동적 SQL

마이바티스의 가장 강력한 기능 중 하나는 동적 SQL을 처리하는 방법이다. JDBC나 다른 유사한 프레임워크를 사용해본 경험이 있다면 동적으로 SQL 을 구성하는 것이 얼마나 힘든 작업인지 이해할 것이다. 간혹 공백이나 콤마를 붙이는 것을 잊어본 적도 있을 것이다. 동적 SQL 은 그만큼 어려운 것이다.

동적 SQL 을 사용하는 것은 결코 파티가 될 수 없을 것이다. 마이바티스는 강력한 동적 SQL 언어로 이 상황은 개선한다.

동적 SQL 엘리먼트들은 JSTL이나 XML기반의 텍스트 프로세서를 사용해 본 사람에게는 친숙할 것이다. 마이바티스의 이전 버전에서는 알고 이해해야 할 엘리먼트가 많았다. 마이바티스 3 에서는 이를 크게 개선했고 실제 사용해야 할 엘리먼트가 반 이하로 줄었다. 마이바티스의 다른 엘리먼트의 사용을 최대한 제거하기 위해 OGNL 기반의 표현식을 가져왔다.

- if
- choose (when, otherwise)
- trim (where, set)
- foreach

if

동적 SQL 에서 가장 공통적으로 사용되는 것으로 where의 일부로 포함될 수 있다. 예를 들면:

```
<select id="findActiveBlogWithTitleLike"
     resultType="Blog">
  SELECT * FROM BLOG
  WHERE state = ‘ACTIVE’
  <if test="title != null">
    AND title like #{title}
  </if>
</select>
```

이 구문은 선택적으로 문자열 검색 기능을 제공할 것이다. 만약에 title 값이 없다면 모든 active 상태의 Blog 가 리턴될 것이다. 하지만 title 값이 있다면 그 값과 비슷한 데이터를 찾게 될 것이다.

title과 author를 사용하여 검색하고 싶다면? 먼저 의미가 좀더 잘 전달되도록 구문의 이름을 변경할 것이다. 그리고 다른 조건을 추가한다.

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG WHERE state = ‘ACTIVE’
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

choose, when, otherwise

우리는 종종 적용 할 모든 조건을 원하는 대신에 한가지 경우만을 원할 수 있다. 자바에서는 switch 구문과 유사하며 마이바티스에서는 choose 엘리먼트를 제공한다.

위 예제를 다시 사용해보자. 지금은 title만으로 검색하고 author가 있다면 그 값으로 검색된다. 둘다 제공하지 않는다면 featured 상태의 blog가 리턴된다.

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG WHERE state = ‘ACTIVE’
  <choose>
    <when test="title != null">
      AND title like #{title}
    </when>
    <when test="author != null and author.name != null">
      AND author_name like #{author.name}
    </when>
    <otherwise>
      AND featured = 1
    </otherwise>
  </choose>
</select>
```

trim, where, set

앞서 예제는 악명높게 다양한 엘리먼트가 사용된 동적 SQL 이다. “if” 예제를 사용해보자.

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  WHERE
  <if test="state != null">
    state = #{state}
  </if>
  <if test="title != null">
    AND title like #{title}
  </if>
  <if test="author != null and author.name != null">
    AND author_name like #{author.name}
  </if>
</select>
```

어떤 조건에도 해당되지 않는다면 어떤 일이 벌어질까? 아마도 다음과 같은 SQL 이 만들어질 것이다.

```
SELECT * FROM BLOG
WHERE
```

아마도 이건 실패할 것이다. 두번째 조건에만 해당된다면 무슨 일이 벌어질까? 아마도 다음과 같은 SQL이 만들어질 것이다.

```
SELECT * FROM BLOG
WHERE
AND title like ‘someTitle’
```

이것도 아마 실패할 것이다. 이 문제는 조건만 가지고는 해결되지 않았다. 이렇게 작성했다면 다시는 이렇게 작성하지 않게 될 것이다.

실패하지 않기 위해서 조금 수정해야 한다. 조금 수정하면 아마도 다음과 같을 것이다.

```
<select id="findActiveBlogLike"
     resultType="Blog">
  SELECT * FROM BLOG
  <where>
    <if test="state != null">
         state = #{state}
    </if>
    <if test="title != null">
        AND title like #{title}
    </if>
    <if test="author != null and author.name != null">
        AND author_name like #{author.name}
    </if>
  </where>
</select>
```

where 엘리먼트는 태그에 의해 컨텐츠가 리턴되면 단순히 “WHERE”만을 추가한다. 게다가 컨텐츠가 “AND”나 “OR”로 시작한다면 그 “AND”나 “OR”를 지워버린다.

만약에 where 엘리먼트가 기대한 것처럼 작동하지 않는다면 trim 엘리먼트를 사용자 정의할 수도 있다. 예를들어 다음은 where 엘리먼트에 대한 trim 기능과 동일하다.

```
<trim prefix="WHERE" prefixOverrides="AND |OR ">
  ...
</trim>
```

override 속성은 오버라이드하는 텍스트의 목록을 제한한다. 결과는 override 속성에 명시된 것들을 지우고 with 속성에 명시된 것을 추가한다.

다음 예제는 동적인 update 구문의 유사한 경우이다. set 엘리먼트는 update 하고자 하는 칼럼을 동적으로 포함시키기 위해 사용될 수 있다. 예를 들어:

```
<update id="updateAuthorIfNecessary">
  update Author
    <set>
      <if test="username != null">username=#{username},</if>
      <if test="password != null">password=#{password},</if>
      <if test="email != null">email=#{email},</if>
      <if test="bio != null">bio=#{bio}</if>
    </set>
  where id=#{id}
</update>
```

여기서 set 엘리먼트는 동적으로 SET 키워드를 붙히고 필요없는 콤마를 제거한다.

아마도 trim 엘리먼트로 처리한다면 아래와 같을 것이다.

```
<trim prefix="SET" suffixOverrides=",">
  ...
</trim>
```

이 경우 접두사는 추가하고 접미사를 오버라이딩 한다.

foreach

동적 SQL 에서 공통적으로 필요한 것은 collection 에 대해 반복처리를 하는 것이다. 종종 IN 조건을 사용하게 된다. 예를들면

```
<select id="selectPostIn" resultType="domain.blog.Post">
  SELECT *
  FROM POST P
  WHERE ID in
  <foreach item="item" index="index" collection="list"
      open="(" separator="," close=")">
        #{item}
  </foreach>
</select>
```

foreach엘리먼트는 매우 강력하고 collection 을 명시하는 것을 허용한다. 엘리먼트 내부에서 사용할 수 있는 item, index두가지 변수를 선언한다. 이 엘리먼트는 또한 열고 닫는 문자열로 명시할 수 있고 반복간에 둘 수 있는 구분자도 추가할 수 있다.

참고 컬렉션 파라미터로 Map이나 배열객체와 더불어 List, Set등과 같은 반복가능한 객체를 전달할 수 있다. 반복가능하거나 배열을 사용할때 index값은 현재 몇번째 반복인지를 나타내고 value항목은 반복과정에서 가져오는 요소를 나타낸다. Map을 사용할때 index는 key객체가 되고 항목은 value객체가 된다.

XML설정 파일과 XML 매핑 파일에 대해서는 이 정도에서 정리하고 다음 섹션에서는 Java API 에 대해 좀더 상세하게 살펴볼 것이다.

bind

`bind` 엘리먼트는 OGNL표현을 사용해서 변수를 만든 뒤 컨텍스트에 바인딩한다. 예를들면

```
<select id="selectBlogsLike" resultType="Blog">
  <bind name="pattern" value="'%' + _parameter.getTitle() + '%'" />
  SELECT * FROM BLOG
  WHERE title LIKE #{pattern}
</select>
```

Multi-db vendor support

"_databaseId" 변수로 설정된 databaseIdProvider가 동적인 코드에도 사용가능하다면 데이터베이스 제품별로 서로다른 구문을 사용할 수 있다. 다음의 예제를 보라:

```
<insert id="insert">
  <selectKey keyProperty="id" resultType="int" order="BEFORE">
    <if test="_databaseId == 'oracle'">
      select seq_users.nextval from dual
    </if>
    <if test="_databaseId == 'db2'">
      select nextval for seq_users from sysibm.sysdummy1"
    </if>
  </selectKey>
  insert into users values (#{id}, #{name})
</insert>
```

Pluggable Scripting Languages For Dynamic SQL

마이바티스 3.2부터는 플러그인 형태로 스크립트 언어를 사용할 수 있다. 그래서 언어별 드라이버를 장착하고 동적 SQL쿼리를 작성할때 그 언어를 사용할 수 있다.

두개의 내장된 언어가 있다.

- xml
- raw

`xml` 언어는 설정하지 않을때 기본으로 사용하는 값이다. `xml` 을 사용하면 이전에 다룬 모든 동적태그를 실행할 수 있다.

`raw` 언어는 사실 기능이 조금 부족하다. `raw` 설정을 사용하면 마이바티스는 파라미터를 치환해서 데이터베이스 드라이버에 구문을 전달한다. 짐작하는 것처럼 `raw` 언어는 `xml` 언어보다 조금더 빠르다.

다음처럼 `lang` 속성을 추가해서 구문에서 사용할 언어를 명시할 수 있다.

```
<select id="selectBlog" lang="raw">
  SELECT * FROM BLOG
</select>
```

또는 매퍼를 사용하는 경우라면 `@Lang` 애노테이션을 사용한다.

```
public interface Mapper {
  @Lang(RawLanguageDriver.class)
  @Select("SELECT * FROM BLOG")
  List<Blog> selectBlog();
}
```

다음의 인터페이스를 구현해서 자신만의 언어 드라이버를 구현할 수도 있다.

```
public interface LanguageDriver {
  ParameterHandler createParameterHandler(MappedStatement mappedStatement, Object parameterObject,
BoundSql boundSql);
  SqlSource createSqlSource(Configuration configuration, XNode script, Class<?> parameterType);
  SqlSource createSqlSource(Configuration configuration, String script, Class<?> parameterType);
}
```
