Day 12 (2019-03-06)

1 Spring Web Application 개발 - JDBC, Mybatis

(1) JDBC 연동

1) JDBC (Java Database Connectivity)

Java 기반 어플리케이션에서 DB와 관련된 작업을 처리할 수 있도록 도와주는 역할을 한다. JDBC에서 제공하는 API를 통해 프로그래밍하며 다양한 DBMS 벤더마다 접근하는 표준이 다른 문제를 하나의 JDBC API를 통해 처리할 수 있도록 도와준다.

DBMS	JDBC Driver
MySQL	com.mysql.jdbc.Driver
Oracle	oracle.jdbc.driver.OracleDriver
MSSQL	com.microsoft.sqlserver.jdbc.SQLServerDriver

JDBC를 통해서 데이터베이스에 접근하여 작업하는 일반적인 과정은 아래와 같다.

1. DriverManager를 통해 DB Connection 객체 생성
2. Connection 객체에서 Statement(주로 PreparedStatement) 객체를 생성하여 쿼리를 실행 (executeQuery, executeUpdate)
3. 결과로 반환 된 ResultSet을 조작
4. Connection close

```
      Connection conn =null;
  Statement stmt =null;
  ResultSet rs =null;

  try{
    Class.forName("com.mysql.jdbc.Driver");

    String url ="jdbc:mysql://localhost/dev";
    conn = DriverManager.getConnection(url,"dev","dev");
    stmt = conn.createStatement();

    String sql ="SELECT name, owner, date_format(birth, '%Y%m%d' date FROM pet";
    rs = stmt.executeQuery(sql);

    while(rs.next()){
      String name = rs.getString(1);
      String owner = rs.getString(2);
      String date = rs.getString(3);

      System.out.println(name +" " + owner +" " + date);
    }
  }
  catch( ClassNotFoundException e){
    System.out.println("  ");
  }
  catch( SQLException e){
    System.out.println(" " + e);
  }
  finally{
        try {
                if( rs !=null && !rs.isClosed()){
                rs.close();
            }
        } catch( SQLException e){
      e.printStackTrace();
    }
        try{
                if( stmt !=null && !stmt.isClosed()){
               stmt.close();
            }
        } catch( SQLException e){
      e.printStackTrace();
    }
    try{
      if( conn !=null && !conn.isClosed()){
        conn.close();
      }
        } catch( SQLException e){
      e.printStackTrace();
    }
  }
}
```

try-with-resource 구문으로 바꾸면 조금 더 깔끔한 코드가 될 것

2) DataSource와 Connection Pool

javax.sql.DataSource는 Connection 인스턴스를 생성하는 표준 인터페이스로 JDBC 명세에 규정되어있다.

이러한 DataSource 인터페이스를 구현하는 대표적인 구현체는 Apache Commons DBCP와 HikariCP 등이 있다.
가장 간단한 구현체로는 Spring에서 제공하는 DriverManagerDataSource로 요청 할 때마다 매번 Connection 객체를 생성하여 새로 접속을 열도록 동작한다

일반적으로 커넥션을 생성하고 연결을 해제하는 작업은 비용이 크기 때문에 DriverManagerDataSource와 같이 것들은 테스트 환경에서나 사용한다.
주로 사용하는 커넥션 풀 라이브러리는 위에 언급한 Apache Commons DBCP, HikariCP가 있다.

※ HikariCP

Spring Boot 2.0 부터 default dependency로 설정되었으며 Commons DBCP와 벤치마킹 성능비교를 통해
좀 더 나은 안정적이고 효율성을 입증한 라이브러리이다. 프로젝트에 적용 시 반드시 부하테스트를 진행하여 성능 개션을 확인작업이 필요하다.

3) JNDI (Java Naming and Directory Interface)

J2EE의 스펙 중 하나로 Directory Service에서 제공하는 데이터 및 객체를 발견(discover)하고 참고(lookup)하기 위한 자바 API이다
대표적인 네이밍 서비스들인 LDAP, DNS 처럼 Naming → Binding 이름을 통해 자원을 바인딩하는 개념이라고 생각하면 편하다.

일반적으로 DataSource를 WAS에 설정하여 JNDI lookup을 통해 Application에서 활용하는 패턴을 사용하는데 아래오 같은 이점이 있다.

1. 협업을 통한 운영의 이점
    a. 보통 시스템 운영은 개발자가 아닌 SE가 담당하는 경우가 많기에
    Application에 설정할 경우 SE 담당자가 Application 구조를 파악해야 하는 문제가 생김
2. 관리포인트 감소
    a. 일반적으로 하나의 컨테이너에 여러 Application이 deploy 되는데 DataSource 설정이
    Application에 각각 설정되어있다면 추후 변경시 하나하나 수정해야하는 문제가 생김
3. 장애대처
    a. DB1 서버가 죽을 것을 대비하여 Active-Standby 구조를 구성하는데 failover와 관련하여 자동전환을
    대부분 WAS에서 제공함
4. 분산처리
    a. 분산 트랜잭션 설정을 위해선 반드시 WAS에서 제공하는 Transaction 기능을 이용해야 함

4) SQLException과 DataAccessException

앞선 JDBC API를 사용하여 DB 접근 처리하는 예제 코드를 보면 boilerplate code가 존재한다.

또한 대부분의 JDBC API는 Checked Exception인 SQLException을 내보내기 때문에 항상 try-catch 또는 try-resource 블록으로 감싸주어야 하는 문제가 있다.

Checked Exception은 강력한 비즈니스 로직을 표현하거나(상품 구매 요청 시 MoneyNotEnoughException) 복구가 가능한 예외(NoSuchAlgorithmException)인 경우에 선언하는 것이 보통이나
SQLException은 연결지연 문제나 Invalid Object로 인한 예외등을 제외하면 더 이상 복구가 가능한 것은 아니다

그렇기에 Spring JDBC에서는 SQLException을 DataAccessException이라는 최상위의 UnCheckedException으로 감싸 던진다.

대표적인 하위 Exception은 TypeMistachDataAccessException, BadSqlGrammerException, DuplicateKeyException 등이 있다.

2. Mybatis

(1) Mybatis란?

개발자가 지정한 SQL, Stored Procedure 등을 실행하고 결과를 매핑하는 것을 지원하는 Persistence Framework로 JDBC를 사용할 때 개발자가 일일히 지정했던 파라매터 매핑, Result mapping, Type handling 등을 쉽고 간편히 하도록 지원한다

(2) 사용예제

XML 형식으로 된 Mybatis mapper를 작성한다

```
<mapper namespace="kr.co.crscube.ctms.persistence.TrackerPdPvMapper">

   <resultMap id="TrackerPDPVMap" type="kr.co.crscube.ctms.model.tracker.TrackerPdPv">
      <id property="tpdKey" column="TPD_KEY"/>
      <result property="invtKey" column="INVT_KEY" />
      <result property="invtId" column="INVT_ID" />
      <result property="invtName" column="INVT_NAME" />
      <result property="tsbKey" column="TSB_KEY" />
      <result property="subjId" column="SUBJ_ID" />
      <result property="scrnNo" column="SCRN_NO" />
      <result property="randomNo" column="RND_NO" />
      <result property="pdCode" column="PD_CODE" />
      <result property="pdDetails" column="PD_DETAILS" />
      <result property="pdStatus" column="PD_STATUS" />
      <result property="capa" column="CAPA" />
      <result property="identifiedDate" column="IDENTIFIED_DT" />
      <result property="completedDate" column="COMPLETE_DT" />
      <result property="irbSubmissionDate" column="IRB_SUB_DT" />
      <result property="comments" column="COMMENTS" />
      <result property="subjKey" column="SUBJ_KEY" />
      <result property="schdKey" column="SCHD_KEY" />
      <result property="itemKey" column="ITEM_KEY" />
      <result property="rowNp" column="ROW_NP" />
      <result property="rowNo" column="ROW_NO" />
      <result property="instKey" column="INST_KEY" />
      <result property="reason" column="REASON" />
      <result property="deleted" column="DEL_FLAG"
            typeHandler="kr.co.crscube.ctms.persistence.typeHandler.BooleanCharTypeHandler"/>
      <result property="inputTime" column="INPUT_TIME" />
      <result property="userKey" column="USER_KEY" />
      <result property="siteKey" column="SITE_KEY" />
   </resultMap>


      <select id="selectTrackerPdPv" parameterType="int" resultMap="TrackerPDPVMap">
         SELECT
            TP.PROJ_KEY
            ,IT.INVT_KEY
            ,IT.INVT_NAME
            ,IT.INVT_ID
            ,TS.TSB_KEY
            ,TS.SUBJ_ID
            ,PD.TPD_KEY
            ,PD.PD_CODE
            ,PD.PD_DETAILS
            ,PD.PD_STATUS
            ,PD.CAPA
            ,PD.IDENTIFIED_DT
            ,PD.COMPLETE_DT
            ,PD.IRB_SUB_DT
```

```
            ,PD.COMMENTS
            ,PD.SUBJ_KEY
            ,PD.SCHD_KEY
            ,PD.ITEM_KEY
            ,PD.ROW_NO
            ,PD.ROW_NP
            ,PD.INST_KEY
            ,PD.REASON
            ,PD.DEL_FLAG
            ,PD.INPUT_TIME
            ,PD.USER_KEY
            ,PD.SITE_KEY
         FROM
            TPJ_PROJECT TP
            ,TPJ_INV_TEAM IT
            ,TAN_SUBJECT TS
            ,TAN_PDPV PD
         WHERE
            TP.PROJ_KEY = IT.PROJ_KEY
            AND IT.INVT_KEY = TS.INVT_KEY
            AND IT.INVT_KEY = PD.INVT_KEY
            AND TS.TSB_KEY = PD.TSB_KEY
       AND PD.TPD_KEY =  #{tpdKey, jdbcType=NUMERIC}
      </select>
</mapper>
```

ResultMap은 쿼리 결과를 우리가 원하는 Model, Dto의 형태로 가공해주는 역할을 한다

TypeHandler는 특정 컬럼의 데이터에 대해서 별도 전처리가 필요한 경우 적용하는 것이다.

ex) DB에 저장된 값은 YES, NO, Y, N, True, False 문자열일 때 이것을 Java의 Boolean 타입으로 변경

```
@MappedJdbcTypes(JdbcType.VARCHAR)
@MappedTypes(Boolean.class)
public class BooleanCharTypeHandler extends BaseTypeHandler<Boolean> {

    @Override
    public void setNonNullParameter(PreparedStatement ps, int i,
            Boolean parameter, JdbcType jdbcType) throws SQLException {
        if (parameter == null) {
            ps.setString(i, null);
        } else if (parameter.equals(Boolean.TRUE)) {
            ps.setString(i, "Y");
        } else {
            ps.setString(i, "N");
        }
    }

    @Override
    public Boolean getNullableResult(ResultSet rs, String columnName)
            throws SQLException {
        return convertStringToBoolean(rs.getString(columnName));
    }

    @Override
    public Boolean getNullableResult(ResultSet rs, int columnIndex)
            throws SQLException {
        return convertStringToBoolean(rs.getString(columnIndex));
    }

    @Override
    public Boolean getNullableResult(CallableStatement cs, int columnIndex)
            throws SQLException {
        return convertStringToBoolean(cs.getString(columnIndex));
    }

    private Boolean convertStringToBoolean(String source) {
        if (source != null && "Y".equals(source)) {
            return Boolean.TRUE;
        }
        return Boolean.FALSE;
    }
}
```

Mapper XML의 namespace에 명시한 인터페이스를 생성하고 first class element에 대응하는 메소드를 만든다.

```
package kr.co.crscube.ctms.persistence;

import kr.co.crscube.ctms.model.entry.EntrySearchData;
import kr.co.crscube.ctms.model.tracker.TrackerPdPv;
import org.apache.ibatis.annotations.Param;

import java.util.List;

public interface TrackerPdPvMapper {
   TrackerPdPv selectTrackerPdPv(@Param("tpdKey") int tpdKey);
}
```

해당 Mapper를 주입받은 서비스에서 실제 사용한다

```
public class TrackerPdPvServiceImpl implements TrackerPdPvService {

   @Autowired
   private TrackerPdPvMapper trackerPdPvMapper;

   @Override
   public TrackerPdPv selectTrackerPdPv(int tpdKey) {
      return trackerPdPvMapper.selectTrackerPdPv(tpdKey);
   }
}
```

(3) 동적 쿼리

1) if문

```
<select id="selectSubjectCompleteVisits" resultMap="subjectVisitRateMap">
   SELECT PROJ_KEY, INVT_KEY, SUBJ_KEY, VISIT_KEY
   FROM (
      SELECT TS.PROJ_KEY, TS.INVT_KEY, SBJ.SUBJ_KEY, SS.VISIT_KEY, SUM(DECODE(ASS.VALUE, 'Y', 0, 1))
INCOMPLETE_CNT
      FROM TAN_SUBJECT SBJ, TAN_SCHEDULE TS, TPJ_VISIT_RATE TVR, ANS_SUPP_SCHD ASS, STD_SCHEDULE SS
      WHERE SBJ.SUBJ_KEY IS NOT NULL
      <if test="projectKeys == null or projectKeys.size == 0">
         AND 1 = 2
      </if>
      <if test="projectKeys != null and projectKeys.size > 0">
        AND SBJ.PROJ_KEY IN
        <foreach collection="projectKeys" item="projectKey" index="index" open="(" close=")" separator=",">
         #{projectKey, jdbcType=NUMERIC}
        </foreach>
       </if>
        AND SBJ.DEL_FLAG = 'N'
        AND TS.TSC_KEY = SBJ.TSC_KEY
        AND TS.DEL_FLAG = 'N'
        AND TVR.PROJ_KEY = TS.PROJ_KEY
        AND TVR.DEL_FLAG = 'N'
        AND ASS.SUBJ_KEY = SBJ.SUBJ_KEY
        AND ASS.SUPP_ID LIKE 'REVIEW_ALL_1'
        AND ASS.DEL_FLAG = 'N'
        AND SS.SCHD_KEY  = ASS.SCHD_KEY
        AND SS.VISIT_KEY = TVR.VISIT_KEY
      GROUP BY TS.PROJ_KEY, TS.INVT_KEY, SBJ.SUBJ_KEY, SS.VISIT_KEY
   )
   WHERE INCOMPLETE_CNT = 0
</select>
```

2) foreach문

```
<select id="selectVisitReportDatasetSchedules" resultMap="VisitReportDatasetScheduleMap">
   SELECT
      TS.TSC_KEY
   ,   '['||TT.INVT_ID||'] '||TT.INVT_NAME TEAM_NAME
   ,     TS.SCHD_NO
   FROM TAN_SCHEDULE TS
   ,     TPJ_INV_TEAM TT
   WHERE
   TT.INVT_KEY = TS.INVT_KEY
   AND TT.DEL_FLAG = 'N'
   <if test="teamKeys != null and teamKeys.size > 0">
      AND   TT.INVT_KEY IN
      <foreach collection="teamKeys" item="teamKey" index="index" open="(" close=")" separator=",">
         #{teamKey}
      </foreach>
   </if>
   <if test="visitNos != null and visitNos.size > 0">
      AND   TS.SCHD_NO IN
      <foreach collection="visitNos" item="visitNo" index="index" open="(" close=")" separator=",">
         #{visitNo}
      </foreach>
   </if>
</select>
```

3) sql 구문을 통한 import

```
<sql id="pagingOracleOpening">
   <if test="pageIndex != null and rowNum != null">
      SELECT * FROM (
         SELECT ROWNUM AS RNUM, R.*
         FROM (
   </if>
</sql>

<sql id="pagingOracleClosing">
   <if test="pageIndex != null and rowNum != null">
         ) R
            WHERE ROWNUM <![CDATA[ <= ]]> #{pageIndex} * #{rowNum}
      ) WHERE RNUM <![CDATA[ > ]]> (#{pageIndex} - 1 ) * #{rowNum}
   </if>
</sql>


.....................


<select id="selectPdpvWithInvestigator" resultMap="TrackerPDPVMap">
   <include refid="kr.co.crscube.ctms.persistence.CommonMapper.pagingOracleOpening"/>
   SELECT
      TT.INVT_KEY
   ,  TT.INVT_ID
   ,  TT.INVT_NAME
   ,  TB.SCRN_NO
   ,  TB.RND_NO
   ,  TP.PD_CODE
   ,  TP.PD_DETAILS
   ,  TP.PD_STATUS
   ,  TP.CAPA
   ,  TP.IDENTIFIED_DT
   ,  TP.COMPLETE_DT
   ,  TP.IRB_SUB_DT
   ,  TP.COMMENTS
   ,  TP.TPD_KEY
   ,  TAX_PDPV.GET_REG_TIME(TP.TPD_KEY) INPUT_TIME
   FROM TPJ_INV_TEAM TT
      , TAN_SUBJECT  TB
      , TAN_PDPV    TP
   WHERE TT.INVT_KEY = #{invtKey, jdbcType=NUMERIC}
     AND TT.DEL_FLAG = 'N'
     AND TT.INVT_ID IS NOT NULL
     AND TB.INVT_KEY = TT.INVT_KEY
     AND TB.DEL_FLAG = 'N'
     AND TP.INVT_KEY = TB.INVT_KEY
     AND TP.TSB_KEY  = TB.TSB_KEY
     AND TP.DEL_FLAG = 'N'
     <if test="pdpvs != null">
      <foreach item="pdpvKey" collection="pdpvs" separator="or" open="AND (" close=")">
         TP.TPD_KEY = #{pdpvKey, jdbcType=NUMERIC}
      </foreach>
     </if>
   <include refid="kr.co.crscube.ctms.persistence.CommonMapper.commonFilter"/>

   ORDER BY TT.INVT_ID, TB.SCRN_NO, INPUT_TIME DESC
   <include refid="kr.co.crscube.ctms.persistence.CommonMapper.pagingOracleClosing"/>
</select>
```

3. Transaction

(1) JDBC에서 Transaction 관리

```
Connection conn = null;
PreparedStatement pstmt = null;


try{
.....
conn.setAutoCommit(false);                        //  false .


pstmt.executeUpdate("update .... ");            //   .
pstmt.executeUpdate("insert ....");

pstmt.executeUpdate("delete ... ");


.....


conn.commit();                                        //   commit() .

...
}catch(SQLException sqle){
        if(conn!=null) {
                try{
                        conn.rollback();
                }
                catch(SQLException sqle){
                }
        }
}

conn.setAutoCommit(true);
```

DBMS에 설정된 Default 내용에 따라 다르지만 보통 Auto commit이 true로 지정되어있으므로 Connection에서 명시적으로 Auto commit을 false로 지정 후
처리 결과에 따라 commit, rollback을 직접 개발자가 적용해주어야 하며 Auto commit도 다시 true로 원복해주어야 한다.

만약 여러 호출 스택을 거쳐 commit, rollback을 관리하게 된다면 매우 유지보수하기 힘들어진다

(2) Spring의 Transaction

여느 Application에서 필요로하는 공통기능인 Transaction을 Spring에선 AOP를 통해 제공한다.

사용법은 아래와 같이 Service layer의 메소드에 @Transactional 어노테이션을 붙이는 것이며
Spring은 TransactionSynchronizationManager 내부에서 ThreadLocal을 통해 멀티스레드 환경에서의 트랜잭션 컨텍스트를 관리한다

AOP 기반으로 구현되므로 이전 세미나에서 진행한 AOP 설정 시 주의점을 숙지하여야 하며
ThreadLocal을 통해 컨텍스트를 관리하므로 비동기 처리와 같이 Thread의 Context가 바뀌는 경우 제대로 Transaction이 처리되지 않을 수 있음을 유의하여야 한다.

```
@Transactional
@Override
public EntryGroup updateSomeEntryGroup(int userKey, int sponsorKey, EntryGroup group, String propertyJsonText) {
    group.setUpdateUserKey(userKey);
    group.setSponsorKey(sponsorKey);
    manageTemplateMapper.callUpdateSomeGroup(group);

    EntryGroup entryGroup = getSomeEntryGroupWithItems(sponsorKey, group.getGroupKey());

    for (GroupPropertyRecord record : getGroupPropertyRecords(entryGroup, propertyJsonText, userKey,
sponsorKey)) {
        validationGroupProperties(record);
        record.setGroupKey(entryGroup.getGroupKey());
        record.setReason(entryGroup.getReason());
        manageTemplateMapper.callUpdateSomeGroupProperty(record);
    }

    if (entryGroup.isForcedCreateRowNumTypeItem()) {
        manageTemplateMapper.callUpdateSomeItem(EntryItem.createForcedRowNumItem(group));
    }
    return entryGroup;
}
```

(3) Transaction 속성

트랜잭션이란 최소한의 작업단위를 의미하며 ACID 특성이있다.

원자성 (Atomicity)

All or Nothing으로 트랜잭션은 모두 성공하거나 모두 실패한다

일관성 (Consistency)

트랜잭션이 성공적으로 완료되면 DB의 일관성이 보존된다.
즉, 기존의 기본키/외래키 같은 무결성 제약조건이나 정합성이 일치하는 상태가 유지된다

격리성 (Isolation)

여러 트랜잭션이 동시에 실행되더라도 각 트랜잭션은 다른 트랜잭션과 독립적으로 수행된다.
멀티스레드 환경에서 병렬적으로 트랜잭션이 수행될 때 한 트랜잭션의 성공이나 실패가 다른 트랜잭션에 영향을 주지 않도록 하기 위함

지속성 (Durability)

성공적으로 수행된 트랜잭션은 영원히 지속된다

속성	설명	속성값	사용 예

isolation	Transaction의 isolation Level. 별도로 정의하지 않으면 DB의 Isolation Level을 따름.	• DEFAULT: DB 설정, 기본 격리 수준(기본설정) • READ_UNCOMMITED : 커밋되지 않는 데이터에 대한 읽기를 허용 • READ_COMMITED : 커밋된 데이터에 대해 읽기 허용 • REPEATEABLE_READ : 동일 필드에 대해 다중 접근 시 모두 동일한 결과를 보장 즉 반복하여 SELECT를 해도 다른 트랜잭션이 변경한 데이터 등에 대해서 영향받지 않는다 • SERIALIZABLE : 가장 높은 격리, 성능 저하의 우려가 있음 트랜잭션이 완료될 때까지 SELECT 문장이 사용하는 모든 데이터에 대해 Shared Lock이 걸려 다른 사용자는 그 영역에 해당하는 데이터에 수정 및 입력이 불가능하다	@Transactional (isolation=Isolation.DEFAULT)

propagation	트랜잭션 전파규칙을 정의	• PROPAGATION_MANDATORY : 작업은 반드시 특정한 트랜잭션이 존재한 상태에서만 가능 • PROPAGATION_NESTED : 중첩된 Transaction을 지원하는 WAS에서만 동작하며 기존에 트랜잭션(TX1)이 있는 경우, 중첩된 트랜잭션(TX2)는 포함되어서 실행되며 독자적으로 commit, rollback이 가능하다 • PROPAGATION_NEVER : 트랜잭션 상황에 실행되면 예외 발생 • PROPAGATION_NOT_SUPPORTED : 트랜잭션이 있는 경우에는 해당 트랜잭션은 보류하고 현재 메소드가 완료된 후에 보류된 트랜잭션을 다시 시작한다 • **PROPAGATION_REQUIRED : 트랜젝션이 있으면 그 상황에서 실행, 없으면 새로운 트랜잭션 실행(기본설정)** • PROPAGATION_REQUIRED_NEW : 대상은 자신만의 고유한 트랜잭션으로 실행 • PROPAGATION_SUPPORTS : 트랜젝션을 필요료 하지 않으나, 트랜잭션 상황에 있다면 포함되어서 실행	@Transactional(propagation=Propagation.REQUIRED)
readOnly	해당 Transaction을 읽기 전용 모드로 처리 JAP에서 조회 성능 개선을 위해 사용함	• true • false(default)	@Transactional(readOnly = true)
rollbackFor	정의된 Exception에 대해서는 rollback을 수행 Spring의 Transaction은 기본적으로 UnCheckedException(+ Error) 처럼 복구 불가능 한 에외나 오류 상황이 발생하면 rollback을 수행하는데 CheckedException인 경우에도 rollback을 수행하도록 하고싶을 때 적용한다		@Transactional(rollbackFor=Exception.class)
noRollbackFor	정의된 Exception에 대해서는 rollback을 수행하지 않음.		@Transactional(noRollbackFor=Exception.class)

timeout	지정한 시간 내에 해당 메소드 수행이 완료되지 않은 경우 rollback 수행. -1일 경우 no timeout (Default = -1)		@Transactional (timeout=10)