Spring Data Access

DataSource

- JDBC를 이용, DB를 사용하려면 Connection 타입의 DB 연결 객체가 필요
- 엔터프라이즈 환경에서는 각 요청마다 Connection을 새롭게 만들고 종료시킨다
- 애플리케이션과 DB 사이의 실제 커넥션을 매번 새롭게 만드는 것은 비효율적이고 성능의 저하를 가져온다
- 풀링(Pooling) 기법 사용
  : 정해진 개수의 DB Connection Pool을 준비하고 애플리케이션이 요청할 때마다 꺼내서 할당하고 사용 후 돌려받아 pool에 저장
- Spring에서는 DataSource를 하나의 독립된 Bean으로 등록하도록 강력하게 권장
- 엔터프라이즈 시스템에서는 반드시 DB 연결 풀 기능을 지원하는 DataSource를 사용해야 한다

DataSource
: 종류

- Spring에서 제공
  - SimpleDriverDataSource, SIngleConnectionDataSource
    학습/테스트용, 실제 서비스에서는 사용하지 않을 것

- Apache Common DBCP
  - 가장 유명한 오픈소스 DB 커넥션 풀 라이브러리
  - http://commons.apache.org/dbcp

- 상용 DB 커넥션 풀
  - 스프링 빈으로 등록 가능하고 프로퍼티를 통해 설정이 가능하다면 어떤 것이든지 사용 가능

DataSource

- 라이브러리 추가 (in pom.xml)

```
   <!-- Spring JDBC -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${org.springframework-version}</version>
  </dependency>
```

- Oracle DataSource를 bean으로 등록하는 예제 (in applicationContext.xml)

```
<!-- Oracle datasource -->
<bean id="oracleDatasource" class="oracle.jdbc.pool.OracleDataSource"
    destroy-method="close">
    <property name="URL" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="user" value="webdb" />
    <property name="password" value="webdb" />
    <property name="connectionCachingEnabled" value="true" />
    <qualifier value="main-db" />
</bean>
```

DataSource

- Common DBCP의 DataSource를 bean으로 등록 (MySQL의 경우)

```
<!-- Common DBCP -->
<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.4</version>
</dependency>
```

- MySQL Driver 설치

```
<!-- MySQL -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.38</version>
</dependency>
```

DataSource

- Common DBCP의 DataSource를 bean으로 등록 (MySQL의 경우)

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url"
    value="jdbc:mysql://localhost:3306/myportal?useSSL=false&amp;useUnicode=true&amp;c
haracterEncoding=utf8" />
    <property name="username" value="webdb" />
    <property name="password" value="webdb" />
</bean>
```

JDBC 연습

- webdb 데이터베이스에 guestbook 테이블과 샘플 데이터를 넣어 봅니다 (MySQL)

```
CREATE TABLE guestbook (
    no int primary key auto_increment,
    name varchar(20) NOT NULL,
    password varchar(20) NOT NULL,
    content varchar(255) NOT NULL,
    regdate datetime NOT NULL DEFAULT now()
);

insert into guestbook (name, password, content)
values ('방문자', 'test', '테스트 방명록입니다.');
```

JDBC 연습

- webdb 데이터베이스에 guestbook 테이블과 샘플 데이터를 넣어 봅니다 (Oracle)

```
CREATE TABLE guestbook (
    no number primary key,
    name varchar2(20) NOT NULL,
    password varchar2(20) NOT NULL,
    content varchar(255) NOT NULL,
    regdate date DEFAULT sysdate
);

CREATE SEQUENCE seq_guestbook_no
    START WITH 1
    INCREMENT BY 1;

insert into guestbook (no, name, password, content)
values (seq_guestbook_no.nextval, '방문자', 'test', '테스트 방명록입니다.');
```

DAO 패턴

- 데이터 액세스 계층은 DAO 패턴이라 불리는 형식으로 분리하는 것이 원칙
- 비즈니스가 단순하거나 없으면 DAO를 서비스 계층과 통합할 수 있다
- 데이터 액세스 기술을 외부로 노출시키지 않는다
- datasource를 주입(DI) 받아야 하고 메서드는 add(), update() 등, 단순하고 CRUD에 따르는 일반적인 이름을 사용하도록 한다

예외 처리

- 데이터 액세스 도중 발생하는 예외는 복구할 수 없다
- DAO 외부로 던지는 예외는 런타임 예외(Runtime Exception)여야 한다
- SQLException은 서비스 계층에서 직접 다뤄야 할 이유가 없으므로 RuntimeException으로 전환해야 한다
- 그러나 받아야 하는 경우가 있다면, 의미를 갖는 예외로 전환하여 전달한다

템플릿과 API

- 데이터 액세스 기술을 사용하는 코드는 try / catch / finally와 반복되는 코드로 작성되는 경우가 많다
- 데이터 액세스 기술은 외부의 리소스와 연동을 통해 이루어지므로 다양한 예외 상황이 발생할 수 있고 예외 상황을 종료하고 리소스를 반환하기 위한 코드가 길고 복잡해지는 경향이 있다 (가독성이 좋지 않음)

- 스프링에서는 DI의 응용 패턴인 템플릿/콜백 패턴을 이용해 반복되는 판에 박힌 코드를 피하고 예외 변환과 트랜잭션 동기화를 위한 템플릿을 제공한다

- 해당 기술의 데이터 액세스 기술 API (MyBatis API, JDBC API)와 스프링 데이터 액세스 템플릿을 조합하여 사용한다

MyBatis

MyBatis 3.x

- iBatis (MyBatis 2.x) 후속으로 등장한 ORM 프레임워크
- XML을 이용한 SQL과 ORM을 지원함
- 본격적인 ORM인 JPA나 Hibernate와 같이 새로운 DB 프로그래밍 패러다임을 이해해야 하는 것은 아님 (MyBatis 3.x에서는 Mapper 인터페이스를 통해 지원)
- 이미 익숙한 SQL을 그대로 사용하고 JDBC 코드의 불편함을 제거
- 가장 큰 특징은 SQL 문장을 자바 코드로부터 분리하여 별도의 XML 파일 안에 작성하고 관리할 수 있다는 것
- Spring 3.0 부터는 MyBatis 3.x(iBatis 2.x) 버전에 스프링 데이터 액세스 기술 대부분을 지원 (DataSource Bean 사용, 스프링 트랜잭션, 예외 자동 변환, 템플릿)

MyBatis 3.x
: 스프링에서 사용

- MyBatis의 DAO는 SQLSession 인터페이스를 구현한 클래스의 객체를 DI 받아 사용
- MyBatis의 DAO는 SQLSessionDaoSupport의 추상 클래스를 상속받아 구현하기도 함
- Mapper 인터페이스를 통한 OR 매핑 기능을 지원
- 이중, SQLSession 인터페이스를 구현한 클래스 객체의 DI 방식을 주로 사용하게 됨
  : SQLSession 인터페이스를 구현한 SQLSessionTemplate 클래스

MyBatis 3.x
: 설정 - 라이브러리 추가

- in pom.xml

```
<!-- MyBatis -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.2.2</version>
</dependency>

<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.2.0</version>
</dependency>
```

MyBatis 3.x
: 설정 - Bean 설정

- SqlSessionFactoryBean 설정 (in applicationContext.xml)

```
<!-- MyBatis SqlSessionFactoryBean -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="oracleDatasource" />
    <property name="configLocation" value="classpath:mybatis/configuration.xml" />
</bean>
```

- SqlSessionTemplate 설정 (in applicationContext.xml)

```
<!-- MyBatis SqlSessionTemplate -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>
```

MyBatis 3.x
: 설정 - Bean 설정

- DAO에 SqlSessionTemplate을 DI

```
public class BoardDao {
    @Autowired
    private SqlSession sqlSession;
}
```

MyBatis 3.x
: 설정 - 설정 파일과 매핑 파일

- MyBatis 설정 파일 (in configuration.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
    <typeAliases>
    </typeAliases>
    <mappers>
        <mapper resource="mybatis/mappers/emaillist.xml" />
    </mappers>
</configuration>
```

- SQL 매핑 파일 (in emaillist.xml)

```
<?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="emaillist">
</mapper>
```

MyBatis 3.x
: SQL 작성

- SELECT
  - 결과의 컬럼 이름과 resultType의 Class 필드 명이 다른 경우

```
<resultMap type="com.example.emaillist.vo.EmailListVo" id="resultMapList">
    <result column="no" property="no" />
    <result column="first_name" property="firstName" />
    <result column="last_name" property="lastName" />
    <result column="email" property="email" />
</resultMap>

<select id="list" resultMap="resultMapList">
    select no,
           first_name,
           last_name,
           email
    from email_list
    order by no desc
</select>
```

MyBatis 3.x
: SQL 작성

- SELECT
  - 결과의 컬럼 이름과 resultType의 Class 필드 명이 다른 경우 (resultMap 사용 안함)

```
<select id="list" resultType="com.example.emaillist.vo.EmailListVo">
    select no,
            first_name as firstName,
            last_name as secondName,
            email
    from email_list
    order by no desc
</select>
```

  - resultType, parameterType의 이름은 configuration.xml alias를 이용해 짧게 줄인다

```
<typeAliases>
    <typeAlias alias="EmailListVo" type="com.example.emaillist.vo.EmailListVo"/>
</typeAliases>
```

MyBatis 3.x
: SQL 작성

- SELECT
  - resultType의 클래스가 존재하지 않을 경우 map을 사용한다

```
<select id="joinlist" resultType="map">
    select  a.no, a.title, a.reg_date, b.no, b.name, b.email
    from  board a,
          member b
    where  a.member_no = b.no
    order by a.reg_date desc
</select>
```

  - Map으로 리턴되고 컬럼 이름이 대문자로 Map의 Key가 된다

MyBatis 3.x
: SQL 작성

- 파라미터 바인딩
  - 객체를 사용한 여러 파라미터 바인딩

```
<select id="search" parameterType="int">
    select no,
           name,
           message,
           to_char( reg_date, 'yyyy-MM-dd hh:mi:ss' ) as regDate
    from guestbook
    where no=#{no }
</select>
```

  - 파라미터의 이름은 임의 지정해도 무방
  - parameterType의 int는 내장된 alias 이다
    (byte, short, long, int, integer, double, float, boolean, string)

MyBatis 3.x
: SQL 작성

- 파라미터 바인딩
  - 파라미터 클래스가 존재하지 않지만 여러 값을 넘겨야 하는 경우

```
<select id="getId" parameterType="map" resultType="string">
    select id from member
    where name=#name# and ssn=#ssn#
</select>
```

```
Map map = new HashMap();
map.put("name", "홍길동");
map.put("ssn", "1234561234567");

String id2 = (String)sqlSession.selectOne( "getId", map );
```

MyBatis 3.x
: SQL 작성

- INSERT 후, 새로 들어간 row의 Primary Key를 얻어와야 하는 경우 (Oracle)

```
<insert id="insert" parameterType="GuestbookVo">
    <selectKey keyProperty="no" resultType="long" order="BEFORE">
        select guestbook_seq.nextval from dual
    </selectKey>
    <![CDATA[
        insert
        into guestbook
        values ( #{no }, #{name }, #{password }, #{message }, SYSDATE )
    ]]>
</insert>
```