SqlmapsOfMaso

ORM의 또 다른 핵 iBATIS SQLMaps

저자 정보 : 이동국(**mailto:fromm0@gmail.com**)

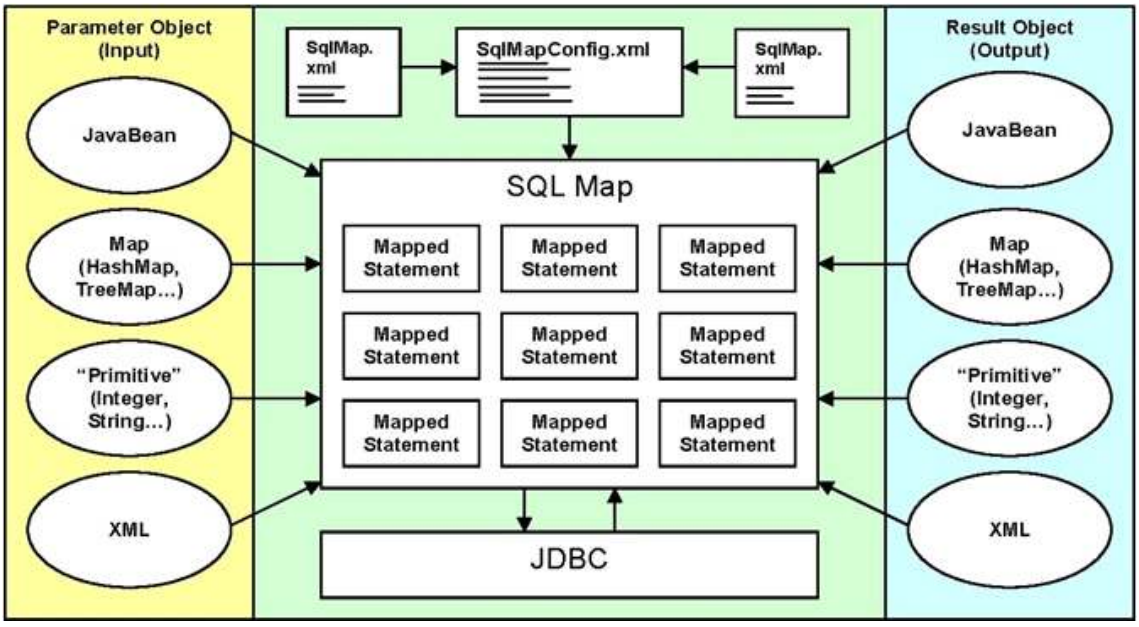
현재 울산에 있는 아이티스타에서 근무하고 있으며 주로 현대자동차 관련 프로젝트에 참여하고 있다. 네이버 자바프레임워크 카페 스탭으로 활동하고 있고 오픈프레임워크 사이트를 관리하고 있다. 평소 개발 프레임워크와 웹서비스에 관심을 가지고 있으며 요즘은 AndroMDA에 관련해서 공부를 시작했다.

ORM의 또 다른 핵 iBATIS SQLMaps

작년을 비롯해서 현재까지 해외 자바 관련 커뮤니티에서 가장 많이 논의가 되고 있는 부분 중 하나는 ORM이다. 그 이유를 생각해보면 자바개발의 많은 부분을 차지하고 있는 분야가 웹프로그래밍이고 그 웹프로그래밍의 거의 대부분을 차지하는 분야가 데이터베이스관련 처리이기 때문일 것이다. 현재 그 ORM을 대표하는 두 가지가 있다. 하나는 hibernate이고 또 하나는 이번에 소개할 iBATIS의 SQLMaps이다. 한때 객체와 관계형 데이터베이스의 관계 맵핑을 설정하면 SQL문을 자동 생성해 주는 hibernate에 비해 SQL문 자동 생성 기능이 없는 SQLMaps는 진정한 의미의 ORM이 아니다 라는 분위기도 있었지만 관계형 데이터베이스와 통신할 수 있는 유일한 매개체인 SQL문과 객체를 맵핑 시켜준다는 의미에서 현재는 SQLMaps또한 ORM이라고 보고 있다. hibernate의 경우 자바 관련 커뮤니티에서 많이 다루어 진 것으로 알고 있다. 우리는 여기서 SQLMaps에 대해서 알아보고자 한다. SQLMaps의 장점은 무엇일까.? iBATIS에서 밝히는 SQLMaps의 최대 장점은 간단함(simple)이다. 이것은 필자를 포함한 SQLMaps를 사용한 대부분의 개발자들이 인정하는 부분이다. 그 간단함이란 기존의 SQL문을 별다른 수정 없이 그대로 사용할 수 있는 상황에서 단순히 객체와 SQL문의 맵핑을 위한 몇몇 설정값만 설정해 주면 간단하게 ORM의 기능을 경험해 볼 수 있다는 것이다. 이러한 장점은 단순한 설명만으로는 잘 느껴지지 않으리라 본다. 잠시 후 예제를 통해서 정말 SQLMaps가 기존의 JDBC프로그램에 비해 얼마나 간단하게 구현이 되는지 보자.

SQLMaps는

앞에서 필자가 SQLMaps는 SQL과 객체를 맵핑시켜주는 도구라고 소개했지만 사실 엄밀하게 따지면 자바빈즈를 PreparedStatement 파라미터와 ResultSet으로 맵핑시켜주는 기능을 담당한다. 즉 PreparedStatement의 ? 에 각각의 파라미터 값을 setString()과 같은 메소드를 사용해서 셋팅하는 과정이나 ResultSet에서 getString()과 같은 메소드를 통해서 임의의 VO객체를 생성하는 과정을 자동으로 해준다. 결과적으로 간단한 xml셋팅만으로 일일이 손으로 작성하던 기존 JDBC형식을 버릴 수가 있다. SQLMaps의 구조를 도식화하면 다음과 같다.



iBATIS에서는 결과적으로 기존 데이터베이스 프로그램 코드의 20%정도만 사용해도 80%이상의 같은 기능을 수행할 수가 있게 된다고 밝히고 있고 필자 또한 그렇게 생각하고 있다.

이론적인 설명은 이쯤에서 잠시 접어두고 실제 사용하는 것을 보고 과연 어떻게 다른가를 알아보자. 아래의 소스는 얼마 전 오픈시드 프로젝트

(http://openseed.net)의 ORM 연구회에서 진행한 1차 테스트 프로젝트의 소스이다. 일단 SQLMaps를 사용하기 위해서는 최소 두 가지의 설정파일이 필요하다. SQLMaps 설정파일과 SQLMaps 맵핑 파일이다. 추가적으로 프라퍼티값을 외부로 빼내기 위해서 사용되는 properties파일이 필요할 수 도 있다. 먼저 SQLMaps설정 파일인 SqlMapsConfig.xml을 보자.

SQLMaps 설정파일

소스 : SqlMapConfig.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
"http://www.ibatis.com/dtd/sql-map-config-2.dtd">
<sqlMapConfig>
   <properties resource="SqlMapConfig.properties" />

   <settings cacheModelsEnabled="true" enhancementEnabled="true"
lazyLoadingEnabled="true" maxRequests="32"
maxSessions="10" maxTransactions="5"
useStatementNamespaces="false" />

   <typeAlias alias="comment" type="net.openseed.orm.openboard.domain.Comment"/>
   <typeAlias alias="message" type="net.openseed.orm.openboard.domain.Message"/>
   <typeAlias alias="user" type="net.openseed.orm.openboard.domain.User"/>

   <transactionManager type="JDBC">
      <dataSource type="DBCP">
         <property name="JDBC.Driver" value="${driver}" />
         <property name="JDBC.ConnectionURL" value="${url}" />
         <property name="JDBC.Username" value="${username}" />
         <property name="JDBC.Password" value="${password}" />
         <property name="JDBC.DefaultAutoCommit" value="false" />
      </dataSource>
   </transactionManager>

   <sqlMap resource="Comment.xml" />
   <sqlMap resource="Message.xml" />
   <sqlMap resource="User.xml" />
</sqlMapConfig>
```

xml파일 내용을 보면 크게 5가지로 분류되어 있다. 첫 번째인 <properties>는 프라퍼티값을 외부로 빼내서 사용하기 위해 프라퍼티파일을 사용할 때 해당 프라퍼티파일의 경로를 지정해준다. 프라퍼티 파일은 '키=값' 의 형태로 값을 지정하고 SqlMapConfig.xml에서는 ${키} 의 형태로 사용할 수 있다.

두 번째인 <settings>은 SQLMaps에서 사용되는 다양한 옵션과 최적화를 위한 값들이다. 각각의 값들은 다음의 표를 참조하길 바란다.

cacheModelsEnabled	SqlMapClient 를 위한 모든 캐시모델을 가능 유무. Default: true (enabled)
enhancementEnabled	런타임시 바이트코드 향상을 가능유무. Default: false (disabled)
lazyLoadingEnabled	모든 늦은(lazy)로딩을 가능유무. Default: true (enabled)
maxRequests	동시에 SQL문을 수행할 수 있는 쓰레드의 수. 셋팅값보다 많은 쓰레드는 다른 쓰레드가 수행을 완료할 때까지 블록 된다. Default: 512
maxSessions	주어진 시간동안 활성화될 수 있는 세션의 수. Default: 128
maxTransactions	한꺼번에 SqlMapClient.startTransaction()에 들어갈 수 있는 쓰레드의 최대갯수. 셋팅값보다 많은 쓰레드는 다른 쓰레드가 나올 때까지 블록 된다. Default: 32
useStatementNamespaces	이 셋팅을 가능하게 하면 당신은 sqlmap이름과 statement이름으로 구성된 전체적인 이름(fully qualified name)으로 맵핑된 statement를 참조해야 한다. 예를 들면: queryForObject("sqlMapName.statementName"); Default: false (disabled)

세 번째인 <typeAlias>는 패키지 명을 포함한 클래스가 너무 길 때 각각의 SQLMaps맵핑 파일에서 사용하기 번거로우므로 별칭을 두어서 간단하게 사용할 수 있다. 하지만 미리 정의된 별칭이 있다. 그 값들은 다음과 같다.

transactionManager에서 사용되는 별칭	
JDBC	com.ibatis.sqlmap.engine.transaction.jdbc.JdbcTransactionConfig

JTA	com.ibatis.sqlmap.engine.transaction.jta.JtaTransactionConfig
EXTERNAL	com.ibatis.sqlmap.engine.transaction.external.ExternalTransactionConfig
dataSource에서 사용되는 별칭	
SIMPLE	com.ibatis.sqlmap.engine.datasource.SimpleDataSourceFactory
DBCP	com.ibatis.sqlmap.engine.datasource.DbcpDataSourceFactory
JNDI	com.ibatis.sqlmap.engine.datasource.JndiDataSourceFactory

네 번째인 <transactionManager>는 트랜잭션에 관련된 값을 셋팅하고 하위 데이터소스값을 지정하는 <dataSource>를 가진다. transactionManager의 type속성 값은 JDBC, JTA, EXTERNAL 중에 하나를 사용할 수 있는데 그 각각의 값은 다음과 같은 특징을 가진다.

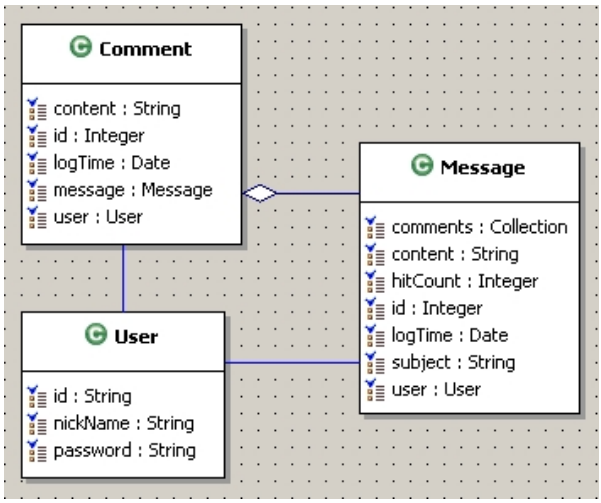
- JDBC - Connection commit()과 rollback()메소드를 통해 트랜잭션을 제어하기 위한 JDBC를 사용하게 된다.
- JTA - 이 트랜잭션관리자는 SQL Maps가 다른 데이터베이스나 트랜잭션 자원을 포함하는 더욱 넓은 범위의 트랜잭션을 포함하도록 하는 JTA전역트랜잭션를 사용한다.
- EXTERNAL - 이것은 개발자가 직접 트랜잭션을 관리할 때 사용하는 것으로 주로 분산 컴포넌트 기반 시스템 환경에서 컨테이너가 트랜잭션을 관리하는 경우에 사용된다.

dataSource의 type속성 값은 SIMPLE, DBCP, JNDI 중에 하나를 사용할 수 있다. 이 값은 각각 iBATIS SimpleDataSource 커넥션 풀링, Jakarta DBCP, JNDI를 통한 데이터소스을 사용하게 한다.

다섯 번째인 <sqlMap>은 SQLMaps맵핑 파일의 위치를 지정한다. resource속성 값에 전체패키지경로명을 포함하는 클래스 명을 써주면 된다.

기본이 되는 SQLMaps의 설정파일은 이렇게 특별히 어려운 설정 없이 간단하게 설정할 수 있다. 그럼 이제부터는 데이터베이스 작업별로 SQLMaps를 어떻게 사용하는지 보여주기 위한 요소인 자바소스와 SQLMaps맵핑 파일을 보도록 하자. SQLMaps맵핑 파일은 SQLMaps설정파일처럼 전체적인 구조를 볼 필요 없이 이후 설명되는 데이터베이스 작업의 종류에 따라 세부적인 부분만을 보도록 하겠다.

먼저 객체간의 관계는 다음과 같다.



SQLMaps를 사용하기 위한 객체 생성

SQLMaps를 사용하기 위해서 기본적으로 생성해줘야 하는 객체는 SqlMapClient 이다. 이 객체를 생성하기 위해서는 SQLMaps설정파일인 SqlMapsConfig.xml을 인자로 다음과 같이 코드를 작성하면 기본적으로 SQLMaps의 기능을 사용할 수 있는 SQLMapClient 객체가 생성된다. 이 소스는 getSqlMapConfig()라는 메소드이며 반환되는 객체는 이 문서에서 계속 사용된다.

```
Reader reader = Resources.getResourceAsReader ("SqlMapConfig.xml");
SqlMapClient sqlMap = SqlMapClientBuilder.buildSqlMapClient (reader);
```

조회

조회 작업의 경우 크게 한 개의 테이블을 조회하는 단순조회와 여러 개의 테이블이 서로 관계를 가지는 상태에서 조회하는 다중 테이블 조회의 두 가지의 경우로 나누어서 보겠다.

1. 테이블 한 개의 대한 단순조회

단순조회의 경우를 보기 위해 일단 사용자 정보를 보는 부분을 살펴보기로 하겠다. 사용자 정보에 해당되는 User.java소스는 위의 클래스다이어그램을 통해 파악을 할 수 있을 것이다.

다음은 조회를 위해 사용된 SQLMaps맵핑 파일의 일부이다.

소스 : User.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN" "http://www.ibatis.com/dtd/sql-map-2.dtd" >
<sqlMap>
  <select id="getUser" parameterClass="string" resultMap="user">
  <![CDATA[
    select
      user_id as id,
      password as password,
      nick_name as nickName
    from userinfo
    where user_id=#id#
  ]]>
  </select>
</sqlMap>
```

dtd문서를 보면 알겠지만 SQL문을 포함할 수 있는 요소는 <statement>, <insert>, <update>, <delete>, <select>, <procedure> 정도가 된다. 각각의 의미는 짐작이 갈 것이다. statement는 모든 SQL문의 형태, insert, update, delete, select는 각각의 데이터베이스 작업, procedure는 프로시저작업을 할때 사용하면 된다. 일단 여기서는 조회작업을 수행하기 때문에 <select>를 사용했다. 속성으로는 id, parameterClass, resultClass가 사용되었는데 id 속성 값은 자바소스에서 해당 쿼리를 참조하기 위한 키의 역할을 담당한다. 키에 대해서는 이 xml 뒤에 나오는 자바소스를 통해 좀 더 자세히 설명할 것이다. parameterClass 속성값이 string이면 인자로 넘어오는 값이 문자열타입임을 나타낸다. 즉 user_id에는 인자로 넘어오는 문자열값이 자동으로 셋팅된다. resultClass속성값인 user는 조회 후 각각의 레코드가 자동으로 user 객체 타입으로 생성이 된다는 것을 의미한다. 여기서 user라는 값은 앞에서 별칭된 net.openseed.orm.openboard.domain.User를 나타낸다. 내부적으로는 user객체가 생성이 되서 각각의 칼럼의 값이 setId(id칼럼값), setPassword(password칼럼값), setNickName(nickName칼럼값)를 차례로 호출하여 값을 할당하게 되는 셈이다.

이렇게 내부적으로 처리가 된다면 DAO계층에서 개발자가 어떻게 코딩을 하면 될까.? 그 형식은 다음과 같다.

```
    public User get(String id) {
        SqlMapClient sqlMap = null;
        User user = null;

        try {
            sqlMap = getSqlMapConfig();
            user = (User) sqlMap.queryForObject("getUser", id);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
        }
        return user;
    }
```

단순히 소스코드에서 SQL문과 자바소스만 분리한 것 같은데도 너무 간단하지 않은가.? 사실. 예제의 SQL문 자체가 너무 단순해서 간단함이 잘 느껴지지 않을 수 있으나 테이블 자체가 굉장히 많은 칼럼을 가지고 있었던 예전 경험을 생각해 보라. 이것은 사용자로 하여 SQLMaps를 사용하면 굉장히 지겨운 JDBC코드작성에서 벗어나고 그 과정에서 발생할 수 있는 많은 버그도 줄일 수가 있을 것이다.

2. 두개이상의 테이블에 대한 조회

그렇다면 다른 테이블과 1:1, 1:M, 또는 M:N 등의 다중 관계를 가지는 경우에는 어떻게 처리할까.? 그 예는 메시지관련 부분을 보면 된다. Message는 게시판에서 사용자에게 보여지게 되는 실제 글에 해당되는 객체로써 Message.java를 사용한다.

이러한 관계를 SQLMaps맵핑 파일에서는 어떻게 표현할까.? 그 표현방식은 다음에서 볼 수 있다.

소스 : Message.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE sqlMap PUBLIC "-//iBATIS.com//DTD SQL Map 2.0//EN" "http://www.ibatis.com/dtd/sql-map-2.dtd" >
<sqlMap namespace="Message">
  <resultMap id="get-message" class="message">
    <result property="id" column="id"/>
    <result property="subject" column="subject"/>
    <result property="content" column="content"/>
    <result property="hitCount" column="hitCount"/>
    <result property="logTime" column="logTime"/>
  </resultMap>
  <resultMap id="get-message-result" class="message" extends="get-message">
    <result property="user.id" column="user_id"/>
    <result property="user.password" column="password"/>
    <result property="user.nickName" column="nickName"/>
    <result property="comments" column="{id=id}" select="getCommentList" />
  </resultMap>

  <select id="getMessageList" resultMap="get-message-result">
  <![CDATA[
    select a.seq as id,
      a.subject as subject,
      a.content as content,
      a.user_id as user_id,
      a.hit_count as hitCount,
      a.log_time as logTime,
      b.password as password,
      b.nick_name as nickName
    from message a, userinfo b
    where a.user_id=b.user_id
```

```
    order by a.seq desc
  ]]>
  </select>
</sqlMap>
```

여기서 눈여겨보아야 하는 것은 resultMap이다. 이는 복합적인 타입을 사용할 때 굉장히 유용하다. 물론 복합타입에만 사용할 필요는 없지만 필자는 기본적으로 복합타입이 아닐 경우 resultClass를 사용한다. 여기에서 첫 번째 resultMap은 기본적인 message정보를 가진다. 두 번째는 짐작할 수 있듯이 메시지 객체의 사용자정보와 여러 개의 덧글에 대한 정보를 가진다. 여기서 두번째 resultMap은 첫번째 resultMap의 정보를 가져오기 위해 extends 속성을 사용한다. 즉 extends는 두개의 resultMap를 연결하는 연결자 역할을 하게 된다. 이 resultMap에는 내포된 객체에 값을 셋팅하는 방법도 포함하고 있다. <result property="user.id" column="user_id"/> 여기서 property값인 user.id가 셋팅될 객체의 프라퍼티를 가리킨다. 다시 말해 Message객체내의 user객체의 id값을 user_id칼럼의 값으로 셋팅하는 것이다. 굳이 자바 소스로 표현해 본다면 다음과 같을 것이다. message.getUser().setId(rs.getString("user_id")); 이런 처리방식으로 내포된 다양한 객체에 대한 접근도 가능하다. 그렇다면 메시지 객체내의 여러 개의 덧글을 의미하는 Collection형태의 객체는 어떻게 생성할까.? 그 방법은 <result property="comments" column="{id=id}" select="getCommentList" /> 에 모두 담겨있다. 여기서는 두 가지가 설명되어야 한다. 먼저 select값인데 이는 getCommentList라는 id값을 가지는 쿼리를 호출한다. getCommentList라는 id값을 가지는 쿼리는 다음과 같다.

소스 : Comment.xml

```
   <select id="getCommentList" parameterClass="comment" resultMap="get-comment-result">
  <![CDATA[
    select cu.seq as seq,
      cu.content as content,
      cu.log_time as logTime,
      cu.user_id as userId,
      cu.message_seq as messageSeq,
      cu.nick_name as nickName,
      cu.password as password,
      m.seq as mseq,
      m.subject as subject,
      m.content as mcontent,
      m.user_id as muserId,
      m.hit_count as hitCount,
      m.log_time as mlogTime
      from (select c.seq, c.content,
      c.log_time, c.user_id, c.message_seq,
      u.nick_name, u.password
    from comment c, userinfo u
    where c.user_id=u.user_id) cu, message m
    where cu.message_seq=m.seq
    and cu.message_seq=#id#
    order by cu.seq asc
  ]]>
  </select>
]]>
</select>
```

이 쿼리는 실제 Comment.xml이라는 xml파일에 있는데 SQLMaps맵핑파일이 SQLMaps설정파일에 모두 선언이 되어 있다면 각각의 맵핑파일내 SQL문은 어디서든 호출이 가능하다. 이것은 굉장히 큰 장점이다. 해당 SQL문이 xml내에서 뿐 아니라 자바소스내에서도 어디서든 호출된다는 것은 해당 테이블의 구조가 변경될시 기존의 JDBC형태의 프로젝트처럼 프로그램마다 쿼리문을 수정했던 어려움 없이 해당 맵핑 파일의 SQL문 하나만을 수정하면 전체 프로그램에 반영이 된다는 것이다. 필자는 독자 여러분이 이런 부분을 정말 큰 장점이라고 인식해주길 바란다. select="getCommentList"을 사용함으로써 자동적으로 <result>에는 List형태의 값이 자동으로 셋팅된다. 그럼 많은 독자분들이 조회조건에 해당되는 값은 어떻게 넘겨야 하는지에 대한 의문이 들것이다. 이에 대한 답은 column속성에 모두 담겨있다. 지금 이 소스의 column값은 {id=id} 이다. 이것은 특정객체의 id값을 id칼럼의 값으로 셋팅한다는 것이다. 즉 {id=id}에서 전자의 id는 값이 셋팅되는 객체의 id라는 변수이고 후자의 id는 id칼럼의 값이다. 결과적으로 현재 getCommentList쿼리의 parameterClass인 comment객체의 id값을 id칼럼의 값으로 셋팅해서 넘기게 된다. 만약에 두개 이상의 조회조건, 예를 들면 comment객체의 id값과 content값을 인자로 넘겨야 한다면 ,(콤마)를 구분자로 주어서 {id=id,content=content} 형태로 column값을 설정하면 된다. 지금까지 설명한 것은 getCommentList의 parameterClass가 원시타입이 아닌 특정객체타입 일때의 경우이고 단순히 원시타입인 string이나 int타입이라면 column값을 id 라고 지정해 주면 된다. 즉 칼럼명만 지정해주면 된다. 이는 id칼럼값을 그냥 그대로 넘기는 것을 의미한다. 결과적으로 id가 getMessageList인 SQL문을 호출하면 메시지에 관련된 값이 자동으로 get-message-result라는 값의 resultMap에 셋팅되고 이에 자동적으로 다시 getCommentList를 호출해서 해당 값을 다시 Collection형태의 값으로 채워준다.

자바소스에서의 처리는 다음과 같다.

```
PaginatedList list = (PaginatedList) sqlMap.queryForPaginatedList("getMessageList", "", range);
for (int i = 0; i < page - 1; i++) {
  list.nextPage();
}
```

여기서 queryForPaginatedList 메소드는 SQLMaps에서 페이지처리를 위해 기본으로 제공하는 메소드이다. 이것을 사용하면 페이지처리를 자동으로 해준다. 조회는 이렇게 두 가지 경우만 살펴보면 거의 대부분의 상황에 대해 설명이 되는 듯 하다. 그럼 입력의 경우를 살펴보도록 하자.

입력

```
  <insert id="insertMessage" parameterClass="message">
    <selectKey resultClass="int" keyProperty="id">
    <![CDATA[
      select nextval('hibernate_sequence');
    ]]>
```

```
    </selectKey>
  <![CDATA[
    insert into message(seq, subject, content, user_id, hit_count, log_time)
    values(#id#, #subject#, #content#, #user.id#, 0, 'today')
  ]]>
  </insert>
```

입력 또한 조회의 경우와 크게 다르지 않다. parameterClass에 해당되는 객체의 값이 각각의 #값# 에 셋팅된다. 조회의 경우와 다르게 추가되는 부분은 <selectKey>요소이다. 이 요소는 선택사항으로 key에 해당되는 값을 임의로 조회결과값으로 설정할 수가 있도록 지원하는 기능을 가진다. 즉 selectKey내의 쿼리문에 의해 반환되는 값이 #id#값에 대치가 된다는 것이다.

자바소스 내에서는 다음과 같이 처리할 수 있다.

```
        try {
            sqlMap = getSqlMapConfig();
            sqlMap.startTransaction();

            result = (Integer) sqlMap.insert("insertMessage", message);

            sqlMap.commitTransaction();
            success = true;
        } catch (NestedException ne) {
            logger.error(ne.getMessage(), ne);
        } catch (Exception e) {
            logger.error(e.getMessage(), e);
        } finally {
            sqlMap.endTransaction();
        }
```

이 소스의 insert()메소드는 결국 다음과 같은 작업을 수행하는 셈이다.

```
String sql = "insert into message(seq, subject, content, user_id, hit_count, log_time)
values(?, ?, ?, ?, 0, 'today')"
PreparedStatement pstmt = connection.prepareStatement(sql);
pstmt.setString(1, message.getId());
pstmt.setString(2, message.getSubject());
pstmt.setString(3, message.getContent());
pstmt.setString(4, message.getUser().getId());
result = pstmt.executeUpdate();
```

여기서 필자는 추가적으로 SQLMaps내에서의 트랜잭션 처리에 대해서 언급하도록 하겠다. SQLMaps의 트랜잭션은 sqlMap.startTransaction() 메소드를 호출함으로써 시작된다. 커밋을 수행하기 위해서는 sqlMap.commitTransaction() 메소드를 호출하면 된다. 하지만 SQLMaps API를 잠시 들여다 보면 SQLMaps에는 롤백에 관련된 메소드가 없다. 아니 없다기 보단 임의로 롤백을 수행하는 메소드가 없다. SQLMaps에서는 예외가 발생할 경우 sqlMap.endTransaction()메소드를 호출함으로써 롤백을 수행할 수 있다. 즉 sqlMap.endTransaction()은 정상적으로 모든 작업이 수행이 되면 트랜잭션을 커밋하고 connection을 닫는 작업을 자동으로 수행하게 되고 만약 예외가 발생했을 경우 트랜잭션을 롤백하고 connection을 닫는 작업을 자동으로 수행한다.

> 이런 트랜잭션처리를 하더라도 결과적으로 제대로 트랜잭션처리가 안되는 경우가 있다. MySQL예전 버전(필자가 테스트한 바로는 4.0.x버전)에서는 정상적인 처리가 되지 않고 각각의 작업이 수행 직후 바로 커밋되어버린다. 이는 iBATIS에서도 공식적으로 알려진 부분이고 다른 데이터베이스나 MySQL 4.1.x버전을 포함한 이후버전에서는 문제가 없다.

수정과 삭제의 작업은 입력의 작업과 사실 거의 유사하다. 그래서 수정과 삭제에 대한 추가적인 설명은 여기서 생략하도록 하겠다.

Spring에서 SQLMaps 사용하기.

Spring은 현재 비즈니스 레이어를 담당하는 오픈소스 프레임워크중에 거의 독보적인 위치를 차지하고 있다. Spring을 사용하면 IoC형태로 트랜잭션을 관리할 수도 있고 SQLMaps를 좀더 쉽게 사용할 수 있도록 도와준다.

> Spring에서는 SQLMaps 1.x와 2.x를 지원하기 위한 클래스가 다르다. 물론 xml설정파일에서 셋팅하는 값이 다르다. 여기서는 2.x버전을 지원하는 내용만을 다룬다.

SqlMapConfig.xml을 applicationContext.xml으로 옮기기

소스 : applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework.org/dtd/spring-beans.dtd">
<!-- 이 설정파일은 openseed의 스터디때 생성된 파일이 아니다.
글을 설명하기 위해 임의로 생성된 파일이나 설정자체는 제대로 작동하는것을 확인했다. -->
<beans>
  <!-- iBATIS SQLMaps의 설정파일 위치를 지정한다.

  class값은
  SQLMaps 1.x버전을 사용할때는 org.springframework.orm.ibatis.SqlMapFactoryBean
  SQLMaps 2.x버전을 사용할때는 org.springframework.orm.ibatis.SqlMapClientFactoryBean 를 사용한다.
  -->
  <bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
    <property name="configLocation"><value>WEB-INF/SqlMapConfig.xml</value></property>
  </bean>
```

```
  <!-- dataSource를 사용하는것에 대한 정보를 나타낸다.
  여기서 사용될수 있는 dataSource타입은 다른 문서를 참조하길 바란다.

  여기선 apache의 DBCP Connection pooling을 사용하는 것이다. -->
  <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
    <property name="driverClassName"><value>org.postgresql.Driver</value></property>
    <property name="url"><value>jdbc:postgresql://localhost:5432/openseed</value></property>
    <property name="username"><value>openseed</value></property>
    <property name="password"><value>openseed</value></property>
    <property name="defaultAutoCommit"><value>false</value></property>
  </bean>

  <!-- DB연결에 관련된 설정을 DataSource형태로 지정을 했기 때문에 트랜잭션 관리를
  org.springframework.jdbc.datasource.DataSourceTransactionManager 가 담당하도록 지정한다. -->
  <bean id="myTransactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
      <property name="dataSource"><ref local="dataSource"/></property>
  </bean>

  <!-- 각각의 메소드 별로 트랜잭션관리 속성을 지정한다. -->
  <bean id="guestService" class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
    <property name="transactionManager"><ref local="myTransactionManager"/></property>
    <property name="target"><ref local="guestTarget"/></property>
    <property name="transactionAttributes">
      <props>
        <prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
        <prop key="save*">PROPAGATION_REQUIRED</prop>
        <prop key="update*">PROPAGATION_REQUIRED</prop>
        <prop key="delete*">PROPAGATION_REQUIRED</prop>
      </props>
    </property>
  </bean>

  <!-- 소위 Spring을 사용하게 되는 비지니스 객체의 클래스를 지정하는 부분이다.
  즉 여기선 gikim.dongguk.guestboard.spring.GuestSpringImpl 클래스가 Spring의 여러가지 기능을 담당하게 되는것이다.
  그리고 관리하게 되는 DAO는 guestDAO로 지정한다.
  -->
  <bean id="guestTarget" class="gikim.dongguk.guestboard.spring.GuestSpringImpl">
    <property name="guestDAO"><ref local="guestDAO"/></property>
  </bean>

  <!-- DAO에 관련된 셋팅이다. 실제로 iBATIS SQLMaps를 사용하게 되는 클래스를 지정하게 된다.
  DB정보인 dataSource값과. iBATIS SQLMaps설정파일의 위치에 해당하는 sqlMapClient를 지정한다.
  -->
  <bean id="guestDAO" class="gikim.dongguk.guestboard.dao.IbatisGuestDAOImpl">
    <property name="dataSource"><ref local="dataSource"/></property>
    <property name="sqlMapClient"><ref local="sqlMapClient"/></property>
  </bean>
</beans>
```

SqlMapConfig.xml의 내용을 applicationContext.xml 로 옮기면 사실 SqlMapConfig.xml에는 <settings>와 <sqlMap>요소만 정의를 해주면 된다. 기존의 <properties>와 <typeAlias>, <transactionManager> 는 applicationContext.xml의 내용으로 대체되는 것이다. 그 외의 SQLMaps맵핑 파일 같은 경우는 그대로 사용가능하다.

SqlMapClient 객체를 대체하는 SqlMapClientTemplate

spring은 SqlMapClient객체를 대체하는 SqlMapClientTemplate를 제공하는데 이를 생성하는 메소드는 getSqlMapClientTemplate()이다.

```
Reader reader = Resources.getResourceAsReader("SqlMapConfig.xml");
SqlMapClient sqlMap = SqlMapClientBuilder.buildSqlMapClient(reader);
user = (User) sqlMap.queryForObject("getUser", id);
```

사용자 정보를 가져오는 소스가 위와 같다고 할 때 spring의 SqlMapClientTemplate를 사용하면 아래와 같이 될 것이다..

```
User user = getSqlMapClientTemplate().queryForObject("getUser", id);
```

눈에 띄게 소스가 간편해진다. 이것밖에 장점이 없을까.? 아니다. 다음의 소스를 보자. spring이 자동적으로 트랜잭션을 처리해 주기 때문에 조금전 입력예제로 사용되었던 소스가 아래처럼 다시 변경될 수 있다.

```
getSqlMapClientTemplate().insert("insertMessage", message);
return true;
```

spring의 트랜잭션 관리를 사용하면 위와 같이 소스가 간단해진다.

이것은 transactionAttributes 속성 하위의 설정값들에 의해 spring의 DI(dependency injection)을 사용하여 가능한 것이지만, 일단 개발자들에게는 SQLMaps로 인해 간단해진 소스가 더욱 심플해졌다는 사실 자체로 행복한 일일 것이다. 이건 개발 프레임워크를 제대로 사용할 때 개발 프레임워크가 개발자에게 주는 혜택이다.

결과적으로 SQLMaps는

지금까지의 설명에서 SQLMaps는 기존의 JDBC프로그래밍 방식과 근본적으로 프로그래밍 방식이 바뀌는 건 아님을 알 수 있다. 사실 이런 부분

이 기존의 코딩 방식을 그대로 가져가기 때문에 획기적인 개선이 없다는 단점을 낳기도 한다. 개인적으로 SQLMaps는 JDBC프로그래밍 방식과 hibernate와 같은 완전히 다른 방식의 데이터베이스 처리 방식 사이의 중간에 위치한다고 생각한다. 둘의 단점을 모두 가지고 있지만 둘의 장점 또한 모두 가지고 있는 ORM이다. 그 둘의 단점을 피하고 장점만을 제대로 사용한다면 정말 좋은 ORM이라는 건 어느 누구도 의심하지 않을 것이다. 필자는 다음과 같은 경우에 hibernate보다는 SQLMaps가 적합할 것 같다고 생각을 한다.

- JDBC코딩의 단점은 버리고 싶으나 hibernate와 같은 전혀 다른 방식을 사용하는 것은 부담스러운 프로젝트
- 기존 JDBC형태의 프로젝트를 ORM 기술을 사용해서 변경하고자 할 때
- 데이터베이스에 관련된 DBA가 회사 내에 있어서 기존의 SQL에 대한 지식과 경험이 많거나 HQL의 성능이 의심되는 경우

개발자는 SQLMaps의 설정방법과 맵핑파일에 관련된 몇몇 지식만 가지면 기존의 JDBC방식을 사용하던 개발자가 SQLMaps를 사용하는 건 크게 어렵지 않다. 하지만 JDBC 코드를 SQLMaps로 바꾸었을 때의 결과는 굉장한 긍정적인 차이를 가질 것이라는 점은 위에서 계속 설명했던 바이다

필자가 설명하는 SQLMaps의 장점이 제대로 전달되었는지는 잘 모르겠다. 사실 이외에도 SQLMaps에서 지원하는 기능은 많다. 물론 여기서 필히 논의가 되었어야 할 dynamic SQL부분이나 로깅관련 사항들은 지면이 모든 내용을 포함하도록 할 수 없는 제한으로 인해 설명되지 못했다. 이 사항들은 관련 문서와 커뮤니티에서 충분한 정보를 얻을 수 있을 것이다. 현재 가장 빠르게 발전하고 있는 ORM은 hibernate임에는 필자도 이견을 달지 않는다. 하지만 오픈소스가 생성된 배경과 의도가 다른 만큼 개발자들이 단순히 기술적인 부분에만 치우쳐 오픈소스를 사용하는 일은 없었으면 한다. 기술적인 부분이 어느 정도 지원하는 수준에서 프로젝트에서 요구하는 상황과 해당 오픈소스의 생성의도가 제대로 맞아떨어질 때 그 오픈소스가 정말 강력한 기능을 발휘할 것이라는 점은 모두가 동의할 것이다. 현재 hibernate는 jboss의 지원을 받고 있고 SQLMaps는 아파치의 지원을 받고 있다. 두 오픈소스가 충분히 발전할 가능성을 가지고 있고 이미 강력한 기능을 지원하는 만큼 적재적소에 해당 오픈소스를 사용하길 바란다.

참고자료

iBATIS 홈페이지 - **http://incubator.apache.org/ibatis/site/index.html**
SQLMaps 2.0개발자 가이드(한글) **http://openframework.or.kr/JSPWiki/attach/Hibernate/iBATIS-SqlMaps-2_ko.pdf**
SQLMaps 2.0튜토리얼(한글) **http://openframework.or.kr/JSPWiki/attach/Hibernate/iBATIS-SqlMaps-2-Tutorial_ko.pdf**
SQLMaps를 이용한 객체-관계맵핑 **http://openframework.or.kr/JSPWiki/Wiki.jsp?page=ObjectRelationalMappingwithSQLMaps**
네이버 자바프레임워크 **http://cafe.naver.com/deve**
네이버 자바프레임워크 위키 **http://openframework.or.kr**

Attachments

1.jpg		189946 bytes
2.jpg		91770 bytes
bonus.html		8280 bytes
그림1.jpg		189946 bytes
그림2.jpg		91770 bytes