이면 ResultSet 루프를 만들어서 반복해주는 것도 Spring JDBC가 해주는 작업이다.

Spring JDBC가 해주는 작업(3)

Exception 처리와 반환

- JDBC 작업 중 발생하는 모든 예외는 Spring JDBC 예외 변환기가 처리한다.
- 체크 예외(Checked Exception)인 SQLException을 런타임 예외 (Runtime Exception)인 DataAccessException 타입으로 변환한다.

Transaction 처리

- Spring JDBC를 사용하면 transaction과 관련된 모든 작업에 대해서는 신경 쓰지 않아도 된다.

3. Spring JDBC의 JdbcTemplate 클래스

JdbcTemplate 클래스

Spring JDBC가 제공하는 클래스 중 JdbcTemplate은 JDBC의 모든 기능을 최대한 활용할 수 있는 **유연성**을 제공하는 클래스이다.

- JdbcTemplate이 제공하는 기능은 **실행, 조회, 배치**의 세 가지 작업이다.

  - **실행** : Insert나 Update같이 DB의 데이터에 변경이 일어나는 쿼리를 수행하는 작업
  - **조회** : Select를 이용해 데이터를 조회하는 작업
  - **배치** : 여러 개의 쿼리를 한 번에 수행해야 하는 작업

JdbcTemplate 클래스 생성

- JdbcTemplate은 DataSource를 파라미터로 받아서 아래와 같이 생성할 수 있다.

```
JdbcTemplate template = new JdbcTemplate(dataSource);
```

- DataSource는 보통 Bean으로 등록해서 사용하므로 JdbcTemplate이 필요한 DAO 클래스에서 DataSource Bean을 DI(의존관계 주입) 받아서 JdbcTemplate을 생성할 때 인자로 넘겨주면 된다.
- JdbcTemplate은 멀티스레드 환경에서도 안전하게 공유해서 쓸 수 있기 때문에 DAO클래스의 인스턴스 변수에 저장해 두고 사용할 수 있다.

JdbcTemplate 클래스 생성 Code

- 아래의 코드는 일반적으로 사용되는 DAO 클래스의 기본구조이다. DataSource에 대한 수정자 메서드에서 직접 JdbcTemplate 객체를 생성해준다.

```
public class UserDAOJdbc {
	JdbcTemplate jdbcTemplate;

	@Autowired
	public void setDataSource(DataSource dataSource) {
		jdbTemplate = new JdbcTemplate(dataSource);
	}
……
```

JdbcTemplate 클래스의 update() 메서드

- INSERT, UPDATE, DELETE와 같은 SQL을 실행할 때는 JdbcTemplate의 update() 메서드를 사용한다.

```
int update(String sql, [SQL 파라미터])
```

- update() 메서드를 호출할 때는 SQL과 함께 바인딩 할 파라미터는 Object 타입 가변인자 (Object … args)를 사용할 수 있다.
- update() 메서드의 리턴되는 값은 SQL 실행으로 영향을 받은 레코드의 개수를 리턴한다.

JdbcTemplate 클래스의 update() 메서드 Code

```
public int update(User user) {
    StringBuffer updateQuery = new StringBuffer();
    updateQuery.append("UPDATE USERS SET ");
    updateQuery.append("password=?, name=? ");
    updateQuery.append("WHERE id=? ");

    int result = this.jdbcTemplate.update(updateQuery.toString(),
    user.getName(),user.getPassword(),user.getId() );

    return result;
}
```

JdbcTemplate 클래스의 queryForObject() 메서드

- SELECT SQL을 실행하여 하나의 Row를 가져올 때는 JdbcTemplate의 queryForObject() 메서드를 사용한다.

```
<T> T queryForObject(String sql, [SQL 파라미터],
RowMapper<T> rm)
```

- SQL 실행 결과는 여러 개의 칼럼(Column)을 가진 하나의 로우(Row)
- T는 VO 객체의 타입에 해당된다.
- SQL 실행 결과로 돌아온 여러 개의 column을 가진 한 개의 Row를 RowMapper 콜백을 이용해 VO 객체로 매핑 해준다.

JdbcTemplate 클래스의 queryForObject() 메서드 Code

```
public User findUser (String id) {
    return this.jdbcTemplate.queryForObject("select * from users
       where id=?",new Object[] {id},
       new RowMapper<User>() {
           public User mapRow(ResultSet rs, int rowNum)
                  throws SQLException{
              User user = new User();
              user.setId(rs.getString("id"));
              user.setName(rs.getString("name"));
              user.setPassword(rs.getString("password"));
              return user;
           }
       }//RowMapper
     );//queryForObject
}//findUser
```

JdbcTemplate 클래스의 query() 메서드

- SELECT SQL을 실행하여 여러 개의 Row를 가져올 때는 JdbcTemplate의 query() 메서드를 사용한다.

```
<T> List<T> query(String sql, [SQL 파라미터],
RowMapper<T> rm)
```

- SQL 실행 결과로 돌아온 여러 개의 column을 가진 여러 개의 Row를 RowMapper 콜백을 이용해 VO 객체로 매핑 해준다.
- 결과 값은 매핑 한 VO 객체를 포함하고 있는 List 형태로 받는다. List의 각 요소가 하나의 Row에 해당된다.

학습정리

지금까지 [Spring JDBC 개요]에 대해서 살펴보았습니다.

데이터 액세스 공통개념

DAO 패턴, 컨넥션 풀링, DataSource

Spring JDBC 개요

- JDBC 개요 및 장단점
- Spring JDBC 개요 및 역할

Spring JDBC 의 JdbcTemplate 클래스

JdbcTemplate 클래스의 update(), queryForObject(), query() 메서드