MyBatis

목차

1.MyBatis 시작하기

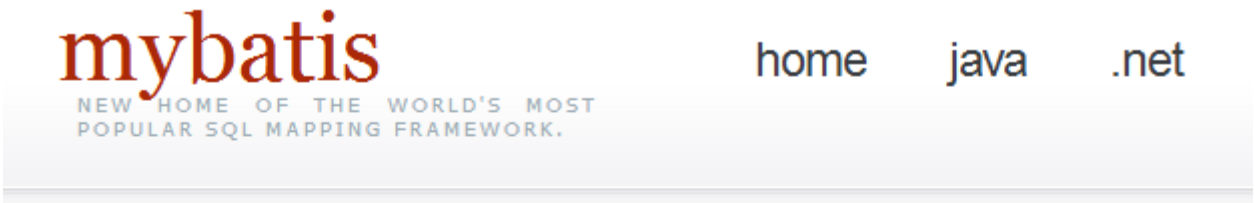
1.1 mybatis 웹 애플리케이션 개발하기

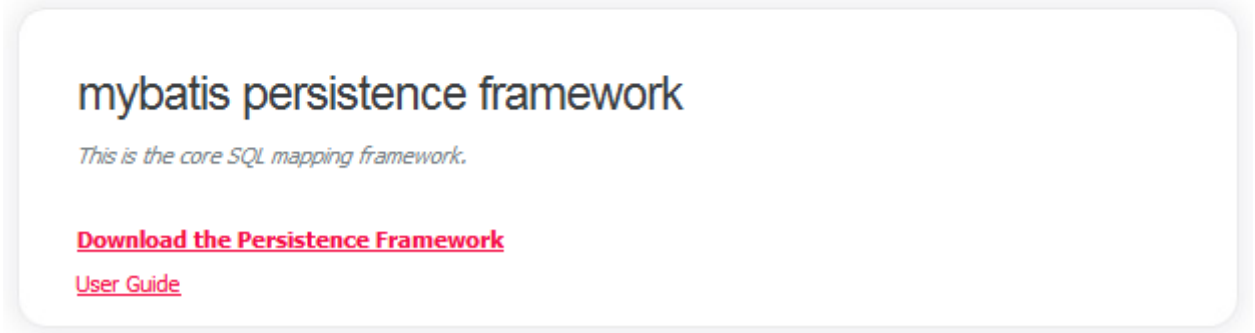
이전 장에서는 간단한 mybatis 애플리케이션을 만들었고, Junit 테스트 코드를 통해 실행해봤다. 실행하는 과정에 출력되는 로그를 통해 mybatis 가 내부에서 JDBC API 를 어떻게 활용하고 있는지도 대략 짐작가능했다.
간단한 애플리케이션 코드에서는 XML 에서 매핑구문을 기술하는 것과 mybatis 자바 API 를 사용해서 실제 호출하는 코드를 본 것이다. 이 간단해 보인 코드가 사실은 실제 mybatis 를 활용하는 대부분의 코드에서 사용될 수 있으며 그 비율은 프로젝트마다 다르겠지만 50%가 넘을수도 있다는 점에서 독자는 mybatis 의 50% 이상을 이미 습득했다고 할 수 있다.
이제부터는 본격적으로 웹애플리케이션을 작성하는 방법을 볼 것이다.
mybatis 코드의 경우, 앞서 본 예제와 크게 다르지 않다.
먼저 본 샘플에서는 “Spring MVC + mybatis 단독” 형태의 예제임을 먼저 알려드린다.

1.1.1 설치하기

A. 수동설치

mybatis 홈페이지의 자바 페이지인 http://www.mybatis.org/java.html 에서 압축 파일을 다운로드한다.





압축파일을 풀어보면 mybatis-x.x.x.jar 형태의 jar 파일이 있는데. 이 파일을 WEB-INF/lib 밑에 두면 사실상 설치는 끝난다.
이 글을 작성하는 시점에 최신버전은 3.1.0 이라 jar 파일명은 mybatis-3.1.0.jar 이다.

B. maven 이용

dependency 에서 groupId 는 org.mybatis 이고, artifactId 는 mybatis 로 설정하면 된다.

```
<dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.1.0</version>
</dependency>
```

C. ivy 이용

maven 과 유사하다. org 는 org.mybatis 이고, name 은 mybatis 이다.

```
<dependency org="org.mybatis" name="mybatis" rev="3.1.0"/>
```

1.1.2 mybatis 설정파일

```
public Integer deleteComment(Long commentNo) {
        SqlSession sqlSession = getSqlSessionFactory().openSession();
        try {
                String statement = "kr.pe.ldg.mybatis.example1.repository.mapper.CommentMapper.deleteComment";
                int result = sqlSession.delete(statement, commentNo);
                if (result > 0) {
                        sqlSession.commit();
                }
                return result;
        } finally {
                sqlSession.close();
        }
}
```

mybatis 설정파일에서 설정가능한 항목을 파악하기 위해서 가장 간단한 방법은 DTD 를 보는 것이다.
DTD 의 위치는 http://mybatis.org/dtd/mybatis-3-config.dtd 이다.
설정가능한 항목은 10 개다.

A. properties

많은 프로젝트에서는 개발 장비와 실제 리얼 장비로 구분해서 개발하는 경우가 많다. 이러한 경우에 설정에 대해서는 외부 프로퍼티 파일로 분리하고 서버별로 프로퍼티 파일을 선택해서 배포하는 형태를 주로 택한다. mybatis 의 properties 는 외부 프로퍼티 파일을 읽어서 설정파일내 변수처럼 사용할 수 있도록 한다.

외부 프로퍼티 파일의 설정을 아래와 같이 하고 파일명은 mybatis.properties 로 정했다.

```
jdbc.url=jdbc:mysql://localhost:3306/mybatis_example
jdbc.driver= com.mysql.jdbc.Driver
jdbc.username=mybatis
jdbc.password=mybatis
```

대개의 경우, properties 의 resource 속성에 파일을 지정하면 된다. resource 속성은 클래스패스 기준으로 프로퍼티를 찾는다.

```
<properties resource="mybatis.properties"></properties>
```

모든 설정을 외부 프로퍼티 파일에 둔다면 상관없지만 프로퍼티 중에 일부는 서버별로 변경없이 사용할 수 있다. 그러한 경우 아래처럼 설정하면 된다.

```
<properties resource="mybatis.properties">
  <property name="jdbc.driver" value="com.mysql.jdbc.Driver"/>
  <property name="jdbc.username" value="mybatis"/>
  <property name="jdbc.password" value="mybatis"/>
</properties>
```

만약에 프로퍼티 파일을 클래스패스 기준이 아닌 절대경로로 잡아야 할 경우, url 속성을 사용하면 된다.

```
<properties url="file:d:₩mybatis.properties"></properties>
```

이렇게 설정한 후 사용할때는 ${key} 형태로 사용하면 된다. 위 프로퍼티 파일의 내용을 사용한다면 아래와 같다.

```
<property name="url" value="${jdbc.url}" />
```

B. settings

settings 에서 설정되는 각종 프로퍼티는 mybatis 자바 API 가 작동할때의 규칙을 정의한다. mybatis 가 마이너 업그레이드를 진행하면서 추가되는 항목도 종종 있고 몇가지 항목은 JDBC 드라이버에 따라 지원되지 항목도 있다.

1. cacheEnabled : 매퍼 XML 또는 매퍼 애노테이션에서 설정하는 캐시 설정을 전체 기본 설정으로 가져갈지에 대한 옵션이다. mybatis 이 기본 캐시는 분산 캐시가 아니라 로컬 캐시라 서버가 여러대인 경우 데이터 변경에 대해 캐시에 반영하는 게 문제가 될 수 있다. 물론 Cacheonix 와 같은 분산 캐시가 개발 중에 있긴 하지만 mybatis 의 기본 캐시가 아니라서 캐시에 대해서는 신중히 결정해야 한다. 디폴트는 true 이다.
2. lazyLoadingEnabled : 늦은 로딩 사용 여부에 대한 옵션이다. 디폴트는 true 이다.
3. aggressiveLazyLoading : 점진적인 늦은 로딩 사용 여부에 대한 옵션이다. 디폴트는 true 이다.
4. multipleResultSetsEnabled : 한개의 구문에서 여러개의 결과셋 허용 여부에 대한 옵션이다. 드라이버에 따라 지원여부가 확인해서 사용해야 한다. 디폴트는 true 이다.
5. useColumnLabel : 칼럼명 대신 칼럼라벨 사용 여부에 대한 옵션이다. 디폴트는 true 이다.
6. useGeneratedKeys : 생성키 사용 여부에 대한 옵션이다. 디폴트는 false 이다.
7. autoMappingBehavior : 자동 매핑에 대한 옵션이다. 선택가능한 값은 NONE, PARTIAL, FULL 인데 디폴트는 PARTIAL 이다. 자동 매핑에 대해서는 뒤에서 좀더 다뤄보도록 한다.
8. defaultExecutorType : 디폴트 실행타입 옵션이다. 선택가능한 값은 Statement 객체를 재사용하지 않는 SIMPLE, PreparedStatement 객체를 재사용하는 REUSE, Statement 를 재사용하고 작업을 배치로 일괄처리하는 BATCH 가 있다. 디폴트는 SIMPLE 이다.
9. defaultStatementTimeout : 데이터베이스 요청에 대한 타임아웃 설정이다. 대부분의 JDBC 드라이버가 자체 타입아웃 설정이 있는데, 이 값보다 짧게 가져가야 할 경우 사용하면 된다. 양수로 셋팅해야 하며 디폴트는 설정하지 않는다.
10. safeRowBoundsEnabled
11. mapUnderscoreToCamelCase : 테이블 칼럼명은 대개 언더바를 통해 구분하는데, 자바의 코딩 컨벤션은 CamelCase 형태를 취한다. 이 경우 언더바 형태를 CamelCase 형태로 자동 매핑할지에 대한 옵션이다. 이 옵션을 사용하지 않으면서 테이블명의 칼럼명은 언더바로 구분하고 모델은 CamelCase 를 사용할 경우 쿼리문에 별칭을 사용하거나 별도의 결과맵을 사용해야 한다. 디폴트는 false 이다.
12. localCacheScope : 캐시의 저장범위를 정한다. SqlSession 객체를 기준으로 캐시할때는 SESSION, 구문별로 캐시할때는 STATEMENT 를 선택하면 된다. 디폴트는 SESSION 이다.
13. jdbcTypeForNull :
14. lazyLoadTriggerMethods :

defaultStatementTimeout 를 제외하면 대부분 디폴트값을 가지고 있다. defaultStatementTimeout 도 사실은 JDBC 별로 디폴트 값을 가지고 있는터라 사실상 대부분의 옵션은 디폴트 값을 가지고 있어서 그대로 사용하면 되는 셈이다. 즉 변경이 필요한 옵션에 대해서만 설정을 하면 되는데 필자의 경우에는 3 개 정도 수정해서 쓰면 대부분의 애플리케이션의 요구사항을 만족할 것으로 생각한다.

```
<settings>
  <setting name="cacheEnabled" value="false"/>
  <setting name="useGeneratedKeys" value="true"/>
  <setting name="mapUnderscoreToCamelCase" value="true"/>
</settings>
```

C. typeAliases

대부분의 모델 클래스는 패키지 경로를 함께 사용하면 굉장히 긴 문자열이 된다. 샘플에서 사용한 Comment 모델은 패키지 경로까지 모두 표기하면 kr.pe.ldg.mybatis.example1.model.Comment 정도이다. 매핑구문의 파라미터나 결과타입에서 작성하려고 보면 굉장히 긴 문자열이라 종종 오타가 날수도 있고 문자열 자체가 길다 보니 귀찮은 작업임에는 충분히 짐작하고 남을만하다.
이를 위해 타입별 별칭을 설정할 수 있도록 한다.

```
<typeAliases>
  <typeAlias type="kr.pe.ldg.mybatis.example1.model.Comment" alias="Comment" />
</typeAliases>
```

이렇게 설정하고 매핑구문에서 파라미터나 결과타입에 Comment 를 사용하면 kr.pe.ldg.mybatis.example1.model.Comment 로 인식하게 된다.
ibatis 에서는 이렇게 XML 에서만 타입별칭을 설정할 수 있었다. 하지만 mybatis 에서는 애노테이션을 설정하는 방법을 추가로 제공한다.

```
@Alias("Comment")
public class Comment {

}
```

원시타입이나 흔하게 사용되는 자바 타입에 대해서는 mybatis 내부적으로 처리된 별칭 또한 존재한다.

별칭	매핑된 타입
_byte	byte
_long	long
_short	short
_int	int
_integer	int
_double	double
_float	float
_boolean	boolean
string	String
byte	Byte
long	Long
short	Short
int	Integer
integer	Integer

double	Double
float	Float
boolean	Boolean
date	Date
decimal	BigDecimal
bigdecimal	BigDecimal
object	Object
map	Map
hashmap	HashMap
list	List
arraylist	ArrayList
collection	Collection
iterator	Iterator

D. typeHandlers

PreparedStatement 에서 파라미터를 셋팅하거나 결과셋을 가져올 때 테이블 칼럼 각각의 값을 자바의 적절한 타입으로 셋팅해서 가져오기 위해 사용한다. mybatis 에서 이미 정의된 타입 핸들러들이다.

타입 핸들러	자바 타입	JDBC 타입
BooleanTypeHandler	java.lang.Boolean, boolean	어떤 호환가능한 BOOLEAN
ByteTypeHandler	java.lang.Byte, byte	어떤 호환가능한 NUMERIC 또는 BYTE
ShortTypeHandler	java.lang.Short, short	어떤 호환가능한 NUMERIC 또는 SHORT INTEGER
IntegerTypeHandler	java.lang.Integer, int	어떤 호환가능한 NUMERIC 또는 INTEGER
LongTypeHandler	java.lang.Long, long	어떤 호환가능한 NUMERIC 또는 LONG INTEGER
FloatTypeHandler	java.lang.Float, float	어떤 호환가능한 NUMERIC 또는 FLOAT
DoubleTypeHandler	java.lang.Double, double	어떤 호환가능한 NUMERIC 또는 DOUBLE
BigDecimalTypeHandler	java.math.BigDecimal	어떤 호환가능한 NUMERIC 또는 DECIMAL
StringTypeHandler	java.lang.String	CHAR, VARCHAR
ClobTypeHandler	java.lang.String	CLOB, LONGVARCHAR
NStringTypeHandler	java.lang.String	NVARCHAR, NCHAR
NClobTypeHandler	java.lang.String	NCLOB
ByteArrayTypeHandler	byte[]	어떤 호환가능한 byte 스트림 타입
BlobTypeHandler	byte[]	BLOB, LONGVARBINARY
DateTypeHandler	java.util.Date	TIMESTAMP
DateOnlyTypeHandler	java.util.Date	DATE

TimeOnlyTypeHandler	java.util.Date	TIME
SqlTimestampTypeHandler	java.sql.Timestamp	TIMESTAMP
SqlDateTypeHandler	java.sql.Date	DATE
SqlTimeTypeHandler	java.sql.Time	TIME
ObjectTypeHandler	Any	OTHER, 또는 명시하지 않는 경우
EnumTypeHandler	Enumeration Type	VARCHAR - 문자열 호환타입.
EnumOrdinalTypeHandler	Enumeration Type	위치가 저장된것과 같은 어떤 호환가능한 NUMERIC 또는 DOUBLE

대부분의 경우 이미 정의된 타입핸들러를 사용하면 된다.
별도의 타입핸들러를 만들기 위해서는 org.apache.ibatis.type.BaseTypeHandler<T> 를 확장하면 된다.

```
@MappedJdbcTypes(JdbcType.VARCHAR)
public class ExampleTypeHandler extends BaseTypeHandler<String> {

  @Override
  public void setNonNullParameter(PreparedStatement ps, int i, String parameter, JdbcType jdbcType) throws SQLException {
    ps.setString(i, parameter);
  }

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

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

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

구현체를 만들어서 타입핸들러를 명시하는 방법은 두가지가 있다.
typeHandler 를 사용하는 방법과 package 를 통해 해당 패키지내 타입핸들러를 자동검색하게 하는 방법이다.
단 package 를 사용해서 자동검색하는 방식은 애노테이션으로 명시한 타입핸들러만 검색이 된다.

```
<typeHandlers>
<typeHandler handler="kr.pe.ldg.mybatis.example.ExampleTypeHandler"/>
<package name=" kr.pe.ldg.mybatis.example "/>
</typeHandlers>
```

앞서 언급하긴 했지만 mybatis 는 SQL 구문과 모델을 매핑하는 역할을 담당한다. 그리고 그 내부에서는 JDBC api 를 사용한다. 이 두가지 설명에서 이해해야 하는 것이 mybatis 는 이 두가지 외 더이상의 것을 하지 않는다. 즉 JDBC 의 결과셋의 메타데이터(java.sql.ResultSetMetaData 와 같은)를 추출해서 분석해서 자동으로 처리하는 기능은 하지 않는다. 즉 테이블의 칼럼 타입과 자바 모델의 변수타입이 명확한 경우에는 상관없으나 혼란의 여지가 있다면 반드시 그에 대한 정보를 셋팅해줘야 의도된 데로 작동하게 된다.

MyBatis 는 제네릭타입을 체크해서 TypeHandler 로 다루고자 하는 자바타입을 알것이다. 하지만 두가지 방법으로 이 행위를 재정의할 수 있다:
typeHandler 요소의 javaType 속성 추가(예제: javaType="String")
TypeHandler 클래스에 관련된 자바타입의 목록을 정의하는 @MappedTypes 애노테이션 추가. javaType 속성도 함께 정의되어 있다면 @MappedTypes 는 무시된다.

관련된 JDBC 타입은 두가지 방법으로 명시할 수 있다:
typeHandler 요소에 jdbcType 속성 추가(예제: jdbcType=VARCHAR).
TypeHandler 클래스에 관련된 JDBC 타입의 목록을 정의하는 @MappedJdbcTypes 애노테이션 추가. jdbcType 속성도 함께 정의되어 있다면 @MappedJdbcTypes 는 무시된다.

E. objectFactory

대개의 경우에는 결과셋에 대응되는 모델을 만들 때는 칼럼명(또는 칼럼별칭)에 대응되는 setter 메서드를 호출해서 모델을 만들어 리턴한다. 그리고 추가로 값을 셋팅하고자 할때는 이 모델에서 다시 셋팅하는 과정을 거치는 게 일반적이라고 생각한다. 하지만 이 과정을 mybatis 매핑 과정에서 한꺼번에 처리하고자 할때는 ObjectFactory 를 사용할 수 있다.
구현방식은 org.apache.ibatis.reflection.factory.ObjectFactory 인터페이스를 구현하거나 간단히 org.apache.ibatis.reflection.factory.DefaultObjectFactory 를 확장하는 방법이 있다.

```
public class ExampleObjectFactory extends DefaultObjectFactory {
  public Object create(Class type) {
    return super.create(type);
  }
  public Object create(Class type, List<Class> constructorArgTypes, List<Object> constructorArgs) {
    return super.create(type, constructorArgTypes, constructorArgs);
  }
  public void setProperties(Properties properties) {
    super.setProperties(properties);
  }
}
```

```
<objectFactory type="kr.pe.ldg.mybatis.example.ExampleObjectFactory">
  <property name="someProperty" value="100"/>
</objectFactory>
```

F. objectWrapperFactory

G. plugins

mybatis 가 매핑구문을 실행하는 과정에서 특정 시점의 처리를 가로채서 부가적인 작업이 가능하도록 해준다. 로그를 찍어 줄수도 있고, 파라미터에 대해 공통적으로 타입체크 또는 결과셋에 대한 처리를 추가할 수도 있을 것이다.

1. Executor (update, query, flushStatements, commit, rollback, getTransaction, close, isClosed)
2. ParameterHandler (getParameterObject, setParameters)
3. ResultSetHandler (handleResultSets, handleOutputParameters)
4. StatementHandler (prepare, parameterize, batch, update, query)

```
@Intercepts({@Signature(
  type= Executor.class,
  method = "update",
  args = {MappedStatement.class,Object.class})})
public class ExamplePlugin implements Interceptor {
  public Object intercept(Invocation invocation) throws Throwable {
    return invocation.proceed();
  }
  public Object plugin(Object target) {
    return Plugin.wrap(target, this);
  }
  public void setProperties(Properties properties) {
  }
}
```

```
<plugins>
  <plugin interceptor="org.mybatis.example.ExamplePlugin">
    <property name="someProperty" value="100"/>
  </plugin>
</plugins>
```

H. environments

mybatis 의 환경설정은 트랜잭션 관리자와 데이터소스 설정이다. 이 환경설정은 mybatis 단독으로 사용할때만 필요하며
Spring 연동모듈을 사용할 경우에는 필요없다. Spring 연동모듈에서는 트랜잭션과 데이터소스 설정을 Spring 에서 제어하기 때문이다.

첫번째로 설정가능한 항목은 transactionManager (트랜잭션 관리자)이다.
트랜잭션 관리자에서 셋팅해야 할 값은 type 과 각종 프로퍼티이다.
type 속성으로 선택가능한 값은 JDBC 와 MANAGED 이다.

1. JDBC : mybatis 자바 API 에서 제공하는 commit(), rollback() 메서드 등을 사용해서 트랜잭션을 관리하는 방식이다.
2. MANAGED : mybatis 자바 API 보다는 컨테이너가 직접 트랜잭션을 관리하는 방식이다.

두번째로 선택가능한 항목은 dataSource(데이터소스)이다.
데이터소스에서 셋팅해야 할 값은 type 과 각종 프로퍼티이다.
type 속성은 선택가능한 값은 UNPOOLED, POOLED, JNDI 이다.

1. UNPOOLED : 요청마다 데이터베이스 연결을 새롭게 생성하고 처리 후 완전히 해제한다. 데이터베이스 연결을 매번 생성하는 것이 성능적으로 영향이 있기 때문에 간단한 테스트용 애플리케이션을 제외하고 대부분의 서비스용 애플리케이션에서는 사용하지 않는다. 이 타입에서 선택가능한 프로퍼티는 5 가지 정도 된다.

- driver - JDBC 드라이버의 클래스명
- url - 데이터베이스 연결을 위한 URL 정보
- username - 데이터베이스 연결을 위한 계정명
- password - 데이터베이스 연결을 위한 계정의 패스워드
- defaultTransactionIsolationLevel - 디폴트 트랜잭션 격리 레벨

2. POOLED : 일정수의 데이터베이스 연결을 풀에 넣어두고 필요할때마가 가져다가 사용하고 사용하고 나면 다시 풀에 넣는다. 매번 데이터베이스 연결을 생성하지 않기 때문에 대부분의 웹 애플리케이션에서 기본적으로 사용한다. 앞서 본 UNPOOLED 에서 설정가능한 프로퍼티와 그외 추가로 프로퍼티를 지정할 수 있다.

- poolMaximumActiveConnections - 주어진 시간에 존재할 수 있는 활성화된(사용중인) 커넥션의 수. 디폴트는 10 이다.
- poolMaximumIdleConnections - 주어진 시간에 존재할 수 있는 유휴 커넥션의 수
- 강제로 리턴되기 전에 풀에서 “체크아웃” 될 수 있는 커넥션의 시간. 디폴트는 20000ms(20 초)
- poolTimeToWait - 풀이 로그 상태를 출력하고 비정상적으로 긴 경우 커넥션을 다시 얻을려고 시도하는 로우 레벨 셋팅. 디폴트는 20000ms(20 초)
- poolPingQuery - 커넥션이 작업하기 좋은 상태이고 요청을 받아서 처리할 준비가 되었는지 체크하기 위해 데이터베이스에 던지는 핑쿼리(Ping Query). 디폴트는 “핑 쿼리가 없음” 이다. 이 설정은 대부분의 데이터베이스로 하여금 에러메시지를 보게 할수도 있다.
- poolPingEnabled - 핑쿼리를 사용할지 말지를 결정. 사용한다면, 오류가 없는(그리고 빠른) SQL 을 사용하여 poolPingQuery 프로퍼티를 셋팅해야 한다. 디폴트는 false 이다.
- poolPingConnectionsNotUsedFor - poolPingQuery 가 얼마나 자주 사용될지 설정한다. 필요이상의 핑을 피하기 위해 데이터베이스의 타임아웃 값과 같을 수 있다. 디폴트는 0 이다. 디폴트 값은 poolPingEnabled 가 true 일 경우에만, 모든 커넥션이 매번 핑을 던지는 값이다.

3. JNDI : 컨테이너의 JNDI 컨텍스트를 참조한다. 앞서 본 두가지 타입과는 다르며, 설정가능한 프로퍼티는 2 개뿐이다.

- initial_context - 이 프로퍼티는 InitialContext 에서 컨텍스트를 찾기(예를 들어, initialContext.lookup(initial_context)) 위해 사용된다. 이 프로퍼티는 선택적인 값이다. 이 설정을 생략하면, data_source 프로퍼티가 InitialContext 에서 직접 찾을 것이다.
- data_source - DataSource 인스턴스의 참조를 찾을 수 있는 컨텍스트 경로이다. initial_context 룩업을 통해 리턴된 컨텍스트에서 찾을 것이다. initial_context 가 지원되지 않는다면, InitialContext 에서 직접 찾을 것이다.

이 환경설정을 ibatis 에서는 한개만 설정가능했으나 mybatis 에서는 여러개 설정하고 SqlSessionFactory 객체를 생성하는 시점에 선택할 수 있도록 했다. 필자의 경우 서버환경마다 대부분 동일하고 JDBC url 과 계정 정보 정도만 달라서 여러개의 환경설정을 두는 방식보다는 프로퍼티 파일을 서버별로 두는 방식을 사용하기 때문에 그 효용성에 대해서는 크게 와닿지 않으나 사용자는 사람들의 목적에 따라 분명 ibatis 때보다는 좋은 점이 있을 것이다.

I. databaseIdProvider

J. mappers

매핑 구문을 지정하는 방법이다. ibatis 에서는 매핑구문을 정의하는 방법이 2 가지뿐이었지만 mybatis 에서는 4 가지를 제공한다.

1. 클래스 패스내 XML 매핑 파일 지정(resource 속성)
2. url 을 사용한 XML 매핑 파일 지정(url 속성)
3. 매핑 애노테이션을 사용하는 인터페이스 위치 지정(class 속성)
4. 패키지 지정으로 패키지내 자동으로 매핑 구문 검색(name 속성)

```
<mappers>
  <mapper resource="org/mybatis/builder/AuthorMapper.xml"/>
  <mapper url="file:///var/mappers/AuthorMapper.xml"/>
  <mapper class="org.mybatis.builder.AuthorMapper"/>
  <package name="org.mybatis.builder"/>
</mappers>
```

K. 정리

여기에서 대부분의 프로젝트를 작성할 때 가장 공통적으로 사용되는 요소는 properties, settings, typeAliases, environments, mappers 의 5 가지이다. 이 요소들은 명칭이나 그 역할이 조금 달라지긴 했으나 이전 버전인 ibatis 에도 거의 동일하게 있었다.. 그외 추가된 요소는 mybatis 의 스펙을 결정하면서 도움이 되리라 생각되고 실제 몇몇 사용자들이 요청해서 생긴 것들이다. 단 공통적으로 사용할 만한 기능이 아닌게 많아서 사용시 원하는 요구사항에 충분히 맞는지 확인 후 사용해야 한다.

1.1.3 매퍼 XML 과 매퍼 애노테이션

ibatis 에서는 매핑 구문 정의가 XML 에서만 가능했다. 하지만 mybatis 에서는 XML 과 애노테이션 두가지 방식을 지원한다. 실제 사용하는 경우로 따져보면 3 가지 형태로 사용된다.

1. XML 만 사용
2. 애노테이션만 사용
3. XML 과 애노테이션을 혼용해서 사용

B. XML 만 사용하는 경우

XML 을 사용하는 경우는 ibatis 를 사용해본 사람이라면 XML 요소의 명칭과 동적 SQL 부분의 변경사항 정도를 제외하면 기존과 거의 동일하기 때문에 익히는데 시간이 얼마 들지 않을 것이다. 앞서 우리는 앞 장에서 XML 을 사용하는 경우에 대한 간단한 에제를 봤다. 이미 본 예제와 함께 동적 SQL 을 사용하는 경우의 예제를 추가해서 다시 보자

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="kr.pe.ldg.mybatis.example1.repository.mapper.CommentMapper">
  <select id="selectCommentByPrimaryKey" parameterType="long" resultType=" Comment">
    SELECT
        comment_no AS commentNo,
```

```
        user_id AS userId,
        comment_content AS commentContent,
        reg_date AS regDate
      FROM COMMENT
      WHERE comment_no = #{commentNo}
  </select>
  <select id="selectCommentByCondition" parameterType="hashmap" resultType="Comment">
    SELECT
        comment_no,
        user_id,
        comment_content,
        reg_date
      FROM COMMENT
      <where>
            <if test="commentNo != null">
        comment_no = #{commentNo}
            </if>
      </where>
  </select>
</mapper>
```

두개의 매핑구문이 있다. 각각을 mybatis 자바 API 를 통해 사용할때는 명명공간을 반드시 붙여서 매핑구문 아이디를 사용해야 한다.

1. selectCommentByPrimaryKey

➔ sqlSession.selectOne("kr.pe.ldg.mybatis.example1.repository.mapper.CommentMapper.selectCommentByPrimaryKey", 1L);

2. selectCommentByCondition

➔ sqlSession.selectList("kr.pe.ldg.mybatis.example1.repository.mapper.CommentMapper.selectCommentByCondition", new HashMap<String, Object>());

C. 애노테이션만 사용하는 경우

XML 을 사용하는 경우에서 본 selectCommentByPrimaryKey, selectCommentByCondition 모두 애노테이션 방식으로 변경해보자.
애노테이션을 사용할 경우의 단점은, 동적 SQL 을 작성하기가 어렵다는 것이다.
이러한 경우를 위해 @SelectProvider 애노테이션을 제공하고 있다.

```
public interface CommentMapper {
      @Select({
            "SELECT ",
            "comment_no, user_id, comment_content, reg_date ",
            "FROM COMMENT ",
            "WHERE comment_no = #{commentNo}"
            })
      Comment selectCommentByPrimaryKey(Long commentNo);
```

```
    @SelectProvider(type=CommentSqlProvider.class, method="selectCommentByCondition")
    List<Comment> selectCommentByCondition(Map<String, Object> condition);
}
```

ibatis 와 달리 mybatis 에서는 동적 SQL 을 자바 코드로 작성할 수 있도록 API 가 추가되었다. 대표적인 경우가 샘플에서 보이는 BEGIN, SELECT, FROM 등이다. 여기서 자세히 다루고 싶지만 @SelectProvider 와 XML 동적 SQL 은 그 내용이 많아 뒤에서 다시 다루도록 하겠다.

```
public class CommentSqlProvider {
    public String selectCommentByCondition(CommentExample example) {
        BEGIN();
        SELECT("comment_no");
        SELECT("user_id");
        SELECT("comment_content");
        SELECT("reg_date");
        FROM("comment");
        applyWhere(example, false);

        return SQL();
    }
}
```

D. XML 과 애노테이션을 혼용해서 사용하는 경우

XML 을 사용하는 경우에서 selectCommentByPrimaryKey, selectCommentByCondition 두가지를 봤다. 애노테이션 형태를 보여주기 위해 임의로 selectCommentByPrimaryKey 를 애노테이션 방식을 변경을 했다.
그렇기 때문에 selectCommentByCondition 은 여전히 XML 에 그대로 남는다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace="kr.pe.ldg.mybatis.example1.repository.mapper.CommentMapper">
  <select id="selectCommentByCondition" parameterType="hashmap" resultType="Comment">
    SELECT
            comment_no,
            user_id,
            comment_content,
            reg_date
          FROM COMMENT
          <where>
                  <if test="commentNo != null">
              comment_no = #{commentNo}
```

```
                </if>
            </where>
        </select>
    </mapper>
```

하지만 selectCommentByPrimaryKey 는 아래처럼 애노테이션으로 수정하였다.
필자의 경우, 쿼리문이 지면에서 보기 편하도록 @Select 애노테이션에 배열형태로 넣었지만 한개의 문자열로 넣어도 된다. 배열 형태로 넣을 경우, 모든 문자열을 순서대로 붙여서 한 문자열로 만들어서 처리한다.
애노테이션을 사용할 경우의 단점은, 동적 SQL 을 작성하기가 어렵다는 것이다.
이러한 경우를 위해 @SelectProvider 애노테이션을 제공하고 있다.
@SelectProvider 와 XML 동적 SQL 은 그 내용이 많아 뒤에서 다시 다루도록 하겠다.

```
public interface CommentMapper {
        @Select({
                "SELECT ",
                "comment_no, user_id, comment_content, reg_date ",
                "FROM COMMENT ",
                "WHERE comment_no = #{commentNo}"
                })
        Comment selectCommentByPrimaryKey(Long commentNo);

        //      @Select("SELECT comment_no, user_id, comment_content, reg_date FROM COMMENT WHERE comment_no = #{commentNo}")
        List<Comment> selectCommentByCondition(Map<String, Object> condition);
}
```

1.1.4 트랜잭션 관리

A. SqlSessionFactory

mybatis 의 전반적인 처리는 SqlSession 객체가 담당한다. 그렇기 때문에 SqlSession 객체가 생성되는 시점에 트랜잭션에 관련한 속성이 셋팅이 된다. 그래서 트랜잭션을 처리하는 방법을 보기 전에 SqlSession 객체가 생성될때의 형태를 봐두는게 좋다. 먼저 SqlSession 객체를 생성하는 SqlSessionFactory 의 메서드를 살펴보자.

1. SqlSession openSession()
2. SqlSession openSession(boolean autoCommit)
3. SqlSession openSession(Connection connection)
4. SqlSession openSession(TransactionIsolationLevel level)
5. SqlSession openSession(ExecutorType execType,TransactionIsolationLevel level)
6. SqlSession openSession(ExecutorType execType)
7. SqlSession openSession(ExecutorType execType, boolean autoCommit)
8. SqlSession openSession(ExecutorType execType, Connection connection)

B. SqlSession 객체의 디폴트 속성

SqlSessionFactory 가 제공하는 메서드가 이렇게 많긴 하지만 실제로는 대부분 openSession()으로 처리가 가능하다.
openSession()메서드를 사용할 경우, 앞서 본 mybatis 설정파일의 내용을 그대로 가져간다. 그외 메서드는 특별한 경우 mybatis 의 설정을 그대로 사용하지 않고 처리하기 위함인데, 실제 개발과정에서 그러한 경우가 많지 않으리라 생각한다.
openSession()메서드를 사용하여 SqlSession 객체를 생성하게 되면 생성된 객체의 성격은 디폴트 상태가 된다.

1. 트랜잭션은 시작된다.
2. mybatis 설정파일의 설정된 형태의 데이터소스를 가진다.
3. 트랜잭션에 관련한 격리 레벨이나 전파 설정은 설정 내용에 의해 셋팅된다.
4. PreparedStatement 는 재사용되지 않고 처리는 배치형태가 아니다.

C. SqlSession 이 제공하는 메서드

SqlSession 객체가 제공하는 메서드는 많지만 대략적으로 보면 아래와 같다.

1. <T> T selectOne(String statement)
2. <E> List<E> selectList(String statement)
3. <K,V> Map<K,V> selectMap(String statement, String mapKey)
4. int insert(String statement)
5. int update(String statement)
6. int delete(String statement)

D. 트랜잭션 제어 메서드

1. void commit()
2. void commit(boolean force)
3. void rollback()
4. void rollback(boolean force)

여기서 독특한 것은 boolean 타입의 파라미터이다. mybatis api 에서 입력/수정/삭제에 대한 호출이 한번도 없었다고 하면 명시적으로 커밋이나 롤백을 하지 않는다. 단 boolean 타입의 파라미터를 true 로 지정하면 입력/수정/삭제의 호출이 있었는지에 대해 체크하지 않고 무조건 커밋이나 롤백을 하도록 처리한다.
mybatis 를 단독으로 사용할때는 mybatis 의 SqlSession 객체의 commit()이나 rollback()메서드를 사용하면 된다. 하지만 Spring 이나 Google Guice 와 연동하는 경우, 트랜잭션에 대한 제어는 mybatis 가 담당하지 않고 그 역할이 Spring 이나 Google Guice 로 위임된다. Spring 연동시 트랜잭션 처리를 뒤에서 다시 다루도록 한다.

1.1.5 좀더 복잡한 매핑 규칙 정의

A. constructor

B. association

C. collection

D. discriminator