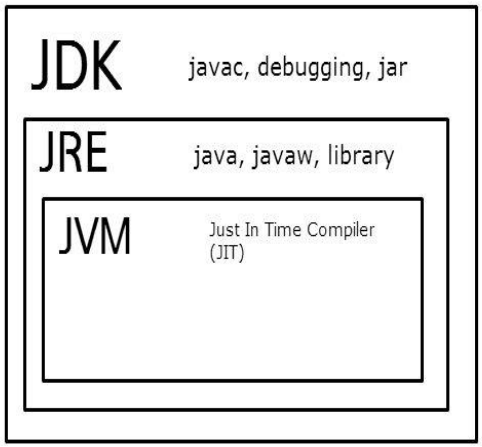
JAVA 프레임워크
MYBATIS

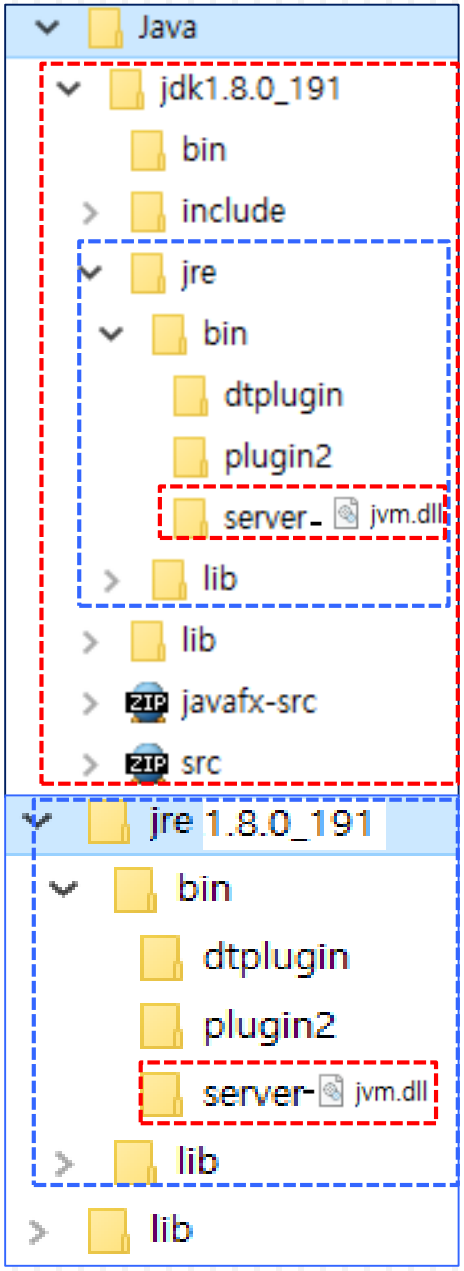
장안대학교
인터넷정보통신과

Oracle JDK와 OpenJDK

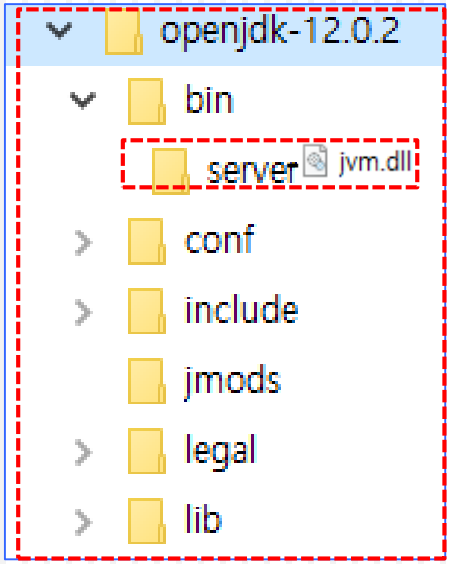
- Java 애플리케이션을 실행하기 위해서는 JVM이 필요하고, 컴파일하기 위해서는 JDK가 필요하다. 일반적으로 JDK를 설치하면 JVM도 함께 설치된다.
- JDK는 폐쇄적인 상업 코드 기반의 Oracle JDK 버전과 오픈 소스 기반의 OpenJDK이다.
- Oracle JDK는 OpenJDK에는 없는 재산권이 걸린 플러그인을 제공한다. 해당 플러그인은 Oracle이 재산권을 보유하고 있다.
- 마지막 공개 업데이트 2019-01-31까지 공개된 JDK 8u202에 해당한다.
- Oracle JDK에 존재하고 OpenJDK에는 없는 대표적 기능으로 글꼴 라이브러리와 Java Web Start가 있다. 서버 애플리케이션 개발에는 쓰이지 않는 기능들이다.
- 과거에는 OpenJDK가 Oracle JDK보다 성능이나 안정성이 크게 떨어졌지만 오늘날에는 Oracle JDK 만이 제공하는 일부 라이브러리를 제외하고는 차이가 없다.

Java - JDK, JRE, JVM

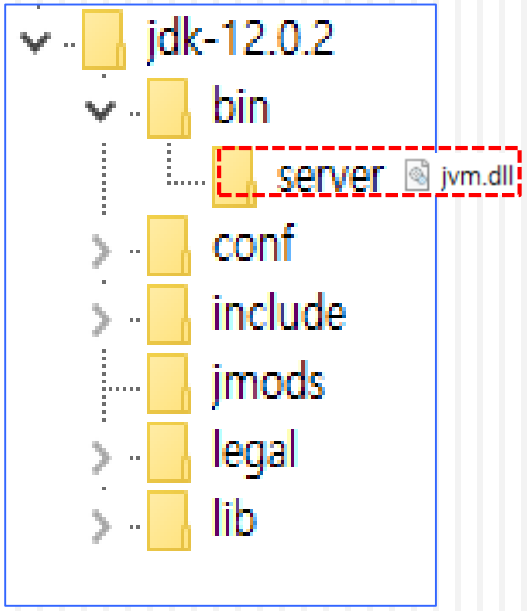




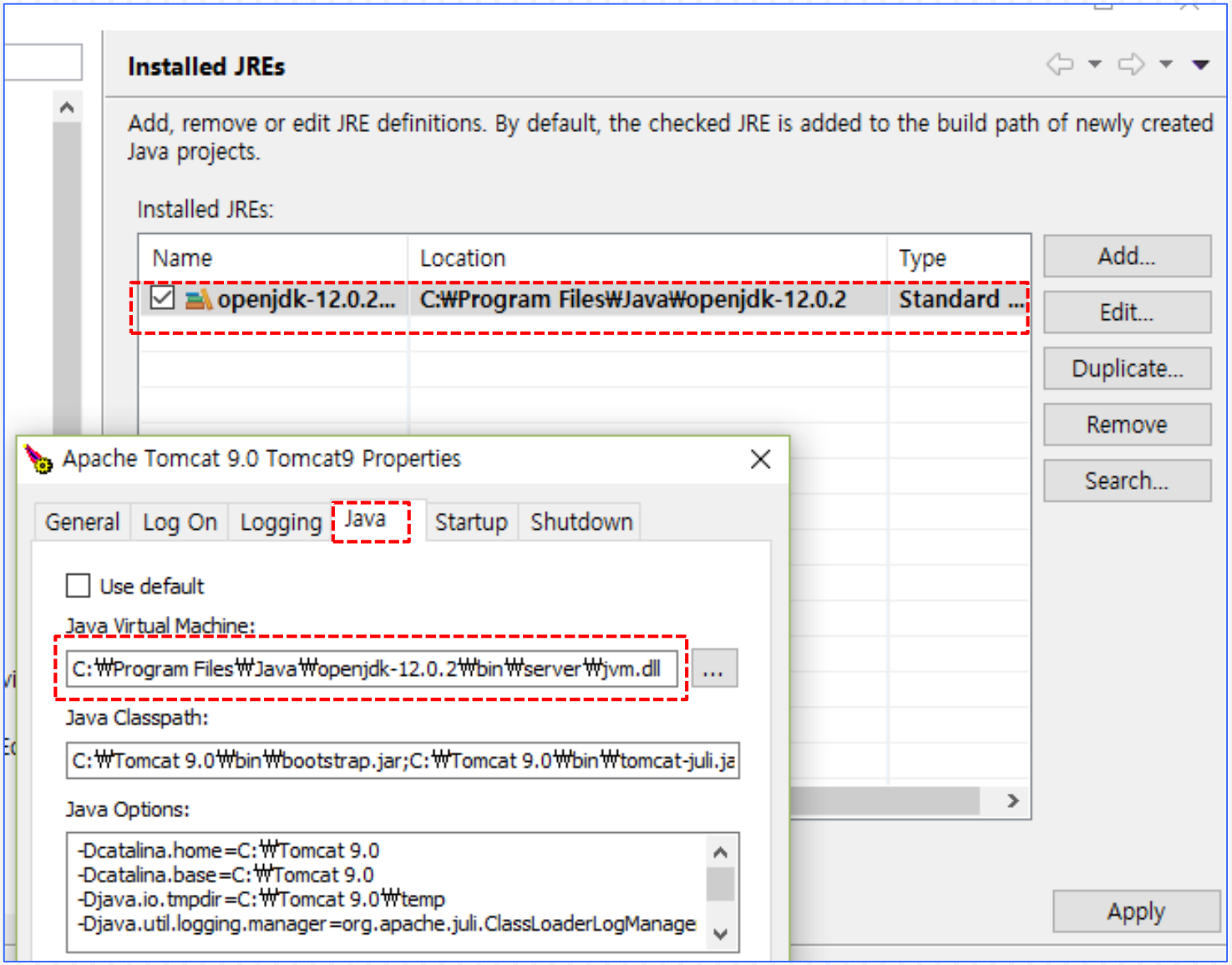
OpenJDK



Oracle JDK

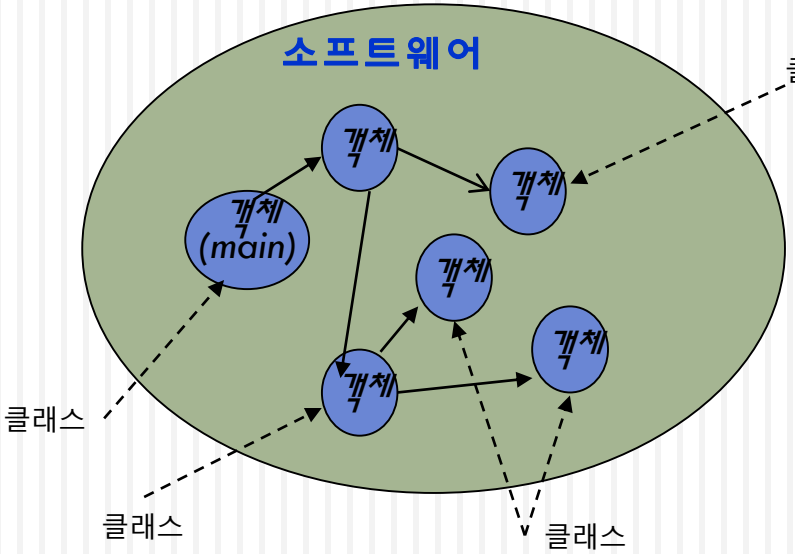


Eclipse 와 tomcat 설정 변경



Java와 객체지향

- 소프트웨어는 객체(Object) 들의 모임으로 이루어진다.
- 객체 - 컴포넌트, bean
- 하나의 클래스(객체)는 다른 객체를 사용하거나 포함하는 등의 관계를 가지고 작동한다.



멤버변수
자료(data)
상태(state)
Attribute
Property
field

멤버함수
Behavior(행위)
Operation(연산)
메서드(method)

객체 생성 : Student std = new Student()

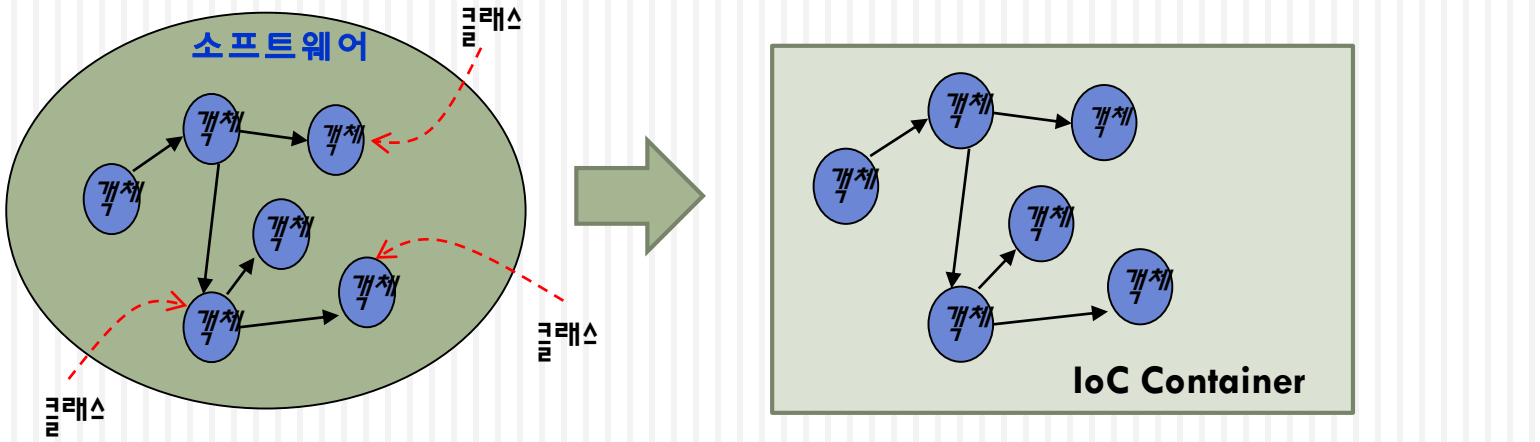
캡슐화, 상속성, 다형성,

Spring framework?

- framework -애플리케이션 개발을 빠르고 효율적으로 할 수 있도록 애플리케이션의 바탕이 되는 틀과 공통 프로그래밍 모델, 기술 API 등을 제공 하는 것.
- Spring framework - 자바 엔터프라이즈(Enterprise) 애플리케이션(Application) 개발에 사용되는 경량형 어플리케이션 Framework

- 스프링프레임워크에서는 BeanFactory 인터페이스와 ApplicationContext 인터페이스를 제공하여 모든 제어권한을 가지고 객체를 관리하도록 하는IoC 컨테이너를 생성한다.
- IoC 컨테이너는 객체의 생성, 초기화, 서비스 소멸에 관한 모든 제어 권한을 가지면서 객체의 생명주기를 관리한다.
- 개발자가 관리하던 객체의 제어권한을 이 객체가 관리하게 되었다는 의미에서 이 객체를 IoC 컨테이너(Inversion of Control Container)라 한다.

IoC 컨테이너

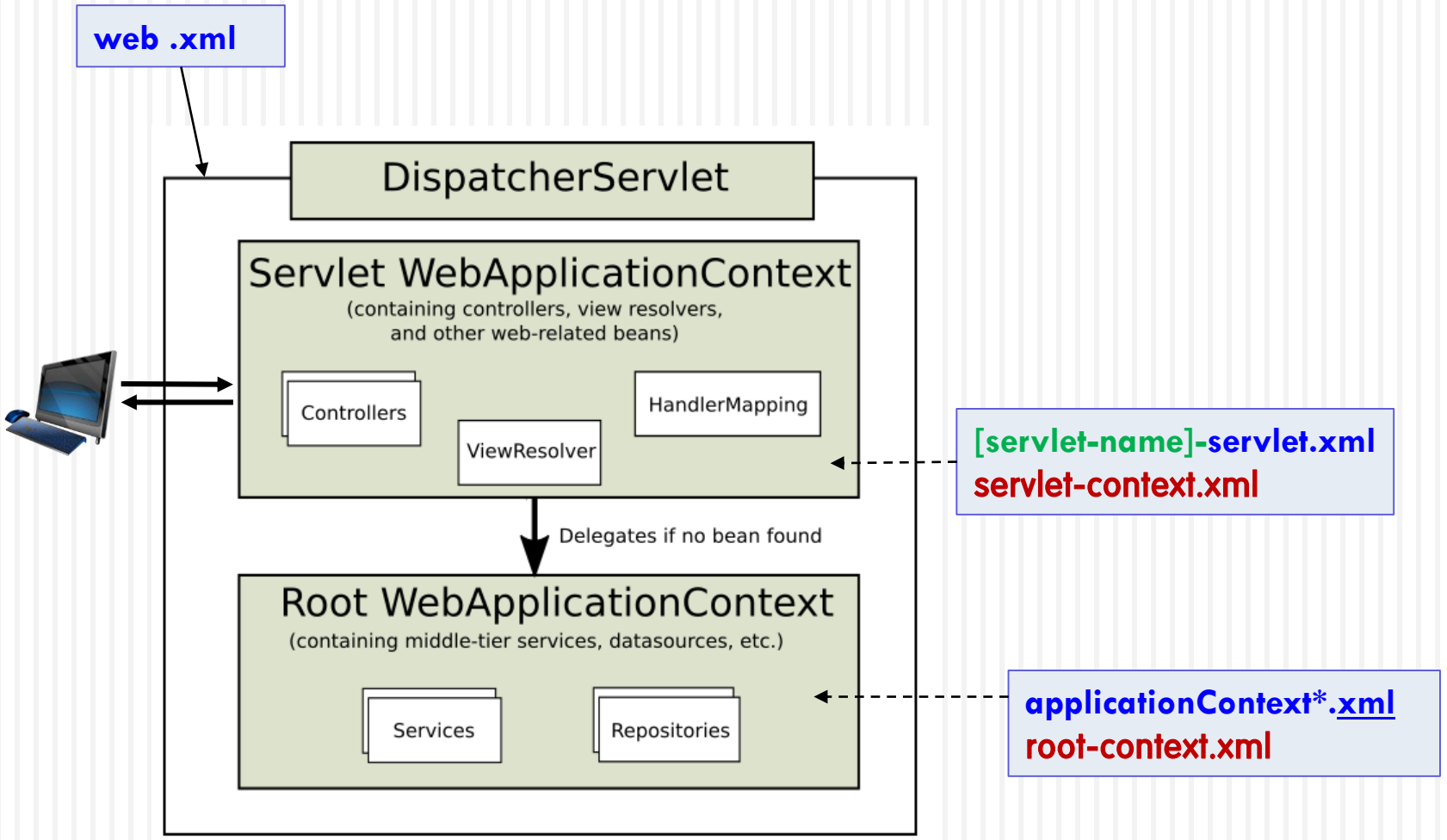


root-context.xml , servlet-context.xml

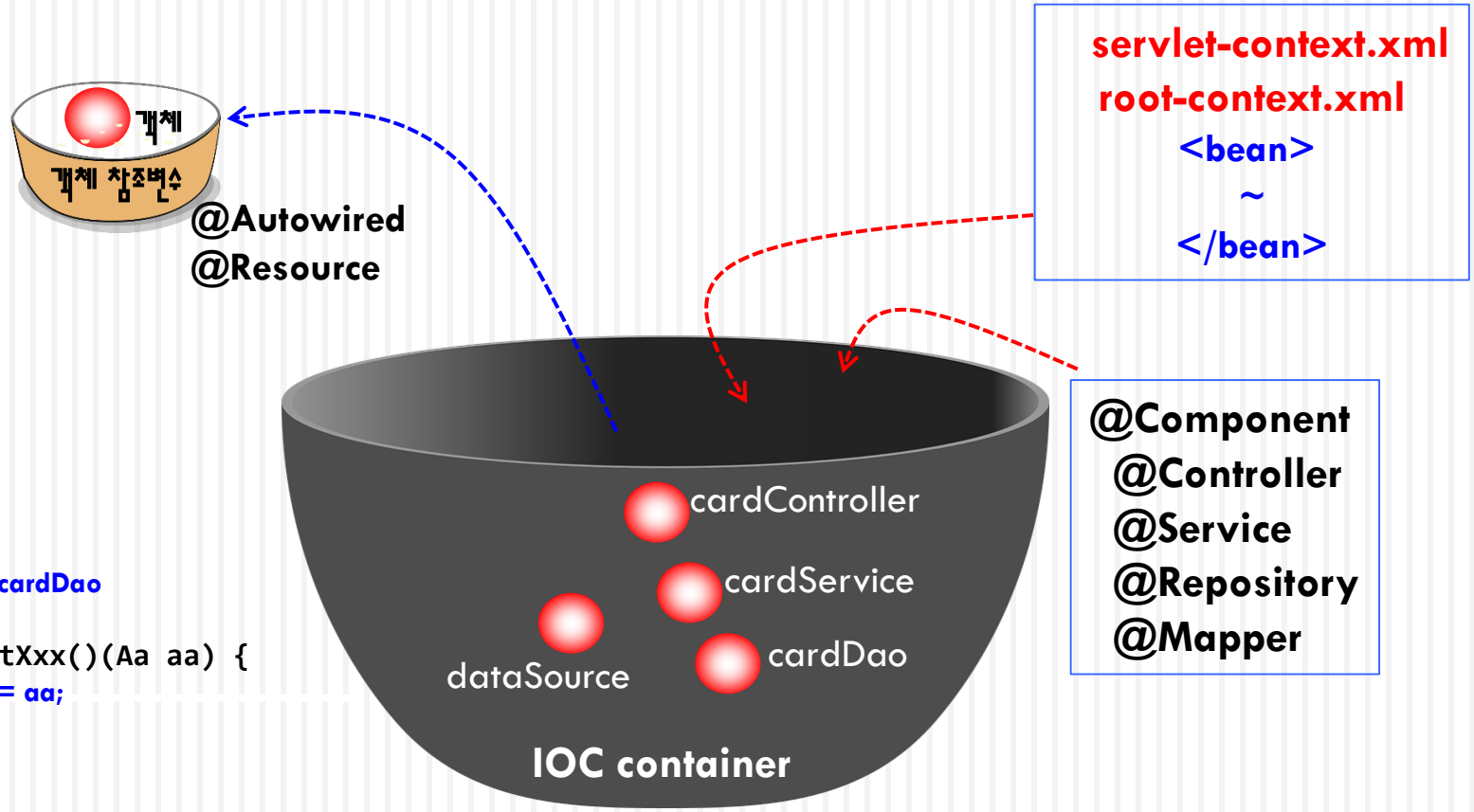
@ 애노테이션

- ApplicationContext 인터페이스는 설정메타데이터(Configuration Metadata)에 따라 스프링 IoC 컨테이너를 구성한다.
  - 자바코드 기반 설정 메타데이터
  - XML 기반 설정 메타데이터
  - 애노테이션 기반 설정 설정 메타데이터

Web IoC 컨테이너 구성방법



IoC 컨테이너에서 객체의 생성과 사용




```
private CardDao  cardDao

public void setXxx()(Aa aa) {
  this.cardDao = aa;
}
```

```
<bean id="cardDao" class=".dao.CardDaoImpl" >
 <property name="dataSource">
      <ref bean="dataSource" />
 </property>
</bean>
```

Dao 객체 생성 메타설정 파일

```xml
<!-- CardDao -->
<bean id="cardDao" class="com.jang.biz.dao.CardDaoImpl" >
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
</bean>
```

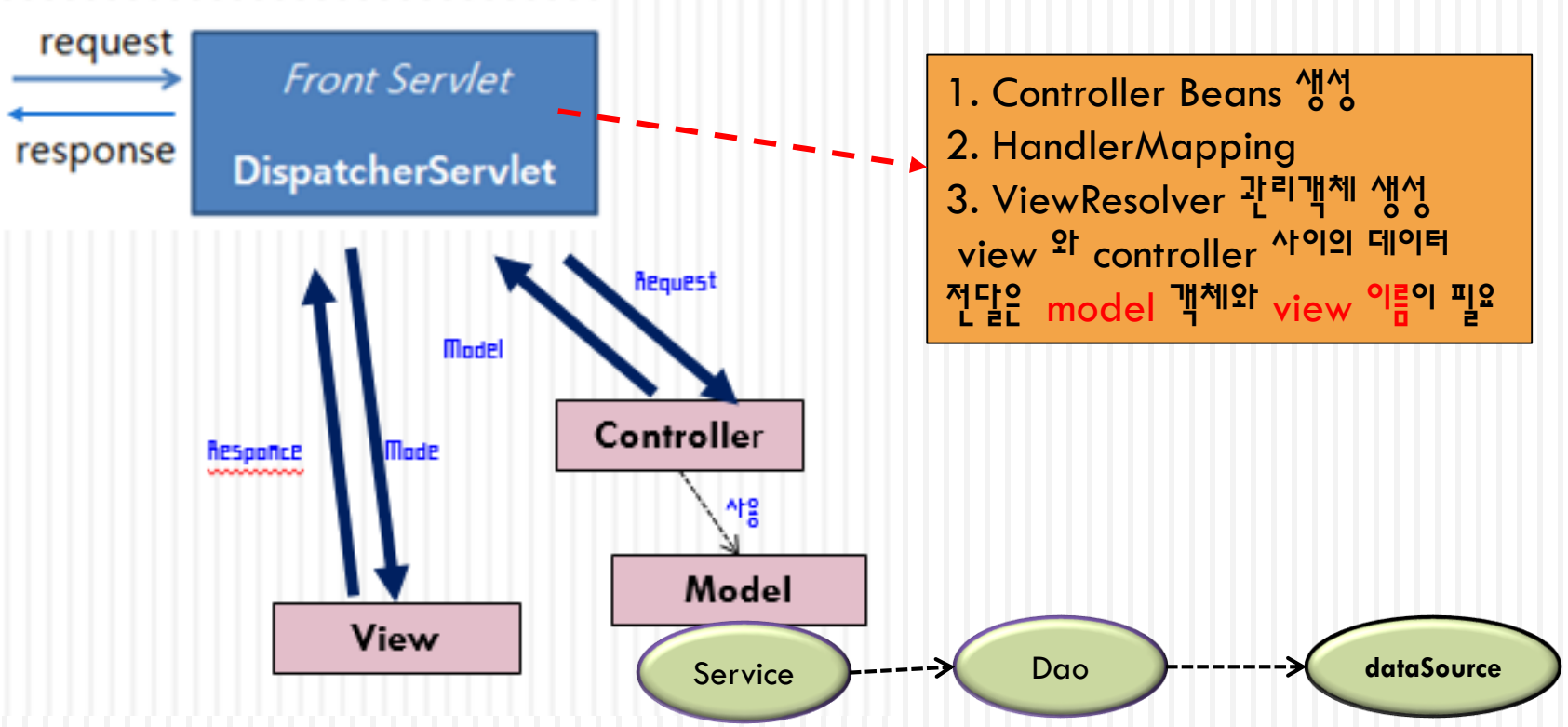
XML 기반 설정 메타데이터

```java
public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.jdbcTemplate2 = new NamedParameterJdbcTemplate(dataSource);
    }
```

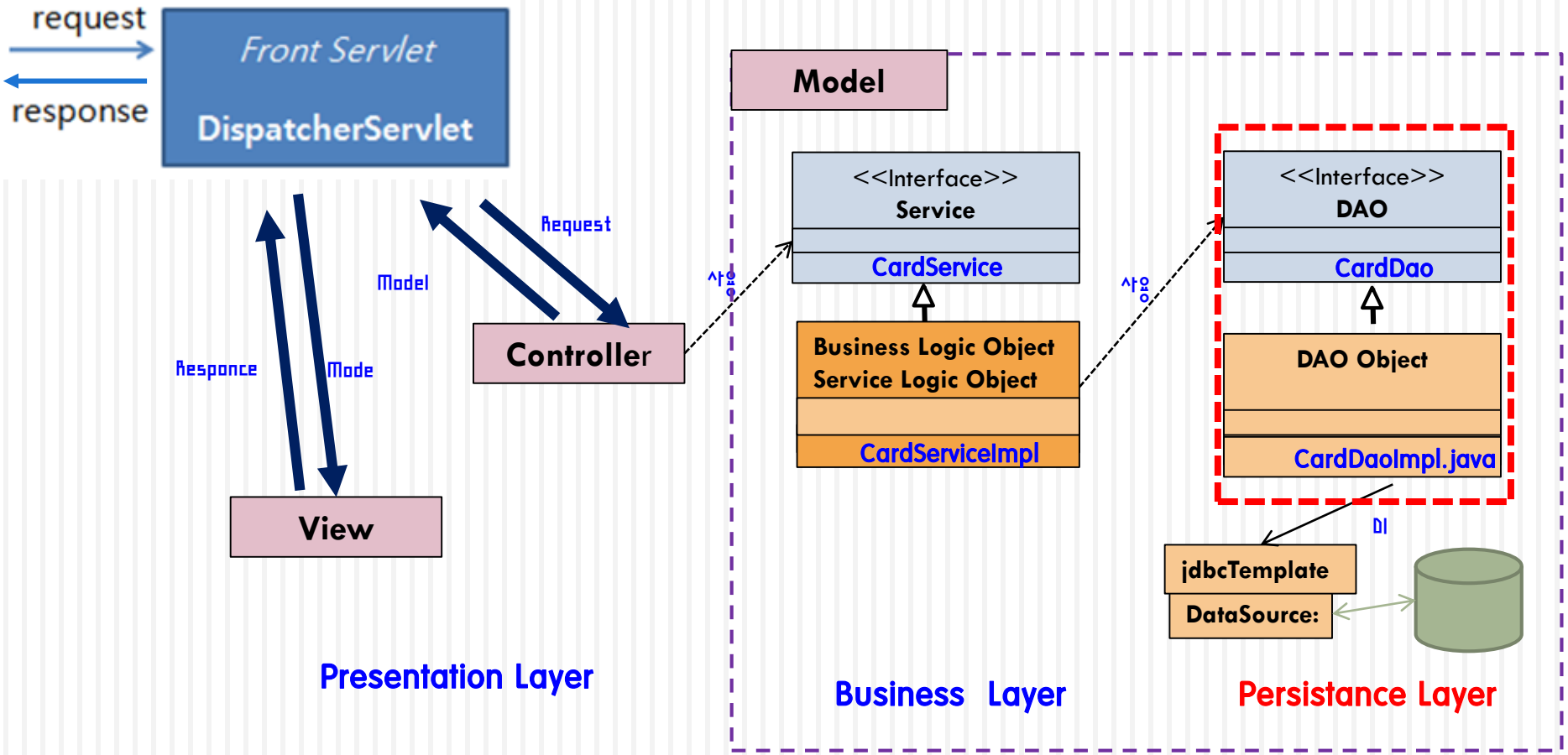
```java
@Repository(value="cardDao")
public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    @Autowired
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
        this.jdbcTemplate2 = new NamedParameterJdbcTemplate(dataSource);
    }
```

애노테이션 기반 설정 설정 메타데이터

스프링 프레임워크의 구성과 MVC



DAO design 패턴 설계 구현



인터페이스 개념

고객관리 미니 프로젝트개요

테이블 구성

열(D): 이름 BCARD

PK	이름	데이터 유형	크기	널이 아님	기본값
🔑	BNO	NUMBER		☑	
	BNAME	VARCHAR2	50	☑	
	PHONE	VARCHAR2	15	☑	
	DESCRIPTION	VARCHAR2	1000	☑	

메시지 :

[신규]

Sample No	Title	Phone	Description	수정	삭제
1	양옥경	010-1234-4321	태진아 부인	수정	삭제
2	이현상	010-2234-2321	저현상 동생	수정	삭제
3	김수현	010-3234-3321	드라마 작가	수정	삭제
4	박상현	010-4234-4421	정치가	수정	삭제
5	강양원	010-5234-5421	장안대학교수	수정	삭제

[Submit]

이름	
전화	
내용	

[Submit]

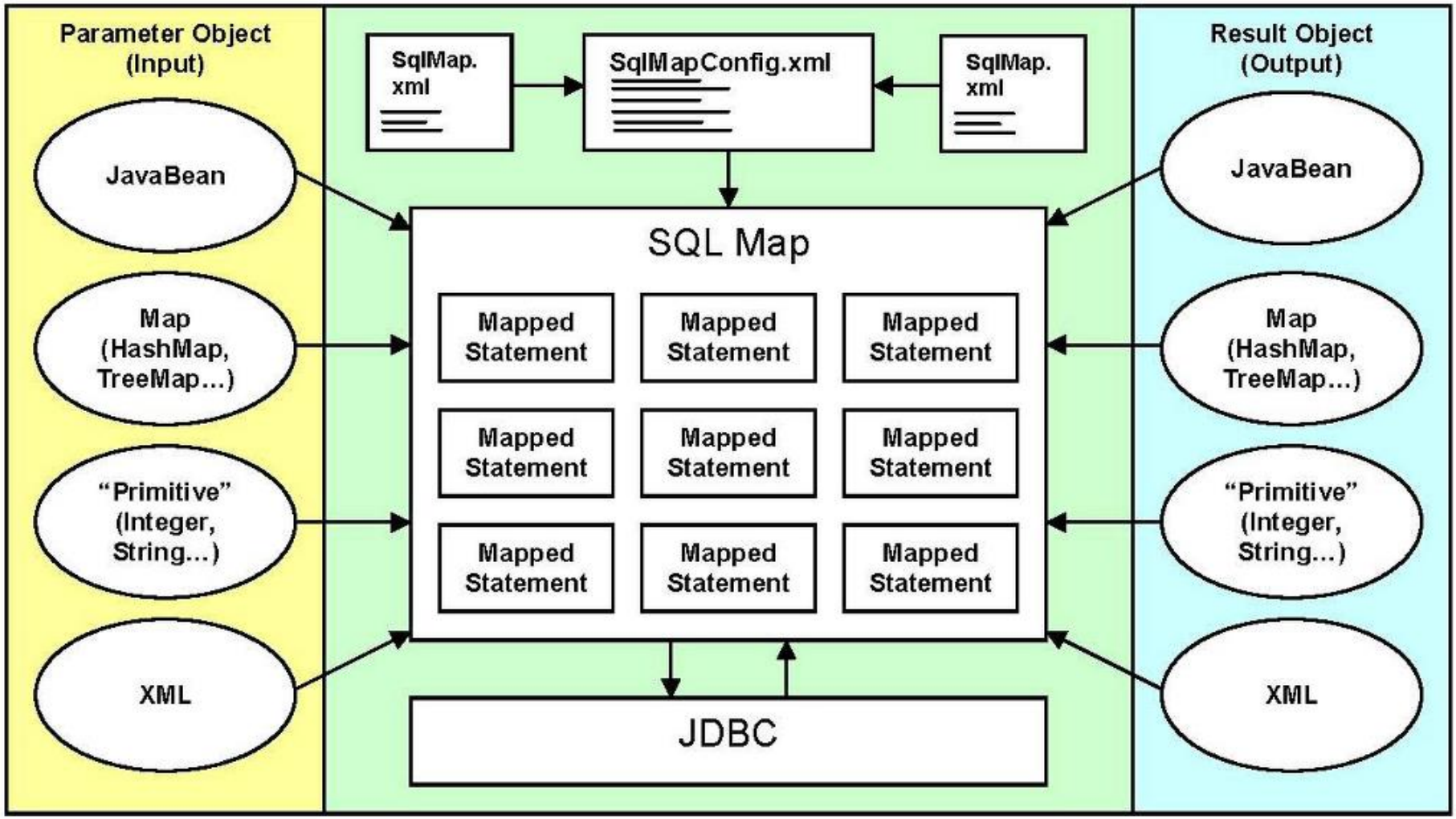
이름	양옥경
전화	010-1234-4321
내용	태진아 부인

Mybatis를 이용한 명함관리 구현

Mybatis 란?

- **persistence framework**
  - Java 소스코드에서 SQL 문장을 분리하여 별도의 XML 파일로 저장하고 Mapper를 이용하여 JDBC처럼 작동 시키는 퍼시스턴스 프레임워크
    - 영속성(persistence) : 데이터를 생성한 프로그램의 실행이 종료되더라도 사라지지 않는 데이터의 특성을 의미
- **생산성의 향상**
  - JDBC와 SQL 기능을 유지하면서도 훨씬 더 적은 코드로도 JDBC처럼 작동.
    자바코드의 20%를 사용하여 JDBC기능의 80%를 제공하는 간단한 프레임워크
- **SQL 문장과 프로그래밍 코드의 분리(관심사의 분리)**
  - 작업의 분배 : 팀을 세분화하는 것을 도움
- **뛰어난 이식성**
  - 다양한 프로그래밍 언어로도 구현. - 자바, C#(iBatis.NET), Ruby(RBATIS)
- **데이터베이스 접근 클래스와 비즈니스 로직을 담은 클래스의 분리**
- **오픈소스이며 무료**
- 2010년 6월 16일 apache의 iBatis 팀 ➔ google code myBatis 팀

Ibatis


Parameter Object
(Input)
JavaBean
Map
(HashMap,
TreeMap...)
"Primitive"
(Integer,
String...)
XML
SqlMap.
xml
SqlMapConfig.xml
SqlMap.
xml
SQL Map
Mapped
Statement
Mapped
Statement
Mapped
Statement
Mapped
Statement
Mapped
Statement
Mapped
Statement
Mapped
Statement
Mapped
Statement
Mapped
Statement
JDBC
Result Object
(Output)
JavaBean
Map
(HashMap,
TreeMap...)
"Primitive"
(Integer,
String...)
XML

SQL 문장 분리(1)

```java
public interface CardDao {

    List<Card> getCardList();

    Card getCard(int bno);

    int addCard(Card card);

    int updateCard(Card card);

    int deleteCard(int bno);
}
```

```java
@Repository(value="cardMapper")
public class CardMapper {

    @Autowired
    private SqlSessionTemplate sqlSession;

    public List<Card> getCardList(){
        return sqlSession.selectList("com.jang.biz.mapper.CardMapper.listCard");
    }
}
```

CardMapper.java

```java
@Repository(value="cardDao")
public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    @Override
    public List<Card> getCardList() {
        String SQL= "SELECT bno,bname,phone,description FROM  bcard orde
        RowMapper mapper = new BeanPropertyRowMapper<Card>(Card.class);
        List<Card> clist = (List)this.jdbcTemplate.query(SQL, mapper);
        return clist;
    }
```

SqlSessionTemplate
sqlSessionFactory

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

<mapper namespace="com.jang.biz.mapper.CardMapper">

    <select id="listCard" resultType="Card">
        SELECT  bno
                ,bname
                ,phone
                ,description
        FROM    bcard
        ORDER BY bno asc
    </select>
```

Cardmapper.xml

SQL 문장 분리(2)

```java
public interface CardDao {

    List<Card> getCardList();

    Card getCard(int bno);

    int addCard(Card card);

    int updateCard(Card card);

    int deleteCard(int bno);
}
```

```java
@Repository(value="cardDao")
public class CardDaoImpl implements CardDao {

    private JdbcTemplate jdbcTemplate;
    private NamedParameterJdbcTemplate jdbcTemplate2;

    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }

    @Override
    public List<Card> getCardList() {
        String SQL= "SELECT bno,bname,phone,description FROM  bcard orde
        RowMapper mapper = new BeanPropertyRowMapper <Card>(Card.class);
        List<Card> cList = (List)this.jdbcTemplate.query(SQL, mapper);
        return  cList ;
    }
```

CardMapper.java

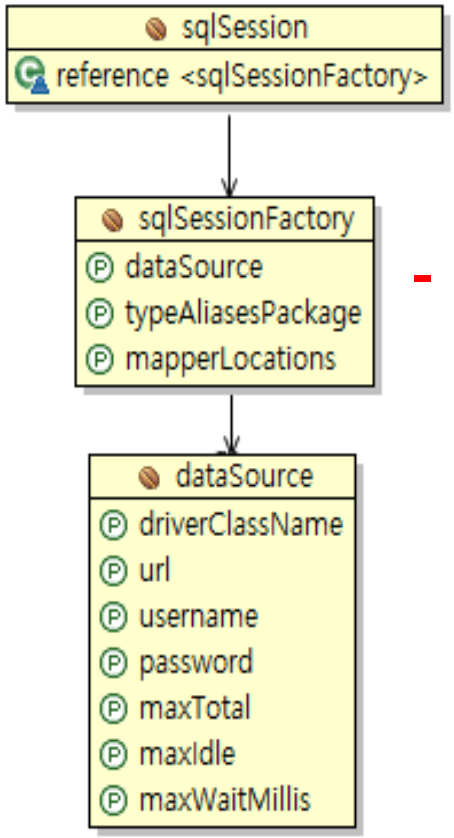
```java
@Repository(value="cardMapper")
public interface CardMapper {

        List<Card> getCardList();

        Card getCard(int bno);

        int addCard(Card card);

        int updateCard(Card card);

        int deleteCard(int bno);
}
```

sqlSessionFactory

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

<mapper namespace="com.jang.biz.mapper.CardMapper">

    <select id="getCardList" resultType="Card">
        SELECT  bno
               ,bname
               ,phone
               ,description
        FROM   bcard
        ORDER BY bno asc
    </select>
```

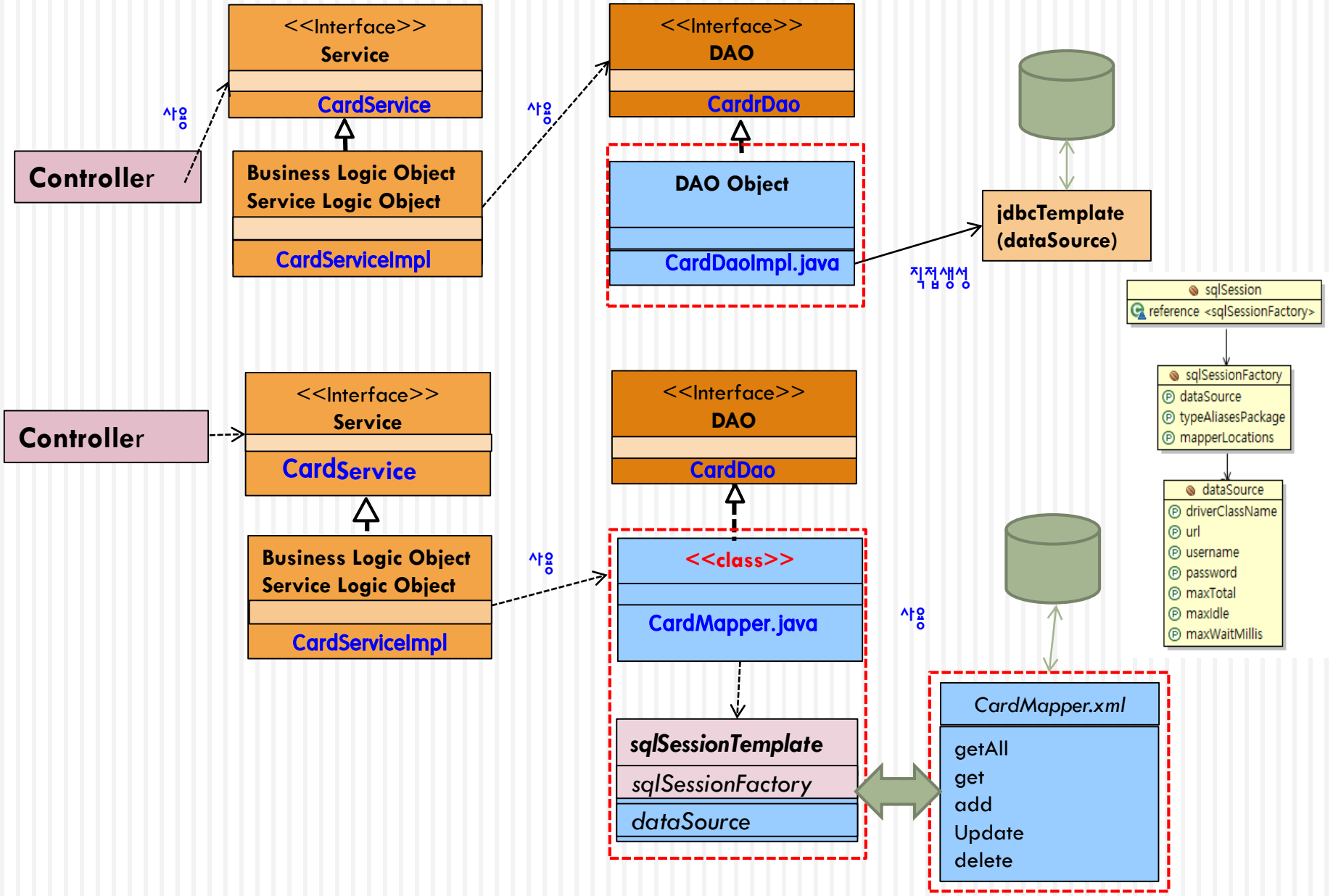
CardMapper.xml

SqlSessionTemplate와 sqlSessionFactory 구조

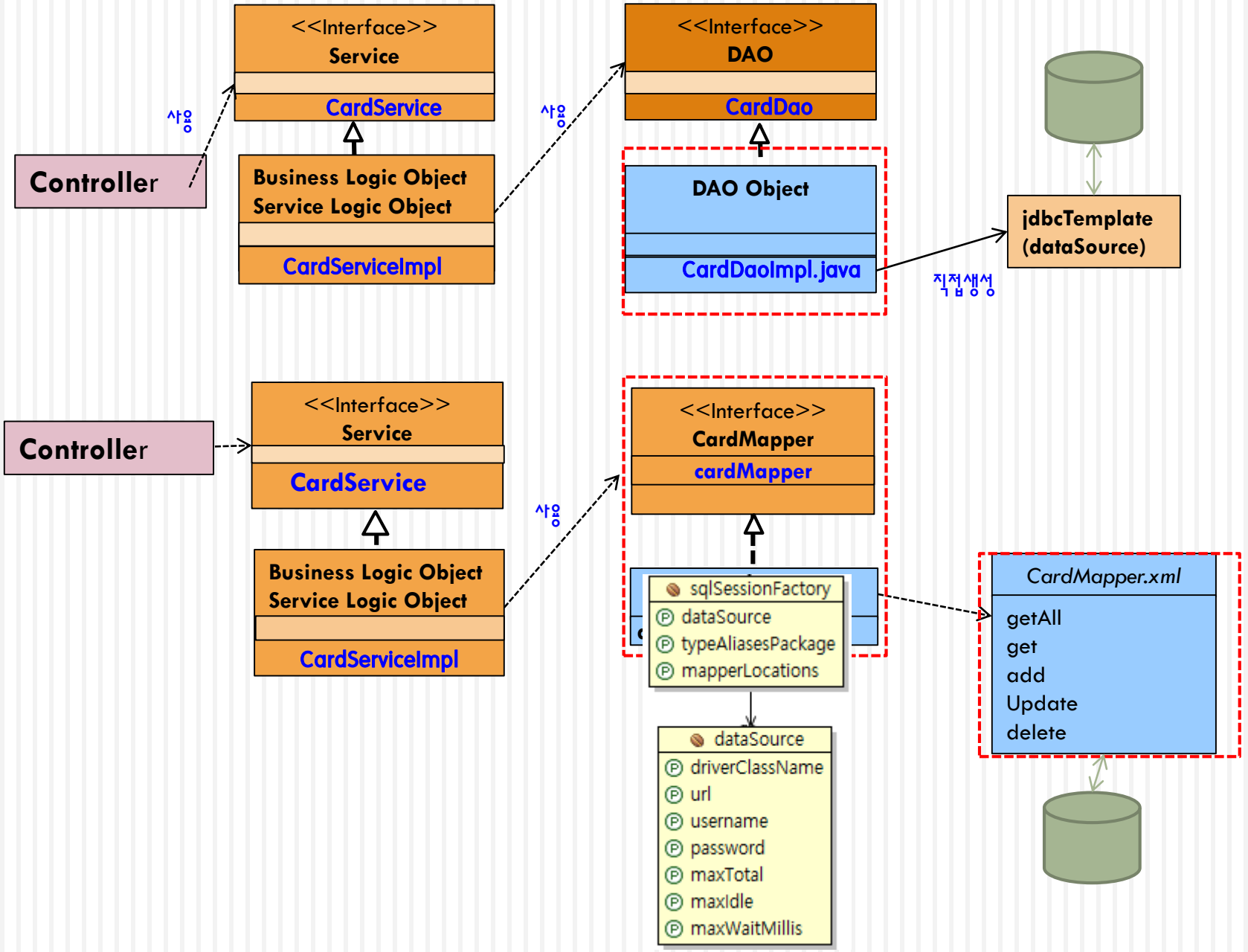


- 이전 사용자들을 위하여 SqlSessionTemplate 클래스와 메서드를 제공하여 SQL 문을 실행한다.
  sqlSession.selectList("매퍼파일")
  jdbcTemplate.query("SQL")
- Xml 매퍼 파일을 파싱하고 dataSource 객체를 이용하여 SQL 문을 실행하는 객체

MyBatis Mapper를 이용한 DAO 객체 구현 (DAO 구현에서 SQL만 XML로 분리)



MyBatis Mapper를 이용한 DAO 객체 구현 (IMPL없이 interfac와 SQL XML 매칭)



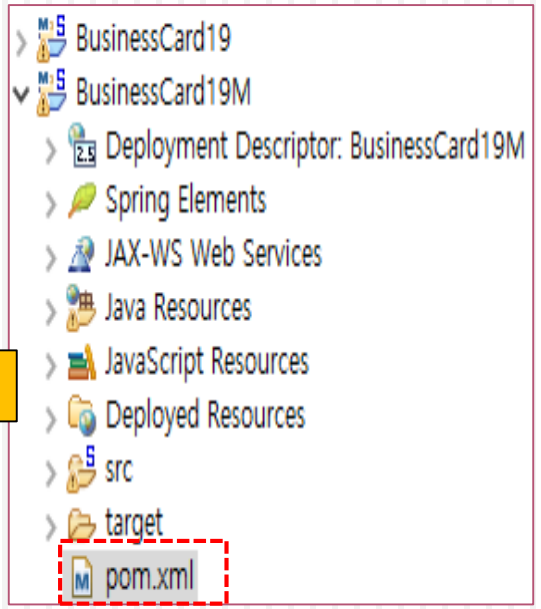
BusinessCardMy19 프로젝트 구성

- 프로젝트 생성 : BusinessCard19 복사 ➔ BusinessCard19M
- POM 추가

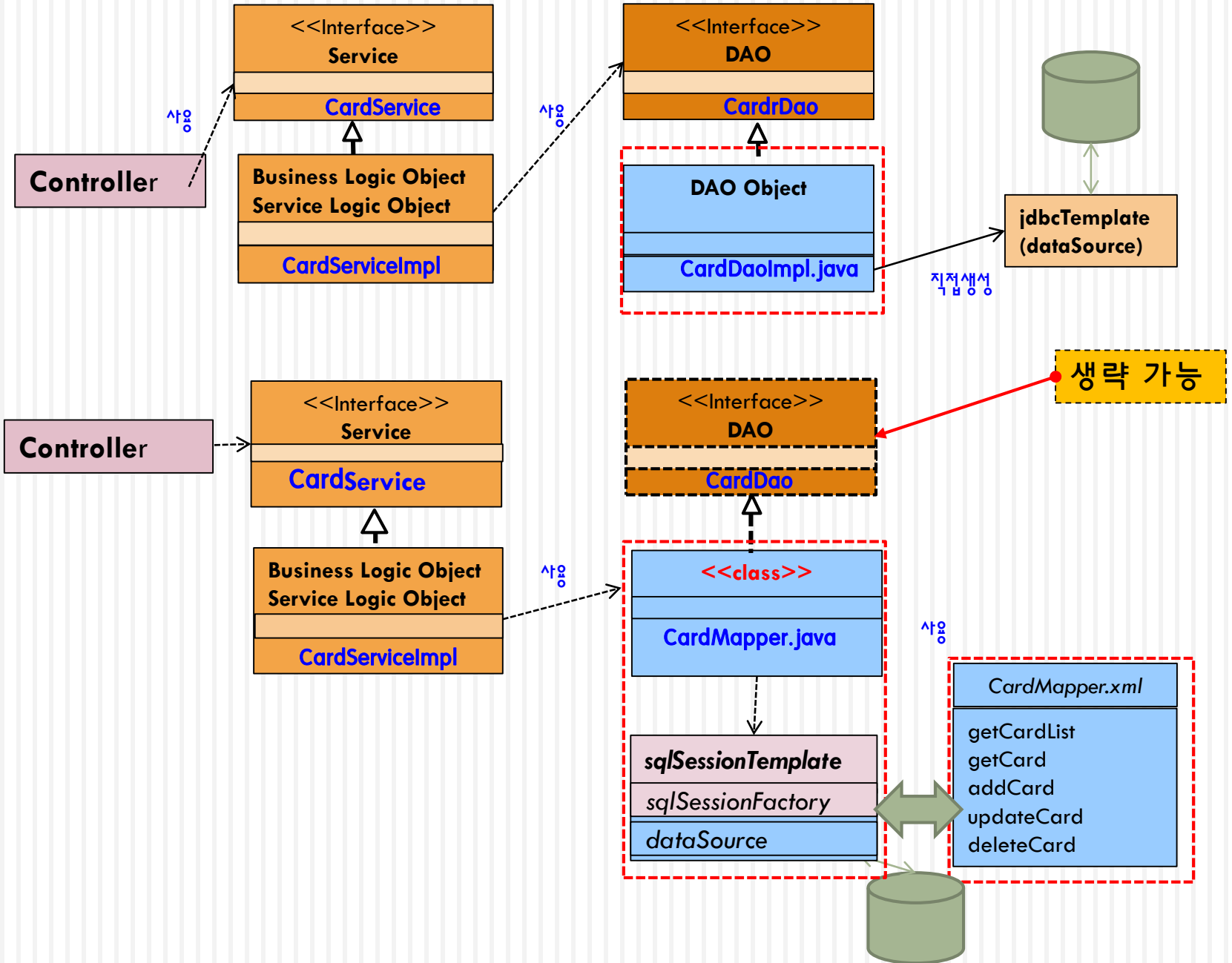
```
<!-- Hibernate Validator -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.12.Final</version>
</dependency>
```

```
<!-- Mybatis -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.4.6</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.3.2</version>
</dependency>
```

1. 추가



MyBatis (1)


<<Interface>>
Service
CardService
사용
Controller
Business Logic Object
Service Logic Object
CardServiceImpl
사용
<<Interface>>
DAO
CardrDao
DAO Object
CardDaoImpl.java
직접생성
jdbcTemplate
(dataSource)
생략 가능
Controller
<<Interface>>
Service
CardService
Business Logic Object
Service Logic Object
CardServiceImpl
사용
<<Interface>>
DAO
CardDao
<<class>>
CardMapper.java
sqlSessionTemplate
sqlSessionFactory
dataSource
사용
CardMapper.xml
getCardList
getCard
addCard
updateCard
deleteCard

MyBatis Dao 객체 개발 순서

1. Dao(MAPPER) (인터페이스) or 클래스 설계
   - 메서드(함수)명, 입력(매개변수:parameter) ,출력(반환값)

2. 메서드에 필요한 sql 문을 XML 파일로 작성
   - Namespace : 연동되는 인터페이스(클래스) java 파일 경로 확인
   - <XML 태그명, id=메서드명, resultType:반환값 , prameterType:매개변수>
     sql 문 ~
     </ XML 태그명 >

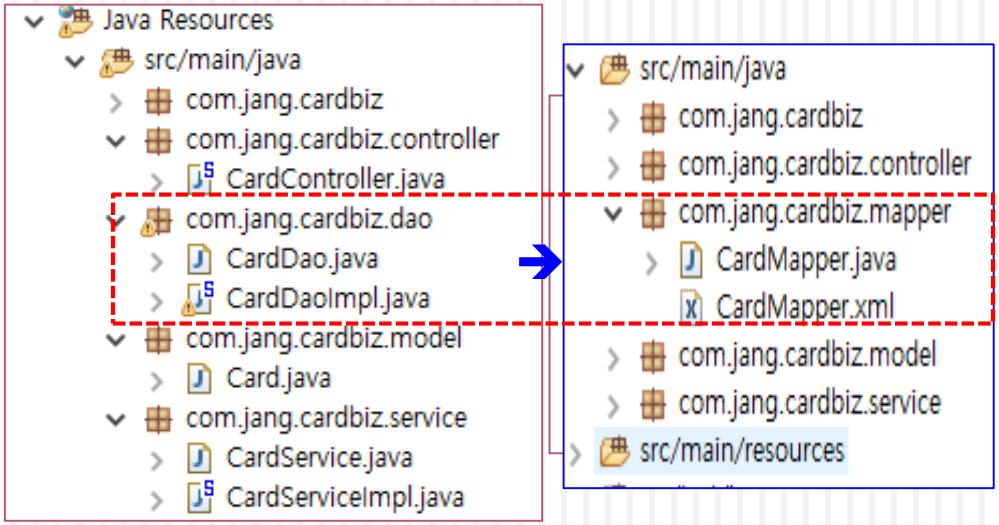
3. sqlSessionFactory 와 sqlSession 객체 생성 (root-context.xml)
   - xml 파일 경로 설정
   - 연계되는 인터페이스(클래스) java 파일 경로 확인

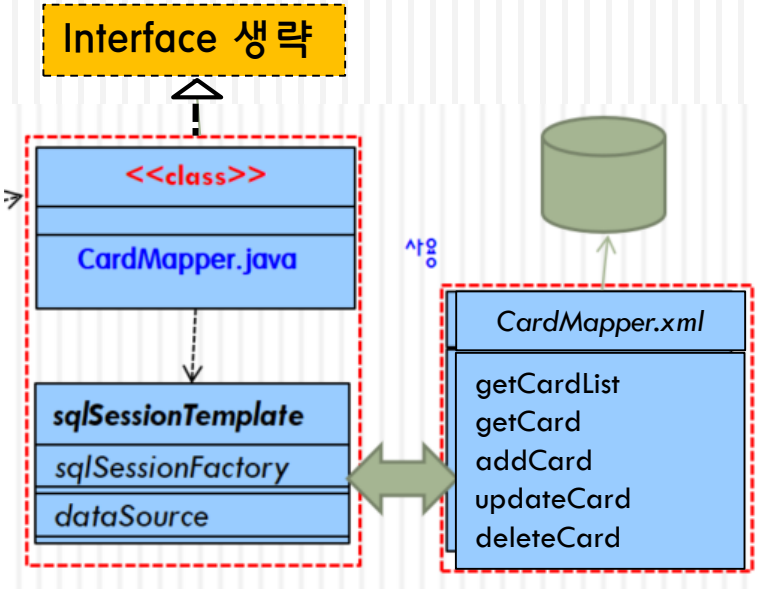
4. Dao(MAPPER) 클래스에서 sqlSession 객체 @Autowired
   - 각 메서드에서 필요한 sqlSession 객체의 메서드 작성

Mybatis(1)

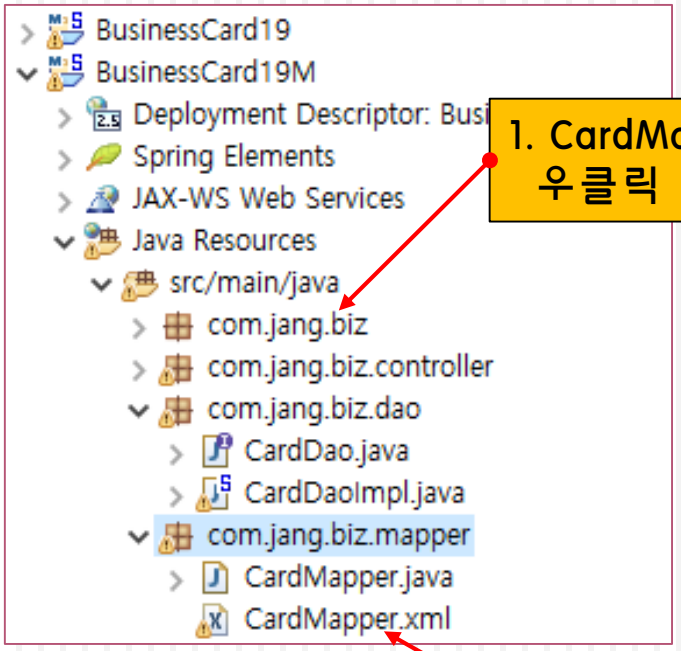
- 프로그램의 소스코드에서 SQL 문장을 분리하여 별도의 XML 파일로 저장하고 Mapper 를 이용하여 서로 연결시키는 방식
- 단계

1. CardDaoImpl 구현 클래스 ➔ CardMapper.java class (@repository) 설계
2. ➔ CardMapper.xml 작성
3. SqlSessionFactory 와 SqlSession 객체 생성 (root-context.xml)
4. CardMapper 클래스에서 SqlSession 객체 Autowired



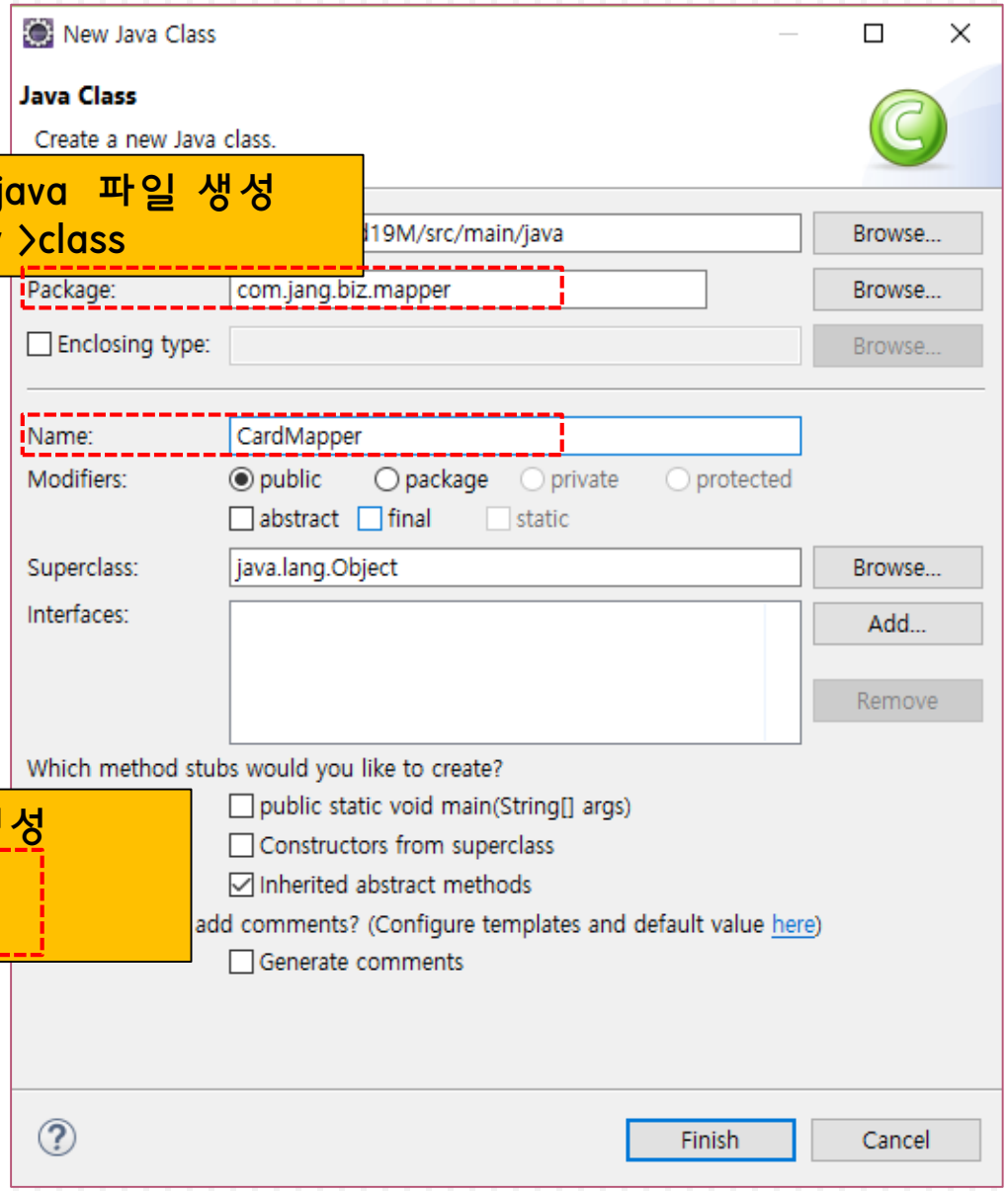


CardMapper.java 클래스와 CardMapper.xml 생성



1. CardMapper.java 파일 생성
   우클릭 > New >class

2. CardMapper.xml 파일 생성
   우클릭 > New > xml
   반드시 명칭 동일

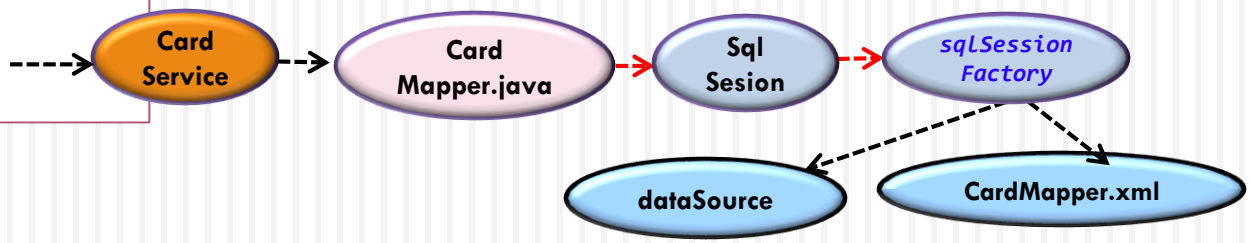


CardMapper.java 클래스 설계

```java
@Repository(value="cardMapper")
public class CardMapper {

    public List<Card> getCardList(){

        return null;
    }

    public Card getCard(int bno) {

        return  null;
    }

    public int addCard(Card card) {
        return  0;

    }

    public int updateCard(Card card) {
        return  0;
    }

    public int deleteCard(int bno) {

        return  0;
    }
}
```

애노테이션에 의한 dao 객체 생성 선언
"cardDao" 대체

```java
public interface CardDao {

    List<Card> listCard();

    Card getCard(int bno);

     int  addCard(Card card);

     int  updateCard(Card card);

     int  deleteCard(int bno);
}
```



CardMapper.xml 작성

```
<?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="com.jang.biz.mapper.CardMapper">

    ...

</mapper>
```

Xml 파서가 문서의 유효성 검사 경우 DOCTYPE 선언은 필수적
uri에서 문서 구조가 정의된 dtd 파일 찾음

명명공간(Namespaces)은 패키지경로를 포함한 전체
이름을 가진 구문을 구분하기 위해 필수로 사용
Xml 파일과 연동되는 java 파일의 경로와 명칭

CardMapper.xml 작성

```
<?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="com.jang.biz.mapper.CardMapper">

    <select id="getCard" parameterType="int"  resultType="Card">
        SELECT  bno
              ,bname
              ,phone
              ,description
        FROM   bcard
        WHERE  bno = #{bno}
    </select>

    ...
</mapper>
```

데이터베이스 결과데이터를 객체에 로드하는 방법을 정의하는 요소
resultMap이나 resultType 사용

```
@Override
public Card getCard(int bno) {

    String SQL= "SELECT bno,bname,phone,description FROM  bcard WHERE  bno = ?";
    RowMapper <Card> mapper = new BeanPropertyRowMapper<Card>(Card.class);
    return this.jdbcTemplate.queryForObject(SQL, mapper, bno);
}
```

CardDaoImpl 메서드명을 를 XML 태그 id로 표현

CardMapper.xml 작성 규칙

```
<?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="com.jang.CardBiz.mapper.CardMapper">

   <select id="listCard" resultType="Card">
      SELECT  bno
            ,bname
            ,phone
            ,description
      FROM    bcard
      ORDER BY bno asc
   </select>

   ...

</mapper>
```

id 는 클래스에서 호출하는 메서드명과 일치

SQL구문 | 메서드명 | 파라메터 형 | 반환값

```
<mapper namespace="com.jang.petstore.mapper.AccountMapper">

  <select id="getAccountByUsername" parameterType="string" resultType="Account">
    SELECT
          SIGNON.USERNAME,
          ACCOUNT.EMAIL,
          ACCOUNT.FIRSTNAME,
          ACCOUNT.LASTNAME,
          ACCOUNT.STATUS,
```

```
public List<Card> listCard() {

    String SQL= "SELECT bno,bname,phone,description FROM  bcard order by bno asc ";

    RowMapper mapper = new BeanPropertyRowMapper <Card>(Card.class);
    List<Card> cList = (List)this.jdbcTemplate.query(SQL, mapper);
    return  cList ;
}
```

Mapper SQL 구문 작성 규칙

속성	설명
id	구문을 찾기 위해 사용될 수 있는 명명공간내 유일한 구분자 메서드명
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭 파라메터
resultType	이 구문에 의해 리턴되는 기대타입의 패키지 경로를 포함한 전체 클래스명이나 별칭. collection 이 경우, collectio 포함된 타입이 될 수 있다. resultType 이나resultMap 을 사용하라. 반환값
resultMap	외부 resultMap 의 참조명. 결과맵은 MyBatis 의 가장 강력한 기능이다. resultType 이나 resultMap 을 사용하라. 반환값
flushCache	이 값을 true 로 셋팅하면, 구문이 호출될때마다 캐시가 지원질것이다(flush). 디폴트는 false 이다.
useCache	이 값을 true 로 셋팅하면, 구문의 결과가 캐시될 것이다. 디폴트는 true 이다.

Mapper.xml sql 구문 파일 구성

```
<?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="com.jang.doc.cardmybatis.mapper.CardMapper">

<select id="listCard" resultType="Card" >
      SELECT  bno
            ,bname
            ,phone
            ,description
      FROM   bcard order by bno asc
</select>
```

```
public List<Card> listCard() {

    String SQL= "SELECT bno,bname,phone,description FROM  bcard order by bno asc ";

    RowMapper mapper = new BeanPropertyRowMapper <Card>(Card.class);
    List<Card> cList = (List)this.jdbcTemplate.query(SQL, mapper);
    return  cList ;
}
```

```
<select id="getCard" parameterType="int" resultType="Card" >
     SELECT  bno
           ,bname
           ,phone
           ,description
     FROM   bcard
     WHERE  bno = #{bno}
</select>
```

```
public Card getCard(int bno) {

    String SQL= "SELECT bno,bname,phone,description FROM  bcard WHERE  bno = ?";
    RowMapper <Card> mapper = new BeanPropertyRowMapper<Card>(Card.class);
    return this.jdbcTemplate.queryForObject(SQL, mapper, bno);
}
```

```
<delete id="deleteCard">
    DELETE
    FROM  bcard
    WHERE  bno = #{bno}
</delete>
```

```
public void deleteCard(int bno) {
    this.jdbcTemplate.update("DELETE FROM bcard WHERE  bno = ?", bno);

}
```

Mapper.xml 파일 구성

```
<insert id=  "addCard" keyProperty="bno">
      <selectKey keyProperty="bno" resultType="int" order="BEFORE">
         SELECT COALESCE(MAX(bno), 0) + 1 AS bno
         FROM   bcard
      </selectKey>

      INSERT INTO bcard (
           bno
          ,bname
          ,phone
          ,description
      ) VALUES (
           #{bno}
          ,#{bname}
          ,#{phone}
          ,#{description}
      )
</insert>

<update id="updateCard" parameterType="Card" >
      UPDATE  bcard
      SET
            bname =  #{bname}
           ,phone = #{phone}
           ,description = #{description}
      WHERE  bno  = #{bno}
 </update>

</mapper>
```

<selectKey> : 자동생성키

첫 번째 인자가 NULL이 아닌 값이면 해당 값을 결과로 반환하고, NULL이면 두 번째 인자를 반환한다.

```
public void addCard(Card card) {
    int MaxNo= (int)this.jdbcTemplate.queryForObject("select max(bno)+1 from bcard",Integer.class);

    card.setBno(MaxNo);
    String SQL="INSERT INTO bcard (bno,bname,phone,description) VALUES (:bno,:bname,:phone,:description)";
    SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(card);
    this.jdbcTemplate2.update(SQL, parameterSource);

}
```

```
public void updateCard(Card card) {
    String SQL="UPDATE bcard SET bname = :bname, phone = :phone, description = :description WHERE  bno  = :bno";
    SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(card);
    this.jdbcTemplate2.update(SQL, parameterSource);

}
```

생략가능

root-context.xml 수정(SqlSession 만들기)

```
    <property name="maxWait">
        <value>10000</value>
    </property>
</bean>

<!-- enable transaction demarcation with annotations -->
<tx:annotation-driven />

<!-- CardDao
 <bean id="cardDao" class="com.jang.biz.dao.CardDaoImpl" >
    <property name="dataSource">
        <ref bean="dataSource" />
    </property>
 </bean>
-->

<!-- cardService bean 생성
 <bean id="cardService" class="com.jang.biz.service.CardServiceImpl" >
    <property name="cardDao">
        <ref bean="cardDao" />
    </property>
 </bean>
-->
```

```
<!-- transaction manager, use JtaTransactionManager for global tx -->
<tx:annotation-driven />
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSou
    <property name="dataSource" ref="dataSource" />
</bean>
```

```
<!-- define the SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="typeAliasesPackage" value="com.jang.cardbiz.model" />
    <property name="mapperLocations">
        <list>
            <value>classpath*:com/jang/cardbiz/mapper/*Mapper.xml</value>
        </list>
    </property>
</bean>
```

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

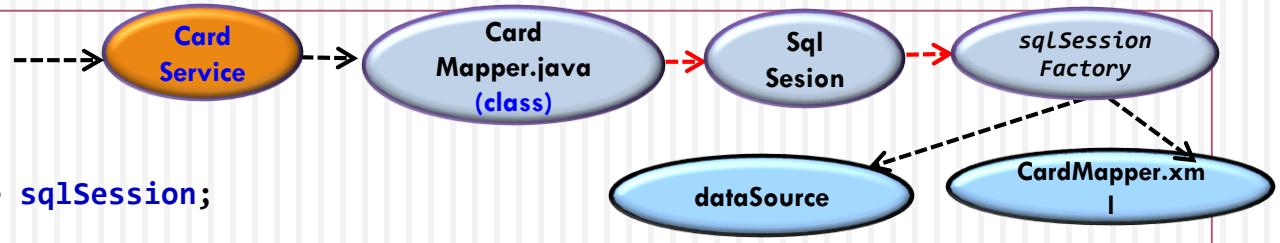
```
<!-- scan for mappers and let them be autowired -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.jang.cardbiz.mapper" />
</bean>
```

SqlSessionFactory

- SqlSessionFactory는 Datasource 와 XML 파일을 읽어 DAO에 주입할 수 있는 객체를 생성한다.
- *SqlSessionTemplate*는 스프링 트랜잭션 설정에 따라 자동으로 커밋 혹은 롤백을 수행하고 닫혀지는, 쓰레드에 안전한 SqlSession 개체를 생성

```
<!-- transaction manager, use JtaTransactionManager for global tx -->
<tx:annotation-driven />
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- define the SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="typeAliasesPackage" value="com.jang.biz.model" />
    <property name="mapperLocations">
        <list>
            <value>classpath*:com/jang/biz/mapper/*Mapper.xml</value>
        </list>
    </property>
</bean>

<bean id="sqlSession"  class="org.mybatis.spring.SqlSessionTemplate">
     <constructor-arg ref="sqlSessionFactory"></constructor-arg>
 </bean>

<!-- scan for mappers and let them be autowired -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.jang.biz.mapper" />
</bean>
```

CardMapper.java 클래스 완성 (CardDaoImpl.java수정)



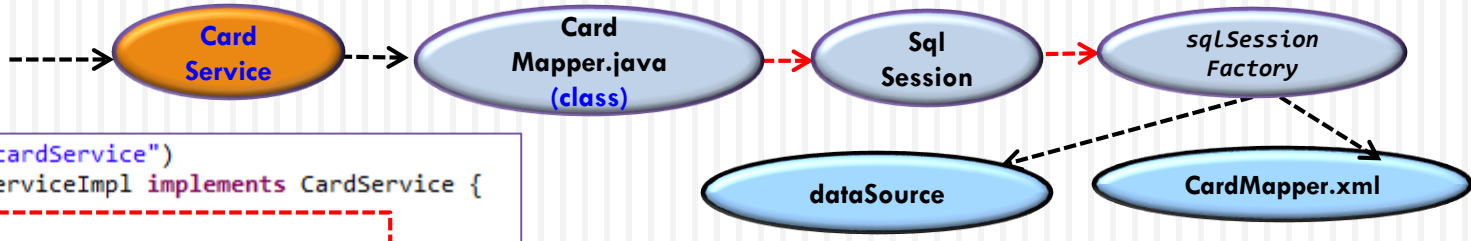

```java
@Repository(value="cardMapper")
public class CardMapper {

	@Autowired
	private SqlSessionTemplate sqlSession;

	public List<Card> listCard(){
		return sqlSession.selectList("com.jang.biz.mapper.CardMapper.listCard");
	}

	public Card getCard(int bno) {
		return sqlSession.selectOne("com.jang.biz.mapper.CardMapper.getCard", bno);
	}

	public void addCard(Card card) {
		sqlSession.insert("com.jang.biz.mapper.CardMapper.addCard", card);
	}

	public void updateCard(Card card) {
		sqlSession.update("com.jang.biz.mapper.CardMapper.updateCard", card);
	}

	public void deleteCard(int bno) {
		sqlSession.delete("com.jang.biz.mapper.CardMapper.deleteCard", bno);
	}

}
```

XML 파일

매개변수

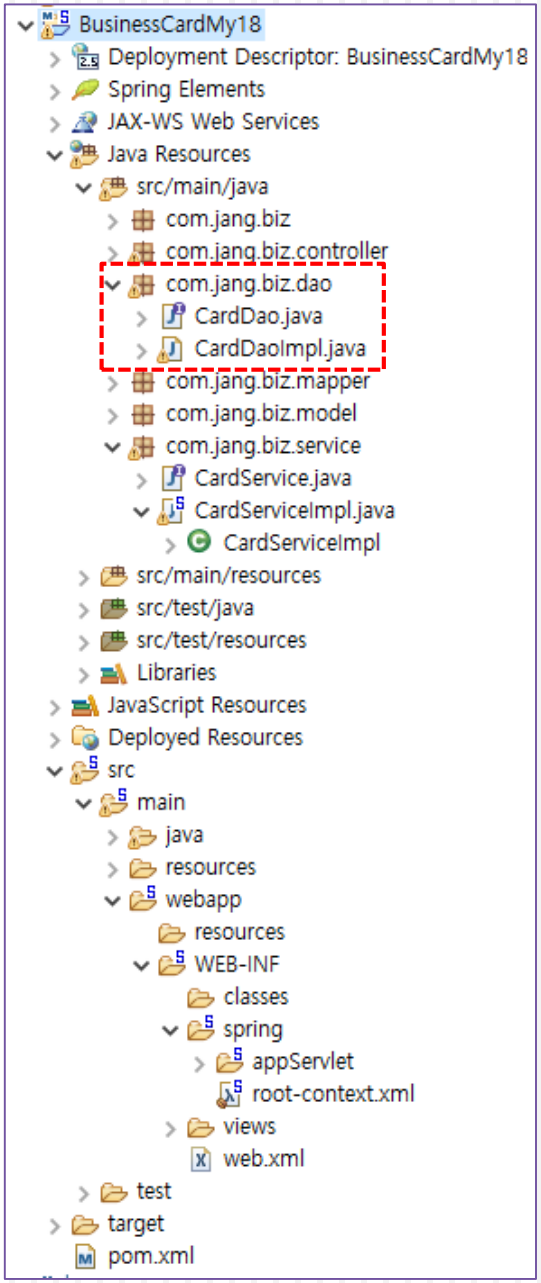
```java
public interface CardDao {

	List<Card> listCard();

	Card getCard(int bno);

	void addCard(Card card);

	void updateCard(Card card);

	void deleteCard(int bno);
}
```

CardServiceImpl.java 의존관계 재설정 (이름변경)

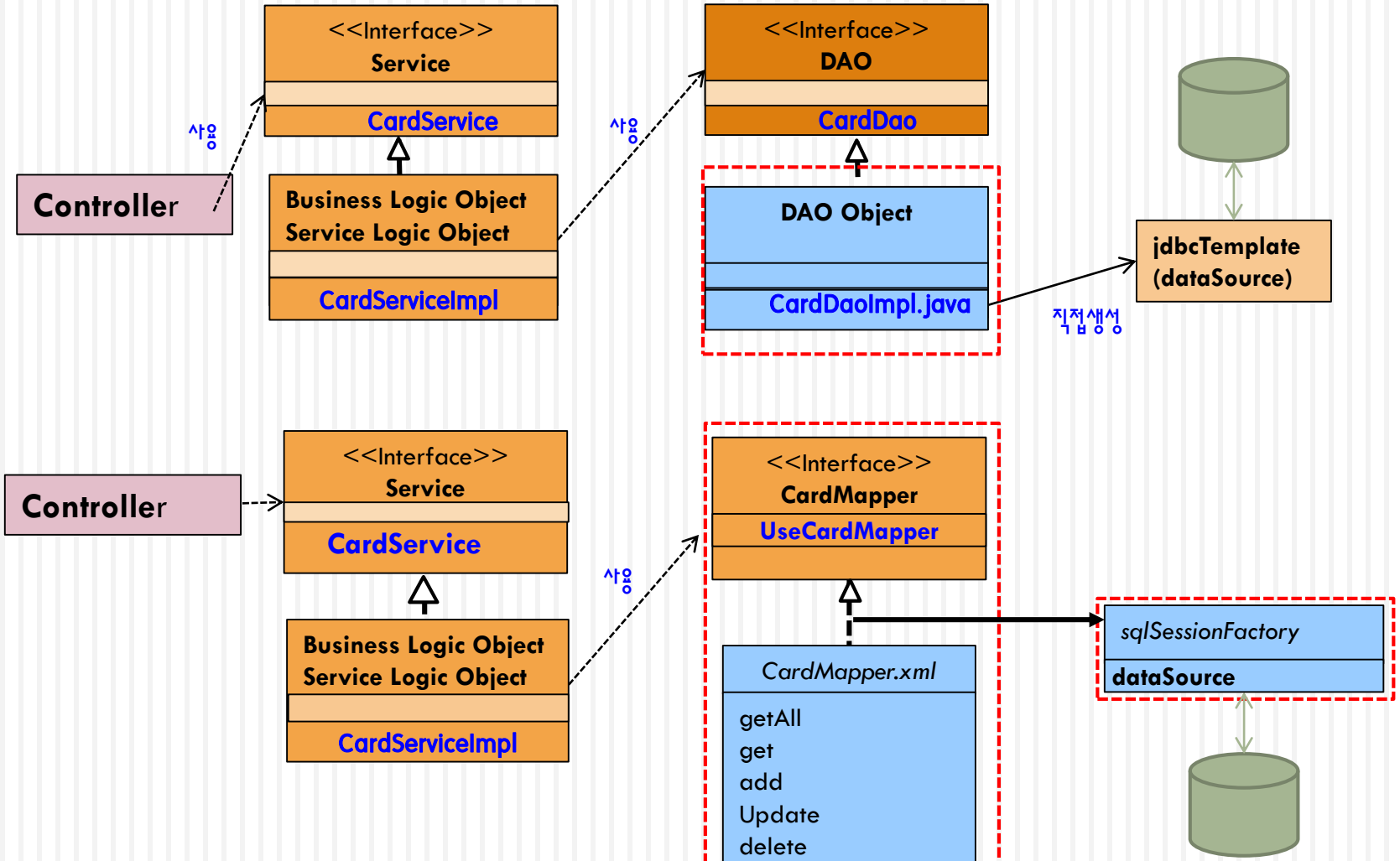



```java
@Service(value = "cardService")
public class CardServiceImpl implements CardService {

    /*
    @Resource(name = "cardDao")
    private CardDao cardDao;
    */

    @Resource(name = "cardMapper") //autowired
    private CardMapper cardMapper;

    @Override
    public List<Card> listCards() {
        return this.cardMapper.listCard();
    }

    @Override
    public Card getCards(int bno) {
        return this.cardMapper.getCard(bno);
    }

    @Override
    public void addCards(Card card) {
        this.cardMapper.addCard(card);
    }

    @Override
    public void updateCards(Card card) {
        this.cardMapper.updateCard(card);
    }

    @Override
    public void deleteCards(int bno) {
        this.cardMapper.deleteCard(bno);
    }

}
```

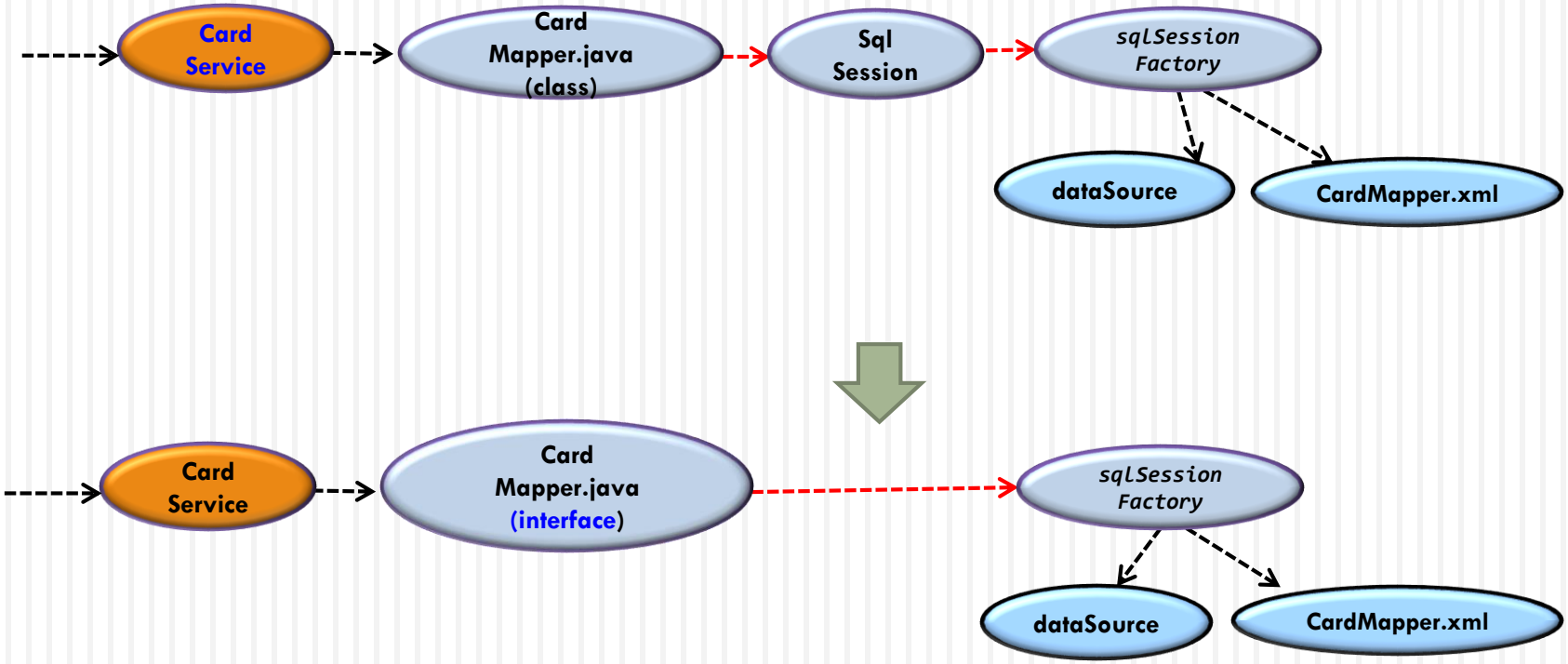
실행 및 결과 확인


BusinessCardMy18
Deployment Descriptor: BusinessCardMy18
Spring Elements
JAX-WS Web Services
Java Resources
src/main/java
com.jang.biz
com.jang.biz.controller
com.jang.biz.dao
CardDao.java
CardDaoImpl.java
com.jang.biz.mapper
com.jang.biz.model
com.jang.biz.service
CardService.java
CardServiceImpl.java
CardServiceImpl
src/main/resources
src/test/java
src/test/resources
Libraries
JavaScript Resources
Deployed Resources
src
main
java
resources
webapp
resources
WEB-INF
classes
spring
appServlet
root-context.xml
views
web.xml
test
target
pom.xml

MyBatis(2) Mapper를 이용한 DAO 객체 구현 (IMPL없이 interfac와 SQL XML 매칭)



인터페이스와 *sqlSessionFactory*를 이용한 Dao 객체생성

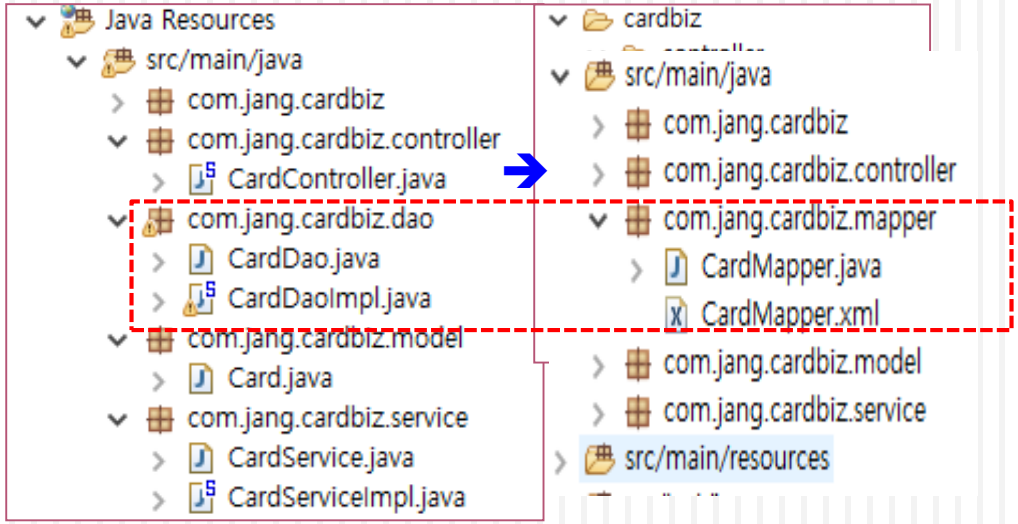


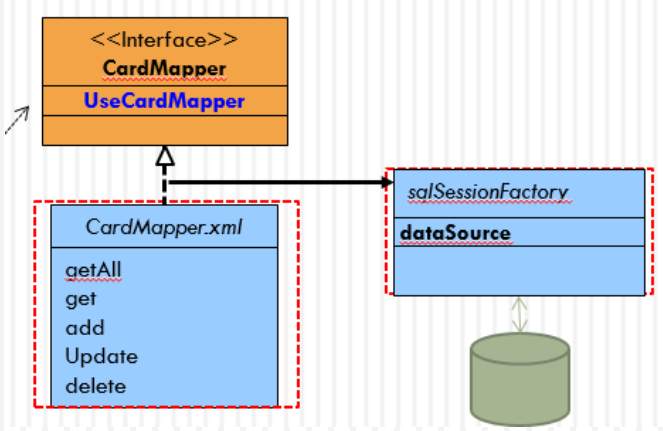
Mybatis(2)

- 프로그램의 소스코드에서 SQL 문장을 분리하여 별도의 XML 파일로 저장하고 Mapper를 이용하여 서로 연결시키는 방식
- Mapper를 이용한 DAO 객체 구현 –구현 클래스(IMPL)없이 interfac와 SQL XML 매칭)
- 단계

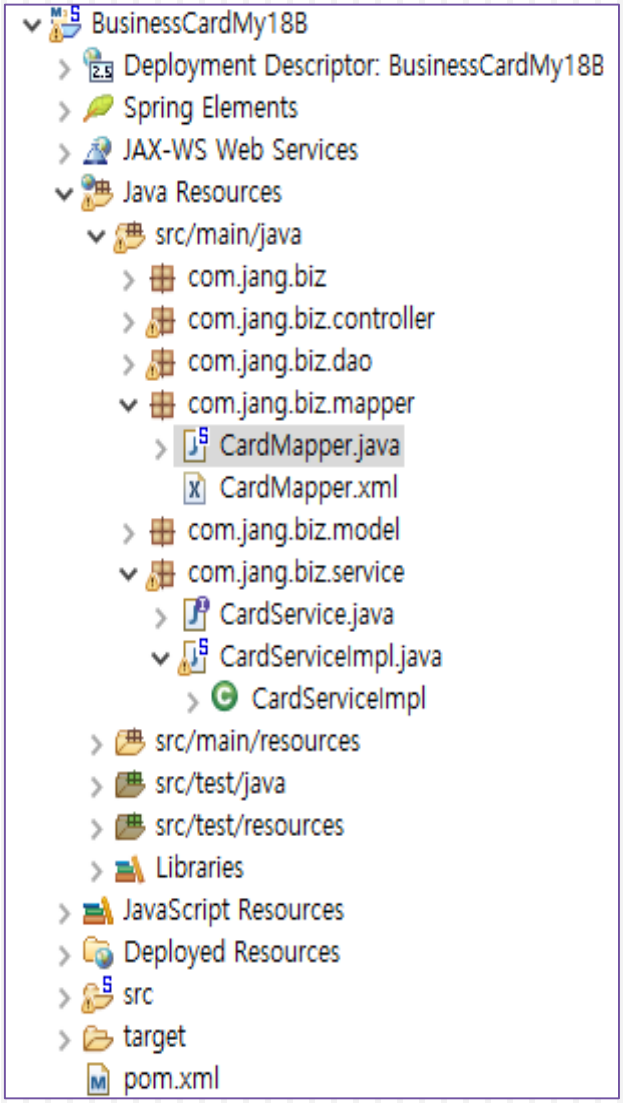
1. CardDao interface ➔ CardMapper interface(@Mapper or @repository)
2. CardDaoImpl 구현 클래스에서 SQL 분리 ➔ CardMapper.xml
3. SqlSessionFactoryBean으로 Dao 객체 생성 (root-context.xml)

-SqlSessionFactory 클래스는 데이터베이스에 대해 SQL 명령어를 실행하기 위해 필요한 모든 메서드를 가지고 있는 객체 (Mapper 객체 생성)





프로젝트 생성과 인터페이스 작성



1. BusinessCardMy18 프로젝트 복사 ➔ BusinessCardMy18B
2. CardMapper.java 클래스 삭제
3. 우클릭 > new > interface ➔ CardMapper
4. CardMapper 인터페이스 작성

```
package com.jang.biz.mapper;

import java.util.List;

@Mapper //@Repository(value="cardMapper")
public interface CardMapper {

        List<Card> listCard();

        Card getCard(int bno);

        void addCard(Card card);

        void updateCard(Card card);

        void deleteCard(int bno);

}
```

어떤 인터페이스가 @Mapper으로 지정되어 있다면 스프링 빈으로 등록

SqlSessionBean(root-context.xml)

SqlSessionFactoryBean은 datasource와 mapper XML 파일을 받아 jdbc와 같은 기능을 수행하는 객체를 생성하고 동일 이름의 interface 파일과 연동하여 dao 구현 객체를 생성한다.

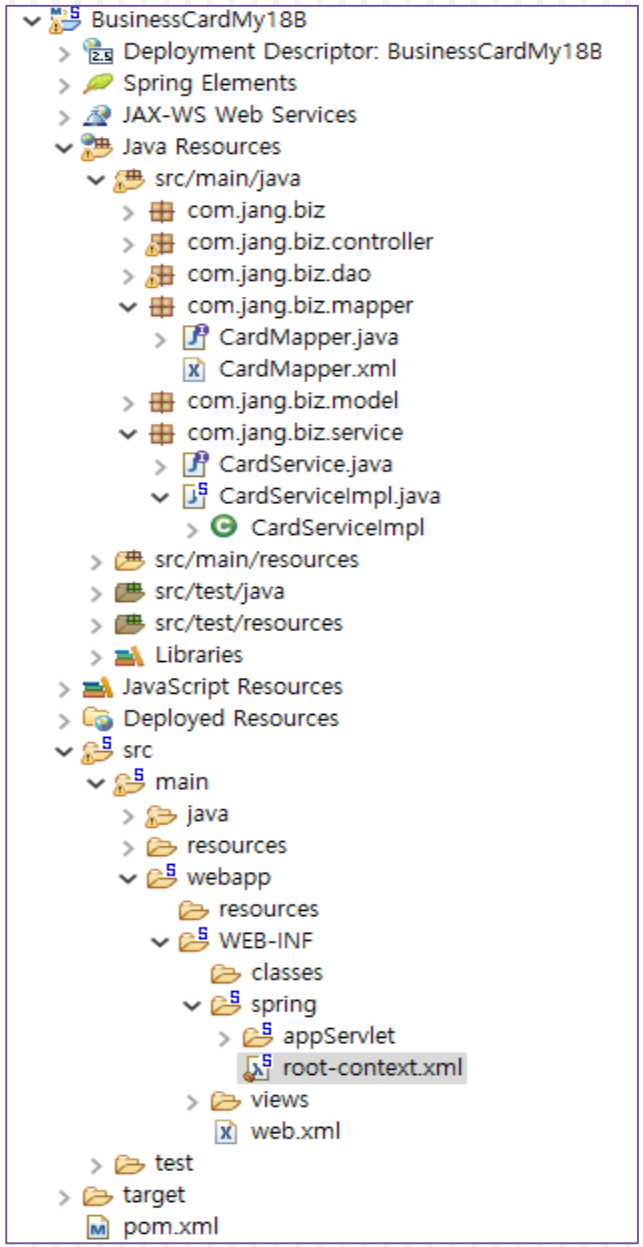
```
<!-- transaction manager, use JtaTransactionManager for global tx -->
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- define the SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="typeAliasesPackage" value="com.jang.biz.model" />
    <property name="mapperLocations">
        <list>
            <value>classpath*:com/jang/biz/mapper/*Mapper.xml</value>
        </list>
    </property>
</bean>

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

<!-- scan for mappers and let them be autowired -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="com.jang.biz.mapper" />
</bean>

/beans>
```

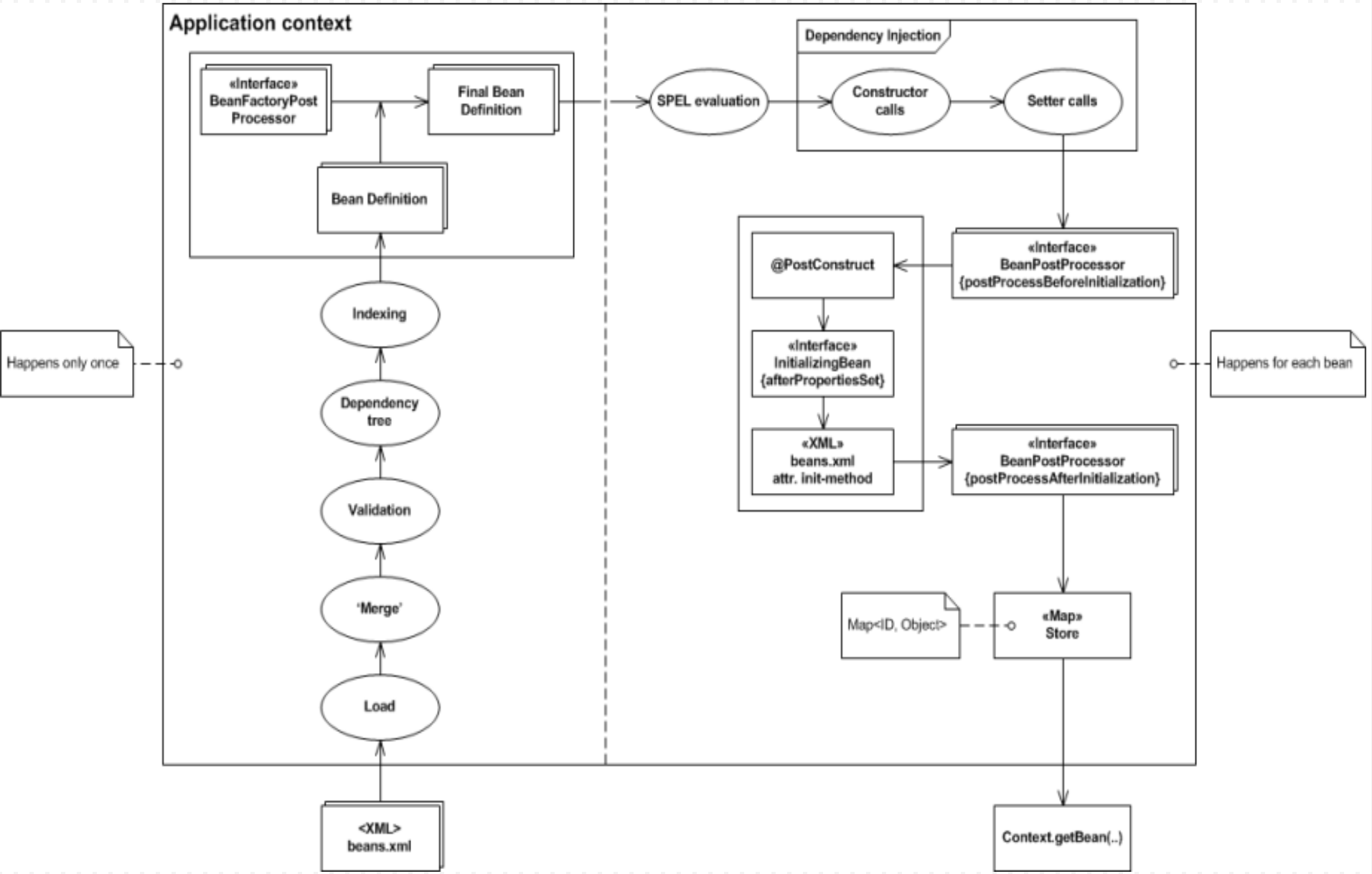
Mapper 클래스에서 직접 주입할 때 사용

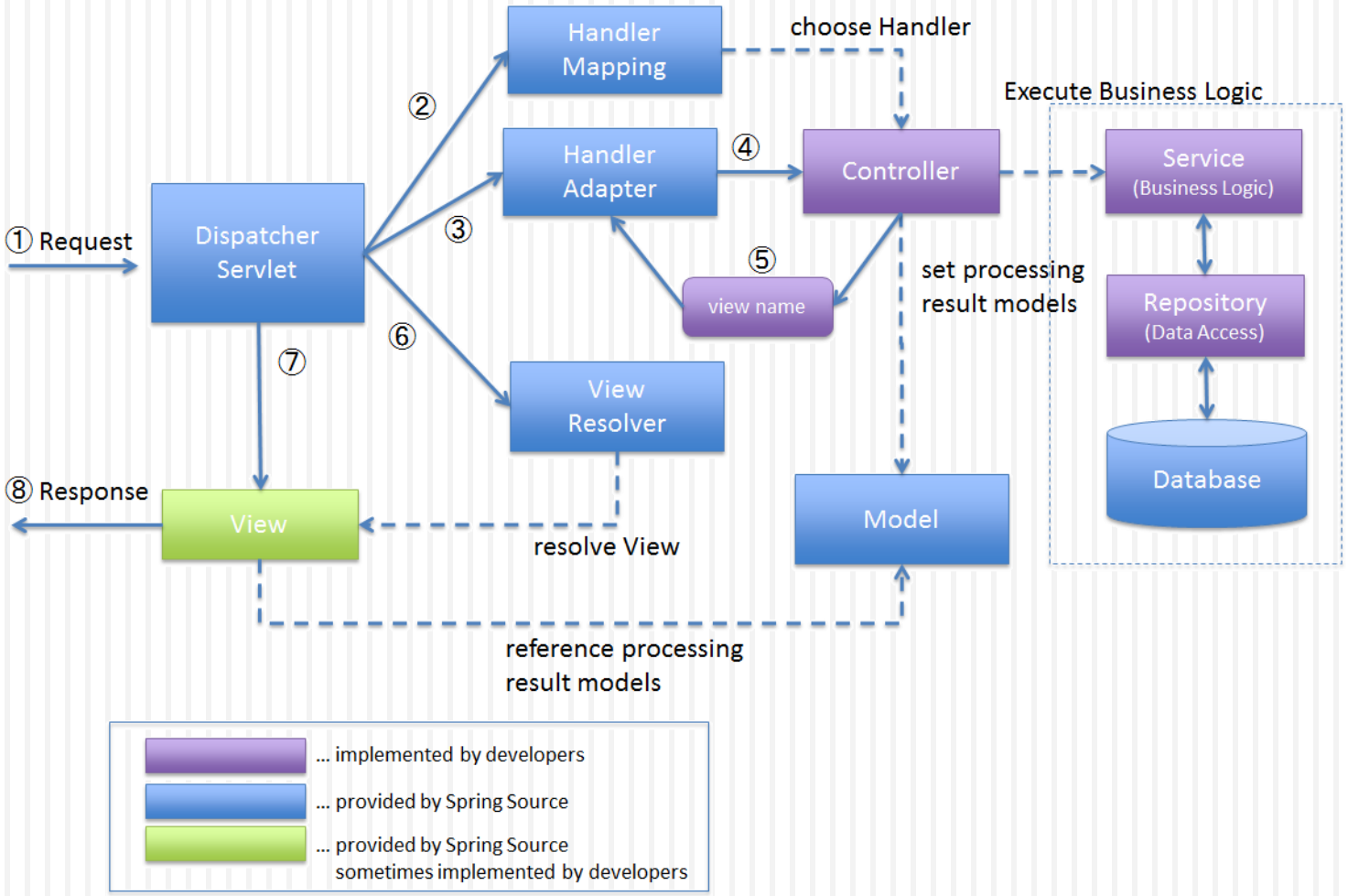
@Autowired

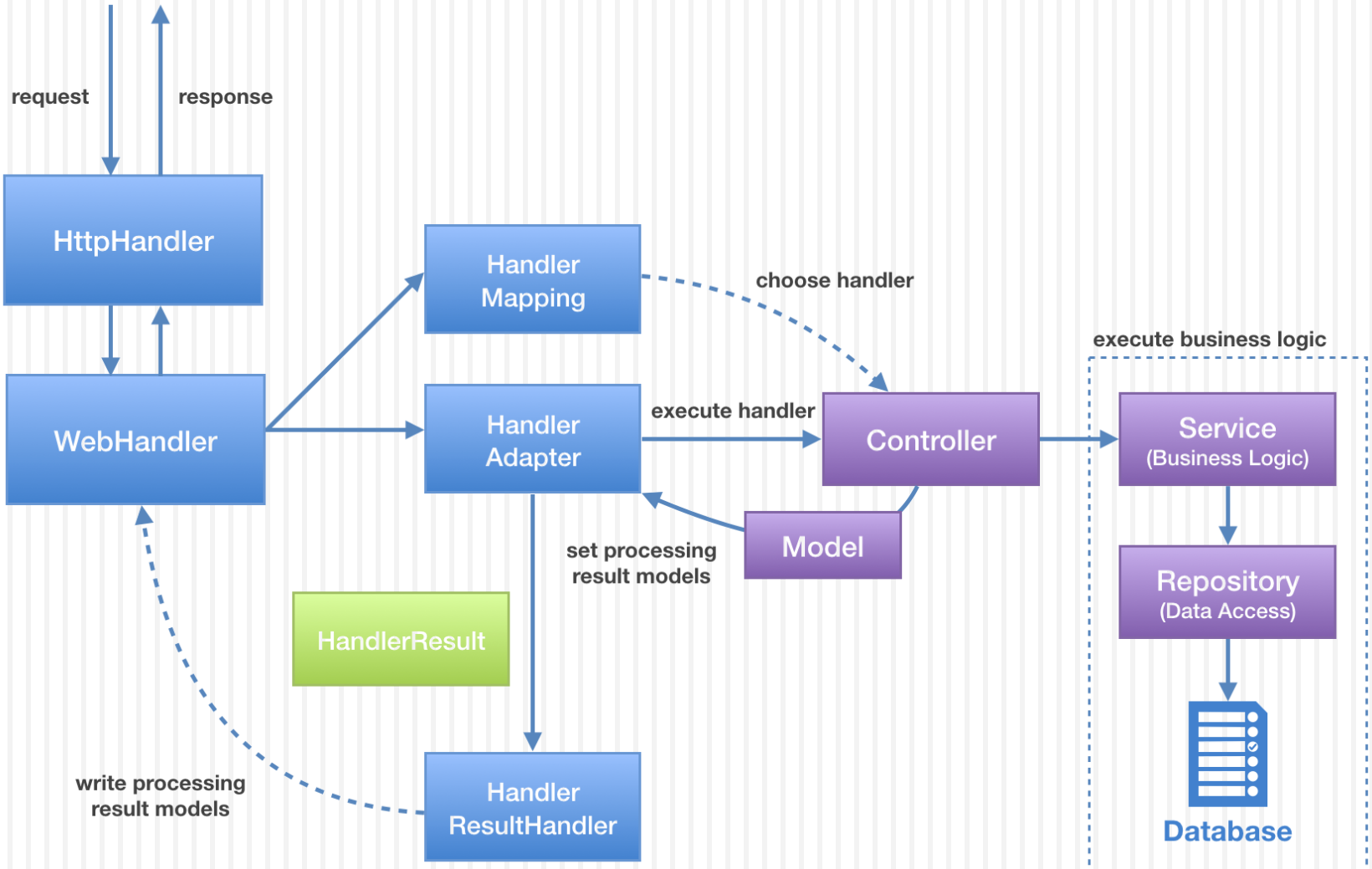
private SqlSessionTemplate sqlSession;

실행 및 결과 확인

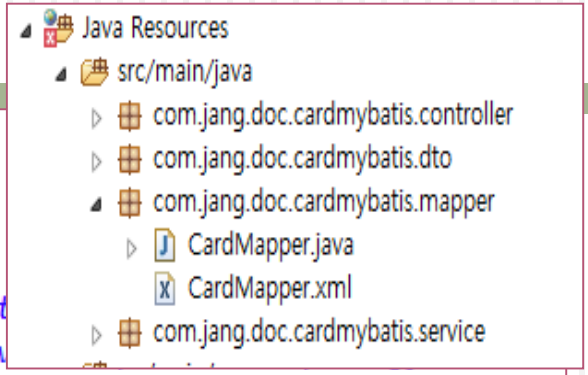

BusinessCardMy18B
Deployment Descriptor: BusinessCardMy18B
Spring Elements
JAX-WS Web Services
Java Resources
src/main/java
com.jang.biz
com.jang.biz.controller
com.jang.biz.dao
com.jang.biz.mapper
CardMapper.java
CardMapper.xml
com.jang.biz.model
com.jang.biz.service
CardService.java
CardServiceImpl.java
CardServiceImpl
src/main/resources
src/test/java
src/test/resources
Libraries
JavaScript Resources
Deployed Resources
src
main
java
resources
webapp
resources
WEB-INF
classes
spring
appServlet
root-context.xml
views
web.xml
test
target
pom.xml


Application context
«Interface»
BeanFactoryPost
Processor
Final Bean
Definition
Bean Definition
Indexing
Dependency
tree
Validation
'Merge'
Load
<XML>
beans.xml
Happens only once
SPEL evaluation
Dependency Injection
Constructor
calls
Setter calls
@PostConstruct
«Interface»
InitializingBean
{afterPropertiesSet}
«XML»
beans.xml
attr. init-method
«Interface»
BeanPostProcessor
{postProcessBeforeInitialization}
«Interface»
BeanPostProcessor
{postProcessAfterInitialization}
Map<ID, Object>
«Map»
Store
Happens for each bean
Context.getBean(..)


Handler Mapping
choose Handler
Execute Business Logic
②
Handler Adapter
④
Controller
Service (Business Logic)
① Request
Dispatcher Servlet
③
⑤
view name
set processing result models
Repository (Data Access)
⑥
⑦
View Resolver
Database
⑧ Response
View
resolve View
Model
reference processing result models
... implemented by developers
... provided by Spring Source
... provided by Spring Source sometimes implemented by developers


request
response
HttpHandler
WebHandler
Handler Mapping
choose handler
Handler Adapter
execute handler
Controller
execute business logic
Service
(Business Logic)
Repository
(Data Access)
Database
Model
set processing result models
HandlerResult
Handler ResultHandler
write processing result models

Servlet-context .xml




```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context
    xsi:schemaLocation="http://www.springframework.org/schema/mv
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/b
        http://www.springframework.org/schema/context http://www.springframework.org/schema

    <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructur

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />

    <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resou
    <resources mapping="/resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WE
    <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <context:component-scan base-package="com.jang.cardbiz" />

</beans:beans>
```

컨테이너에 생성할 객체: Auto scan
@Controller , @Service , @Repository **@Component**