데이터베이스 연동 (2) - Log4Jdbc, Service Layer

#01. SQL 처리 과정에 대한 상세 로그 남기기 (log4jdbc)

1. 의존성 추가

> https://mvnrepository.com/artifact/org.bgee.log4jdbc-log4j2/log4jdbc-log4j2-jdbc4.1

```
implementation 'org.bgee.log4jdbc-log4j2:log4jdbc-log4j2-jdbc4.1:1.16'
```

log4jdbc 설정파일 생성

/src/main/resources/log4jdbc.log4j2.properties 파일을 생성하고 아래 구문 추가

```
#-----------------------------------------------
# log4jdbc 설정 정보
#-----------------------------------------------
log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
log4jdbc.dump.sql.maxlinelength=0
log4jdbc.auto.load.popular.drivers=false
log4jdbc.drivers=com.mysql.cj.jdbc.Driver
```

Database 접속 정보 수정

/src/main/resources/application.properties 파일의 DATABASE 접속 정보에서 url과 driver를 수정

```
#-----------------------------------------------
# DATABASE 접속 정보
#-----------------------------------------------
# 기본 구성
#spring.datasource.url=jdbc:mysql://127.0.0.1:3306/myschool?characterEncoding=UTF8
#spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:log4jdbc:mysql://127.0.0.1:3306/myschool?
characterEncoding=UTF8
spring.datasource.driver-class-name=net.sf.log4jdbc.sql.jdbcapi.DriverSpy
spring.datasource.username=root
spring.datasource.password=123qwe!@#
```

logback-spring.xml 파일에서 로깅 대상 패키지 정보 추가

```
<!-- 데이터베이스 관련 패키지에 대한 로그 레벨 설정 -->
<logger name="jdbc.sqlonly" level="INFO" additivity="false">
    <appender-ref ref="CONSOLE" />
```

```
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</logger>
<logger name="jdbc.sqltiming" level="OFF">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</logger>
<logger name="jdbc.audit" level="OFF">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</logger>
<logger name="jdbc.resultset" level="ERROR">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</logger>
<logger name="jdbc.resultsettable" level="DEBUG" additivity="false">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</logger>
<logger name="jdbc.connection" level="ERROR">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</logger>
<logger name="log4jdbc.debug" level="ERROR">
    <appender-ref ref="CONSOLE" />
    <appender-ref ref="FILE" />
    <appender-ref ref="Error" />
</logger>
```

#02. Service Layer 구현

1. 인터페이스 정의

```
package kr.hossam.database.services;

import java.util.List;

import kr.hossam.database.exceptions.ServiceNoResultException;
import kr.hossam.database.models.${Model클래스};
/**
 * ${OOO} 관리 기능과 관련된 MyBatis Mapper를 간접적으로 호출하기 위한 기능 명세.
 * 하나의 처리를 위해서 두 개 이상의 Mapper 기능을 호출할 필요가 있을 경우,
 * 이 인터페이스의 구현체(Impl)을 통해서 처리한다.
 *
 * 1) delete 기능의 경우 삭제된 데이터의 수를 의미하는 int를 리턴
 * 2) 목록 조회 기능의 경우 List<DTO클래스>를 리턴
```

```
 * 3) 입력, 수정, 상세조회의 경우는 DTO 클래스를 리턴
 * 4) 모든 메소드는 throws Exception을 기술하여 예외가 발생할 경우 호출한 쪽에서 처리하
도록 유도한다.
 */
public interface ${Model클래스}Service {
    /**
     * ${OOO} 정보를 새로 저장하고 저장된 정보를 조회하여 리턴한다.
     *
     * @param input - 저장할 정보를 담고 있는 Beans
     * @return ${Model클래스} - 저장된 데이터
     * @throws MyBatisException - SQL처리에 실패한 경우
     * @throws ServiceNoResultException - 저장된 데이터가 없는 경우
     */
    public ${Model클래스} addItem(${Model클래스} input) throws
ServiceNoResultException, Exception;

    /**
     * ${OOO} 정보를 수정하고 수정된 정보를 조회하여 리턴한다.
     *
     * @param input - 수정할 정보를 담고 있는 Beans
     * @return ${Model클래스} - 수정된 데이터
     * @throws MyBatisException - SQL처리에 실패한 경우
     * @throws ServiceNoResultException - 수정된 데이터가 없는 경우
     */
    public ${Model클래스} editItem(${Model클래스} input) throws
ServiceNoResultException, Exception;

    /**
     * ${OOO} 정보를 삭제한다. 삭제된 데이터의 수가 리턴된다.
     *
     * @param input - 삭제할 조건을 담고 있는 Beans
     * @return int - 삭제된 데이터의 수
     * @throws MyBatisException - SQL처리에 실패한 경우
     * @throws ServiceNoResultException - 삭제된 데이터가 없는 경우
     */
    public int deleteItem(${Model클래스} input) throws ServiceNoResultException,
Exception;

    /**
     * ${OOO} 정보를 조회한다. 조회된 데이터가 없는 경우 예외가 발생한다.
     *
     * @param input - 조회할 ${OOO}의 일련번호를 담고 있는 Beans
     * @return ${Model클래스} - 조회된 데이터
     * @throws MyBatisException - SQL처리에 실패한 경우
     * @throws ServiceNoResultException - 조회된 데이터가 없는 경우
     */
    public ${Model클래스} getItem(${Model클래스} input) throws
ServiceNoResultException, Exception;

    /**
     * ${OOO} 정보를 조회한다. 조회된 데이터가 없는 경우 예외가 발생한다.
     *
     * @param input - 조회할 ${OOO}의 ${OOO}번호를 담고 있는 Beans
     * @return ${Model클래스} - 조회된 데이터
```

```
     * @throws MyBatisException - SQL처리에 실패한 경우
     * @throws ServiceNoResultException - 조회된 데이터가 없는 경우
     */
    public List<${Model클래스}> getList(${Model클래스} input) throws
ServiceNoResultException, Exception;
}
```

2. 구현체 클래스

```
package kr.hossam.database.services.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import kr.hossam.database.exceptions.ServiceNoResultException;
import kr.hossam.database.mappers.${모델클래스}Mapper;
import kr.hossam.database.mappers.ProfessorMapper;
import kr.hossam.database.mappers.StudentMapper;
import kr.hossam.database.models.${모델클래스};
import kr.hossam.database.models.Professor;
import kr.hossam.database.models.Student;
import kr.hossam.database.services.${모델클래스}Service;

/**
 * ${OOO} 관리 기능에 대한 비즈니스 로직 처리를 담당하는 서비스 계층의 구현체
 */
@Service
public class ${모델클래스}ServiceImpl implements ${모델클래스}Service {

    /** SQL문을 구현하고 있는 Mapper 객체 주입 */
    @Autowired
    private ${모델클래스}Mapper ${mapper객체};

    @Autowired
    private ProfessorMapper professorMapper;

    @Autowired
    private StudentMapper studentMapper;

    @Override
    public ${모델클래스} addItem(${모델클래스} input) throws
ServiceNoResultException, Exception {
        int rows = ${mapper객체}.insert(input);

        if (rows == 0) {
            throw new ServiceNoResultException("저장된 데이터가 없습니다.");
        }

        return ${mapper객체}.selectItem(input);
```

```
    }

    @Override
    public ${모델클래스} editItem(${모델클래스} input) throws
ServiceNoResultException, Exception {
        int rows = ${mapper객체}.update(input);

        if (rows == 0) {
            throw new ServiceNoResultException("수정된 데이터가 없습니다.");
        }

        return ${mapper객체}.selectItem(input);
    }

    @Override
    public int deleteItem(${모델클래스} input) throws ServiceNoResultException,
Exception {
        // ... 삭제 전 참조관계 확인 코드 추가

        int rows = ${mapper객체}.delete(input);

        if (rows == 0) {
            throw new ServiceNoResultException("삭제된 데이터가 없습니다.");
        }

        return rows;
    }

    @Override
    public ${모델클래스} getItem(${모델클래스} input) throws
ServiceNoResultException, Exception {
        ${모델클래스} output = ${mapper객체}.selectItem(input);

        if (output == null) {
            throw new ServiceNoResultException("조회된 데이터가 없습니다.");
        }

        return output;
    }

    @Override
    public List<${모델클래스}> getList(${모델클래스} input) throws
ServiceNoResultException, Exception {
        return ${mapper객체}.selectList(input);
    }
}
```

3. 단위 테스트

```
package kr.hossam.database.services;
```

```
import java.util.List;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import kr.hossam.database.exceptions.ServiceNoResultException;
import kr.hossam.database.models.${Model클래스};
import lombok.extern.slf4j.Slf4j;

@Slf4j
@SpringBootTest
public class ${Model클래스}ServiceTest {
    @Autowired
    private ${Model클래스}Service ${service객체};

    @Test
    @DisplayName("${OOO} 추가 테스트")
    void insert${Model클래스}() {
        ${Model클래스} input = new ${Model클래스}();
        input.setDname("스프링학과");
        input.setLoc("G강의실");

        ${Model클래스} output = null;

        try {
            output = ${service객체}.addItem(input);
        } catch (ServiceNoResultException e) {
            log.error("SQL문 처리 결과 없음", e);
        } catch (Exception e) {
            log.error("Mapper 구현 에러", e);
        }

        if (output != null) {
            log.debug("output: " + output.toString());
        }
    }

    @Test
    @DisplayName("${OOO} 수정 테스트")
    void update${Model클래스}() {
        ${Model클래스} input = new ${Model클래스}();
        input.setDeptNo(102);
        input.setDname("스프링${OOO}");
        input.setLoc("G강의실");

        ${Model클래스} output = null;

        try {
            output = ${service객체}.editItem(input);
        } catch (ServiceNoResultException e) {
            log.error("SQL문 처리 결과 없음", e);
        } catch (Exception e) {
```

```
            log.error("Mapper 구현 에러", e);
        }

        if (output != null) {
            log.debug("output: " + output.toString());
        }
    }

    @Test
    @DisplayName("${OOO} 삭제 테스트")
    void delete${Model클래스}() {
        ${Model클래스} input = new ${Model클래스}();
        input.setDeptNo(203);

        try {
            ${service객체}.deleteItem(input);
        } catch (ServiceNoResultException e) {
            log.error("SQL문 처리 결과 없음", e);
        } catch (Exception e) {
            log.error("Mapper 구현 에러", e);
        }
    }

    @Test
    @DisplayName("하나의 ${OOO} 조회 테스트")
    void selectOne${Model클래스}() {
        ${Model클래스} input = new ${Model클래스}();
        input.setDeptNo(102);

        ${Model클래스} output = null;
        try {
            output = ${service객체}.getItem(input);
        } catch (ServiceNoResultException e) {
            log.error("SQL문 처리 결과 없음", e);
        } catch (Exception e) {
            log.error("Mapper 구현 에러", e);
        }

        if (output != null) {
            log.debug("output: " + output.toString());
        }
    }

    @Test
    @DisplayName("${OOO} 목록 조회 테스트")
    void selectList${Model클래스}() {
        List<${Model클래스}> output = null;

        ${Model클래스} input = new ${Model클래스}();
        input.setDname("${OOO}");

        try {
            output = ${service객체}.getList(input);
        } catch (ServiceNoResultException e) {
```

```
            log.error("SQL문 처리 결과 없음", e);
        } catch (Exception e) {
            log.error("Mapper 구현 에러", e);
        }

        if (output != null) {
            for (${Model클래스} item : output) {
                log.debug("output: " + item.toString());
            }
        }
    }
}
```