Mybatis를 통한 DB 연동하기

한국기술교육대학교
온라인평생교육원

학습내용

- Mybatis 알아보기
- Transaction
- 동적 SQL

학습목표

- Mybatis에 대해 설명할 수 있다.
- 데이터베이스 Transaction에 대해 설명할 수 있다.
- 동적 SQL을 구성하는 방법을 파악할 수 있다.

Mybatis 알아보기

1 Mybatis란

iBatis

- JDBC를 대체하는 프레임워크(Persistence Framework)
- SQL에 기반한 데이터베이스와 자바, 닷넷(.NET), 루비(Ruby) 등을 연결시켜 주는 역할
- 소스코드에서 SQL 문장을 분리하여 별도의 XML 파일로 저장하고 이 둘을 연결시켜주는 방식으로 작동

Mybatis

1. 아파치 소프트웨어 재단의 iBatis 개발자 팀이 구글 코드로 이전하기로 결정하고 구글 코드에서 새로 만들어지는 이름이 MyBatis로 변경됨
2. MyBatis는 개발자가 지정한 SQL, 저장프로시저, 그리고 몇 가지 고급 매핑을 지원하는 영속성 프레임워크
3. MyBatis는 JDBC 코드와 수동으로 세팅하는 parameter와 결과 매핑을 제거
4. MyBatis는 데이터베이스 레코드에 원시타입과 Map 인터페이스, 자바 Object를 설정하고 매핑하기 위해 XML과 annotation을 사용할 수 있음

Mybatis 알아보기

2 Mybatis 설정 및 활용

《1》 pom.xml에 dependency 추가

```
<!- Spring jdbc -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>4.2.0.RELEASE</version>
  </dependency>
  <!-- Apache Commons -->
  <dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.4</version>
  </dependency>
  <!-- MySQL Connector -->
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.34</version>
  </dependency>
<!-- MyBatis -->
  <dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.2.8</version>
  </dependency>
  <dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.2.2</version>
  </dependency>
```

Mybatis 알아보기

2 Mybatis 설정 및 활용

《2》 /src/resources/common에 mybatis.xml 추가

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <settings>
    <setting name="mapUnderscoreToCamelCase" value="true" />
  </settings>
</configuration>
```

《3》 spring 설정 파일에 다음과 같이 추가

```
 <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close">
      <property name="driverClassName" value="${jdbc.driverClassName}" />
      <property name="url" value="${jdbc.url}" />
      <property name="username" value="${jdbc.username}" />
      <property name="password" value="${jdbc.password}" />
   </bean>

   <!-- MyBatis -->
   <mybatis:scan base-package="com.spring.repository" />
   <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
      <property name="dataSource" ref="dataSource" />
      <property name="configLocation" value="classpath:mybatis.xml" />
   </bean>
```

Mybatis 알아보기

2 Mybatis 설정 및 활용

《4》 config.properties에 다음과 같이 추가(DataSource정보)

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url =
jdbc:mysql://localhost:3306/spring?characterEncoding=utf8&useUnicode=tr
ue&mysqlEncoding=utf8
jdbc.username = DB 아이디
jdbc.password = DB 비밀번호
```

《5》 기본 문법

문법	설명
@Insert	Mysql Insert문
@Update	Mysql Update문
@Select	Mysql Select문
@Delete	Mysql Delete문
@SelectKey	▪ Statement : 방금 삽입한 ID를 가져오기 위한 문구 ▪ KeyProperty : ID 프로퍼티 ▪ ResultType : 결과값 형태
@Param	▪ 매퍼 메소드가 여러 개의 Parameter를 가진다면, annotation은 이름에 일치하는 매퍼 메소드 Parameter에 적용 ▪ @Param(“person”)를 사용하면, Parameter는 #{person}으로 명명

Mybatis 알아보기

2 Mybatis 설정 및 활용

6 예시(UserMapper)

```
package com.spring.repository;


import com.spring.domain.User;
import org.apache.ibatis.annotations.*;
import org.springframework.stereotype.Component;
import org.springframework.stereotype.Repository;

@Repository
public interface UserMapper {
    @Insert("INSERT INTO USERS (NAME, EMAIL, PASSWORD, AGE) VALUES
(#{name}, #{email}, #{password}, #{age})")
    @SelectKey(statement = "SELECT LAST_INSERT_ID()", keyProperty = "id",
before=false, resultType = int.class)
    void insert(User user);
 @Update("UPDATE USERS SET NAME = #{name}, EMAIL = #{email},
PASSWORD = #{password}, AGE = #{age} WHERE ID = #{id}")
    void update(User user);

    @Select("SELECT * FROM USERS WHERE ID = #{id}")
    User findOne(@Param("id") int id);

    @Delete("DELETE FROM USERS WHERE ID = #{id}")
    void delete(@Param("id") int id);

}
```

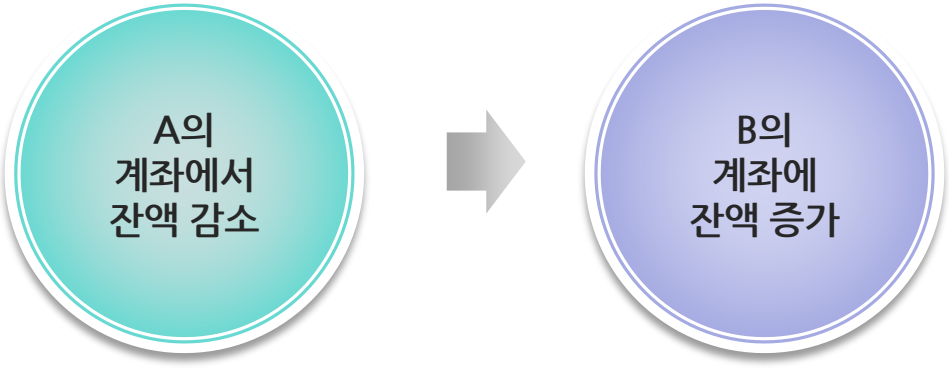
Transaction

1 Transaction이란

Transaction

- 데이터베이스 내에서 한꺼번에 수행되어야 할 일련의 연산들
- DB와 JAVA 언어가 데이터를 주고 받는 과정에 원자성을 부여하는 수단
- 트랜잭션의 모든 연산들은 한꺼번에 모두 완료되거나 그렇지 않으면 모두 취소되어야 함

예를 들어, A가 B에게 금액을 이체하는 상황을 살펴보면?



이 과정 사이에서 오류가 발생한다면?

Transaction의 원리

- Transaction기술은 하나의 커넥션에서 자동으로 Commit을 하던 것을 해제하고 수동으로 제어하는 기술
- 즉, 이용자가 직접 쿼리 결과를 컨트롤할 수 있는 기술

커밋(Commit)	해당 Connection의 요청을 완료하고 특별한 에러가 없다면 결과를 DB에 반영
롤백(RollBack)	해당 Connection 수행 중 예기치 않은 에러 발생 시 모든 과정을 취소하고 DB를 Connection이 수행되기 이전 상태로 변경

Transaction

2 @Transactional 적용

⟨1⟩ @Transactional annotation

스프링에서는 @Transactional이라는 annotation을 이용해 자동 적용 가능

@Transactional annotation은 컨트롤러, 서비스, 메소드 레벨에 적용 가능

⟨2⟩ Transaction Manager 설정

1. Spring 설정 파일에 아래와 같이 추가
2. proxy-target-class="true": Spring에서는 Interface 기반의 다이나믹 프록시를 지원하는데 Interface를 사용하지 않고 구체 클래스에 직접 트랜잭션을 적용하려면 아래와 같이 설정해줌

```
<tx:annotation-driven proxy-target-class="true" />
<bean id="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager
">
  <property name="dataSource" ref="dataSource" />
</bean>
```

동적 SQL

1 Provider 방식

Provider 방식

- SqlProvider와 SqlBuilder를 이용해 Java에서 동적으로 SQL을 생성하는 방법
- 예 : 동적으로 Select문을 구성할 때는 @SelectProvider annotation 이용
- 그 외 @UpdateProvider, @InsertProvider, @DeleteProvider가 있음

```
@SelectProvider(type = UserSqlProvider.class, method = "findAllByProvider")
List<User> findByProvider(Searchable searchable);
```

동적 SQL

1 Provider 방식

동적 쿼리를 위한 객체(Searchable.java)

```
public class Searchable {
    private String name;
    private String email;
        private String orderParam;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getOrderParam() {
        return orderParam;
    }

    public void setOrderParam(String orderParam) {
        this.orderParam = orderParam;
    }
}
```

동적 SQL

1 Provider 방식

동적 쿼리를 작성하는 객체(UserSqlProvider.java)

```
package com.spring. repository.provider;

import com.spring.domain.Searchable;
import org.apache.ibatis.jdbc.SQL;

public class UserSqlProvider {

    public String findAllByProvider(final Searchable searchable) {
        return new SQL() {
            {
                SELECT("*");
                FROM("USERS");
                if(searchable.getName() != null) {
                    WHERE("NAME = #{name}");
                    if(searchable.getEmail() != null) {
 OR();
                        WHERE("EMAIL = #{email}");
                    }
                }
                if(searchable.getOrderParam() != null) {

                    ORDER_BY(searchable.getOrderParam() + " DESC");
                }
            }
        }.toString();
    }
}
```

동적 SQL

1 Provider 방식

동적 쿼리를 위한 Controller(UserController.java)

```
@RequestMapping(value = "/list", method = RequestMethod.GET)
    public String list(Model model, @RequestParam(required=false) String
name, @RequestParam(required=false) String email,
    @RequestParam(required=false) String order) {
        Searchable searchable = new Searchable();
        searchable.setName(name);
        searchable.setEmail(email);
        searchable.setOrderParam(order);
        model.addAttribute("users", userMapper.findByProvider(searchable));
        return "list";
    }
```

2 Script 방식

1 Script 적용

1. Mybatis는 JSTL과 비슷한 형식의 스크립트를 제공
2. 앞서 UserSqlProvider와 비슷한 동작을 하는 스크립트는 다음과 같음

```
//@formatter off
 @Select("<script>"
            + "SELECT * FROM USERS"
            + "<if test='name != null'> WHERE NAME = #{name}</if>"
            + "<if test='name != null and email != null'> OR EMAIL =
#{email}</if>"
            + "<if test='orderParam != null'>ORDER BY ${orderParam} DESC</if>"
            + "</script>")
//@formatter on
    List<User> findByScript(Searchable searchable);
```

동적 SQL

2 Script 방식

《2》 List와 같은 자료구조 탐색

```
//@formatter off
 @Select("<script>"
          + "SELECT * FROM USERS"
          + "<if test='stringList != null and !stringList.empty'> WHERE NAME IN
<foreach item='item' collection='stringList' open='(' separator=','
close=')'>#{item}</foreach></if>"
          + "</script>")
//@formatter on
   List<User> findByList(@Param("stringList") List<String> stringList);
```

학습정리

1. Mybatis 알아보기

- Mybatis
  - 아파치 소프트웨어 재단의 iBatis 개발자 팀이 구글 코드로 이전하기로 결정하고 구글 코드에서 새로 만들어지는 이름이 MyBatis로 변경
  - MyBatis는 개발자가 지정한 SQL, 저장프로시저, 그리고 몇 가지 고급 매핑을 지원하는 영속성 프레임워크
- 기본 문법
  - @Insert : Mysql Insert문
  - @Update : Mysql Update문
  - @Select : Mysql Select문
  - @Delete : Mysql Delete문
  - @SelectKey
    - ✓ Statement(방금 삽입한 ID를 가져오기 위한 문구)
    - ✓ KeyProperty(ID 프로퍼티)
    - ✓ ResultType(결과값 형태)
  - @Param
    - ✓ 매퍼 메소드가 여러 개의 Parameter를 가진다면, 이 Annotation은 이름에 일치하는 매퍼 메소드 Parameter에 적용
    - ✓ @Param("person")를 사용하면, Parameter #{person}으로 명명

학습정리

2. Transaction

- Transaction
  - 데이터베이스 내에서 한꺼번에 수행되어야 할 일련의 연산들
  - DB와 JAVA 언어가 데이터를 주고 받는 과정에 원자성을 부여하는 수단
  - 트랜잭션의 모든 연산들은 한꺼번에 모두 완료되거나 그렇지 않으면 모두 취소되어야 함

- @Transactional 적용
  - 스프링에서는 @Transactional이라는 annotation을 이용하여 간단하게 자동으로 적용 가능
  - @Transactional annotation은 컨트롤러나 서비스, 메소드 레벨에 적용 가능

학습정리

3. 동적 SQL

- Provider 방식
  - SqlProvider와 SqlBuilder를 이용하여 Java에서 동적으로 SQL을 생성하는 방법
  - 그 외 @UpdateProvider, @InsertProvider, @DeleteProvider가 있음

- Script 방식
  - Mybatis는 JSTL과 비슷한 형식의 스크립트 제공
  - List와 같은 자료구조 탐색