설치

마이바티스를 사용하기 위해 mybatis-x.x.x.jar 파일을 클래스패스에 두어야 한다.

메이븐을 사용한다면 pom.xml에 다음의 설정을 추가하자.

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

XML에서 SqlSessionFactory 빌드하기

모든 마이바티스 애플리케이션은 SqlSessionFactory 인스턴스를 사용한다. SqlSessionFactory인스턴스는 SqlSessionFactoryBuilder를 사용하여 만들수 있다. SqlSessionFactoryBuilder는 XML설정파일에서 SqlSessionFactory인스턴스를 빌드할 수 있다.

XML파일에서 SqlSessionFactory인스턴스를 빌드하는 것은 매우 간단한다. 설정을 위해 클래스패스 자원을 사용하는 것을 추천하나 파일 경로나 `file://` URL로부터 만들어진 InputStream인스턴스를 사용할 수도 있다. 마이바티스는 클래스패스와 다른 위치에서 자원을 로드하는 것으로 좀더 쉽게 해주는 Resources 라는 유틸성 클래스를 가지고 있다.

```
String resource = "org/mybatis/example/mybatis-config.xml";
InputStream inputStream = Resources.getResourceAsStream(resource);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(inputStream);
```

XML설정파일에서 지정하는 마이바티스의 핵심이 되는 설정은 트랜잭션을 제어하기 위한 TransactionManager과 함께 데이터베이스 Connection인스턴스를 가져오기 위한 DataSource 를 포함한다. 세부적인 설정은 조금 뒤에 보고 간단한 예제를 먼저보자.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "https://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

좀 더 많은 XML 설정방법이 있지만 위 예제에서는 가장 핵심적인 부분만을 보여주고 있다. XML 가장 위부분에서는 XML 문서의 유효성체크를 위해 필요하다. environment엘리먼트는 트랜잭션 관리와 커넥션 풀링을 위한 환경적인 설정을 나타낸다. mappers엘리먼트는 SQL 코드와 매핑 정의를 가지는 XML 파일인 mapper 의 목록을 지정한다.

XML 을 사용하지 않고 SqlSessionFactory 빌드하기

XML 보다 자바를 사용해서 직접 설정하길 원한다면 XML 파일과 같은 모든 설정을 제공하는 Configuration 클래스를 사용하면 된다.

```
DataSource dataSource = BlogDataSourceFactory.getBlogDataSource();
TransactionFactory transactionFactory = new JdbcTransactionFactory();
Environment environment = new Environment("development", transactionFactory, dataSource);
Configuration configuration = new Configuration(environment);
configuration.addMapper(BlogMapper.class);
SqlSessionFactory sqlSessionFactory = new SqlSessionFactoryBuilder().build(configuration);
```

이 설정에서 추가로 해야 할 일은 Mapper클래스를 추가하는 것이다. Mapper클래스는 SQL 매핑 애노테이션을 가진 자바 클래스이다. 어쨌든 자바 애노테이션의 몇가지 제약과 몇가지 설정방법의 복잡함에도 불구하고 XML 매핑은 세부적인 매핑을 위해 언제든 필요하다. 좀더 세부적인 내용은 조금 뒤 다시 보도록 하자.

SqlSessionFactory 에서 SqlSession 만들기

SqlSessionFactory 이름에서 보듯이 SqlSession 인스턴스를 만들수 있다. SqlSession 은 데이터베이스에 대해 SQL명령어를 실행하기 위해 필요한 모든 메소드를 가지고 있다. 그래서 SqlSession 인스턴스를 통해 직접 SQL 구문을 실행할 수 있다. 예를 들면 :

```
try (SqlSession session = sqlSessionFactory.openSession()) {
  Blog blog = session.selectOne("org.mybatis.example.BlogMapper.selectBlog", 101);
}
```

이 방법이 마이바티스의 이전버전을 사용한 사람이라면 굉장히 친숙할 것이다. 하지만 좀더 좋은 방법이 생겼다. 주어진 SQL 구문의 파라미터와 리턴값을 설명하는 인터페이스(예를 들면, BlogMapper.class)를 사용하여 문자열 처리 오류나 타입 캐스팅 오류 없이 좀더 타입에 안전하고 깔끔하게 실행할 수 있다.

예를들면:

```
try (SqlSession session = sqlSessionFactory.openSession()) {
  BlogMapper mapper = session.getMapper(BlogMapper.class);
  Blog blog = mapper.selectBlog(101);
}
```

그럼 자세히 살펴보자.

매핑된 SQL 구문 살펴보기

이 시점에 SqlSession이나 Mapper클래스가 정확히 어떻게 실행되는지 궁금할 것이다. 매핑된 SQL 구문에 대한 내용이 가장 중요하다. 그래서 이 문서 전반에서 가장 자주 다루어진다. 하지만 다음의 두가지 예제를 통해 정확히 어떻게 작동하는지에 대해서는 충분히 이해하게 될 것이다.

위 예제처럼 구문은 XML이나 애노테이션을 사용해서 정의할 수 있다. 그럼 먼저 XML 을 보자. 마이바티스가 제공하는 대부분의 기능은 XML을 통해 매핑 기법을 사용한다. 이전에 마이바티스를 사용했었다면 쉽게 이해되겠지만 XML 매핑 문서에 이전보다 많은 기능이 추가되었다. SqlSession을 호출하는 XML 기반의 매핑 구문이다.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "https://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" resultType="Blog">
    select * from Blog where id = #{id}
  </select>
</mapper>
```

간단한 예제치고는 조금 복잡해 보이지만 실제로는 굉장히 간단하다. 한 개의 매퍼 XML 파일에는 많은 수의 매핑 구문을 정의할 수 있다. XML 도입무의 헤더와 doctype 을 제외하면 나머지는 쉽게 이해되는 구문의 형태이다. 여기선 org.mybatis.example.BlogMapper 네임스페이스에서 selectBlog라는 매핑 구문을 정의했고 이는 결과적으로 org.mybatis.example.BlogMapper.selectBlog형태로 실제 명시되게 된다. 그래서 다음처럼 사용하게 되는 셈이다.

```
Blog blog = (Blog) session.selectOne("org.mybatis.example.BlogMapper.selectBlog", 101);
```

이건 마치 패키지를 포함한 전체 경로의 클래스내 메소드를 호출하는 것과 비슷한 형태이다. 이 이름은 매핑된 select구문의 이름과 파라미터 그리고 리턴타입을 가진 네임스페이스과 같은 이름의 Mapper 클래스와 직접 매핑될 수 있다. 이건 위에서 본것과 같은 Mapper 인터페이스의 메소드를 간단히 호출하도록 허용한다. 위 예제에 대응되는 형태는 아래와 같다.

```
BlogMapper mapper = session.getMapper(BlogMapper.class);
Blog blog = mapper.selectBlog(101);
```

두번째 방법은 많은 장점을 가진다. 먼저 문자열에 의존하지 않는다는 것이다. 이는 애플리케이션을 좀더 안전하게 만든다. 두번째는 개발자가 IDE 를 사용할 때 매핑된 SQL 구문을 사용할때의 수고를 덜어준다. 세번째는 리턴타입에 대해 타입 캐스팅을 하지 않아도 된다. 그래서 BlogMapper 인터페이스는 깔끔하고 리턴 타입에 대해 타입에 안전하며 이는 파라미터에도 그대로 적용된다.

참고 네임스페이스(Namespaces)에 대한 설명

네임스페이스(Namespaces) 가 이전버전에서는 사실 선택사항이었다. 하지만 이제는 패키지경로를 포함한 전체 이름을 가진 구문을 구분하기 위해 필수로 사용해야 한다.

네임스페이스은 인터페이스 바인딩을 가능하게 한다. 네임스페이스을 사용하고 자바 패키지의 네임스페이스을 두면 코드가 깔끔해지고 마이바티스의 사용성이 크게 향상될 것이다.

이름 분석(Name Resolution): 타이핑을 줄이기 위해 마이바티스는 구문과 결과매핑, 캐시등의 모든 설정엘리먼트를 위한 이름 분석 규칙을 사용한다.

- “com.mypackage.MyMapper.selectAllThings”과 같은 패키지를 포함한 전체 경로명(Fully qualified names)은 같은 형태의 경로가 있다면 그 경로내에서 직접 찾는다.
- “selectAllThings”과 같은 짧은 형태의 이름은 모호하지 않은 엔트리를 참고하기 위해 사용될 수 있다. 그래서 짧은 이름은 모호해서 에러를 자주 보게 되니 되도록 이면 전체 경로를 사용해야 할 것이다.

BlogMapper와 같은 Mapper클래스에는 몇가지 트릭이 있다. 매핑된 구문은 XML 에 전혀 매핑될 필요가 없다. 대신 자바 애노테이션을 사용할 수 있다. 예를들어 위 XML 예제는 다음과 같은 형태로 대체될 수 있다.

```
package org.mybatis.example;
public interface BlogMapper {
  @Select("SELECT * FROM blog WHERE id = #{id}")
  Blog selectBlog(int id);
}
```

간단한 구문에서는 애노테이션이 좀더 깔끔하다. 어쨌든 자바 애노테이션은 좀더 복잡한 구문에서는 제한적이고 코드를 엉망으로 만들 수 있다. 그러므로 복잡하게 사용하고자 한다면 XML 매핑 구문을 사용하는 것이 좀더 나을 것이다.

여기서 당신과 당신의 프로젝트 팀은 어떤 방법이 좋은지 결정해야 할 것이다. 매핑된 구문을 일관된 방식으로 정의하는 것이 중요하다. 앞서 언급했지만 반드시 한가지만 고집해야 하는 건 아니다. 애노테이션은 XML로 쉽게 변환할 수 있고 그 반대의 형태 역시 쉽게 처리할 수 있다.

스코프(Scope) 와 생명주기(Lifecycle)

이제부터 다룰 스코프와 생명주기에 대해서 이해하는 것은 매우 중요하다. 스코프와 생명주기를 잘못 사용하는 것은 다양한 동시성 문제를 야기할 수 있다.

참고 객체 생명주기와 의존성 삽입 프레임워크

의존성 삽입 프레임워크는 쓰레드에 안전하도록 해준다. 트랜잭션 성질을 가지는 SqlSessions과 매퍼들 그리고 그것들을 직접 빈에 삽입하면 생명주기에 대해 기억하지 않아도 되게 해준다. DI프레임워크와 마이바티스를 사용하기 위해 좀더 많은 정보를 보기 위해서는 MyBatis-Spring이나 MyBatis-Guice 하위 프로젝트를 찾아보면 된다.

SqlSessionFactoryBuilder

이 클래스는 인스턴스화되어 사용되고 던져질 수 있다. SqlSessionFactory 를 생성한 후 유지할 필요는 없다. 그러므로 SqlSessionFactoryBuilder 인스턴스의 가장 좋은 스코프는 메소드 스코프(예를들면 메소드 지역변수)이다. 여러개의

SqlSessionFactory 인스턴스를 빌드하기 위해 SqlSessionFactoryBuilder를 재사용할 수도 있지만 유지하지 않는 것이 가장 좋다.

SqlSessionFactory

한번 만든뒤 SqlSessionFactory는 애플리케이션을 실행하는 동안 존재해야만 한다. 그래서 삭제하거나 재생성할 필요가 없다. 애플리케이션이 실행되는 동안 여러 차례 SqlSessionFactory 를 다시 빌드하지 않는 것이 가장 좋은 형태이다. 재빌드하는 형태는 결과적으로 “나쁜냄새” 가 나도록 한다. 그러므로 SqlSessionFactory 의 가장 좋은 스코프는 애플리케이션 스코프이다. 애플리케이션 스코프로 유지하기 위한 다양한 방법이 존재한다. 가장 간단한 방법은 싱글턴 패턴이나 static 싱글턴 패턴을 사용하는 것이다. 또는 구글 쥬스나 스프링과 같은 의존성 삽입 컨테이너를 선호할 수도 있다. 이러한 프레임워크는 SqlSessionFactory의 생명주기를 싱글턴으로 관리할 것이다.

SqlSession

각각의 쓰레드는 자체적으로 SqlSession인스턴스를 가져야 한다. SqlSession인스턴스는 공유되지 않고 쓰레드에 안전하지도 않다. 그러므로 가장 좋은 스코프는 요청 또는 메소드 스코프이다. SqlSession 을 static 필드나 클래스의 인스턴스 필드로 지정해서는 안된다. 그리고 서블릿 프레임워크의 HttpSession 과 같은 관리 스코프에 둬서도 안된다. 어떠한 종류의 웹 프레임워크를 사용한다면 HTTP 요청과 유사한 스코프에 두는 것으로 고려해야 한다. 달리 말해서 HTTP 요청을 받을때마다 만들고 응답을 리턴할때마다 SqlSession 을 닫을 수 있다. SqlSession 을 닫는 것은 중요하다. 언제나 finally 블록에서 닫아야만 한다. 다음은 SqlSession을 닫는 것을 확인하는 표준적인 형태다.

```
try (SqlSession session = sqlSessionFactory.openSession()) {
  // do work
}
```

코드전반에 이런 형태를 사용하는 것은 모든 데이터베이스 자원을 잘 닫는 것으로 보장하게 할 것이다.

Mapper 인스턴스

Mapper는 매핑된 구문을 바인딩 하기 위해 만들어야 할 인터페이스이다. mapper 인터페이스의 인스턴스는 SqlSession 에서 생성한다. 그래서 mapper 인스턴스의 가장 좋은 스코프는 SqlSession 과 동일하다. 어쨌든 mapper 인스턴스의 가장 좋은 스코프는 메소드 스코프이다. 사용할 메소드가 호출되면 생성되고 끝난다. 명시적으로 닫을 필요는 없다.

```
try (SqlSession session = sqlSessionFactory.openSession()) {
  BlogMapper mapper = session.getMapper(BlogMapper.class);
  // do work
}
```