

# Spring Framework

# Dependency Relationship

항목	설명
정의	한 클래스가 다른 클래스를 사용하는 사용 관계이다.
사용 예시	MemberDao가 MemberBean을 사용한다.
특징	하나의 클래스의 변화가 다른 클래스에 영향을 주는 관계이다. UML 표기법은 점선으로 된 화살표로 표현한다.

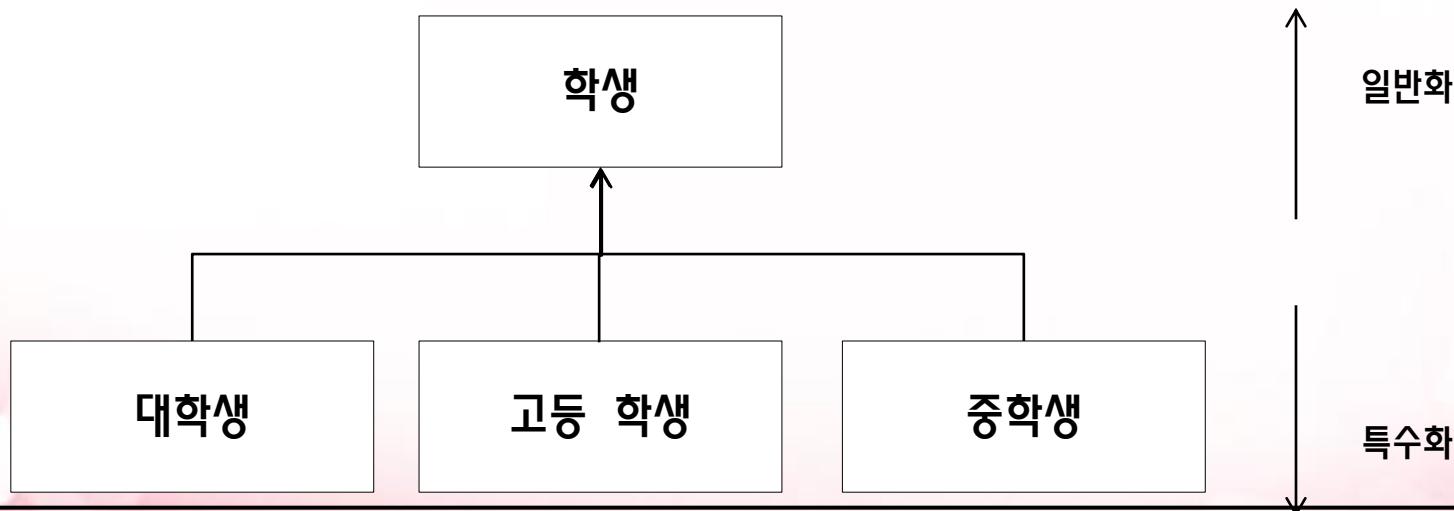


```
class Programmer{  
    //프로그래머가 컴퓨터를 사용하고 있다  
    private Computer computer ;  
    ...  
}
```

# Generalization Relationship

항목	설명
정의	일반화( Generalization), 특수화( Specialization) 관계 객체 지향의 상속 관계를 말한다.
일반화	보편적이고 일반적인 항목을 말한다. 모든 학생은 등록금을 낸다.
특수화	해당 환경에 대한 특수한 내용만 언급한다. 중고생은 교복을 입는다.

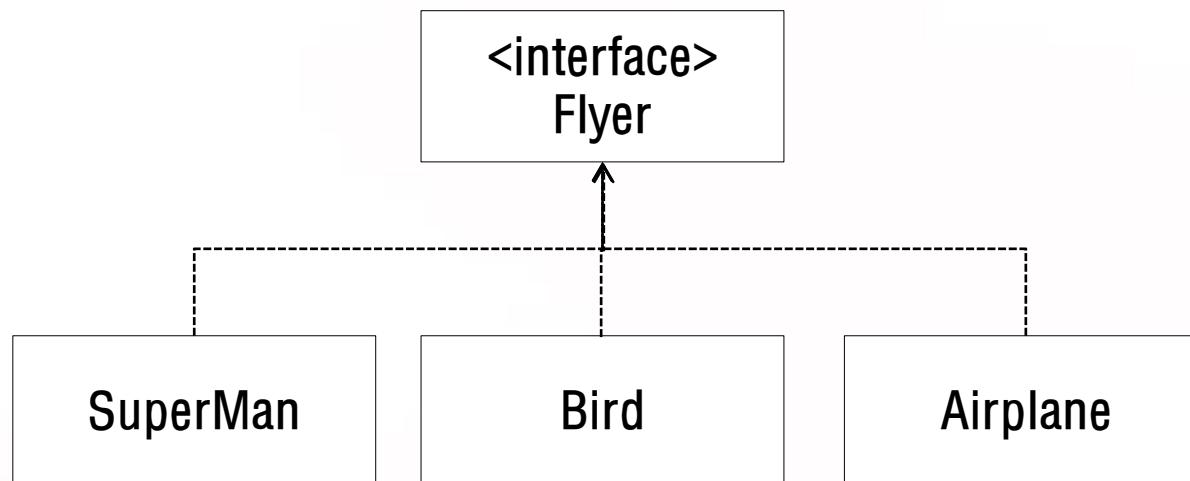
```
class University extends Student{  
    ...  
}
```



# Realization Relationship

항목	설명
정의	인터페이스와 실제 구현된 클래스간의 관계

```
class SuperMan implements Flyer{  
    ...  
}
```



# 프레임워크(Framework) 란?

항목	설명
프레임워크 정의	[어떤 것을 구성하는 구조, 뼈대]를 의미. (사전적 의미) [미리 만들어 놓은 클래스와 인터페이스, 정보 파일(XML)등으로 구성된 집합] 동적인 웹 페이지/웹 애플리케이션/웹 서비스를 개발을 유ти리티 모음.
대표적 JAVA 프레임워크	struts1 struts2 spring2.5 spring3.0 MyBatis(마이바티스), iBatis(아이바티스) SQL 문장으로 직접 DB 데이터를 다루는 SQL 맵퍼(mapper) Hibernate(하이버네이트) 자바 객체를 통해 간접적으로 DB 데이터를 다루는 객체 관계 맵퍼(Object-Relational mapper)이다. 탑-링크(TopLink)도 ORM에 속한다.

# 프레임 워크의 등장 배경

- 코드를 재사용한다면 많은 시간과 노력이 절약된다.
- 재사용성 높은 프로그램을 만드는 개발자가 우대되어야 한다.



# 재사용 방식

- Copy & Paste 재사용
- 함수의 재사용
  - 동일한 코드를 함수로 선언 후 재사용하기
- 클래스 상속과 재정의를 통한 재사용
  - 객체의 등장은 한 단계 높은 수준의 재활용 방식 지원
  - 주상 객체의 클래스
  - 상속을 통한 공통점의 공유
  - 변경이 필요한 부분만 재정의
- 디자인 패턴과 프레임 워크
  - 개발자간 공유하는 경험들의 모음(디자인 패턴)들을 모아서 적용하기 쉽도록 묶어둔 집합체
  - 특정 문제를 위한 클래스와 인터페이스의 집합체

# 프레임워크

- 어떤 문제를 해결하기 위한 잘 설계된 재사용 가능한 모듈들의 집합.
- 동작에 필요한 구조를 어느 정도 완성해 놓은 반제품 형태의 도구로써 완전한 애플리케이션 S/W는 아니다.
- 이 뼈대(프레임워크)에 살을 붙이는 작업이 필요하다.
- 프레임워크를 사용하면 JDBC 프로그래밍의 복잡함이나 번거로움 없이 간단한 작업만으로 데이터베이스와 연동되는 시스템을 빠르게 개발할 수 있다.
- **프레임워크의 구분**
  - 기능 프레임워크
    - 특정 기능 구현에 사용
  - 지원 프레임워크
    - 개발을 지원하는 프레임워크
    - ANT, JUnit 등등
  - 통합 프레임워크
    - 여러 기능의 프레임워크를 한 곳에 통합한 프레임워크

# 좋은 프레임워크의 조건

- 재사용성
- 영속성(Persistence)
  - 지속성이라고도 하며, 애플리케이션을 종료 후 다시 시작하면 이전에 저장한 데이터를 다시 불러올 수 있는 기술을 의미한다.
- 확장성
- 용이성
  - 사용하기 쉬워야 한다.
- 독립성
  - 특정 밴더의 플랫폼에 독립적이어야 한다.
- 꾸준한 지원
  - 버전 업
  - 패치 등

# Spring Framework

## spring 개요

레퍼런스 사이트 :

<http://docs.spring.io/spring/docs/1.2.9/api/>

<http://docs.spring.io/spring/docs/3.0.x/javadoc-api/>

<http://docs.spring.io/spring/docs/3.1.x/spring-framework-reference/html/>

<http://docs.spring.io/spring/docs/3.0.0.RELEASE/spring-framework-reference/htmlsingle/>

<http://docs.spring.io/spring/docs/4.0.0.RELEASE/javadoc-api/>

# 기본 개념

용어	항목에 대한 설명
POJO	Plain Old Java Object 순수한 형태의 자바 클래스 형태를 말한다. 다른 클래스를 상속 받아서 작성하는 클래스가 아니다.
IoC	Inversion of Control( 제어의 역전 ) 개발자가 직접 객체를 언제 생성하고 없앨지 결정하는 것이 아니라 컨테이너에게 맡긴다는 일임 시킨다. POJO(Plain Old Java Object) 객체의 생성에서 생명 주기의 관리까지를 IoC Container에 게 담당시킴으로써 (XML 이용) 개발에 있어서 편의성과 재사용성의 극대화를 추구하는 개념이다.
AOP	Aspect Oriented Programming( 관점 지향 프로그래밍 ) 기존의 객체 지향 언어에 의해 추구된 모듈화에 따라 많아진 중복된 코드라든지 공통된 처리에 관련한 것들을 관점으로 뽑아 내어 공통으로 처리하는 방식을 말한다.

# 주요 기능과 특징

용어	항목에 대한 설명
경량 컨테이너	자바 객체를 담고 있는 컨테이너이다. 자바 객체의 생성, 소멸과 같은 라이프 사이클을 관리한다. 스프링으로부터 필요한 객체를 가져와 사용할 수 있다.
DI 지원	Dependency Injection 설정 파일을 통해서 객체간의 의존 관계를 설정한다. 객체는 직접 의존하고 있는 객체를 생성하거나 검색할 필요가 없다.
AOP 지원	Aspect Oriented Programming 트랜잭션이나 로깅, 보안과 같이 여러 모듈에서 공통으로 필요로 하지만 실제 모듈의 핵심은 아닌 기능들을 분리해서 각 모듈에 적용
POJO 지원	Plain Old Java Object 스프링 컨테이너에 저장되는 자바 객체는 특정한 인터페이스를 구현하거나 클래스를 상속받지 않아도 된다. 기존에 작성한 코드를 수정할 필요 없이 스프링에서 사용 가능
트랜잭션 처리	설정 파일을 통해 트랜잭션 관련 정보를 입력하기 때문에 ,트랜잭션 구현에 상관없이 동일한 코드를 여러 환경에서 사용 가능하다.
영속성과 관련된 다양한 API를 지원	JDBC 를 비롯하여 iBATIS, 하이버네이트 ,JPA,JDO 등 데이터베이스 처리와 관련하여 널리 사용되는 라이브러리와의 연동을 지원한다.
다양한 API에 대한 연동을 지원	JMS,메일 ,스케줄링 등 엔터프라이즈 어플리케이션을 개발하는데 필요한 다양한 API를 설정 파일을 통해서 손쉽게 사용 가능

# Spring Framework

## Spring DI(Dependency Injection)

# DI(Dependency Injection)

용어	항목에 대한 설명
Bean	클래스로부터 생성된 객체를 스프링에서는 [ 빈 ] 이라고 부른다.
DI 개념	객체(Bean) 간의 의존 관계를 외부의 파일 ( 스프링 설정 파일 )에서 설정하겠다.
DI 컨테이너	어떤 클래스가 필요로 하는 인스턴스를 자동으로 생성 / 취득하여 연결해준다. 스프링도 일종의 DI 컨테이너이다.
스프링 설정 파일	Bean을 관리한다. 기본 파일 명은 applicationContext.xml( 일반적으로 비즈니스 로직 계층)이다.
BeanFactory 인터페이스	빈을 생성하고 설정, 관리하는 역할을 수행한다. 스프링에서 org.springframework.beans.factory.BeanFactory가 기본 IoC 컨테이너이다. getBean() 메소드를 지원한다. 가장 일반적으로 많이 사용하는 클래스는 XmlBeanFactory 이다.
Injection	[주입] 이라고 한다. 각 클래스간의 의존 관계를 관리하기 위한 방법이다. 생성자 Injection, setter Injection 이 있다.
wiring	DI 를 이용 (xml 설정 파일 ) 하여 객체간의 연관 관계를 형성하는 작업이다.

# 스프링 관련 인터페이스/클래스

항목	설명
BeanFactory 인터페이스	Bean 객체를 관리, Bean 객체 간의 의존 관계를 설정해주는 기능을 제공한다. 구현된 클래스로는 XmlBeanFactory 클래스가 있다.
XmlBeanFactory 클래스	외부 자원으로부터 설정 정보를 읽어 와서 Bean 객체를 설정한다.
Resource 인터페이스	외부의 다양한 자원을 동일한 방식으로 읽어 들일 수 있는 방법을 제시한다. Resource를 이용하여 XmlBeanFactory에 설정 정보를 전달한다. 구현된 클래스들 FileSystemResource : 파일 시스템의 특정 파일로부터 정보를 읽어 온다. ClassPathResource : 클래스 패스(src 폴더)에 있는 자원으로부터 정보를 읽어 온다.
ApplicationContext 인터페이스	BeanFactory 인터페이스를 상속 받는 하위 인터페이스 Bean 객체 라이프 사이클, 파일과 같은 자원 처리 추상화, 메시지 지원 및 국제화 지원, 이벤트 지원, xml 스키마 확장을 통한 편리한 설정
WebApplicationContext 인터페이스	웹 애플리케이션을 위한 ApplicationContext 인터페이스 중의 1개이다. ServletContext 마다 1개 이상의 WebApplicationContext를 가질 수 있다. 구현된 클래스들 ClassPathXmlApplicationContext 클래스 패스(src 폴더)에 위치한 xml 파일로부터 설정 정보를 로딩한다. FileSystemXmlApplicationContext 파일 시스템에 위치한 xml 파일로부터 설정 정보를 로딩한다. XmlWebApplicationContext 웹 애플리케이션에 위치한 xml 파일로부터 설정 정보를 로딩한다.

# xml 설정 파일 읽기

- 필요한 jar 파일 : spring.jar, commons-logging.jar
- 메인 메소드 내에서 다음과 같이 코딩한다.
  - //applicationContext.xml 파일은 /src/ 폴더 아래에 둘 것
  - //데이터 출처를 리소스 (resource)으로 해석
  - Resource resource = new ClassPathResource("applicationContext.xml");
  - //Bean : 자바의 객체를 Spring에서는 Bean이라고 부른다
  - //Bean 공장 (BeanFactory)을 이용하여 객체 생성
  - BeanFactory factory = new XmlBeanFactory(resource);
  - //해당되는 타입으로 다운 캐스팅을 해줘야 한다.
  - Player player = (Player)factory.getBean("player");
  - player.forward();

# 자바 코드와 스프링 코드 비교

항목	사용 예시
일반적인 자바 코드	import pkgSample3.CDPlayer ; CDPlayer player = new CDPlayer();
스프링 설정 파일	<bean id="player" class="pkgSample3.CDPlayer"/>

```
<beans xmlns="http://www.springframework.org/schema/beans"  
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
       xsi:schemaLocation=  
           "http://www.springframework.org/schema/beans  
           http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
    <bean id="player" class="pkgSample3.CDPlayer" />  
</beans>
```

```
Resource resource = new ClassPathResource("applicationContext.xml");  
  
BeanFactory factory = new XmlBeanFactory(resource);  
  
Player bean = (Player)factory.getBean("player");  
  
bean.forward();  
bean.reverse();
```

# applicationContext.xml 파일의 <bean> 요소

항목	사용 예시
일반적인 자바 코드를 이용한 방법	<pre>import pkgSampleCDPlayer ; CDPlayer player = new CDPlayer();</pre>
인스턴스 설정을 위한 xml 설정 파일을 이용한 방법	<pre>&lt;bean id="player" class="pkgSampleCDPlayer"/&gt;</pre>

속성	의미	디폴트 값
id	Bean에 대해 붙이는 식별자(정해진 이름)	
name	id에 대한 별명(Alias), 복수 정의가 가능	
class	Bean의 클래스명, 완전 수식명으로 기술함	
parent	Bean 정의를 계승할때에 지정하는 부모 Bean의 ID	
abstract	Bean 클래스가 추상 클래스인지 아닌지	false
singleton	Bean이 Singleton으로서 관리될지 안될지	false
lazy-init	Bean의 지연 로딩을 행할지 안 할지	false
autowire	구현 클래스를 자동적으로 연결	no
dependency-check	의존 관계 체크 방법	none
dependence-on	이 Bean이 의존하는 Bean의 이름, 먼저 초기화된 것이 보증됨	
init-method	Bean의 초기화 때에 실행시킬 메소드	
destory-method	Bean 컨테이너의 종료시에 실행시킬 메소드	

# 빈(Bean)의 유효 범위

- IoC Container에 의해 관리되는 Bean의 기본 범위는 singleton이다.
- scope 속성으로 다르게 지정하는 것이 가능하다.

유효 범위	설명	예시 및 비교
Singleton	스프링 컨테이너에 한 개의 빈 객체만 존재한다.	<bean id="" class="" /> default 가 singleton
Prototype	빈을 사용할 때마다 객체 생성 호출 때마다 서로 다른 객체가 생성	<bean id="" class="" scope="prototype"/>
Request	HTTP 요청마다 빈 객체를 생성	<bean id="" class="" scope="request"/>
Session	HTTP 세션마다 빈 객체를 생성	<bean id="" class="" scope="session"/>
Global Session	글로벌 HTTP 세션에 대해 빈 객체를 생성	<bean id="" class="" scope="globalSession"/>

# Annotation을 이용한 와이어링(wiring)

- SpringDI 어노테이션을 이용하여 자동으로 wiring을 수행하려면 다음과 같이 xml 파일을 설정해야 한다.
  - <!-- 다음 문장은 Annotation 기반 Wiring을 사용하겠다고 선언하는 것이다. -->
  - <context:annotation-config></context:annotation-config>

The diagram illustrates the mapping between XML configuration and Java code for annotation-based wiring.

**XML Configuration:**

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:p="http://www.springframework.org/schema/p"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
                          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                          http://www.springframework.org/schema/context
                          http://www.springframework.org/schema/context/spring-context-3.0.xsd">
    <!-- 다음 문장은 Annotation 기반 Wiring을 사용하겠다고 선언하는 것이다. -->
    <context:annotation-config></context:annotation-config>
</beans>
```

A callout box points from the XML configuration to the corresponding Java code, containing the text: "다음과 같이 코드를 하게 되면 ListController 컨트롤러는 자동으로 mainLogic 객체를 Wiring하게 된다."

**Java Code:**

```
@Controller
public class ListController {
    private MainLogic mainLogic;
    @Autowired
    public void setMainLogic(MainLogic mainLogic) {
        this.mainLogic = mainLogic;
    }
}
```

A callout box points from the Java code to the XML configuration, containing the text: "<!-- 다음 문장은 Annotation 기반 Wiring을 사용하겠다고 선언하는 것이다. -->

# Injection의 종류

## • Constructor( 생성자 ) Injection

- 생성자를 통해 클래스 간의 의존 관계를 연결하여 항상 인스턴스를 참조할 수 있게 한다.
- 참조하는 클래스의 인스턴스가 생성되지 않아도 설정 파일을 이용하여 의존 관계를 정의할 수 있다
- XML 설정 파일에서 <bean> 요소의 하위 요소로 <constructor-arg>를 사용한다.

Foo 클래스가  
Bar 클래스를 참조하고 있다.



속성	설명
index	Constructor의 몇 번째의 인수에 값을 전달할 것인지를 지정
type	Constructor의 어느 데이터형의 인수에 값을 전달할 것인지를 지정
ref	자식 요소 <ref bean = "bean 명"/> 대신 사용 가능
value	자식 요소 <value>값</value> 대신 사용 가능

예) 생성자를 이용하여 연결 관계를 설정하는 경우 - ref를 이용하여 객체를 전달

Foo.java	applicationContext.xml
public class Foo { private Bar bar; public Foo(Bar bar) { this.bar = bar; } }	<bean id = "foo" class = "Foo"> <constructor-arg> <ref bean = "bar"/> </constructor-arg> </bean> <bean id = "bar" class = "Bar"/>

예) 전달 인자가 두 개인 생성자에 연결 관계 설정하기 - **index, value** 속성으로 값을 지정

Foo.java	applicationContext.xml
public class Foo { public Foo(int a, String b) { ... } }	<bean id = "foo" class = "Foo"> <constructor-arg index = "0"> <value>25</value> </constructor-arg> <constructor-arg index = "1" value = "Hello"> </bean>

# Injection의 종류

- **setter Injection**

- Setter 메소드를 이용하여 클래스 간의 의존 관계를 연결시킨다.
- 예제에서 Foo 클래스의 멤버로 존재하는 Bar형 레퍼런스 변수 bar에 대한 setter는 setBar() 메소드가 된다.

Foo.java	applicationContext.xml
<pre>public class Foo {     private Bar bar;     public setBar(Bar bar) {         this.bar = bar;     } }</pre>	<pre>&lt;bean id = "foo" class = "Foo"&gt;     &lt;property name = "bar" ref = "bar"&gt; &lt;/property&gt; &lt;/bean&gt; &lt;bean id = "bar" class = "Bar"/&gt;</pre> <p>setBar 메소드를 의미한다. 인스턴스 변수나 매개 변수 이름은 달라도 상관 없다.</p>

- setBar() 메소드를 호출할 때 전달해 준 Bar 인스턴스를 이용하여 Foo는 Bar 인스턴스를 참조한다.
- Setter injection에서는 <property> 요소의 name 속성을 이용하여 값의의 존 관계를 연결 시킬 대상이 되는 필드 값을 지정한다.
- 객체를 전달해야 할 경우에는 <ref> 요소를 사용한다.

property 요소의 속성은 다음과 같다.

속성	설명
ref	자식 요소 <ref bean = "bean 명"/> 대신에 사용할 수 있다.
value	자식 요소 <value>값</value> 대신에 사용할 수 있다.

# Injection의 종류

- p 네임스페이스

- XML의 P 네임 스페이스를 이용하면 <property> 태그를 사용하지 않고 좀 더 간단한 방법으로 setter Injection 시킬 수 있다.

xml 네임 스페이스에 p 네임스페이스를 명시한다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:p="http://www.springframework.org/schema/p"
       ... 이하 헉득>

    <!-- Dependency Injection에 사용될 setter 설정하기 -->
    <!-- xml 네임 스페이스를 이용한 Injection 기법 Injection에 사용될 setter 설정하기 -->
    <bean name="/hello.htm"
          class="controller.HelloController" p:myBean-ref="myBean"/>

    ... 이하 증략

</beans>
```

# 라이프-사이클 관련 인터페이스

인터페이스	항목에 대한 설명
InitializingBean 인터페이스	객체 생성, 프로퍼티 초기화 프로퍼티가 올바르게 설정이 되었는가 여부를 검사하는 기능 빈 객체에 프로퍼티에 의존하는 객체 값의 주입이 완료된 후 수행 afterPropertiesSet() 메소드를 오버라이딩해야 한다. 초기화 후 프로퍼티 값을 검정하는 영역이다. <bean> 태그의 init-met hod 속성과 동일한 기능을 수행한다.
BeanNameAware 인터페이스	스프링 컨테이너가 이 메소드를 호출하여 Bean의 이름을 얻어 낸다. setBeanName(String beanName) 메소드를 오버라이딩해야 한다.
BeanFactoryAware 인터페이스	BeanFactory를 빈에게 전달할 때 사용된다. public void setBeanFactory(BeanFactory beanFactory)
DisposableBean 인터페이스	사용된 객체의 자원을 반납하는 경우에 사용 (소멸 메소드) 빈 객체가 소멸되기 전에 자원 해제와 관련된 코드를 기술할 목적 destroy() 메소드 <bean> 태그의 destroy-met hod 속성과 동일한 기능을 수행한다.

# Spring Framework

## Spring Annotation

# Annotation

- Annotation(어노테이션)

- 프로그램에 대한 부연 설명, 덧붙이는 말
- compiler 나 외부 Tool에게 프로그램에 대한 부가적인 설명을 제공
- 코드 수행에 영향을 미치지 않는다
- JSP 나 프레임 워크 (스프링 등)에 xml 파일 내의 복잡한 설정을 없애기 위한 용도로 사용

```
@Override  
public void paranemethod() {  
    // 이 메소드는 오버라이딩되었다  
}
```

기존 자바에서 사용하던 기법

```
//@Aspect : 설정 파일에 설정하지 않고, Advice를 적용할 수 있게 하는 어노테이션  
@Aspect  
public class MyAspect {  
  
    //@Before : Before Advice로 사용할 메소드에 추가하는 어노테이션  
    @Before("execution(public void annotation.BizService.*())")  
    public void beforeMethod(JoinPoint jp){  
    }  
}
```

스프링에서 사용되는 예시

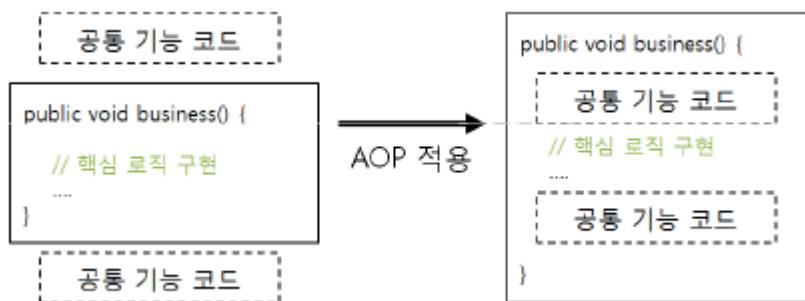
# Spring Framework

## AOP(Aspect Oriented Programming)

# AOP 소개

- AOP(Aspect Oriented Programming)

- 문제를 해결하기 위한 핵심 관심 사항(core concern, 핵심 로직)과 전체에 적용되는 공통 관심 사항(cross-cutting concern, 공통 기능 )을 기준으로 프로그래밍함으로써 공통 모듈을 여러 코드에 쉽게 적용할 수 있도록 해 준다.



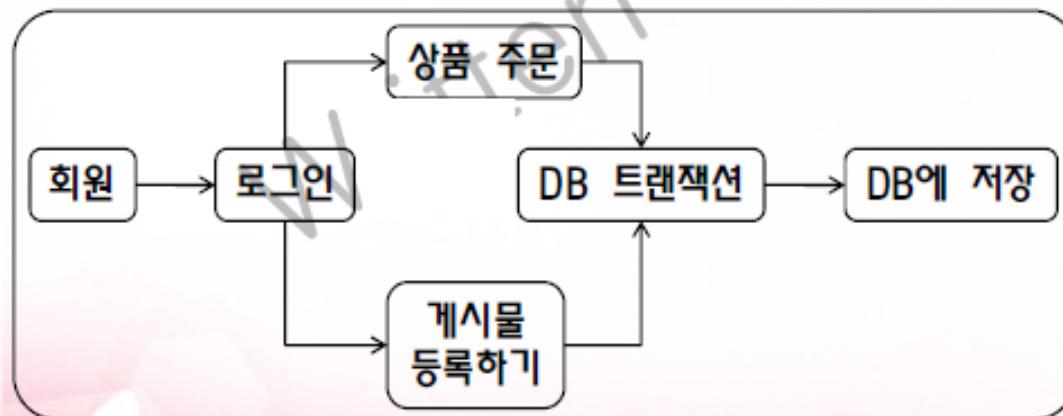
- 공통 관심 사항을 구현한 코드를 핵심 로직을 구현한 코드 안에 삽입

- 핵심 로직을 구현한 코드에서 공통 기능을 직접적으로 호출하지 않는다.
- 핵심 로직을 구현한 코드를 컴파일하거나, 컴파일된 클래스를 로딩하거나, 로딩된 클래스의 객체를 생성할 때 AOP가 적용되어 핵심 로직 구현 코드 안에 공통 기능이 삽입된다.
- 공통 기능이 변경되더라도 핵심 로직을 구현한 코드를 변경할 필요가 없다.  
(공통 기능 코드를 변경한 뒤 핵심 로직 구현 코드에 적용만 하면 됨)

# AOP 이론

- 코드를 기능별로 분리하는 것이 유리 (유지 보수 용이)
- AOP의 개념:
  - 기능별로 코드를 완전히 분리하자
  - 관점의 분리 (Separation of Concerns)
- 기능의 통합은 AOP Framework 가 담당한다.

이 그림에서 [로그인]과 [DB 트랜잭션]등은 **공통 관심 사항**에 속한다.  
예를 들어서 [상품 주문]을 하거나, [게시물 등록하기]를 수행하려면  
반드시 [로그인]이라는 액션을 취해야 한다.



## 공통 관심사항(Aspect)

1. 로그인
2. 트랜잭션
3. 보안 문제
4. 사용 권한 검사 등등

# AOP 용어

용어	항목에 대한 설명
Core-concerns	핵심 관심 사항(상품 주문 하기 , 게시물 등록하기 등등)
Aspect	여러 객체에 공통으로 적용되는 공통 관심 사항(Cross-cutting-concerns) 예를 들어 [로그인 검사]나 [사용 권한 검사] logic 등이 여기에 속한다. 즉 여러 개의 클래스에서 공통적으로 사용되는 소스 모듈을 의미한다.
Joinpoint	Advice를 적용할 수 있는 모든 요소들을 의미 메소드 호출 ,필드 값 수정 ,예외 발생 등이 Joinpoint에 해당 각각의 구체적인 적용 지점을 Pointcut 이라고 한다.
Advice	Aspect가 수행해야 할 어떠한 작업 목록 Aspect(공통 관심 사항)을 언제 (~하기 전에 ,~한 직후에)핵심 로직에 적용시킬 것인가?를 정의함
Pointcut	Joinpoint를 구성하는 각각의 구체적인 적용 지점(메소드 실행, 예외 처리 ,속성 변경 )을 말한다. 스프링에서는 AspectJ Pointcut Expression을 사용하여 Pointcut을 표현한다. 스프링에서 핵심 관심 사항을 표현한 클래스 (Target Object)내의 특정 메소드를 표현함 Aspect 가 Advice 할 Joinpoint의 영역을 좁히는 일
Weaving	Advice를 핵심 로직 코드에 적용하는 것을 의미 공통 코드를 핵심 로직 코드에 삽입하는 것.

# 스프링의 Advice 타입

Advice 타입	항목에 대한 설명
Around Advice	Joinpoint 앞과 뒤에서 실행되는 Advice org.aopalliance.intercept.MethodInterceptor
Before Advice	Joinpoint 앞에서 실행되는 Advice org.springframework.aop.MethodBeforeAdvice
After Returning Advice	Joinpoint 메서드 호출이 정상적으로 종료된 뒤에 실행이 되는 Advice org.springframework.aop.MethodReturningAdvice
After Throwing Advice	예외가 던져질 때 실행되는 Advice org.springframework.aop.ThrowsAdvice
After Advice	메서드가 정상적으로 실행되는지 또는 예외를 발생시키는지 여부에 상관없이 적용되는 Advice try~catch~finally에서 finally와 비슷
Introduction	클래스에 인터페이스와 구현을 추가하는 특수한 Advice org.springframework.aop.Introduction.Interceptor

# 스프링에서의 AOP

- 엔터프라이즈 어플리케이션을 구현하는데 필요한 기능을 제공
- 특징
  - 스프링은 자체적으로 프록시 기반의 AOP를 지원 (메소드 호출 Joinpoint 만을 지원)
  - 필드 값 변경과 같은 Joinpoint를 사용하고 싶다면 AspectJ와 같이 풍부한 기능을 지원하는 AOP 도구를 사용
  - 자바 기반 (별도의 문법을 익힐 필요가 없음)
- AOP 구현 방식
  - 1) 스프링 API를 이용한 AOP 구현
  - 2) POJO 클래스를 이용한 AOP 구현
  - 3) AspectJ 5 에서 정의한 @Aspect 어노테이션 기반의 AOP 구현
- 어떤 방식을 사용하더라도 내부적으로는 프록시를 이용하여 AOP 가 구현되므로 메소드 호출에 대해서만 AOP를 적용할 수 있음.

# xml 스키마를 이용한 AOP 설정

- Spring2 버전부터 AOP 설정과 관련된 `aop` 네임스페이스 및 `aop` 네임스페이스와 관련된 XML 스키마가 추가되었다.
- `aop` 네임스페이스와 관련된 XML 스키마는 다음과 같이 `<bean>` 태그에 명시할 수 있다.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"    <-- AOP 관련 항목
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
                           http://www.springframework.org/schema/aop           <-- AOP 관련 항목
                           http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">   <-- AOP 관련 항목
...
</beans>
```

# xml 스키마를 이용한 AOP 설정

태그	항목에 대한 설명
<aop:config>	aop 설정을 하겠다는 의미이다.
<aop:aspect>	aspect(공통 관심 사항)을 지정 ref 속성 : 공통 기능을 구현하고 있는 Bean을 명시(사용할 Aspect를 명시한다.)
<aop:pointcut>	Pointcut을 설정 expression 속성 : Advice를 적용할 Pointcut에 대한 정보를 지정 (AspectJ 표현식)
<aop:어드바이스>	Advice를 설정한다. pointcut 속성 : 직접 AspectJ 표현식을 사용한 Pointcut 지정 pointcut-ref 속성 : <aop:pointcut> 태그를 이용하여 작성한 Pointcut 지정

```
<!-- 사용 예시 -->
<bean id="logging" class="spring.pojo.LoggingAspect" />

<aop:config><!-- aop 설정을 하겠다 -->
    <!-- logging이라는 빈을 Aspect으로 사용하겠다 -->
    <aop:aspect id="loggingAspect" ref="logging">
        <!-- expression 속성에는 Pointcut을 정의하는 AspectJ의 표현식을 명시한다. -->
        <aop:pointcut id="publicMethod"
            expression="execution(public * kame.spring.chap03.core..*(..))" />
        <!-- 해당 메소드(logging)를 Around Advice으로 설정 -->
        <!-- pointcut-ref 속성에는 사용할 Pointcut을 명시한다. -->
        <aop:around pointcut-ref="publicMethod" method="logging" />
    </aop:aspect>
</aop:config>
```

# 프록시를 이용한 AOP 구현

- **특징**

- 스프링은 Aspect의 적용 대상(target)이 되는 객체에 대한 프록시를 만들어 제공한다.
- 대상 객체를 사용하는 코드는 대상 객체를 프록시를 통해서 간접적으로 접근한다.
- 프록시는 공통 기능을 실행한 뒤 대상 객체의 실제 메소드를 호출하거나 또는 대상 객체의 실제 메소드가 호출된 뒤 공통 기능을 실행한다.

- **AOP 적용**

- 어떤 대상 객체에 대해 AOP를 적용할지의 여부는 설정 파일을 통해서 지정한다.
- 설정 정보를 이용하여 런타임에 대상 객체에 대한 프록시 객체를 생성한다.

# @Aspect 어노테이션을 이용한 AOP

- AspectJ 5 버전에 새롭게 추가된 어노테이션
- 사용 방법
  - xml 설정 파일에 <aop:aspectj-autoproxy> 태그를 추가한다.
  - 자바 소스에서 @Aspect 어노테이션을 사용한다.
  - 이렇게 적용된 빈은 Aspect로 사용이 가능하게 된다.
- 추가 설명
  - <aop:aspectj-autoproxy> 태그는 @Aspect 어노테이션이 적용된 클래스를 로딩한다.
  - 해당 클래스에 명시된 Advice 및 Pointcut 정보를 이용하여 알맞은 빈 객체에 Advice를 적용한다.

# @Aspect 어노테이션을 이용한 AOP

어노테이션	설명
@Aspect 어노테이션이 적용된 클래스	Advice 로 사용될 메소드에는 Advice를 의미하는 어노테이션을 적용
@Before 어노테이션	Before Advice 로 사용할 메소드에 적용
@After 어노테이션	After Advice 로 사용할 메소드에 적용
@AfterReturning 어노테이션과 @AfterThrowing 어노테이션	각각 리턴 값과 예외 객체를 전달받을 파라미터의 이름을 지정 Advice 메소드에서 리턴 값과 예외 객체를 사용할 수 있도록 함
@Around 어노테이션을 제외한 나머지 Advice를 표시하는 어노테이션들	메소드는 프록시 대상 객체에 대한 정보가 필요한 경우 첫 번째 파라미터로 JoinPoint를 사용.
@Around 어노테이션 사용	ProceedingJoinPointproceed() 메소드를 호출하여 프록시 대상 객체의 메소드를 호출한다.

# @Aspect 어노테이션을 이용한 AOP

//@Aspect 어노테이션이 적용된 클래스는 Advice 구현 메소드나 Pointcut 정의를 파일에 포함할 수 있다.

@Aspect //Aspect 클래스 구현하겠다.

@Order(3) //@Order : Advice 적용 순서를 명시적으로 지정하고자 하는 경우에 사용

```
public class ProfilingAspect {
```

//Pointcut 설정하기 : aop 패키지 하위의 모든 메소드에 대하여 ...

@Pointcut("execution(public \* aop..\*(..))")

private void profileTarget() {

//@Around : Around Advice를 적용하겠다.

@Around("profileTarget()")//profileTarget() 메소드에 정의된 Pointcut의 Advice를 적용하겠다.

//Around Advice를 적용한 메소드는 반드시 ProceedingJoinPoint를 첫번째 매개변수로 사용해야 한다.

//그렇지 않으면, 스프링은 예외를 발생시킨다.

public Object trace(ProceedingJoinPoint joinPoint) throws Throwable {

... 이하 중략

}

}

## 설명

aop 패키지 이하의 모든 메소드에 대하여  
실행 전후에 trace() 메소드를 수행하겠다.

# MethodBeforeAdvice 구현

- MethodBeforeAdvice 인터페이스
  - org.springframework.aop.MethodBeforeAdvice interface
  - 객체의 메소드 수행 전에 공통 기능을 적용하고 싶을 때에 사용되는 Advice 인터페이스
- 메소드 정의 =>

```
public interface MethodBeforeAdvice extends BeforeAdvice {  
    void before(Method method, Object[] args, Object target) throws Throwable;  
}
```
- before() 메소드
  - 파라미터
    - Method method : 대상 객체에서 실제로 호출될 메소드를 나타내는 Method 객체
    - Object[] args : 메소드 호출 시 전달된 인자 목록
    - Object target : 실제 대상 객체
  - 특징
    - 대상 객체의 메소드가 호출되기 전에 실행
    - 대상 객체의 메소드는 MethodBeforeAdvice의 before() 메소드가 실행된 후에 실행
- 주의 사항
  - before() 메소드에서 Method를 이용하여 대상 객체의 메소드를 호출하게 되면 대상 객체의 메소드가 중복 실행됨
  - MethodBeforeAdvice의 before() 메소드에서 예외를 발생시키면 대상 객체가 실행되지 않음

# MethodInterceptor 인터페이스

- MethodInterceptor 인터페이스
  - org.springframework.intercept.MethodInterceptor interface
  - AOP Alliance 에서 정의한 인터페이스
  - MethodBeforeAdvice,AfterReturningAdvice,ThrowsAdvice를 합쳐 놓은 인터페이스
  - 필요에 따라서 대상 객체의 메서드를 호출할지의 여부를 결정할 수 있고 ,원하는 대로 제어할 수 있기 때문에 다양한 용도로 활용이 가능
- 메서드 정의

```
public interface MethodInterceptor extends Interceptor {  
    Object invoke(MethodInvocation invocation) throws Throwable;  
}
```
- Advice 와의 차이점
  - invoke() 메서드가 호출되기 이전이나 이후에 대상 객체의 메서드가 호출되지 않는다
  - invoke() 메서드에서 직접 대상 객체의 메서드를 호출해 주어야 한다.

# AfterReturningAdvice 인터페이스

- AfterReturningAdvice 구현
  - org.springframework.aop.AfterReturningAdvice interface
  - 대상 객체의 메서드를 실행한 후에 공통 기능을 적용하고 싶을 때에 사용되는 Advice 인터페이스
- afterReturning() 메서드
  - 파라미터
    - Object returnValue : 대상 객체의 메서드가 리턴한 값
    - Method method : 대상 객체에서 실제로 호출될 메서드를 나타내는 Method 객체
    - Object[] args : 메서드 호출 시 전달된 인자 목록
    - Object target : 실제 대상 객체
  - 특징
    - 대상 객체의 메서드가 정상적으로 실행을 완료한 이후에 실행
    - 대상 객체의 메서드가 예외를 발생시킬 경우에는 afterReturning() 메서드는 실행되지 않음
    - 대상 객체의 리턴 값을 첫번째 파라미터로 전달 받긴 하지만 리턴 값을 변경할 수는 없음

# JoinPoint 인터페이스

- JoinPoint 인터페이스의 메소드

- 호출되는 대상 객체, 메소드 그리고 전달되는 파라미터 목록에 대해 접근할 수 있는 메소드 제공
- Signature getSignature() : 호출되는 메소드에 대한 정보를 구함
- Object getTarget() : 대상 객체를 구함
- Object[] getArgs() : 파라미터 목록을 구함

- Signature 인터페이스의 메소드

- 호출되는 메소드와 관련된 정보를 제공
- String getName() : 메소드의 이름을 구함
- String toLongString() :
  - 메소드를 완전하게 표현한 문장을 구함 (메소드의 리턴 타입, 파라미터 타입이 모두 표시됨)
- String toShortString() :
  - 메소드를 축약해서 표현한 문장을 구함 (기본 구현은 메소드의 이름만을 구함)

# AspectJ Pointcut Expression

- POJO 클래스를 이용하여 AOP를 적용하는 두 가지 방법
  - XML 스키마를 이용하여 Aspect를 설정하는 방법
  - @Aspect 어노테이션을 이용하여 Aspect를 설정하는 방법
- 두 방법의 공통점
  - AspectJ의 문법을 이용하여 Pointcut을 설정한다.
  - <aop:태그>를 이용하여 Aspect를 설정하는 경우 execution 명시자를 이용하여 Advice가 적용될 Pointcut을 설정한다.
- 사용 예시

```
<aop:aspect id="cacheAspect" ref="cache">
    <aop:around method="read"
        pointcut='execution(public * kame.spring.chap03.core.*.readArticle(..))' />
</aop:aspect>
```

# AspectJ Pointcut Expression

- AspectJ는 Pointcut을 명시할 수 있는 다양한 명시자를 제공한다.
- 스프링은 메소드 호출과 관련된 명시자만을 지원한다.
- execution 명시자
  - Advice를 적용할 메소드를 명시할 때 사용
  - 기본 형식
    - execution(수식어패턴? 리턴타입패턴 패키지패턴?이름패턴 (파라미터패턴))
    - 수식어 패턴 : 생략가능한 부분(public,protected 등이 옴)
    - 리턴타입패턴 : 리턴 타입을 명시
    - 클래스이름패턴 ,이름패턴 : 클래스 이름 및 메소드 이름을 패턴으로 명시
    - 파라미터패턴 : 매칭될 파라미터에 대해서 명시
  - 특징
    - 각 패턴은 '\*'를 이용하여 모든 값을 표현
    - '..'을 이용하여 0개 이상이라는 의미를 표현

# AspectJ Pointcut Expression 예시

사용 예시	설명
execution(public void set*(..))	리턴 타입이 void이고 메소드 이름이 set으로 시작하고, 파라미터가 0개 이상인 메소드 호출.
execution(* mypkg.core.*.*())	mypkg.core 패키지의 파라미터가 없는 모든 메소드 호출.
execution(* .mypkg.core..*.*(..))	mypkg.core 패키지, 하위 패키지에 있는 파라미터가 0개 이상인 메소드 호출.
execution(Integer mypkg.WriteArticleService.write(..))	리턴 타입이 Integer인 WriteArticleService 인터페이스의 write() 메소드 호출.
execution(* get*(*))	이름이 get으로 시작하고 1개의 파라미터를 갖는 메소드 호출.
execution(* get*(*, *))	이름이 get으로 시작하고 2개의 파라미터를 갖는 메소드 호출.
execution(* read*(Integer, ..))	메소드 이름이 read로 시작하고, 첫 번째 파라미터 타입이 Integer이며, 1개 이상의 파라미터를 갖는 메소드 호출.

# AspectJ의 Pointcut 표현식 정리

- execution([접근자제어패턴], 리턴타입패턴 [패키지패턴]메서드이름패턴(파라메터패턴)) [ ] 안의 패턴은 생략 가능
- execution(public void set\*(..))
  - public에 리턴값이 없으면, 패키지명은 없고, 메서드는 set으로 시작하며 인자값은 0개 이상인 메서드 호출
- execution(\* com.people.\*.\*())
  - 리턴타입에 상관없이 com.people 패키지의 인자값이 없는 모든 메서드 호출
- execution(\* com.people..\*.\*(..))
  - 리턴타입에 상관없이 com.people 패키지 및 하위 패키지에 있는, 인자값이 0개 이상인 메서드 호출
- execution(Integer com.people.WriteService.write(..))
  - 리턴 타입이 Integer인 WriteService의 인자값이 0개 이상인 write() 호출
- execution(\* get\*(\*))
  - 메서드 이름이 get으로 시작하는 인자값이 1개인 메서드 호출
- execution(\* get\*(\*, \*))
  - 메서드 이름이 get으로 시작하는 인자값이 2개인 메서드 호출
- execution(\* get\*(Integer, ..))
  - 메서드 이름이 get으로 시작하고 첫번째 인자값의 데이터타입이 Integer이며, 1개 이상의 인자값을 갖는 메서드 호출
- execution(\* com..\*(..)) && @annotation(@annotation)
  - @annotation이 있는 모든 메소드 호출
- execution(\* \*(..,@annotation (\*), ..))
  - @annotation을 파라메터로 갖고 있는 모든 메소드 호출

# Spring Framework

## MVC(Model View Controller)

# 스프링 MVC 개요

- 스프링 트랜잭션 처리, DI 및 AOP 적용을 손쉽게 사용할 수 있다.
- 각자 역할이 분명한 컴포넌트로 분리하여 담당한다.
  - Controller, Validation Checker, Command 객체, Form 객체, Model 객체
  - DispatcherServlet, HandlerMapping, ViewResolver 등등
- 요청 파라미터로부터 모델 객체 값을 자동으로 채워주는 기능이 있다.
- Controller를 사용하여 클라이언트의 요청을 처리한다.
- Struts와 상당 부분 유사하나 Controller 부분에서 많은 기능을 제공한다.
- Spring 프레임워크에서 기타 다른 프레임워크로 쉽게 대체가 가능하다.
  - MVC : Struts, WebWork, JSF
  - View : jsp, jstl, velocity, excel, pdf

# Spring MVC vs Struts

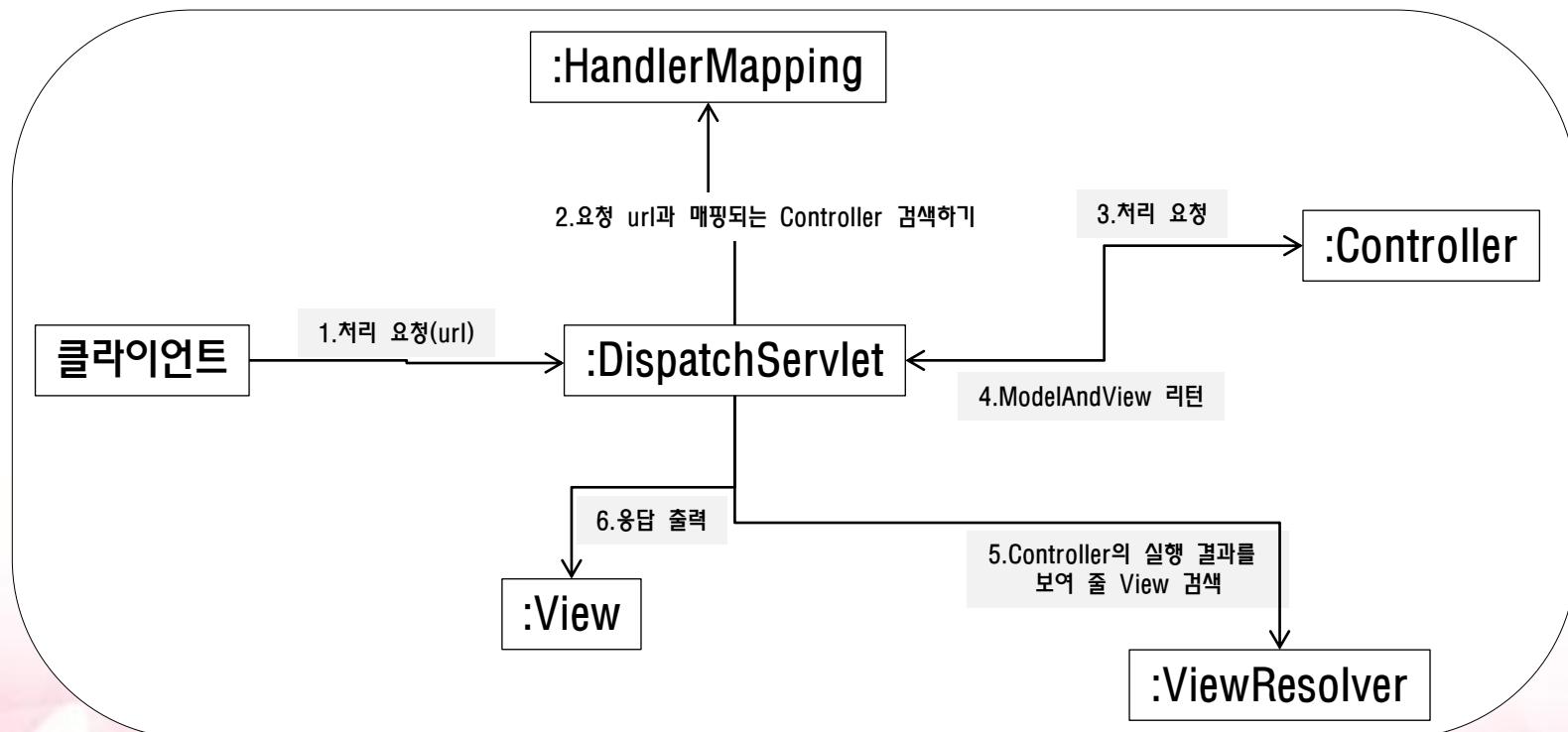
구분	Struts	Spring
동작 패턴	Front Controller	Front Controller
Controller 서블릿	ActionServlet	DispatcherServlet
URL 처리	forward,type 형태로 지원	Controller 클래스로 지원
비즈니스 로직 처리	Action 클래스 상속	Controller 클래스 상속
리턴 형태	forward 객체 (URL)	ModelAndView 객체

# MVC 구성 요소

구성 요소	설명
1) DispatcherServlet (스프링이 기본 제공)	클라이언트의 웹브라우저에서 요청을 전송하게 되면, 이 요청을 받아 들이는 서블릿이 DispatcherServlet이다. 즉, 최초의 클라이언트의 요청을 전달 받는 최초 접수자(서블릿)이다.
2) HandlerMapping (스프링이 기본 제공)	웹 요청 url과 컨트롤러를 맵핑한 정보 관리 기능을 담당한다. 요청된 URL을 분석하여 어떤 컨트롤러가 해당 URL에 대한 처리를 담당할지 의뢰한다. HandlerMapping은 정의된 맵핑 파일을 분석하여 해당 컨트롤러가 처리할 것임을 알려 준다.
3) Controller (개발자가 직접 구현)	담당할 컨트롤러에서 비즈니스 로직에 대한 부분의 처리를 의뢰한다. 이때 처리를 하는 컴포넌트는 Controller 객체가 아니라 스프링에서 제공하는 Controller 계열 클래스 중 하나를 고른다. 적당한 Controller 클래스를 고른 다음 해당 클래스를 상속 받아서 비즈니스 로직을 처리를 한다.
4) ModelAndView	Model(전달할 어떤 내용/값)/ View(전달될 곳)을 정의하고 있는 객체이다. Controller 클래스에서 비즈니스 로직 처리 후에 ModelAndView 객체 (Model + View)를 리턴 한다.
5) ViewResolver (스프링이 기본 제공)	Model 과 View 가 결정이 되었다면 화면에 어떤 식으로 표시 (Jsp,Velocity) 할 지에 대하여 의뢰를 한다.
6) View	결정되어진 표시할 방법으로 요청을 의뢰한 클라이언트의 브라우저로 응답이 전송되어 진다.

# MVC를 이용한 웹 요청 처리

구성 요소	설명
개발자가 직접 개발해야 할 부분	클라이언트의 요청을 처리할 컨트롤러 클라이언트에 응답결과 화면을 전송할 JSP 파일이나 Velocity 템플릿 등의 뷰 코드
스프링이 기본적으로 제공하는 구현 클래스	DispatcherServlet / HandlerMapping /ViewResolver



# MVC 개발 순서

- 일반적인 MVC의 개발 순서는 다음과 같다
  - DispatcherServlet 서블릿을 web.xml 파일에 설정한다.
  - HandlerMapping을 사용하여 요청을 처리해줄 컨트롤러 객체를 구한다.
  - 컨트롤러를 작성한다. (클라이언트 요청 처리)
  - 컨트롤러는 ModelAndView 객체를 리턴한다.
  - ViewResolver를 설정한다.
  - 응답 결과를 보여줄 View 객체를 구한다.

# web.xml(배포 서술자)

- 배포를 위한 서술자 파일로써 DD(Deployment Descriptor)라고 불리운다.
- web Application의 환경 설정을 위한 XML 형식의 파일이다.
- web.xml에 작성되는 내용은 다음과 같은 것들이 있다.
  - ServletContext의 초기 파라미터 정보
  - Session의 유효 시간 설정
  - Servlet/JSP에 대한 정의
  - Servlet/JSP 매팅
  - Mime Type 매팅
  - Welcome File List
  - Error Pages 처리
  - 리스너 / 필터 설정
  - 보안 등등

# DispatcherServlet

- 최초의 클라이언트의 요청을 전달 받는 서블릿이다.
- Controller나 View와 같은 스프링 MVC의 구성 요소를 이용하여 클라이언트에게 서비스를 제공해준다.
- /WEB-INF/web.xml 파일에 서블릿과 서블릿 매팅 정보를 추가한다.
- DispatcherServlet 정의 예시

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app...>
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
      org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.htm</url-pattern>
  </servlet-mapping>
</web-app>
```

Servlet 이름이다.

컨텍스트 파일은 [서블릿명 + -servlet.xml]이므로 dispatcher-servlet.xml이 된다  
스프링 MVC의 설정용 파일이다

확장자가 htm인 요청에 대해서만 처리가 가능하다.

# 캐릭터 인코딩 처리를 위한 필터 설정

항목	설명
Encoding(인코딩)	비(非) 아스키 문자열들을 특수한 문자열로 변환하는 것
서블릿 필터(filter)	요청이나 응답 처리 전에 사전에 가로 채서 정보를 변환하거나 이용하기 위한 용도
jsp에서 캐릭터 인코딩 처리 방식	요청 파라미터의 캐릭터 인코딩이 ISO-8859-1이 아닌 경우 response.setCharacterEncoding("UTF-8");
스프링에서 캐릭터 인코딩 필터 설정하기	요청 파라미터의 캐릭터 인코딩을 설정할 수 있는 필터 클래스인 CharacterEncodingFilter 클래스를 제공한다.

```
<filter>
    <filter-name>encodingFilter</filter-name>
    <filter-class>
        org.springframework.web.filter.CharacterEncodingFilter
    </filter-class>
    <init-param>
        <param-name>encoding</param-name>
        <param-value>EUC-KR</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*<url-pattern>
</filter-mapping>
```

모든 요청에 대한 캐릭터 인코딩 처리

한글 폼 필드의 값이나 한글 파일명이 서버에서 깨지지 않고  
제대로 업-로드될 수 있도록 필터를 설정해 준다.

# ApplicationContext(스프링) 설정 파일

- web.xml 파일에 스프링 설정 파일 위치에 대한 정보를 작성한다.

항목	설명
<context-param> 태그	컨텍스트의 위치를 표시하기 위한 컨텍스트 인스턴스이다. 전달되는 문자열이 여러 개인 경우 콤마를 분리자로 하여 정의할 수 있다. contextConfigLocation 파라미터를 사용한다.
ContextLoader Listener	스프링의 Web-ApplicationContext를 시작시키기 위한 Bootstrap listener이다. ServletContext 인스턴스 생성(애플리케이션 최초 시작)시 호출이 된다. <listener> 태그에 명시한다. 디플트 설정 파일 이름은 "/WEB-INF/applicationcontext.xml"이다. <context-param> 태그 내의 <param-name> 태그를 사용하게 되면 변경이 가능하다.

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/applicationContext.xml</param-value>
</context-param>

<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
```

스프링 관련 설정 파일의 위치와 이름을 명시.  
contextConfigLocation을 사용하여 설정 파일 목록을 다른 것으로 지정하고자 하는 경우에 사용

## 작업 순서

1) 리스너를 등록한다.

2) contextConfigLocation 파라미터를 이용하여 공유할 설정 파일 목록을 명시한다.

# DispatcherServlet과 ApplicationContext 설정 파일

```
<!-- 요청된 url 이름으로 동일한 이름의 컨트롤러를 호출하도록 설정 -->
<bean id="handlerMapping"
      class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<!-- 컨트롤러 설정하기. -->
<!-- 데이터 인서트 -->
<bean name="/insert.do">
    파일 이름 : [서블릿명-servlet.xml]
    파일 위치 : web.xml 파일과 동일한 폴더
    파일 용도 : MVC 용 설정 파일
    <!-- MemDAO 객체 주입에 대한 설정 -->
    <property name="dao" ref="dao"/>
    <!-- 파라미터로 넘어오는 값들을 저장한 DTO빈의 클래스 -->
    <property name="commandClass" value="myPkg.bean.Command"/>
</bean>
<!-- 리스트 보기 -->
<!-- 항목 삭제하기 -->
<!-- 항목 수정하기 -->
<!-- 수정할 멤버 찾기 -->
<!-- 페이징 처리하기 -->
...
... 이하 중략

<!-- ViewResolver 설정 -->
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceView">
    <property name="prefix" value="/jsp/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

파일 이름 : [서블릿명-servlet.xml]

파일 위치 : web.xml 파일과 동일한 폴더

파일 용도 : MVC 용 설정 파일

다음과 같은 경우에는 contextConfigLocation 초기화 파라미터를 이용하여 설정 정보를 변경할 수 있다.

- \* 한 개 이상의 설정 파일을 사용하고자 하는 경우
- \* 기본 설정 파일 이름이 아닌 다른 이름을 설정 파일을 사용하고자 하는 경우

```
<!-- 선수 관리 서블릿 -->
<servlet>
    <servlet-name>baseBallPro</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <!-- 서블릿 설정 파일들을 알려 주는 셋팅 -->
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/servlet-config/baseBallPro-servlet.xml
        </param-value>
    </init-param>
</servlet>
```

# HandlerMapping

- 클라이언트의 요청과 컨트롤러(업무 처리 영역)의 맵핑을 담당하는 인터페이스이다.
- HandlerMapping 인터페이스를 구현한 AbstractHandlerMapping 클래스를 확장한 여러 개의 HandlerMapping 클래스가 존재한다.
- DispatcherServlet이 사용하는 스프링 설정 파일에 사용할 HandlerMapping을 Bean 객체로 등록하면 된다.
- 스프링 설정 파일에 등록하지 않게 되면 기본 값으로 BeanNameUrlHandlerMapping 클래스를 사용한다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       ...
       http://www.springframework.org/schema/context/spring-context-2.5.xsd">
    <bean id="beanNameUrlMapping"
          class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />
    <bean name="/hello.htm" class="mvcPackage.HelloController" />
    ...
</beans>
```

# 설정이 필요한 주요 Property(속성)

용어	항목에 대한 설명
Interceptors	요청을 가로채어 선처리 같은 작업을 담당할 인터셉터들의 목록, 서블릿의 필터와 유사하게 동작
defaultHandler	핸들러 맵핑이 적합한 핸들러를 찾지 못할 경우 사용하기 위한 디폴트 핸들러
order	org.springframework.core.Ordered 인터페이스의 속성을 구현 사용 가능한 핸들러 맵핑이 다수일 경우 순서를 결정하기 위한 인덱스 값 (정수)
alwaysUseFullPath	적당한 핸들러를 찾기 위해 현재 서블릿 컨텍스트 내 완전한 경로 (full path)를 사용할 것인지의 여부 false(디폴트 값) : 현재 서블릿 맵핑내 경로가 사용 true : 서블릿 컨텍스트내 완전한 경로를 사용
urlPathHelper	URL을 조사할 때 사용되는 헬퍼 클래스, 부분적으로 수정이 가능하지만 디폴트 값 사용 권장
urlDecode	HttpServletRequest는 Decode 되지 않은 요청(request)을 반환하는데 이 요청(request)의 URI가 Decode 되어야 할지의 여부를 결정 기본값은 false, JDK 1.4 이상에서 사용 가능 Decode 방식은 request에 명시된 인코딩이나 디폴트인 ISO-8859-1 인코딩 스키마를 사용
lazyInitHandlers	Prototype(Non Singleton) 핸들러는 언제나 늦은 초기화 (Lazy Loading) 방식으로 동작하는 반면 singleton 핸들러임에도 불구하고 Lazy Loading을 하도록 허락한다. 디폴트 값은 false이다.

# BeanNameUrlHandlerMapping

용어	항목에 대한 설명
정의	URL과 매칭되는 이름을 갖는 빈을 컨트롤러로 사용하는 HandlerMapping이다.
특징	<p>bean의 이름을 이용하여 맵핑을 수행한다. AbstractUrlHandlerMapping 클래스를 상속 받고 있다. alwaysUseFullPath 프로퍼티의 값을 true로 지정하면 전체 경로를 이용하여 매핑되는 컨트롤러 검색이 가능하다.</p> <p>사용자가 HandlerMapping을 정의하지 않으면 기본 핸들러 맵핑으로 사용된다. 하지만, 어떤 핸들러 맵핑이 사용되는지를 명확히 명시하기 위해 선언을 해 두는 것이 좋다 다수의 핸들러 맵핑을 사용할 경우 순서(order)를 명시하기 위해 설정을 권장한다. 컨트롤러의 맵핑이 많을 경우에는 다른 맵핑 방식을 권장한다.</p>

```
<!-- 요청된 url 이름으로 동일한 이름의 컨트롤러를 호출하도록 설정 -->
<bean id="handlerMapping"
      class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>

<!-- 리스트 보기 -->
<bean name="/list.do" class="myPkg.controller.ListController">
    <property name="dao" ref="dao"/>
</bean>
```

name 속성으로 지정된 값을 그대로 경로 값으로 처리하겠다.

http://localhost:8989//shoppingMall/list.do

사용자가 list.do를 요청하게 되면  
ListController 컨트롤러가 수행된다.

이미 만들어 놓은 dao 객체를  
setter injection을 수행한다.

# SimpleUrlHandlerMapping

- 컨트롤러와 URL의 맵핑을 설정 파일 위에 합쳐서 기술하는 방법이다.
- 요청하는 컨트롤러 파일의 이름들을 한 장소에서 관리해주는 HandlerMapping이다.

```
<bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <props>
            <prop key="/index.html">indexController</prop>
            <prop key="/detail.html">detailController</prop>
        </props>
    </property>
</bean>
<bean id="indexController" class="controller.IndexController">
    <property name="shopService"><ref bean="shopService" /></property>
</bean>

<bean id="detailController" class="controller.DetailController">
    <property name="shopService"><ref bean="shopService" /></property>
    <property name="cacheSeconds"><value>30</value></property>
</bean>
```

Properties 형의 mappings 프로퍼티를 이용한다.

각 요청에 따른 다른 컨트롤러를 수행한다.

# 다수의 HandlerMapping 사용하기

- 모든 핸들러 맵핑 클래스는 `org.springframework.core.Ordered` 인터페이스를 구현하고 있다.
- 애플리케이션 내에서 여러 개의 핸들러 맵핑을 선언할 수 있으며, `order` 특성을 설정하면 맵핑의 우선 순위(`order` 값이 작을수록 우선 순위가 높음)를 지정이 가능하다.

```
<bean id="beanNameUrlHandlerMapping"
      class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
    <property name="order" value="1"/>
</bean>
<bean id="simpleUrlHandlerMapping"
      class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="order" value="0"/>
    <property name="mappings">
      ...
    </property>
</bean>
```

SimpleUrlHandlerMapping을 우선 순위로 하고 발견하지 못했을 경우 BeanNameUrlHandlerMapping으로 처리하는 예제이다.

# Controller 인터페이스

- MVC 디자인 패턴의 일부 (MVC의 [C]부분에 해당)
- Model과 View의 교량 역할 (비즈니스 규칙을 수행/검증)
- 일반적인 구현 방법
  - 상속을 이용하여 스프링이 제공하는 몇 가지 기본 컨트롤러 클래스 중에서 알맞은 클래스를 상속 받아 구현한다.
  - 단순 컨트롤러, Form 컨트롤러, Command 기반 컨트롤러,
  - Wizard 기반 컨트롤러 등으로 구현이 가능하다.
- 처리된 결과
  - 사용자 입력을 해석하고 사용자 입력을 view에 의해 모델로 변형된다.
  - ModelAndView에 담아 DispatcherServlet에 전달해 준다.
- Controller 인터페이스의 정의

```
public interface Controller {  
    ModelAndView handleRequest(HttpServletRequest request,  
        HttpServletResponse response) throws Exception;  
  
    //어떠한 작업 ...  
    return null ;  
}
```

# Controller 인터페이스 사용 예시

- 처리된 결과

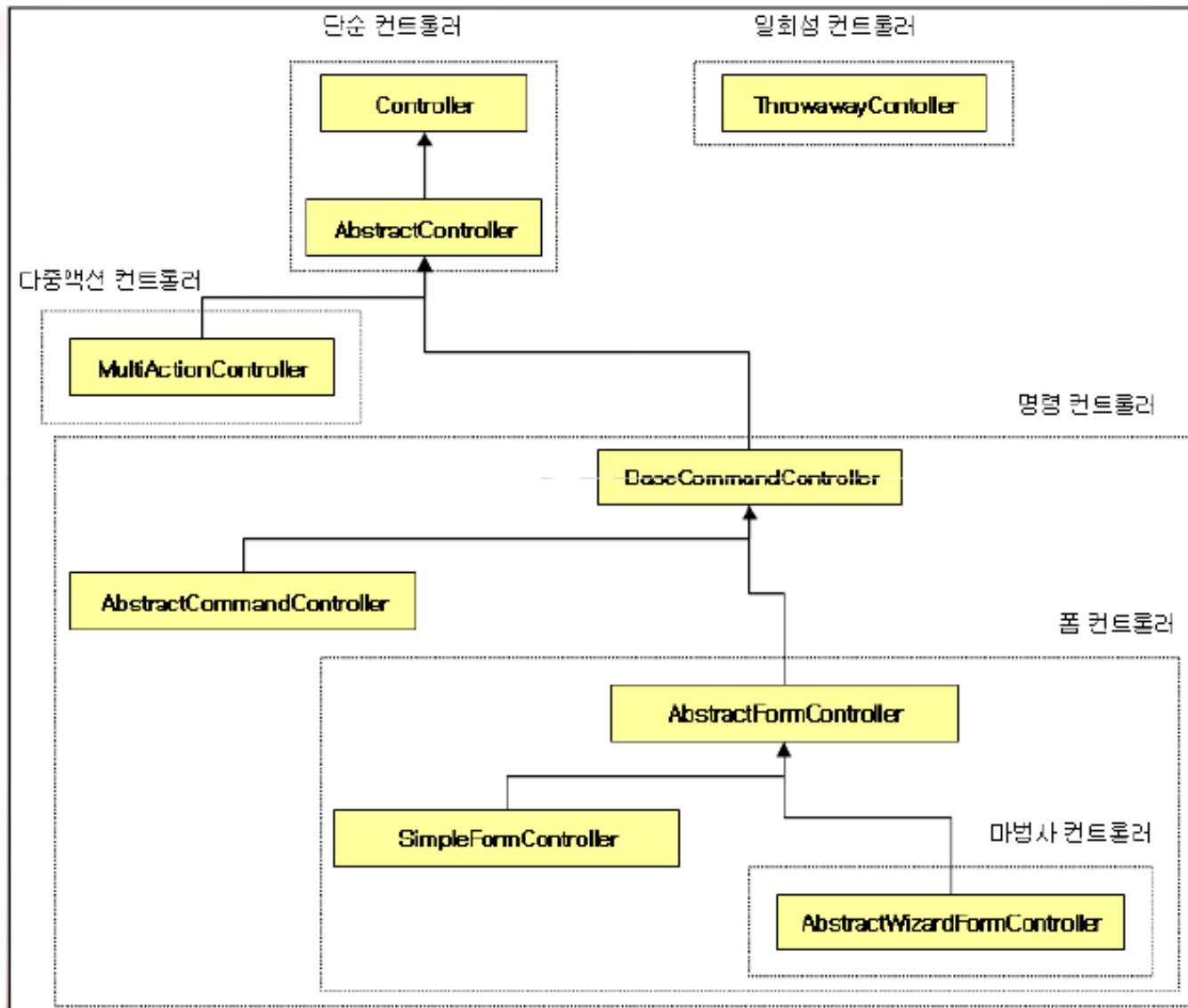
- 사용자 입력을 해석하고 사용자 입력을 view에 의해 모델로 변형된다.
- ModelAndView를 담아 DispatcherServlet에 전달해 준다.

```
//모든 회원들을 조회하여 보여준다.  
public class ListController implements Controller{  
    ... 이하 중략  
  
    @Override  
    public ModelAndView handleRequest(HttpServletRequest arg0,  
        HttpServletResponse arg1) throws Exception {  
        //비즈니스 로직 수행하기  
        List<MemberBean> memberList = dao.getMemberList();  
  
        //결과 값과 이동할 뷰 이름을 ModelAndView 객체에 담기  
        ModelAndView mv=new ModelAndView();  
        mv.setViewName("list");//뷰이름  
        mv.addObject("memberList", memberList);  
  
        return mv;  
    }  
}
```

Model : 저장될 어떠한 데이터  
View : 이동할 곳(장소)

memberList라는 객체를 mv 영역에 저장시키고, list.jsp라는 파일로 제어를 이동시킨다

# Controller 인터페이스의 계층도



# 컨트롤러의 종류

컨트롤러	클래스(인터페이스)	설명
단순 컨트롤러	Controller AbstractController	가장 간단한 형태로 별도 기능을 제공하지 않는 컨트롤러 파라미터를 받지 않거나 적은 수의 파라미터를 받아서 처리를 할 때 필요 요청 파라미터 처리 등의 작업을 직접 구현해 주여야 한다.
일회성 컨트롤러	ThrowawayController	요청을 명령으로서 처리하는 단순한 방법을 사용하고자 할 때 사용한다. WebWork의 Action과 비슷하다.
다중 액션 컨트롤러	MultiActionController	연관성이 있거나, 비슷한 로직을 수행하는 다수의 액션을 하나의 컨트롤러에서 구현하고 자 하는 경우에 사용된다.
명령 (Command) 컨트롤러	AbstractCommandController	요청으로부터 하나 이상의 파라미터를 받아서 다른 객체 (Command 객체)로 바인딩 시킬 때 사용한다. 파라미터의 유효성 검증 기능을 사용할 수 있다.
폼 컨트롤러	-	입력 폼을 사용자에게 보여주며 입력된 데이터를 처리해야 하는 경우에 사용한다.
마법사 컨트롤러	-	사용자로 하여금 다수의 페이지에 걸친 복잡한 입력 폼들을 거치도록 해야 할 경우에 사용한다. 최종적으로는 하나의 폼으로서 처리된다.
입력 폼 처리	SimpleFormController	폼을 출력하고 폼에 입력한 데이터를 처리할 때 사용한다.
정적 뷰 매핑	ParameterizableViewController UrlFilenameViewContr oller	컨트롤러에서 어떤 기능도 수행하지 않고, 단순히 클라이언트의 요청을 뷰로 전달할 때 사용한다.

# AbstractController

- Controller 인터페이스를 구현해 놓은 가장 간단한 Controller 클래스이다.
- 단순히 클라이언트의 요청을 처리한 뒤에 ModelAndView를 리턴해준다
- 요청으로부터 파라미터가 필요하지 않는 경우에 사용 가능한 Controller이다.
- 상속 받는 컨트롤러 클래스는 handleRequestInternal() 주상 메소드를 구현해야 한다.

```
package samples;

public class SomeController extends AbstractController {
    protected ModelAndView handleRequestInternal(
        HttpServletRequest request,
        HttpServletResponse response) throws Exception {

        // 필요한 파라미터 값 구함.
        String requiredValue = request.getParameter("someParam");
        ...

        // 클라이언트의 요청을 처리하는데 필요한 로직 실행.
        processSomeLogic();

        // 뷰 선택 및 뷰 생성시 필요한 값 입력.
        ModelAndView mav = new ModelAndView();
        mav.setViewName("뷰이름");
        mav.addObject(...);

        return mav;
    }
}
```

# AbstractController에 의해 제공되는 특징

- DI(Dependency Injection)을 사용하여 프로퍼티로 설정이 가능하다.

용어	항목에 대한 설명
supportedMethods	이 컨트롤러가 지원하는 Http 메소드 목록을 콤마로 구분하여 입력한다. 언제나 GET과 POST로 셋팅이 되지만 만약에 원하는 방식을 바꾸고자 할 경우 변경이 가능하다.
requiresSession	컨트롤러가 작업을 하기 위해 HTTP 세션을 필요로 하는지 하지 않는지를 표시한다. 값을 true으로 지정할 경우 HttpSession이 존재하지 않으면 ServletException을 발생 시킨다.
synchronizeSession	HTTP Session에서 동기화되는 컨트롤러에 의해 핸들링하기 원할 경우 사용한다.
cacheSeconds	응답에 포함될 캐시 헤더의 값을 초 단위로 입력한다. HTTP Response에서 cache를 생성할 컨트롤러를 원할 경우 양수 값을 명시한다. 디폴트 : -1
useExpiresHeader	HTTP 1.0 호환 "Expires" 헤더에 대한 처리를 한다면 변경을 해야 한다. 디폴트에 의해 이 프로퍼티의 값은 true이다.
useCacheHeader	HTTP 1.1 호환 "Cache-Control" 헤더에 대한 처리를 한다면 변경을 해야 한다. 디폴트에 의해 이 프로퍼티 값은 true 이다.
사용 예시	다음의 예시는 캐시를 수행하지 않도록 응답 헤더를 설정하고 HTTP GET 메소드만을 지원하도록 컨트롤러를 설정한다. <pre>&lt;bean name="/hellohtm"       class="kamespringchap04controllerHelloController"       p:supportedMethods="GET" p:cacheSeconds="0" /&gt; &lt;/bean&gt;</pre>

# MultiActionController

- 여러 개의 리퀘스트를 처리할 수 있는 특별한 컨트롤러이다.
- 복수 개의 리퀘스트에 대하여 각각의 처리를 하나의 클래스에 기술한 다음 리퀘스트별로 할당할 수 있다.
- 스프링 설정 파일에 리퀘스트 Url과 메소드의 관련성을 명시해야 한다.
- 메소드 이름 결정 방식

항목	설명
InternalPathMethodNameResolver	URL 패턴에 기초하여 메소드 이름을 결정한다.
ParameterMethodNameResolver	요청 파라미터를 기초로 하여 실행할 메소드 이름을 결정한다.
PropertiesMethodNameResolver	프로퍼티파일 내용 중에 키와 값(쌍의 목록)을 기초로 하여 실행

```
<!-- MethodNameResolver -->
<bean id="methodNameResolver"
      class="org.springframework.web.servlet.mvc.mutiaction.PropertiesMethodNameResolver">
    <property name="mappings">
      <props>
        <prop key="/cartAdd.html">add</prop>
        <prop key="/cartClear.html">clear</prop>
        <prop key="/cartConfirm.html">confirm</prop>
      </props>
    </property>
</bean>
```

# MultiActionController 예시

```
<!-- MethodNameResolver -->
<bean id="methodNameResolver"
class="org.springframework.web.servlet.mvc.mutiaction.PropertiesMethodNameResolver">
    <property name="mappings">
        <props>
            <prop key="/cartAdd.html">add</prop>
            <prop key="/cartClear.html">clear</prop>
            <prop key="/cartConfirm.html">confirm</prop>
        </props>
    </property>
</bean>
```

스프링 설정 파일

```
<bean id="cartController" class="controller.CartController">
    <property name="methodNameResolver">
        <ref bean="methodNameResolver" />
    </property>
    <property name="shopService">
        <ref bean="shopService" />
    </property>
</bean>
```

```
public class CartController extends MultiActionController{
    public ModelAndView add(HttpServletRequest request, HttpServletResponse response) throws Exception {
        ...
        return modelAndView;
    }

    public ModelAndView clear(HttpServletRequest request, HttpServletResponse response) throws Exception {
        ...
        return modelAndView;
    }

    public ModelAndView confirm(HttpServletRequest request, HttpServletResponse response) throws Exception {
        ...
        return modelAndView;
    }
}
```

CartController.java

MultiActionController를 상속 받는 CartController 클래스는  
methodNameResolver을 사용하는데, 요청 가능한 페이지는 종3개이다.  
예를 들어서 /cartConfirm.html을 요청하게 되면 confirm() 메소드가 수행된다.

이때 리퀘스트 url에 관련된 메소드는 시그너쳐를 동일하게 해야 한다.

# AbstractCommandController

- 특정 form에서 넘어온 파라미터들을 특정한 객체(Bean 객체)에 담고 싶을 때 사용한다.
  - request.getParameter("파라미터이름") 형태로 받지 않는다.
  - command 객체(Http 요청 정보를 저장하기 위한 객체)를 사용한다.
- 프로퍼티를 설정하는 방법
  - 지정한 커맨드 클래스의 객체를 생성한다.
  - 커맨드 객체의 프로퍼티의 이름과 일치하는 이름을 갖는 파라미터 값으로 한다.
- Bean 설정
  - 설정 파일에서 commandClass 프로퍼티와 commandName 프로퍼티에 값을 입력한다.
  - 설정 파일 대신 자바에서 메소드를 사용하는 경우에는 setCommandClass(), setCommandName()를 사용하면 된다.
  - commandName 프로퍼티의 값을 설정하지 않으면 기본 값은 "command"이다.

```
<bean id="listController" class="myPkgcontrollerListController">
    <!--commandName : 한 세트의 파라미터들을 처리해 줄 객체 이름-->
    <property name="commandName" value="loginCommand"/>

    <!--commandClass : 객체를 만들어 낼 클래스 이름 지정 -->
    <property name="commandClass" value="testLoginCommand"/>
</bean>
```

# AbstractCommandController

- 구현 클래스의 특징
  - handle() 추상 메소드를 오버라이딩해야 한다.
    - protected ModelAndView handle(HttpServletRequest request, HttpServletResponse response, Object command, BindException errors) throws Exception;
    - BindException : 검증된 결과가 여기에 전달이 된다.
    - 전달된 command 파라미터는 setCommandClass() 메소드를 통해서 지정한 커맨드 클래스의 인스턴스가 전달된다.
- 구현 방법
  - setCommandClass(Class commandClass) 메소드
    - 커맨드 클래스를 지정한다.
  - setCommandName(String commandName) 메소드
    - 커맨드 객체의 이름을 지정한다.
    - 생성된 커맨드 객체를 request.setAttribute()를 이용하여 저장할 때 사용할 속성 이름
    - 커맨드 객체의 이름을 ModelAndView에 저장한다.

# AbstractCommandController 예시

http://localhost:8089/mySpringJdbc/jsp/Insert.jsp

회원 등록하기

이름 풍길동  
주소 서울 마포 흥대

회원 등록

```
<center>
  <h1>회원 등록하기</h1>
  <form method="post" action="<% = myContext %>/insert.do">
    이름 <input type="text" name="name"/><br/>
    주소 <input type="text" name="addr" /><br/><br/>
    <input type="submit" value="회원 등록"/>
  </form>
</center>
```

xxx.jsp 파일의 일부 코드

커맨드 객체 : 품 내부의 파라미터들에 대한 정보를 저장하고 있는 객체.

이 예제에서는 name, addr의 커맨드 객체에 저장된다.

```
<bean name="/insert.do" class="myPkg.controller.InsertController"
  <!-- MemDAO 객체 주입에 대한 설정 -->
  <property name="dao" ref="dao"/>

  <!-- 파라미터로 넘어오는 값들을 커맨드 이름 설정 -->
  <property name="commandName" value="mycommand"/>

  <!-- 파라미터로 넘어오는 값들을 저장한 MemberBean의 클래스 타입 설정 -->
  <property name="commandClass" value="myPkg.bean.MemberBean"/>
</bean>
```

```
package myPkg.bean;

public class MemberBean {
  // 회원을 다루기 위한 Bean 클래스(mem 테이블 생성)
  private int num;
  private String name;
  private String addr;
```

등록 자바 코드

```
MemberBean mycommand = new MemberBean();
mycommand.setName("풍길동");
mycommand.setPwd("1234");
```

# AbstractCommandController 예시

```
//회원 1명에 대한 자료를 인서트하고, 목록 보기로 이동한다.  
public class InsertController extends AbstractCommandController{  
    ... 중략  
  
    @Override  
    protected ModelAndView handle(HttpServletRequest request,  
        HttpServletResponse response, Object obj, BindException error)  
        throws Exception {  
  
        MemberBean bean = (MemberBean) obj; //obj를 원래의 형태로 형변환 한 후  
        dao.insert( bean ); //db 저장하기.  
  
        //저장하고 난후 이동할 페이지에 대한 정보 설정  
        ModelAndView mv = new ModelAndView();  
        mv.setViewName("redirect:/list.do");  
        return mv;  
    }  
}
```

Handle 메소드의 3번째 매개 변수에 command 객체의 정보가 들어온다.

```
public class MemberDAO {  
    ... 중략  
  
    public void insert(MemberBean bean){ //회원 정보 저장하기.  
        Object obj[] = { bean.getName(), bean.getAddr()};  
        String sql = "insert into mem values( seqmem.nextval, ?, ? )";  
        jdbcTemplate.update( sql, obj );  
    }  
  
    2개의 파라미터가 입력되고 있다.
```

# SimpleFormController

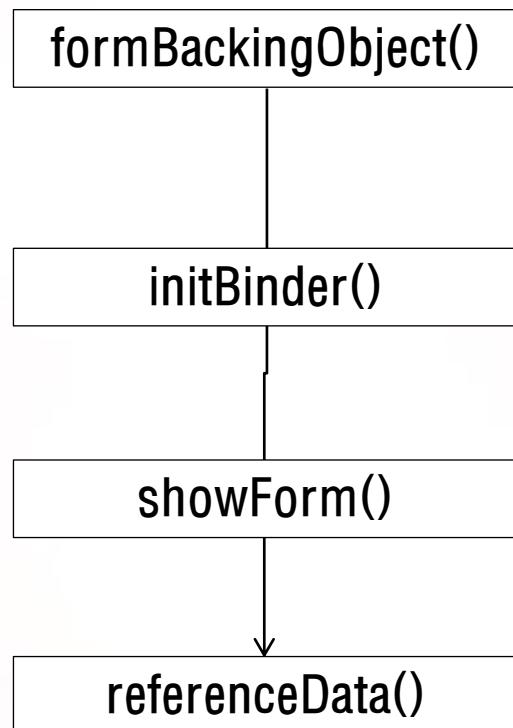
- form 데이터의 표시/입력/수정 처리가 필요한 경우에 사용된다.
- BaseCommandController를 상속 받고 있으므로, 유효성 검증이 가능하고, command 클래스를 사용할 수 있다.
- 파라미터 방식(GET/POST)에 대한 처리가 가능하다.
- form 요소와의 데이터 바인딩이 가능하다.(예 : 최초 수행시 콤보 박스에 목록 넣기)

프로퍼티	설명
sessionForm	세션에 command를 유지할 것인지를 결정(true/false)
commandName	리퀘스트에 바인드하는 command 이름
commandClass	Command 클래스의 완전 수식명(fullPath name)
validator	Command를 검증하는 Validator 클래스의 참조
formView	화면 표시 때, 또는 입력 정보 검증 에러가 발생한 경우에 이동할 View 페이지
successView	입력 정보 송신이 정상적으로 종료(폼 검증 성공)한 경우 이동 장소가 되는 View 페이지

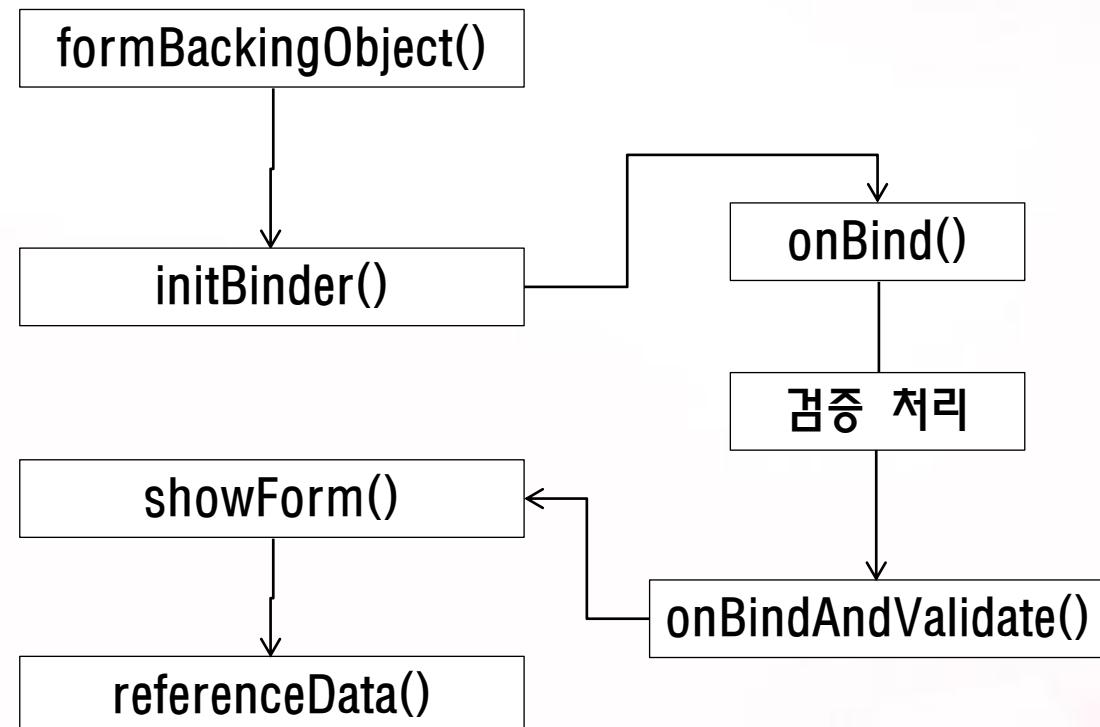
```
<bean id="loginFormController" class="controller.LoginFormController">
    <property name="sessionForm"><value>false</value></property>
    <property name="commandName"><value>user</value></property>
    <property name="commandClass"><value>logic.User</value></property>
    <property name="validator"><ref bean="loginValidator"/></property>
    <property name="formView"><value>login</value></property>
    <property name="successView"><value>loginSuccess</value></property>
    <property name="shopService"><ref bean="shopService" /></property>
</bean>
```

# SimpleFormController

- GET 요청에 대한 처리 흐름



- POST 요청에 대한 처리 흐름



# SimpleFormController

메소드 이름	설명
formBackingObject()	<p>Get/post 요청모두에 응답하는 메소드 Get : 폼에서 디폴트로 채워 줘야 하는 필드의 기본 데이터 준비 Post : 커맨드 클래스의 객체를 생성하여 리턴한다. 사용하는 command를 커스터마이즈함 이 메소드를 오버라이딩해서 출력품의 내용을 표시할 때 userId 와 userName 속성에 메시지에서 취득한 디폴트 값을 표시</p> <pre>protected Object formBackingObject(HttpServletRequest request) throws Exception {     //메시지 소스로부터 초기 표시 데이터 취득     MessageSourceAccessor accessor = getMessageSourceAccessor();     User user = new User();     user.setUserId(accessor.getMessage("user.userId.default"));     user.setUserName(accessor.getMessage("user.userName.default"));     return user; }</pre>
initBinder()	<p>Command에 Date형, Number 형, Boolean 형 등의 속성이 존재하는 경우에 필요한 변환처리를 기술한다.</p> <pre>protected void initBinder(HttpServletRequest request,                            ServletRequestDataBinder binder) throws Exception {      //Date형의 birthDay프로퍼티를 커스터마이즈     DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");     dateFormat.setLenient(false);     binder.registerCustomEditor(Date.class, "birthDay",         new CustomDateEditor(dateFormat, false)); }</pre>

# SimpleFormController

메소드 이름	설명
showForm()	Get() 방식의 요청이 들어온 경우 Post 방식에서 데이터 검증에 실패한 경우 개발자가 에러를 추가하는 경우(Errors 인터페이스 사용하여)
referenceData()	폼에서 사용할 컨트롤의 참조 데이터를 Map 객체로 저장 가능하다. 예시 : 콤보 박스에 들어갈 내용 채우기 JSP 파일에서 \${EL}을 사용하여 참조가 가능하다. ShowForm() 메소드가 이 메소드를 호출한다. 아래의 예시는 [직업] 항목을 콤보 박스에 표시하기 위해서 데이터를 모델로 작성한다.

```
protected Map referenceData(HttpServletRequest request) throws Exception {  
    //콤보박스데이터를 작성  
    Map model = new HashMap();  
    String[] jobs = new String[]{"변호사", "의사", "교수", "주부", "학생", "the others"};  
    model.put("jobs", jobs);  
    return model;  
}
```

# SimpleFormController

메소드 이름	설명
onBind()	입력 폼으로부터 송신된 데이터를 command로 바인드한 후에 호출된다. 필요하다면 오버라이드하여 처리 내용을 추가할 수 있다.
검증 처리	입력한 정보에 대한 검증 처리가 실행되면 검증 처리는 Validate() 인터페이스를 구현한 검증 클래스로 전달한다.
onBindAndValidate()	검증 처리를 마친 후에 호출되며 디폴트에서는 아무것도 처리되지 않는다 필요하다면 오버라이드하여 처리내용을 추가할 수 있다
onSubmit() 메소드	Post 방식으로 요청이 들어온 경우에 수행된다. 검증 결과가 정상이라고 판단된 경우에 한해서 호출된다. Command 객체를 저장할 수 있다 Jsp 파일에서 request.getAttribute("커맨드이름"); 또는 \${커맨드이름 프로퍼티이름 }

# @Controller와 @RequestMapping Annotation

- 컨트롤러 클래스를 구현하려면 다음과 같이 수행하면 된다.
- 일반적인 작업절차
  - 컨트롤러 클래스에 @Controller Annotation을 적용한다.
  - 기본 값으로 DefaultAnnotationHandlerMapping을 통해 컨트롤로 사용된다.
  - 클라이언트의 요청을 처리할 메소드에 @RequestMapping Annotation을 적용한다.
  - 설정 파일에 컨트롤러클래스를 Bean으로 등록한다.
- 자동 로딩
  - @Controller 어노테이션을 적용한 클래스는 컴포넌트 스캔 대상이다.
  - <context:component-scan> 태그를 이용하여 해당 클래스를 자동으로 로딩할 수 있다.

```
//가장 간단한 컨트롤러 클래스

@Controller //컨트롤러 클래스로 사용하겠다
public class HelloController {
    @RequestMapping("/hello.do") //이 요청이 들어오면 처리하겠다.
    public String hello() {
        return "hello" ; //이것이 View의 이름이 된다.
    }
}

//스프링 설정 파일에 해당 컨트롤러 클래스를 다음과 같이 등록해 준다.

<bean id="helloController" class="controller.HelloController"/>
```

# @RequestMapping

- 컨트롤러 클래스나 메소드가 특정 HTTP Request URL을 처리한다.

속성 이름	설명
value	"value='/getMovie.do'"와 같은 형식의 맵핑 URL 값이다. 디폴트 속성이기 때문에 value만 정의하는 경우에는 'value='은 생략할 수 있다. @RequestMapping(value={"/addMoviedo","/updateMoviedo" })와 같은 2개의 URL 모두를 처리한다.
method	GET, POST, HEAD 등으로 표현 되는 HTTP Request method에 따라 requestMapping을 할 수 있다. 'method=RequestMethod.GET' 형식으로 사용한다. method 값을 정의하지 않는 경우 모든 HTTP Request method에 대해서 처리하겠다는 의미이다.
params	HTTP Request로 들어오는 파라미터 표현이다. 'params={"param1=a","param2","!myParam"}'로 다양하게 표현 가능하다. @RequestMapping(params = {"param1=a","param2","!myParam"}) 위의 경우 HTTP Request에 param1과 param2 파라미터가 존재해야 하고 param1의 값은 'a'이어야 하며, myParam이라는 파라미터는 존재하지 않아야 한다.

# @RequestMapping

```
// 요청한 인터넷 주소가 다음과 같다면  
http://host/ContextEx/search/internal.do?query=spring&p=3  
  
@Controller  
public class SearchAnnotController {  
  
    @RequestMapping("/search/internal.do")  
    //RequestParam : http 요청 파라미터를 메소드의 파라미터로 전달 받을 때 사용한다.  
    //value : 파라미터 이름, required : 필수 여부, defaultValue : 기본 값을 의미한다.  
    public String searchInternal(  
        @RequestParam(value="query", required=true, defaultValue="some") String query,  
        @RequestParam(value="p", required=false, defaultValue='1') int pageNo,  
        ModelMap modelMap) {  
  
        return search(internalSearchService, query, pageNo, modelMap);  
    }  
}
```

# 컨트롤러 메소드의 파라미터 타입

파라미터 타입	설명
HttpServletRequest, HttpServletResponse, HttpSession	서블릿 API
java.util.Locale	현재 요청에 대한 Locale
InputStream, Reader	요청 컨텐츠에 직접 접근할 때 사용
OutputStream, Writer	응답 컨텐츠를 생성할 때 사용
@PathVariable 어노테이션 적용 파라미터	URI 템플릿 변수에 접근할 때 사용
@RequestParam 어노테이션 적용 파라미터	Http 요청 파라미터를 맵핑
@RequestHeader 어노테이션 적용 파라미터	Http 요청 헤더를 맵핑
@CookieValue 어노테이션 적용 파라미터	Http 쿠키 맵핑
@RequestBody 어노테이션 적용 파라미터	Http 요청의 몸체 내용에 접근할 때 사용 HttpMessageConverter을 이용해서 Http 요청 데이터를 해당 타입으로 변환한다.
Map, Model, ModelMap	뷰에 전달할 모델 데이터를 설정할 때 사용한다.
커맨드 객체	Http 요청 파라미터를 저장한 객체 기본적으로 클래스 이름을 모델명으로 사용한다. @ModelAttribute 어노테이션을 사용하여 모델명을 설정할 수 있다.
Errors, BindingResult	Http 요청 파라미터를 커맨드 객체에 저장한 결과 커맨드 객체를 위한 파라미터 바로 다음에 위치
SessionStatus	폼 처리를 완료 했음을 처리하기 위해 사용한다. @SessionAttribute 어노테이션을 명시한 session 속성을 제거하도록 이벤트를 발생시킨다.

# 컨트롤러 메소드의 리턴 타입

리턴 타입	설명
ModelAndView	뷰 정보 및 모델 정보를 담고 있는 ModelAndView 객체
Model	뷰에 전달될 객체 정보를 담고 있는 Model 객체 이때 View 이름은 요청 URL으로 부터 결정된다. RequestToViewNameTranslator를 통해서 뷰 결정
Map	뷰에 전달된 객체 정보를 담고 있는 Map 이때 View 이름은 요청 URL으로 부터 결정된다. RequestToViewNameTranslator를 통해서 뷰 결정
String	뷰 이름을 리턴한다.
View 객체	View 객체를 직접 리턴한다. 해당 View 객체를 이용해서 뷰를 생성한다.
void	메소드가 ServletResponse나 HttpServletResponse 타입의 파라미터를 갖는 메소드가 직접 응답을 처리한다고 가정한다. 그렇지 않을 경우 요청 URL으로부터 결정된 뷰를 보여준다. RequestToViewNameTranslator를 통해서 뷰 결정
@ResponseBody 어노테이션 적용	메소드에서 @ResponseBody 어노테이션이 적용된 경우, 리턴 객체를 Http 응답으로 전송한다. HttpMessageConverter를 이용해서 객체를 Http 응답 스트림으로 변환한다.

# ModelAndView

- 컨트롤러에 의하여 리턴되는 처리해야 할 결과 정보를 저장한다.
- Model(전달할 어떤 값의 목록) / View(전달될 곳)
- 뷰 정보 + 응답 화면 생성에 필요한 정보를 저장하는 영역이다.
- 특징**
  - DispatcherServlet은 ModelAndView의 뷰 이름을 이용하여 클라이언트에 출력할 결과를 생성해 주는 View 객체를 선택한다.
  - 저장된 객체는 jsp의 request 영역에서 접근이 가능하다.
  - 뷰 이름이 뷰 구현 기술에 의존하지는 않고, 단지 추상적으로 어떤 뷰를 사용한다는 정보만 저장하고 있다.
  - 실제 응답 결과는 ViewResolver를 통해서 결정 된다.
- 관련 메소드**

setViewName(String viewName)	뷰 이름을 지정하는 메소드이다.
addObject(String attributeName, Object attributeValue)	뷰에 전달할 값을 추가한다.
addAllObjects(Map modelMap)	<키, 값>의 구조로 되어 있는 Map을 ModelAndView 객체에 추가한다.

# ModelAndView 객체 생성

- 생성자를 사용하는 경우

```
@Override  
protected ModelAndView handleRequestInternal(HttpServletRequest request,  
HttpServletResponse response) throws Exception {  
  
    ModelAndView mav = new ModelAndView();  
    mav.setViewName("hello"); //뷰 이름을 설정.  
    mav.addObject("greeting", getGreeting()); //뷰에 전달할 값을 추가.  
  
    return mav;  
}
```

- Map 객체를 저장하는 경우

```
Map referenceMap = referenceData();  
  
mav.addAllObjects(referenceMap);
```

예시	설명
생성자를 사용하여 뷰 이름과 Map 전달	Map referenceMap = referenceData(); return new ModelAndView("bbs/list", referenceMap);
뷰에 전달할 객체가 한개 뿐일 경우	//뷰이름, 저장될이름, 저장될값 return new ModelAndView("help/faq", "faqs", faqs);
리다이렉트 뷰설정 뷰이름에 "redirect:" 접두어를 붙이면, 지정한 페이지로 리다이렉트 된다.	//리다이렉트 할 url : errorhtm ModelAndView mav =new ModelAndView(); mav.setViewName("redirect:/errorhtm"); return mav;

# ViewResolver

- 정의 및 특징

- 뷰 이름을 이용하여 결과를 출력할 View 객체를 구한다.
- JSP, Velocity, FreeMarker, PDF, Microsoft Excel 등 다양한 뷰를 지원한다.
- org.springframework.web.servlet.ViewResolver 인터페이스를 구현한 ViewResolver를 지원

파라미터 타입	설명
AbstractCachingViewResolver	Caching View를 다루는 추상 (Abstract) ViewResolver 해당 View를 확장 (extends)하는 View는 Caching 기능을 제공한다.
XmlViewResolver	Spring의 bean 팩토리처럼 DTD를 가진 XML 내 쓰여진 사항을 기초로 동작하는 ViewResolver 디폴트 설정 파일은 /WEB-INF/view.xml
ResourceBundleViewResolver 디폴트 설정 파일 : views.properties	ResourceBundle의 basename 속성에 명시된 bean 정의를 사용하는 ViewResolver 다른 ViewResolver와 혼합해서 사용 가능 [viewname].class = [viewname].url 형태로 설정 최신 버전에서는 .class를 .(class)으로 바꿔 줘야 한다.
UrlBasedViewResolver	주가적인 어떤 맵핑 작업을 하지 않고 URL의 상장적인 view 이름을 사용하는 ViewResolver 단순 JSP만 사용할 경우 사용이 가능하다. 보통 해당 클래스를 확장하여 제공하는 별도의 ViewResolver를 사용한다.
InternalResourceViewResolver	JSP, 서블릿, JstlView, TilesView 같은 View 기능을 제공하는 UrlBasedViewResolver의 편리한 하위 클래스
VelocityViewResolver	Velocity View 기능을 제공하는 UrlBasedViewResolver의 편리한 하위 클래스
FreeMarkerViewResolver	FreeMarker View 기능을 제공하는 UrlBasedViewResolver의 편리한 하위 클래스
ViewResolvers Chaining(혼합 사용)	여러 개의 ViewResolver가 있을 경우에 혼합해서 사용 가능 defaultParentView: 상위 ViewResolver의 설정사항을 오버라이드 가능 order : 여러 개의 ViewResolver가 존재할 경우 순서를 결정

# InternalResourceViewResolver 사용 예제

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/view/" />
    <property name="suffix" value=".jsp" />
</bean>
```

컨트롤러가 리턴한 ModelAndView 객체의 뷰 이름이 "hello"인 경우  
/WEB-INF/view/hello.jsp를 뷰로 사용하는 View 객체를 생성하도록 설정한다.

InternalResourceViewResolver 사용 예제

접두어(prefix) + view이름(hello) + 접미사(.jsp)  
/WEB-INF/view/ hello .jsp

```
@Override
public ModelAndView handleRequest(HttpServletRequest arg0,
    HttpServletResponse arg1) throws Exception {
    // 결과 값과 이동할 뷰 이름을 ModelAndView 객체에 담기
    ModelAndView mv=new ModelAndView();
    mv.setViewName("hello");//뷰이름
    return mv;
}
```

# ResourceBundleViewResolver 사용 예제

- View 관련 Url 맵핑을 별도의 프로퍼티 파일에서 관리하겠다.

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.ResourceBundleViewResolver">

    <!-- views.properties에 기술 -->
    <property name="basename" value="views"/>

    <!-- views.properties 파일에 정의되지 않을 경우에는 parentView에 정의된 사항을 따른다 -->
    <property name="defaultParentView" value="parentView"/>
</bean>
```

프로퍼티 파일 이름은 basename 프로퍼티로 지정한다.

## ResourceBundleViewResolver 사용 예제

### views.properties 파일의 내용 일부

```
bookView.class=org.springframework.web.servlet.view.JstlView
bookView.url=/WEB-INF/jsp/book/bookView.jsp
```

```
bookEdit.class=org.springframework.web.servlet.view.JstlView
bookEdit.url=/WEB-INF/jsp/book/bookEdit.jsp
```

뷰이름.(class) : View 인터페이스를 구현하는 클래스 이름

뷰이름.url : 이동시킬 뷰 이름을 지정한다.

View 인터페이스의 구현 클래스 (viewName.class)와 이동 장소의 url(viewName.url)을 입력한다.

# ViewResolver 사용 예제

## UrlBasedViewResolver 사용 예제

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
```

## xmlViewResolver 사용 예제

```
<bean id="jspViewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/jsp/" />
    <property name="suffix" value=".jsp" />
</bean>
<bean id="excelViewResolver" class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="order" value="1" />
    <property name="location" value="/WEB-INF/simpleviews.xml" />
</bean>
```

# Properties 파일 만들기

항목	설명
필요 유틸리티 파일	native2asciiedxe (JDK설치 폴더의 하위 \bin 폴더에 있음)
사용 방법	native2asciiedxe 사용자가 직접 작성한 파일 생성될 파일 이름 native2asciiedxe messagetxt messageproperties native2asciiedxe reverse messageproperties messagetxt (역변환시키기)
작성 순서	메모장을 이용하여 텍스트 파일을 작성한다. JDK설치 폴더의 하위 \bin 폴더로 파일을 복사한다. native2asciiedxe 파일을 이용하여 파일을 생성한다. (도스 창에서 수행)

## 파일 예시

```
name=설진욱  
password=1234  
user.id=홍길동
```

```
name=\uc124\uc9c4\uc6b1  
password=1234  
user.id=\ud64d\uae38\ub3d9
```

# Validator를 이용한 값 검증

항목	설명	
정의	org.springframework.validation.Validator 인터페이스 객체에 대한 유효성 검사를 수행하고자하는 경우에 사용	
특징	커맨드 객체를 사용하는 컨트롤러는 커맨드 객체를 생성한 뒤 Validator에 커맨드 객체를 전달하여 검증을 요청 BaseCommandController 확장 클래스 이후 클래스는 모두 가능	
관련 메소드	boolean supports(Class clazz)	Validator가 해당 클래스에 대한 값 검증을 지원하는지의 여부를 리턴 검증할 객체의 타입 (클래스) 정보를 파라미터로 받음
	void validate(Object target, Errors errors)	target 객체에 대한 검증을 실행 검증 결과 문제가 있을 경우 errors 객체에 어떤 문제인지에 대한 정보 저장
등록 순서	컨트롤러가 커맨드 객체를 검증할 수 있도록 Validator를 컨트롤러에 등록 커맨드 객체를 검사하기 위하여 Validator 인터페이스를 상속 받는 클래스를 작성한다.	

```
<!-- Controller -->
<bean id="loginFormController" class="controller.LoginFormController">
    ... 이하 중략

    <property name="validator"><ref bean="loginValidator"/></property>

    ... 이하 중략
</bean>
```

```
<!-- Validator -->
<bean id="loginValidator" class="utils.LoginValidator" />
```

# 유효성 검증 기능 구현

```
<!-- MessageSource -->
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>messages</value>
        </list>
    </property>
</bean>
```

error.input.user=사용자 정보가 누락되었습니다.  
error.required.user.userId=아이디 입력이 필요합니다.  
error.required.user.password=비밀번호 입력이 필요합니다.  
error.login.user=로그인 정보가 누락되었습니다.

message.properties 파일

```
//Validator 인터페이스 : 객체가 유효한지 검사해 주는 인터페이스
public class LoginValidator implements Validator{
    //supports : 해당 클래스가 검증을 지원하는지의 여부
    public boolean supports(Class clazz) {
        return User.class.isAssignableFrom(clazz);
    }
    //validate : 실제 로직을 검증해주는 메소드
    //command : 검증해야 할 어떤 대상
    //errors : 문제가 있는 경우에 이 객체에 저장된다.
    public void validate(Object command, Errors errors) {
        User user = (User)command;
        if(!StringUtils.hasLength(user.getUserId())){
            //reject : 커맨드 객체에 대한 일반적인 에러 설정
            errors.reject("error.input");
            //rejectValue : 특정 필드에 대한 에러 처리
            errors.rejectValue("userId", "error.required");
        }
        if(!StringUtils.hasLength(user.getPassword())){
            errors.reject("error.input");
            errors.rejectValue("password", "error.required");
        }
    }
}
```

# 유효성 검증 관련 사용 예시

```
<!-- MessageSource -->
<bean id="messageSource"
    class="org.springframework.context.support.ResourceBundleMessageSource">
    <property name="basenames">
        <list>
            <value>messages</value>
        </list>
    </property>
</bean>
```

스프링은 파일 이름에 확장자 properties를 붙여서 해석해준다.  
/src 폴더 내에 message.properties라는 파일이 존재한다.

```
messages.txt
1error.input.member=입력 정보가 잘못되었습니다.
2error.login=입력 정보가 누락되었습니다.
3error.required=회원 아이디를 입력해 주십시오.
4error.required.member.memberId=회원 ID를 입력해 주십시오.
5error.required.member.password=패스워드를 입력해 주십시오.
6error.required.member.memberName=이름
7error.required.member.postCode=우편 번호
```

Messages.properties 파일의 일부 내용  
[키=값]의 형태로 데이터가 저장되어 있다.

```
public void validate(Object command, Errors errors) {
    if( !StringUtils.hasLength( myPassword ) ){
        errors.reject("error.input");
        errors.rejectValue("password", "error.required");
    }
}
```

Validator 파일 내에서의 검증하기

```
<spring:bind path='member.password'>
    <input type="password" class="password"
        name='<c:out value="${status.expression}" />'
        value='<c:out value='${status.value}' />' maxlength='20'
        <font color="red"><c:out value='${status.errorMessage}' /></font>
    </spring:bind>
```

jsp 파일 내의 코드 사용 예시

# ValidationUtils

- 검증 관련 코드를 제공하는 유틸리티

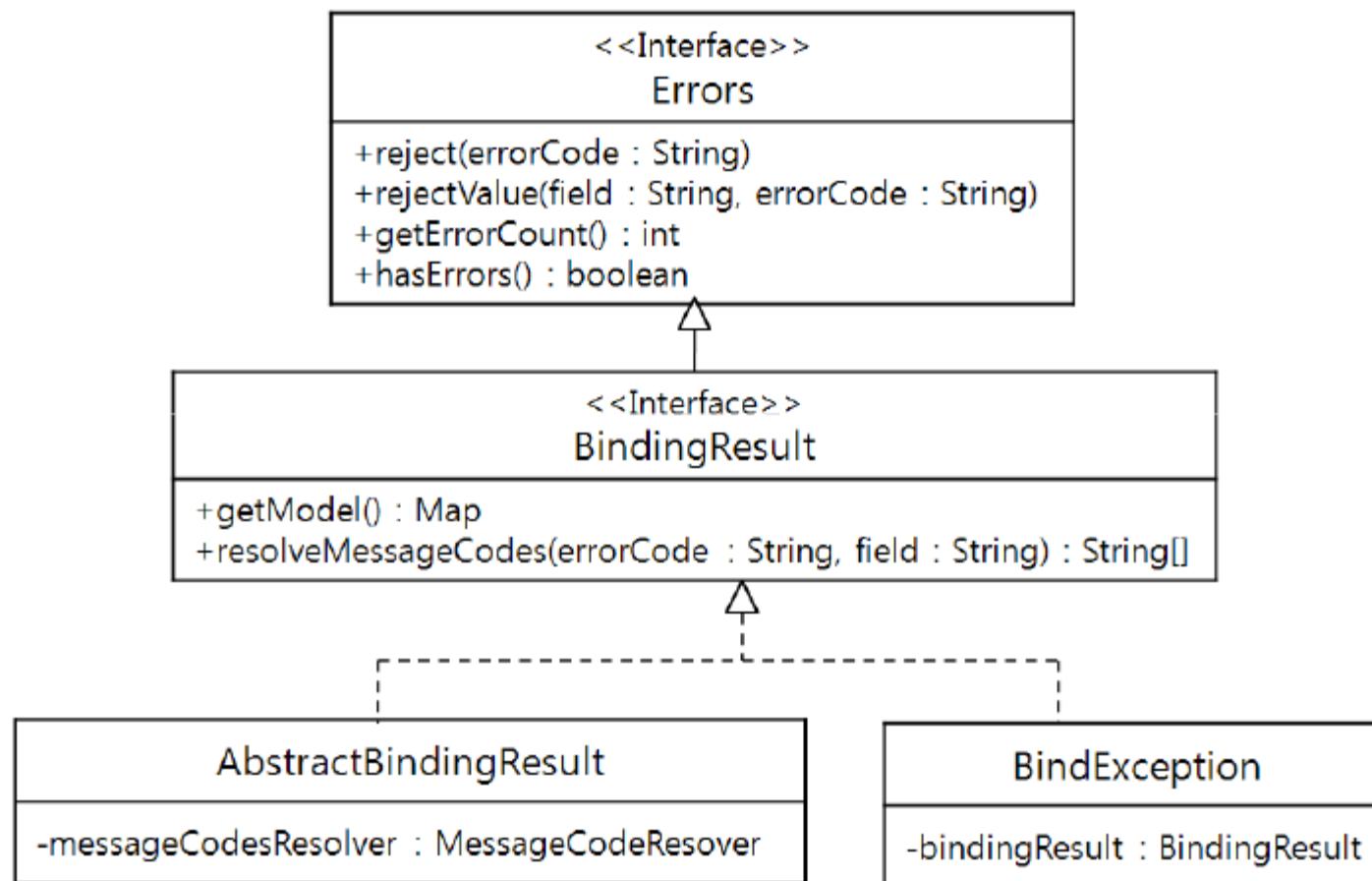
## 주요 메소드

static void rejectIfEmpty(Errors errors, String field, String errorCode, String defaultMessage)	값이 null 이거나, 길이가 0 인 경우 에러 코드를 추가한다.
---	--------------------------------------

static void rejectIfEmptyOrWhitespace(Errors errors, String field, String errorCode)	값이 null 이거나 길이가 0 인 경우 혹은 공백 문자만을 포함한 경우 잘못된 값으로 처리하고 싶을 때 사용
--	---

```
public class MemberInfoValidator implements Validator {  
    @Override  
    public void validate(Object target, Errors errors) {  
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "Id", "required");  
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "required");  
    }  
}
```

# Errors 인터페이스와 BindException 클래스



# Errors 인터페이스

- 커맨드(command) 객체의 검증 결과를 저장하고 있는 인터페이스
- 특징
  - 폼 자체(즉, 커맨드 객체 전체)에 대한 에러를 저장하기 위한 reject() 메소드 제공한다.
  - 커맨드 객체의 각 프로퍼티에 대한 에러를 명시할 수 있는 rejectValue() 메소드를 제공한다.

메소드	설명
hasErrors()	파라미터 값을 저장한 커맨드 객체가 검증을 통과 했는지의 여부를 확인
reject()	커맨드 객체 자체에 대한 에러 코드를 등록 메소드 종류 <pre>void reject(String errorCode) void reject(String errorCode, String defaultMessage) void reject(String errorCode, Object[] errorArgs, String defaultMessage)</pre> 에러 코드에 해당하는 에러 메시지가 존재하지 않는 경우 defaultMessage를 에러 메시지로 사용 errorArgs 파라미터 에러 페시지를 생성할 때 사용될 값
rejectValue() 사용 예	특정 검사할 객체의 특정 프로퍼티에 대한 값에 대한 에러 코드를 저장 메소드 종류 <pre>void rejectValue(String field, String errorCode) void rejectValue(String field, String errorCode, String defaultMessage) void rejectValue(String field, String errorCode, Object[] errorArgs, String defaultMessage)</pre> errors.rejectValue("homepageUrl", "invalidValue");

# BindingResult 인터페이스

- 요청 파라미터 값을 커맨드 객체에 복사한 결과를 저장
- 여러 코드로부터 여러 메시지를 가져온다.
- 특징
  - SimpleFormController 클래스와 AbstractWizardFormController 클래스는 getModel() 메소드로부터 구한 값을 입력폼에서 사용할 수 있도록 알맞게 처리하므로 직접 getModel() 메소드를 호출할 필요가 없다
  - AbstractCommandController 클래스는 getModel() 메소드를 직접적으로 사용하지 않기 때문에 직접 getModel() 메소드를 호출해서 ModelAndView 객체에 여러 관련 모델 정보를 추가.

항목	설명
getModel()	커맨드 객체에 대한 검증 결과 및 커맨드 객체 자체를 저장하고 있는 Map을 리턴한다. 여러 정보 및 커맨드 객체 정보를 ModelAndView에 전달한다.

- AbstractBindingResult 클래스
  - BindingResult 인터페이스의 기본 구현 클래스이다.
  - 검증 결과를 저장하고 여러 메시지를 추출하는 등의 기능을 제공한다.

# BindException 클래스

- Errors 인터페이스를 구현해 놓은 클래스
- 컨트롤러에 에러 정보를 전달할 때 사용한다.
- 내부적으로 bindingResult 필드를 사용하여 실제 요청 처리를 위임.
- Exception 클래스를 상속 받고 있기도 함.
- Errors에 오류가 추가되면 Controller에서 BindException 객체를 통해 오류가 있는지를 확인할 수 있다.

# WebUtils 클래스

- session 에 들어 있는 객체들을 보다 짧은 코드로 읽거나 쓸 수 있다.
- 세션 객체나 쿠키 객체에 대한 접근이 가능하다.

- 이전 사용 방식

```
UserSession userSession
```

```
= (UserSession)request.getSession().getAttribute("userSession");
```

- WebUtils 클래스를 사용한 방식

```
UserSession userSession
```

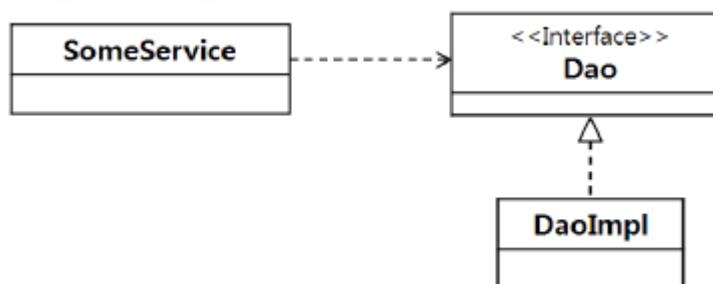
```
= (UserSession)WebUtils.getSessionAttribute(request, "userSession");
```

# Spring Framework

## Spring JDBC

# 데이터베이스 연동 지원

- 데이터베이스 연동을 구현하는 전형적인 방법
  - 데이터 베이스에 접근하는 클래스와 서비스 로직을 구현한 클래스를 구분.
  - 데이터베이스 접근을 위한 DAO(Data Access Object)를 만들고 , 서비스 클래스에서는 DAO를 호출해서 데이터에 대한 CRUD를 처리.
- 데이터베이스 연동을 위한 일반적인 구현
- JDBC, 하이버네이트, iBATIS 등의 다양한 기술을 이용해서 손쉽게 DAO 클래스를 구현할 수 있도록 지원



- 지원 내용
  - 템플릿 클래스를 통한 데이터 접근 : 중복된 코드를 매번 입력하는 성가신 작업을 줄인다.
  - DaoSupport 클래스를 이용한 DAO 클래스 구현
  - 의미 있는 예외 클래스를 제공해준다.
  - 즉, 단순히 SQLException이 아닌 수많은 xxxException을 지원한다.

# DAO 클래스를 위한 DaoSupport 클래스

클래스	설명
DAO	<p>Data Access Object</p> <p>데이터베이스 커넥션을 이용해서 데이터 접근을 처리</p> <p>DataSource와 같은 것으로부터 Connection을 구해야 한다.</p>
DaoSupport	<p>DAO에서 기본적으로 필요하는 기능을 제공하는 클래스이다.</p> <p>DaoSupport 클래스를 상속 받아서 DAO 클래스를 구현한다.</p> <p>JDBC : JdbcDaoSupport 클래스를 제공하고 있다.</p> <p>하이버네이트: HibernateDaoSupport 클래스를 제공하고 있다.</p>

# 스프링의 예외 지원

항목	설명
특징	<p>데이터베이스 처리와 관련된 예외 클래스를 제공한다.</p> <p>OptimisticLockingFailureException이나 DataRetrievalFailureException과 같이 보다 구체적인 실패 원인을 설명해 주는 예외를 준비해 두고 있다.</p> <p>JDBC에서는 예외의 종류에 상관없이 무조건 SQLException이었다.</p>
클래스 제공	<p>템플릿 클래스는 내부적으로 발생하는 예외 클래스를 스프링이 제공하는 예외 클래스로 맞게 변환해서 예외를 발생시킨다.</p> <p>데이터베이스 연동을 위해 사용하는 기술에 상관없이 동일한 방식으로 예외 처리를 수행해야 한다.</p>
DataAccessException 클래스	<p>스프링이 제공하는 데이터베이스 관련 예외 클래스들은 모두 DataAccessException 클래스를 상속 받는다.</p> <p>RuntimeException 이므로 필요한 경우에만 try-catch 블록을 이용해서 예외를 처리해야 한다.</p>

# 일반적인 JDBC 코딩 예시

```
public int UpdateBoard( BoardBean board ) {
    //해당 게시물을 수정한다.
    PreparedStatement pstmt = null ;
    String sql = " update boards set subject=?, writer=? " ;
    sql += " where no = ? " ;

    int cnt = -1 ;
    try {
        conn.setAutoCommit(false);
        pstmt = conn.prepareStatement( sql )
        pstmt.setString(1, board.getSubject());
        pstmt.setString(2, board.getWriter());
        pstmt.setInt(3, board.getNo());
        cnt = pstmt.executeUpdate() ;
        conn.commit();

    } catch (SQLException e) {
        try {
            conn.rollback();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        e.printStackTrace();
    } finally{
        try {
            if( pstmt != null ){ pstmt.close(); }
            closeConnection();
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }
    return cnt ;
}
```

```
public int DeleteBoard( int no ){
    //넘겨진 글 번호를 이용하여 해당 글을 삭제한다.
    PreparedStatement pstmt = null ;
    String sql = "delete from boards where no = ? " ;
    int cnt = -1 ;
    try {
        conn.setAutoCommit(false);
        pstmt = conn.prepareStatement(sql) ;
        pstmt.setInt(1, no) ;
        cnt = pstmt.executeUpdate() ;
        conn.commit();

    } catch (SQLException e) {
        try {
            conn.rollback();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
        e.printStackTrace();
    } finally{
        try {
            if( pstmt != null ){ pstmt.close(); }
            closeConnection();
        } catch (Exception e2) {
            e2.printStackTrace();
        }
    }
}
```

가변적인 코딩 부분이다.  
해당 업무에 따른 SQL 구문을  
정의하고 있다.

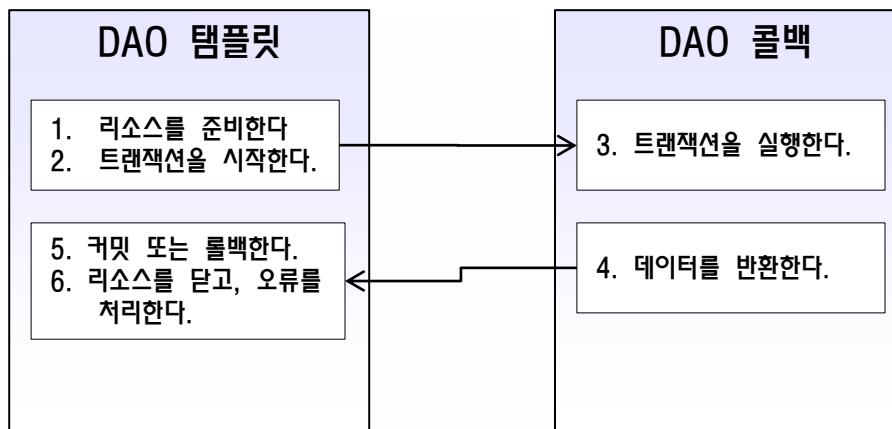
고정적인 코딩 부분이다.  
양쪽 코드가 거의 동일하다.  
개발자로서는 매우 성가신 단순 작업이다.

스프링은 이것을 Template(템플릿) 클래스로 만들  
어서 내부적으로 숨겨 버렸다.

# 데이터 액세스 Template화

- 스프링은 데이터 액세스 절차상 고정적인 단계와 가변적인 단계를 Template(탬플릿)과 callback(콜백)이라는 2가지 단계로 분리했다.

항목	설명
탬플릿 클래스	트랜잭션 제어, 자원 관리, 예외 처리와 같은 고정적인 부분
콜백 구현	질의 객체(statement) 생성, 파라미터 바인딩, 질의 결과 추출 및 반환



콜백이란 특정 이벤트에 의하여 시스템이 알아서 실행하게 되는 어떤 한 함수, 메소드를 말한다.

예를 들어서, 자바의 GUI에서 Button을 클릭하게 되면 ActionListener의 actionPerformed() 메소드가 자동으로 콜백된다.

# 스프링을 위한 템플릿 클래스

- Connection 및 try-catch-finally로 자원을 관리하는 등의 중복된 코드를 제거한다.

## 스프링을 위한 템플릿 클래스 목록

Template	설명
JdbcTemplate	기본적인 JDBC 템플릿 클래스 jdbc.core.JdbcTemplate JDBC(색인된 파라미터 기반)의 쿼리를 통하여 Database에 액세스하는 기능을 제공
NamedParameterJdbcTemplate	PreparedStatement에서 인덱스 기반의 파라미터가 아닌 이름을 가진 파라미터를 사용할 수 있도록 지원하는 템플릿 클래스
SimpleJdbcTemplate	자바 5의 AutoBoxing, Generics, 가변 파라미터 목록 등을 이용해서 쿼리를 실행할 때 사용되는 데이터를 전달할 수 있는 템플릿 클래스. 자바 1.4 이하의 버전에서는 사용할 수 없다. jdbc.core.simple.SimpleJdbcTemplate
HibernateTemplate	orm.hibernate3.HibernateTemplate 하이버 네이트 연결
SqlMapClientTemplate	아이바티스 연결 orm.ibatis.SqlMapClientTemplate

# JdbcTemplate 클래스를 이용한 JDBC 프로그래밍

- **JdbcTemplate 클래스**

- DataSource를 필요로 한다.
- 설정 파일을 이용해서 JdbcTemplate의 dataSource 프로퍼티에 DataSource를 전달한다.

```
<!-- 1.DataSource 설정 -->
<bean id="dataSource" class="org.apache.tomcat.dbcp.dbcp.BasicDataSource">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@mycomputer:1521:orcl999"/>
    <property name="username" value="scott" />
    <property name="password" value="tiger" />
</bean>

<!-- 2.JdbcTemplate 객체설정 -->
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
    <property name="dataSource" ref="dataSource"/>
</bean>

<!-- 3.DAO 빈 객체 설정 -->
<bean id="dao" class="myPkg.dao.MemberDAO">
    <!-- setter Injection -->
    <property name="jdbcTemplate" ref="jdbcTemplate"/>
</bean>
```

# DAO 클래스에서 JdbcTemplate 사용

```
// set 메소드나 생성자를 통해서 JdbcTemplate을 전달 받을 수 있도록 함.
public class JdbcTemplateMessageDao implements MessageDao {

    private JdbcTemplate jdbcTemplate;

    //setter을 사용한 객체 전달 받기
    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
    ...
}
```

```
<!-- 설정 파일에서는 DAO에 JdbcTemplate을 전달하도록 설정. -->

<bean id="messageDao" class="myPkg.dao.jdbc.JdbcTemplateMessageDao">

    <property
        name="jdbcTemplate"
        ref="jdbcTemplate" />
</bean>

<!-- 설정 완료 후 JdbcTemplate이 제공하는 기능 사용 가능. -->
```

# NamedParameterJdbcTemplate을 이용한 JDBC

- NamedParameterJdbcTemplate 클래스
  - 인덱스 기반의 파라미터가 아닌 이름 기반의 파라미터를 설정할 수 있도록 해주는 템플릿 클래스.
  - 인덱스 기반의 파라미터를 전달받는 물음표를 사용하지 않고 이름 기반의 파라미터를 쿼리에서 사용할 수 있도록 지원.
- 설정
  - 생성자를 이용해서 DataSource를 전달 받음.

```
<bean id="namedParameterJdbcTemplate"
      class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate">

    <constructor-arg>
        <ref bean="dataSource" />
    </constructor-arg>
</bean>
```

# NamedParameterJdbcTemplate 관련 메소드

- JdbcTemplate과 비슷한 메서드를 제공한다.
- Object[] 대신에 Map을 전달 받는다.
- 종류

```
public List query(String sql, Map paramMap, RowMapper rowMapper)  
public Object queryForObject(String sql, Map paramMap, RowMapper rowMapper)  
public int queryForInt(String sql, Map paramMap)  
public List queryForList(String sql, Map paramMap)  
public int update(String sql, Map paramMap)
```

- paramMap 파라미터
  - 이름 기반의 파라미터에 삽입될 값을 설정하기 위한 Map
  - 쿼리에서 사용되는 이름과 값을 저장한다.

# NamedParameterJdbcTemplate 사용 예시

```
public class NamedMessageDao
    extends NamedParameterJdbcDaoSupport
    implements MessageDao {

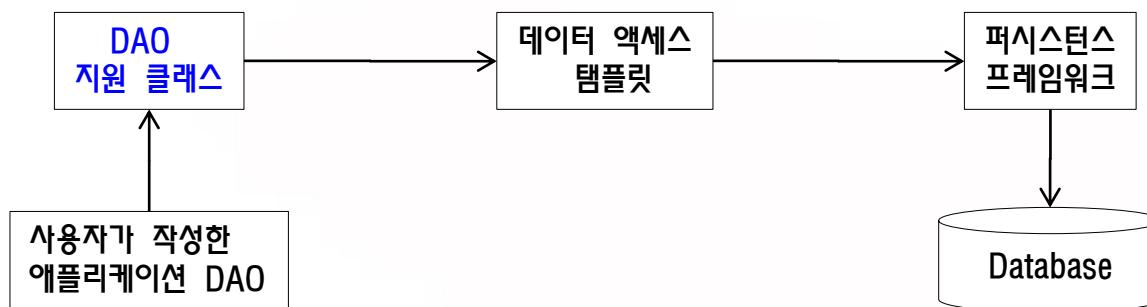
    ... 중간 생략

    @Override
    public void insert(Message message) {
        Map<String, Object> params = new HashMap<String, Object>();
        params.put("guestName", message.getGuestName());
        params.put("content", message.getContent());

        getNamedParameterJdbcTemplate().update(INSERT_SQL, params);
    }
    ... 중간 생략
}
```

# Dao 지원 클래스 사용하기

- 템플릿- 콜백 설계를 기반으로 확장을 통해 DAO 지원 클래스를 제공하고 있다.
- 스프링 DAO 클래스를 구현할 때 상위 클래스로 사용한다.
- 예를 들어서, 애플리케이션 DAO가 JdbcDaoSupport를 상속 받았다면 필요한 JdbcTemplate을 획득하기 위해서는 getJdbcTemplate() 메소드를 호출하면 된다.
- 아래 그림은 템플릿 클래스와 DAO 지원 클래스와 개발자가 작성한 맞춤형 DAO 간의 구현 관계를 묘사하고 있다.



항목	설명
JdbcDaoSupport	DataSource 관리 JdbcTemplate을 지원하는 DaoSupport 클래스
NamedParameterJdbcDaoSupport	NamedParameterJdbcTemplate을 지원하는 DaoSupport 클래스
SimpleJdbcDaoSupport	SimpleJdbcTemplate을 지원하는 DaoSupport 클래스
SqIMapClientDaoSupport	iBatis SqIMap Client

# JdbcDaoSupport 클래스를 상속 받은 클래스

```
public class JdbcMessageDao  
    extends JdbcDaoSupport implements MessageDao {  
  
    private String SELECT_COUNT_SQL  
        = "select count(*) from GUESTBOOK_MESSAGE";  
  
    @Override  
    public int selectCount() {  
        return getJdbcTemplate().queryForInt(SELECT_COUNT_SQL);  
    }  
}
```

예시에서는 JdbcDaoSupport 클래스를 상속 받고 있다.  
따라서, getJdbcTemplate()라는 메소드를 이용하여  
JdbcTemplate 객체를 얻어낼 수 있다.  
이를 이용하여 queryForInt() 메소드를 활용하고 있다.

# DataSource 설정

- 어떠한 DAO를 사용하든 간에 데이터 소스 빈을 설정해야 한다.

항목	설명
DataSource 설정	DaoSupport 클래스나 템플릿 클래스 그리고 프레임워크와의 연동을 위해 제공되는 클래스를 사용할 경우 스프링은 DataSource를 통해 Connection을 제공하고 있다.
DataSource 방식	JDBC 드라이버를 통한 DataSource 설정 JNDI를 이용한 DataSource 설정(Java Naming and Directory Interface) 커넥션 풀(pooling)을 이용한 DataSource 설정

# DataSource 설정

- 풀링 기능이 있는 Data Source 사용하기
- 스프링 자체가 제공하는 것은 아니고, Apache Commons Database Connection Pool(DBCP 라고 한다.)이 이런 기능을 제공한다.
- 가장 많이 쓰이는 것으로 BasicDataSource가 있다.

```
<!-- DBCP 를 이용한 DataSource설정 -->
<bean id="dataSource"
      class="org.apache.tomcat.dbcp.dbcp.BasicDataSource">
    <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
    <property name="username" value="hr"/>
    <property name="password" value="hr"/>
</bean>
```

설정 이름	값의 예시	설명
driverClassName	oracle.jdbc.OracleDriver	JDBC의 드라이브 클래스명 설정
url	jdbc:oracle:thin:@localhost:1521:xe	실제 데이터베이스 파일의 위치
username	hr	사용자 계정 설정
password	hr	사용자 패스워드 설정

# DataSource 설정

- JDBC 드라이버 기반 Data Source

항목	설명
DriverManagerDataSource	커넥션을 요청할 때마다 새로운 커넥션을 반환한다. 커넥션 풀링 기능은 제공하지 않는다. 새로운 커넥션이 만들어질 때마다 커넥션을 생성하므로 심각한 성능 저하를 초래한다.
SingleConnectionDataSource	커넥션을 요청하면 항상 동일한 커넥션을 반환한다. 오직 1개의 커넥션을 제공하는 데이터 소스이다. MultiThread 애플리케이션에서는 사용 불가능하다.

# DriverManager를 이용한 DataSource 설정

- DriverManagerDataSource를 이용한 데이터 베이스 설정

```
<!-- 데이터소스 설정 : DriverManagerDataSource 클래스를 이용한다. -->
<bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">

    <!-- JDBC드라이버클래스명 설정 -->
    <property name="driverClassName">
        <value>oracle.jdbc.driver.OracleDriver</value>
    </property>

    <!-- 데이터베이스 파일의 위치 설정 -->
    <property name="url">
        <value>jdbc:oracle:thin:@192.168.0.3:1521:orcl999</value>
    </property>

    <!-- 사용자 계정 설정 -->
    <property name="username">
        <value>scott</value>
    </property>

    <!-- 사용자 패스워드 설정 -->
    <property name="password">
        <value>tiger</value>
    </property>
</bean>
```

# 콜백 인터페이스

인터페이스	기능
PreparedStatement Creator	createPreparedStatement() 메소드에 인수로써 전달된 Connection에서 PreparedStatement를 생성하여 매개 변수를 설정한다.
PreparedStatement Setter	PreparedStatementCreator와 거의 동일한 기능을 수행하지만 setValues() 메소드에는 PreparedStatement가 인수로써 전달된다.
BatchPreparedStatementSetter	Batch 처리에서 같은 SQL을 여러 번 실행시 사용한다. getBatchSize()에서 Batch의 사이즈를 반환한다. setValues()에는 PreparedStatement와 인덱스가 전달되므로 인덱스에 대응한 매개 변수를 PreparedStatement로 설정한다.
RowMapper	mapRow() 메소드에서 ResultSet을 사용한 처리를 행한다. SQL의 실행의 결과와 행수가 같은 횟수만큼 이 메소드가 호출된다. ResultSet#next()는 스프링이 호출하기 때문에 이 구현 클래스를 호출할 필요는 없다.
ResultSetExtractor	extractData() 메소드에서 ResultSet을 사용한 처리를 한다. 이 메소드는 1회 SQL 실행에 대해 한번만 호출된다. ResultSet#next()는 이 구현 클래스가 호출해야 한다.

# ResultSetExtractor & RowMapper

인터페이스	RowMapper	ResultSetExtractor
ResultSet을 취급하는 메소드	Object mapRow(ResultSet rs)	Object extractData(ResultSet rs)
그 메소드가 호출된 횟수	SQL 실행 결과의 행 수 만큼	1 번만
ResultSet의 next() 메소드	Spring 이 호출	구현 클래스가 호출
사용에 적합한 경우	List 형으로 결과를 반환하고 싶을 경우	임의 형으로 결과를 반환하고 싶을 경우

# RowMapper 인터페이스

- ResultSet에서 값을 가져와 원하는 타입으로 매팅할 때 사용
- 콜백 처리
  - 개발자가 정의해 놓은 메소드를 프레임워크가 호출하는 것
- 콜백 인터페이스
  - 프레임워크가 콜백을 위하여 제공해주는 인터페이스
  - RowMapper 인터페이스는 콜백 인터페이스이다.
  - 사용자가 해당 메소드 mapRow()를 정의해 두면 프레임워크에서 이것을 자동으로 호출해준다.
- 인터페이스 정의

```
public interface RowMapper {  
    Object mapRow(ResultSet rs, int rowNum) throws SQLException;  
}
```

- mapRow() 메소드
  - ResultSet에서 읽어온 값을 이용해서 원하는 타입의 객체를 생성한 뒤 리턴
  - rowNum은 행 번호를 의미 (0부터 시작)

# RowMapper 인터페이스

- 여러 메소드에서 공통으로 사용되는 코드가 있다면 RowMapper 구현 클래스를 별도로 구현해서 재사용

```
public List<MemberBean> getMemberList() { //회원 목록 얻어오기
    String sql = "select num, name, addr from mem";

    //하나의 로우에 매핑되는 의명의 이너클래스 생성
    RowMapper rowmapper=new RowMapper() {
        @Override
        public Object mapRow(ResultSet rs, int index)
            throws SQLException {
            MemberBean bean = new MemberBean(rs.getInt(1),
                rs.getString(2),
                rs.getString(3));
            return bean;
        }
    };

    //query() : 추출된 데이터가 여러 개인 경우에 해당하는 로우들을 bean에 담고
    // 그 bean를 List 컬렉션에 담아서 리턴해준다.
    List<MemberBean> list
        = (List<MemberBean>)jdbcTemplate.query(sql, rowmapper);

    return list;
}
```

# PreparedStatementSetter 인터페이스

- PreparedStatementCreator와 거의 동일한 기능을 수행한다.
- setValues() 메소드에는 PreparedStatement가 인수로써 전달된다.

```
private class ItemPreparedStatementSetterForPrimaryKey  
    implements PreparedStatementSetter{  
  
    private Integer itemId;  
  
    public void setValues(PreparedStatement ps) throws SQLException {  
        //상품ID를 PreparedStatement인스턴스로 설정  
        ps.setInt(1, this.itemId.intValue());  
    }  
}
```

setValues 메소드에서 내용을 처리한다.  
JDBC에서의 setXXX와 동일한 개념이다.

```
@Override //상품 id에 의한 Item 검색하기  
public Item findByPrimaryKey(Integer itemId) {  
    return (Item) getJdbcTemplate().query(  
        ItemDaoImpl.SELECT_BY_PRIMARY_KEY,  
        new ItemPreparedStatementSetterForPrimaryKey(itemId),  
        new ItemResultSetExtractor());  
}
```

# JdbcTemplate에서 제공하는 메소드

- JdbcTemplate 클래스를 이용해서 select 쿼리를 실행한다.
- query() 메소드를 사용한다.
  - sql 파라미터로 전달 받은 select 쿼리를 실행한 뒤 결과를 리턴한다.
  - PreparedStatement를 사용할 경우 쿼리에 포함된 물음표(인덱스 파라미터)에 전달할 값을 Object[] 배열 형태로 전달한다.

항목	설명
List query(String sql, Object[] args, RowMapper rowMapper)	PreparedStatement를 이용해서 SELECT 쿼리를 실행할 때 사용 (즉, ?가 존재하는 경우)
List query(String sql, RowMapper rowMapper)	정적 SQL을 이용해서 SELECT 쿼리를 실행할 때 사용(?가 존재하지 않는 경우)

# JdbcTemplate에서 제공하는 메소드

- `queryForObject()` 메소드
- 정의
  - 쿼리 실행시 1건의 결과가 조회된다면 이 메소드를 사용한다.
- 종류
  - `public Object queryForObject(String sql, RowMapper rowMapper)`
  - `public Object queryForObject(String sql, Object[] args, RowMapper rowMapper)`
- 특징
  - 전달되는 각각의 파라미터는 `query()` 메소드와 동일하다.
  - 차이점 : 리턴되는 데이터는 한 개의 Bean 객체이다.(List 컬렉션이 아니고)
  - 쿼리 실행 결과의 행 개수가 한 개가 아닌 경우에는 `IncorrectResultSizeDataAccessException` 예외를 발생한다.

# JdbcTemplate에서 제공하는 메소드

```
//회원 한명의 정보 얻어오기
public MemberBean getInfo(int num){
    //하나의 로우에 매핑되는 의명의 이너클래스 생성
    RowMapper myRowMapper = new RowMapper() {
        @Override
        public Object mapRow(ResultSet rs, int index)
            throws SQLException {
            MemberBean bean = new MemberBean(rs.getInt(1),
                rs.getString(2),
                rs.getString(3));
            return bean;
        }
    };
    //queryForObject() : 추출된 데이터가 하나인 경우, 해당하는 로우를
    //Object 형태로 리턴한다.
    MemberBean member
        = (MemberBean)jdbcTemplate.queryForObject(
            "select * from mem where num=?",
            new Object[]{num}, myRowMapper);
    return member;
}
```

메소드의 종류

```
public Object queryForObject(String sql, RowMapper rowMapper)
public Object queryForObject(String sql, Object[] args, RowMapper rowMapper)
```

# JdbcTemplate에서 제공하는 메소드

- Object가 아닌 int나 long 타입을 결과를 구할 때 사용할 수 있는 메소드
- 종류
  - public int queryForInt(String sql)
  - public int queryForInt(String sql, Object[] args)
  - public long queryForLong(String sql)
  - public long queryForLong(String sql, Object[] args)
- 특징
  - 결과 행 개수가 한 개가 아닌 경우 예외가 발생한다.
- 사용 예

```
private String sql = "select count(*) from guestbook" ;  
  
public int selectCount() {  
    return jdbcTemplatequeryForInt( sql );  
}
```

# JdbcTemplate에서 제공하는 메소드

- update() 메소드
  - DML 구문(insert, update, delete) 쿼리를 실행할 때에 사용한다.
- 종류
  - public int update(String sql)
  - public int update(String sql, Object[] args)
- 특징
  - 쿼리 실행 결과 변경된 행의 개수를 리턴한다.
- 사용 예

```
public void delete(int num){ //회원 정보 삭제하기.  
    Object obj[] = { num };  
    String sql = "delete from mem where num=?";  
    jdbcTemplate.update( sql, obj );  
}
```

```
public void insert(MemberBean bean){ //회원 정보 저장하기.  
    Object obj[]={ bean.getName(), bean.getAddr()};  
  
    String sql = "insert into mem values( seqmem.nextval, ?, ? )";  
  
    jdbcTemplate.update( sql, obj );  
}
```

# RDBMS 조작 클래스

- 스프링의 JDBC 추상 프레임워크에는 RDBMS 조작 클래스를 다음과 같이 지원하고 있다.

RDBMS 조작 클래스	사용에 적합한 상황
MappingSqlQuery	SQL의 실행 결과를 Java 클래스로 맵핑할 경우
MappingSqlQueryWithParameters	MappingSqlQuery와 동일하지만 매개 변수가 필요한 경우
SqlUpdate	DML(Insert, update, delete) 등 간접적인 작업을 하는 SQL을 실행하는 경우
UpdatableSqlQuery	검색을 실행하여 그 결과에 대하여 간접적인 작업을 할 경우
BatchSqlUpdate	Batch 간접적인 작업을 할 경우
SqlFunction	단일 결과만을 반환하는 SQL을 실행할 경우
StoredProcedure	StoredProcedure를 이용할 경우

# SqlUpdate을 사용하는 예시

- 생성자에서 슈퍼 클래스의 생성자를 호출하여 데이터 소스와 SQL 문을 인수로 전달한다.
- declareParameter() 메소드를 호출하여 매개 변수 객체 (SqlParameter 인스턴스)를 치환한다.
- Compile() 메소드를 호출한다.

```
class InsertPlayer extends SqlUpdate {  
    public InsertPlayer(DataSource ds) {  
        super(ds, "INSERT INTO player(name, team_id) VALUES(?, ?)");  
        super.declareParameter(new SqlParameter("name", Types.VARCHAR));  
        super.declareParameter(new SqlParameter("team_id", Types.INTEGER));  
        compile();  
    }  
}
```

# execute() 메소드

- Connection을 직접 사용해야 할 경우 사용한다.
- 파라미터로 전달받은 ConnectionCallback 인터페이스 구현 객체의 doInConnection() 메소드를 호출하는데 이때 Connection을 doInConnection() 메소드에 전달해야 한다.
- 커넥션 생성과 종료는 JdbcTemplate이 처리하므로 Connection 종료할 필요가 없다.

```
jdbcTemplate.execute(new ConnectionCallback() {
    @Override
    public Object doInConnection(Connection con) throws SQLException, DataAccessException
    {
        Statement stmt = null;
        try {
            stmt = concreateStatement();
            ...
        } finally {
            JdbcUtils.close.Statement(stmt);
        }
        return someValue;
    }
}
```

# iBatis 지원

- 스프링에서 iBatis를 사용하는 경우에는 SqlMapClient를 사용한다.
- SqlMapClient 역시 JDBC와 마찬가지로 try ... catch() 처리를 해줘야 한다.
- 이러한 단점을 없애고자 SqlMapClientTemplate 클래스를 제공하고 있다.

클래스	설명
SqlMapClientTemplate 클래스	try ... catch() 시에 발생하는 코드의 중복을 없애 준다. 스프링이 제공하는 다양한 예외 클래스로 변환해 준다.
SqlMapClientDaoSupport 클래스	SqlMapClientTemplate를 DAO 클래스에 좀 더 쉽게 사용할 수 있도록 제공해주는 클래스
SqlMapClientFactoryBean 클래스	스프링 설정 파일에서 SqlMapClient을 쉽게 설정할 수 있도록 제공하는 클래스이다. dataSource 프로퍼티에는 DataSource를 전달한다. configLocation 프로퍼티에는 iBatis 설정 파일 이름을 명시한다.

```
<!-- DBCP 를 이용한 DataSource설정 -->  
... 코드 생략
```

```
<!-- 2.SqlMapClient 객체 설정 -->  
<bean id="sqlMapClient"  
      class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">  
    <property name="dataSource" ref="dataSource"/>  
    <property name="configLocation"  
              value="classpath:ibatis/SqMapConfig.xml"/>  
</bean>  
  
<!-- DAO 빈 설정 -->  
<bean id="dao" class="myPkg.dao.MemberDao">  
  <property name="sqlMapClient" ref="sqlMapClient"/>  
</bean>
```

# 실습 : JavaReview

- 스프링 framework를 공부하다 보면 Map 인터페이스가 많이 나오게 된다.
- 이를 위하여 미리 자바의 Map 인터페이스에 대하여 사전 연습을 해보도록 한다.
- Map : 키 (key)와 값 (value) 2개의 쌍으로 구성된 인터페이스이다.
- Map 인터페이스를 구체화시킨 클래스 : HashMap, Hashtable 등등이 존재한다.

Map 인터페이스의 주요 메소드	
메소드	항목에 대한 설명
put("name", "홍길동")	key와 value을 이용하여 데이터를 추가한다.
map.containsKey( 키이름 )	해당 목록에 [키이름]이 존재하면 true 이다.
map.remove( 삭제될키 )	[삭제될키] 를 원소에서 제거한다.
map.get(key)	해당 키를 이용하여 value 를 얻어 온다.

# 실습 : JavaReview

- Inner 클래스 및 인터페이스 상속을 이용한 형변환
- 인터페이스
  - 도형 (Shape)을 나타내기 위한 인터페이스
- 구체화 시킨 클래스
  - Circle 클래스 , Rectangle 클래스
- 메인 클래스
  - InnerTest 클래스
    - Anonymous Inner Type( 익명 내부 타입 ) 으로 작성하기
    - 객체 변수없는 Anonymous Inner Type : 1 회용
    - // 완전 구현한 클래스를 이용하는 방법
    - Circle circle = new Circle();
    - // 인터페이스이름 객체 = new 구체화시킨클래스
    - Shape myshape = new Circle();

# 실습 : 원의 면적과 반지름

## • 프로젝트 : CircleExam\sample1 패키지

메인 메소드에서 Circle 클래스를 직접 이용하여 객체를 생성하고 있다.  
이것을 강한 결합(tight coupling)이라고 부른다

강한 결합이란 클래스간의 결합력이 강해지고 의존력이 높은 결합을 의미한다.  
클래스에 어떤 변경이 생기면, 코드를 수정해야 하는 범위가 넓어질 가능성이 높다.

예를 들어 Circle 클래스 대신 다른 클래스를 사용한다고 가정하자.  
이때 다른 클래스에 `Display()`라는 메소드가 존재한다는 보장이 없다.  
그럼 CircleMain 클래스는 코드를 수정해야 한다.  
그래서, 약한 결합(loose coupling)을 만들기 위해서 인터페이스를 사용한다.

```
public class Circle {  
    private double radius ; //반지름  
    private Point point ; //점에 대한 객체  
  
    public Circle(double radius, Point point) {  
        this.radius = radius ;  
        this.point = point ;  
    }  
  
    public void Display() {  
        System.out.println("원의 중심 : " + point.getXpos() + ", " + point.getYpos());  
        System.out.println("원의 면적 : " + getArea());  
    }  
  
    private double getArea() {  
        return Math.PI * Math.pow(radius, 2.0) ;  
    }  
}
```

원(Circle) 클래스

```
public class CircleMain {  
    public static void main(String[] args) {  
        final double radius = 10.0 ;  
        Point point = new Point() ;  
        //setter을 사용한 데이터 넣기  
        point.setXpos(3.0) ;  
        point.setYpos(4.0) ;  
    }  
}
```

메인 파일

//생성자를 이용한 데이터 넣기  
Circle circle = new Circle( radius, point ) ;  
circle.Display(); //원들의 정보를 출력한다.

```
package sample1;  
public class Point {  
    private double xpos ; //x 좌표  
    private double ypos ; //y 좌표  
  
    //getter, setter 생략  
}
```

점(Point) 클래스

Circle 생성자의 매개 변수는 2개이다.  
매개 변수란 생성자/메소드의 외부에서 넣어 주는 데이터를  
말하는 데, 이것을 주입(Injection)이라고 부른다.  
Injection은 크게 생성자, setter Injection이 존재한다.

# 실습 : 원의 면적과 반지름

- 프로젝트 : CircleExam\sample2 패키지

## 원(Circle) 인터페이스

```
public interface Circle {  
    public void Display();  
    public double getArea();  
}
```

```
public class CircleImpl implements Circle {  
    private double radius ; //반지름  
    private Point point ; //점에 대한 객체  
  
    public CircleImpl(double radius, Point point) {  
        System.out.println("sample2 패키지---CircleImpl 생성자 호출됨");  
        this.radius = radius ;  
        this.point = point ;  
    }  
    @Override  
    public void Display() {  
        System.out.println("원의 중심 : " + this.point.getXpos() + ", "  
        System.out.println("원의 면적 : " + getArea());  
    }  
    @Override  
    public double getArea() {  
        return Math.PI * Math.pow(radius, 2.0) ;  
    }  
}
```

CircleImpl 클래스

```
final double radius = 10.0 ;  
Point point = new PointImpl() ;
```

//setter을 사용한 데이터 넣기 메인 파일  
point.setXpos(3.0) ;  
point.setYpos(4.0) ;

Circle circle = new CircleImpl( radius, point ) ;  
circle.Display(); //원들의 정보를 출력한다.

순수 클래스만을 이용한 방법보다는  
의존 관계가 조금 약해졌다.

인터페이스를 이용하게 되면 상대적으로 의존 관계가 약해지게 된다.  
인터페이스는 코딩상 강제 규약(주상 메소드)의 기능을 가지고 있기 때문에 강  
한 애플리케이션을 만들수 있다.  
하지만 방대한 애플리케이션일 경우 인스턴스 생성을 여러 군데서 하는 경우에 역시  
변경할 코드의 양도 많아지게 된다.

이러한 문제점은 **스프링 framework**로 해결할 수 있다.

# 실습 : 원의 면적과 반지름

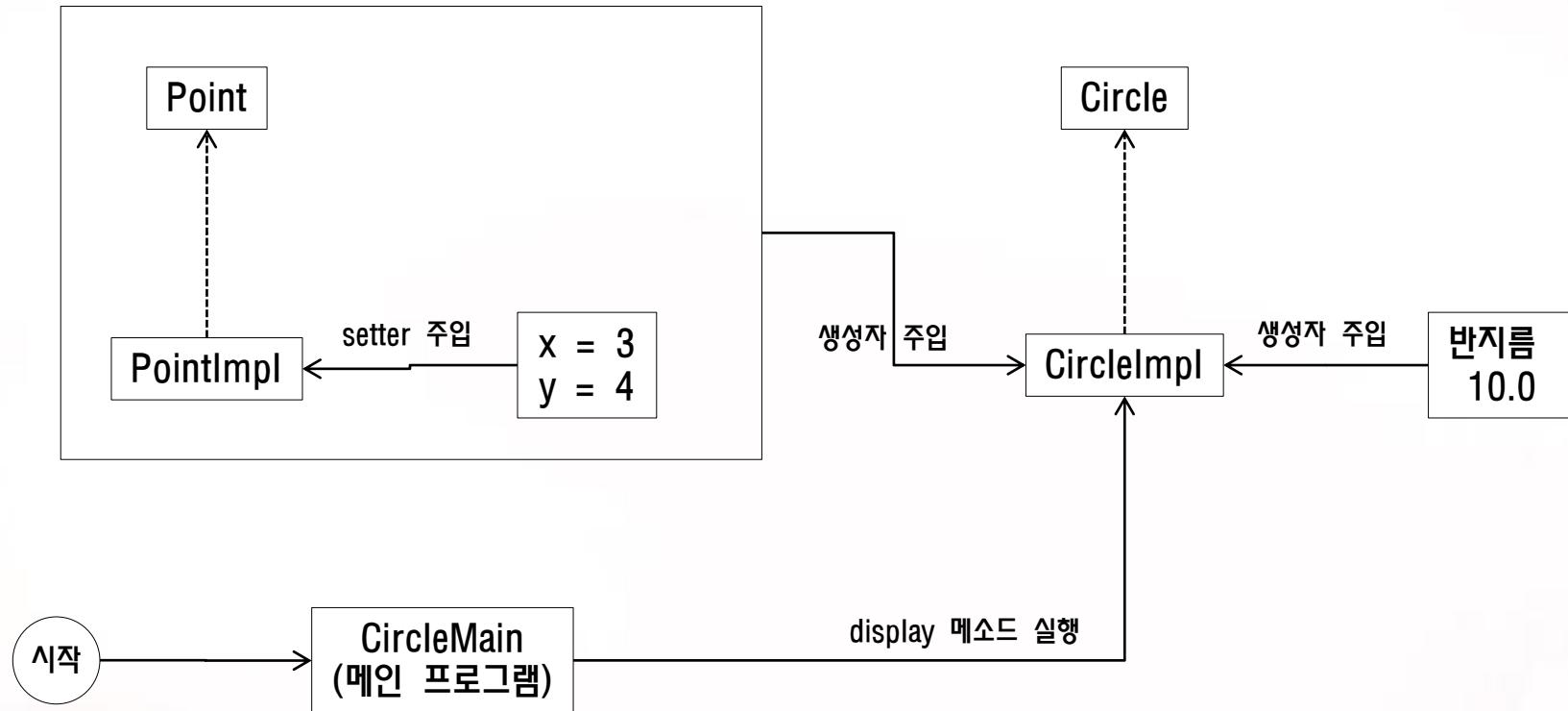
- 프로젝트 : CircleExam\sample3 패키지
- 스프링 설정 파일
  - Bean을 생성하고 관리한다.
  - xml 파일(조립기)를 이용한 의존 관계를 주입한다.
  - 기본 파일 명은 applicationContext.xml이다.

자바와 jsp와 스프링 간의 용어 비교			
	자바	JSP	스프링
용어	객체	JavaBean	bean
객체 생성	인터페이스 객체 = new 클래스();	<jsp:useBean /> 액션 태그	<bean> 태그
-	setter	<jsp:setProperty /> 액션 태그	<property> 태그
-	getter	<jsp:getProperty /> 액션 태그	<constructor-arg> 태그

- 일반적인 작성 절차
- 구현하고자하는 인터페이스 정의
- 추상 메소드(비즈니스 규칙)
- 클래스 구현
- 오버라이딩
- 생성자와 setter/getter 및 구현하고자 하는 메소드 정의
- 스프링 설정 파일 만들기
- xxx.xml
- 메인 메소드 구현

# 실습 : 원의 면적과 반지름

- 프로젝트 : CircleExam\sample3 패키지



# 실습 : 원의 면적과 반지름

- 프로젝트 : CircleExam\sample3 패키지

```
<bean id="point" class="sample3.PointImpl">
    <property name="xpos">
        <value type="double">3.0</value>
    </property>
    <property name="ypos">
        <value type="double">4.0</value>
    </property>
</bean>
```

Xml 파일

```
<bean id="circle" class="sample3.CircleImpl">
    <constructor-arg>
        <value type="double">10.0</value>
    </constructor-arg>
    <constructor-arg>
        <ref bean="point" />
    </constructor-arg>
</bean>
```

```
public class CircleMain {
    public static void main(String[] args) {
        final double radius = 10.0 ;
        Point point = new PointImpl() ;
        //setter을 사용한 데이터 넣기.
        point.setXpos(3.0) ;
        point.setYpos(4.0) ;
        Circle circle = new CircleImpl( radius, point ) ;
        circle.Display(); //원들의 정보를 출력한다.
```

```
public class CircleMain {
    public static void main(String[] args) {
        //applicationContext.xml 파일은 /src/ 폴더 아래에 둘 것
        Resource resource = new ClassPathResource("applicationContext.xml");
        BeanFactory factory = new XmlBeanFactory(resource);
        Circle circle = (Circle)factory.getBean("circle");
        circle.Display();
    }
}
```

Spring DI를 사용한 예시

# 실습 : 사각형의 너비와 대각선

- 프로젝트 : RectangleExam\myPkg 패키지
- 해결 과제
  - 사각형의 밑변(100)과 높이(40)를 생성자 주입시켜 다음과 같이 처리하세요.

The diagram illustrates a Java application structure. On the left, a Java class named `RectangleMain` is shown with its main method. A red diagonal line crosses out the entire class body. On the right, an XML configuration file (`applicationContext.xml`) defines a bean named `rectangleBean` with a class of `myPkg.RectangleImpl`. It has two constructor arguments: a double value of 10.0 and a double value of 4.0. Below the XML is a screenshot of the Eclipse IDE's Console view. The output shows the application starting up and printing the width and diagonal length of the rectangle.

```
public class RectangleMain {  
    public static void main(String[] args) {  
  
        Resource resource = new ClassPathResource("applicationContext.xml");  
        BeanFactory factory = new XmlBeanFactory(resource);  
  
        Rectangle rectangle = (Rectangle)factory.getBean("rectangleBean");  
        rectangle.Print();  
  
        rectangle.Diagonal();  
    }  
}  
  
<beans ... >  
  <bean id="rectangleBean" class="myPkg.RectangleImpl">  
    <constructor-arg>  
      <value type="double">10.0</value>  
    </constructor-arg>  
  
    <constructor-arg>  
      <value type="double">4.0</value>  
    </constructor-arg>  
  </bean>  
</beans>
```

Markers Properties Servers Data Source Explorer Snippets Console  
<terminated> RectangleMain [Java Application] C:\Program Files (x86)\Java\jdk1.7.0\_03\bin\javaw.exe (2012. 10. 11. 오후 3:55:17)  
10월 11, 2012 3:55:17 오후 org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions  
정보: Loading XML bean definitions from class path resource [applicationContext.xml]  
사각형의 넓이 : 40.0  
사각형의 대각선 길이 : 10.770329614269007

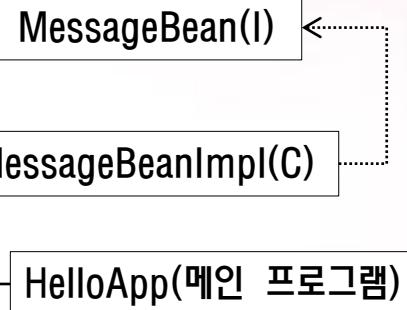
# 실습 : DI 예제

- 프로젝트 : DIExam\sample1 패키지
- Constructor(생성자) Injection과 Setter Injection을 테스트해보는 예시이다.

```
public class HelloApp {  
    public static void main(String[] args) {  
        Resource resource = new ClassPathResource("applicationContext.xml");  
        BeanFactory factory = new XmlBeanFactory(resource);  
        MessageBean bean = (MessageBean)factory.getBean("messageBean");  
        bean.sayHello();  
  
        //다음과 같이 [빈]을 만들어 보시오.  
        CalcBean calcBean = (CalcBean)factory.getBean("calcBean");  
  
        //calculate() : 정수 2개의 사칙 연산  
        //정수 1은 setter 인젝션, 정수 2는 생성자 인젝션으로 처리하시오  
        calcBean.calculate();  
    }  
}
```

```
7월 30, 2014 8:18:01 오후 org.springframework.context.support.DefaultLifecycleProcessor.preInstantiateSingletons  
정보: Loading XML bean definitions from URL [file:/D:/JavaApplication1/src/HelloApp.java]  
안녕하세요 흥길동씨^^  
--- 가끔 승제 실습 ---  
덧셈 : 19  
뺄셈 : 9  
곱셈 : 70  
나눗셈 : 2  
나머지 : 4
```

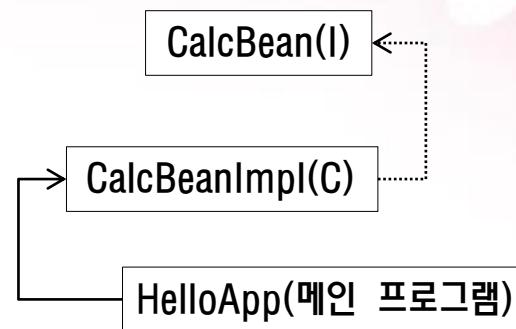
```
<bean id="messageBean" class="sample1.MessageBeanImpl">  
    <constructor-arg>  
        <value>흥길동</value>  
    </constructor-arg>  
    <property name="greeting">  
        <value>안녕하세요</value>  
    </property>  
</bean>  
  
<bean id="calcBean" class="sample1.CalcBeanImpl">  
    <property name="su1">  
        <value type="int">14</value>  
    </property>  
    <constructor-arg>  
        <value type="int">5</value>  
    </constructor-arg>  
</bean>
```



# 실습 : DI 예제

- 프로젝트 : DIExam\sample1 패키지

```
public class MessageBeanImpl implements MessageBean {  
    private String name; // 생성자 주입  
    private String greeting; // setter 주입  
  
    @Override  
    public void sayHello() {  
        System.out.println(greeting + " " + name + "^^");  
    }  
    //스프링 설정 파일에서 <constructor-arg> 태그 사용  
    public MessageBeanImpl(String name) {  
        super();  
        this.name = name;  
    }  
  
    public String getGreeting() {  
        return greeting;  
    }  
    //스프링 설정 파일에서 <property name="greeting"> 태그 사용  
    public void setGreeting(String greeting) {  
        this.greeting = greeting;  
    }  
}
```



```
public class CalcBeanImpl implements CalcBean {  
    private int su1;  
    private int su2;  
  
    public CalcBeanImpl(int su2) {  
        this.su2 = su2;  
    }  
    public void setSu1(int su1) {  
        this.su1 = su1;  
    }  
    @Override  
    public void calculate() {  
        System.out.println("--- 가감 승제 실습 ---");  
        System.out.println("덧셈 : " + (su1 + su2));  
        System.out.println("뺄셈 : " + (su1 - su2));  
        System.out.println("곱셈 : " + (su1 * su2));  
        System.out.println("나눗셈 : " + (su1 / su2));  
        System.out.println("나머지 : " + (su1 % su2));  
    }  
}
```

# 실습 : xml을 이용한 Bean 묶기

- 프로젝트 : XmlBeanBind
- 주문 배송 시스템을 구현한다고 가정한다.
- 이때 xml 파일을 이용하여 Bean들 간의 관계를 묶기 위한 예시이다.

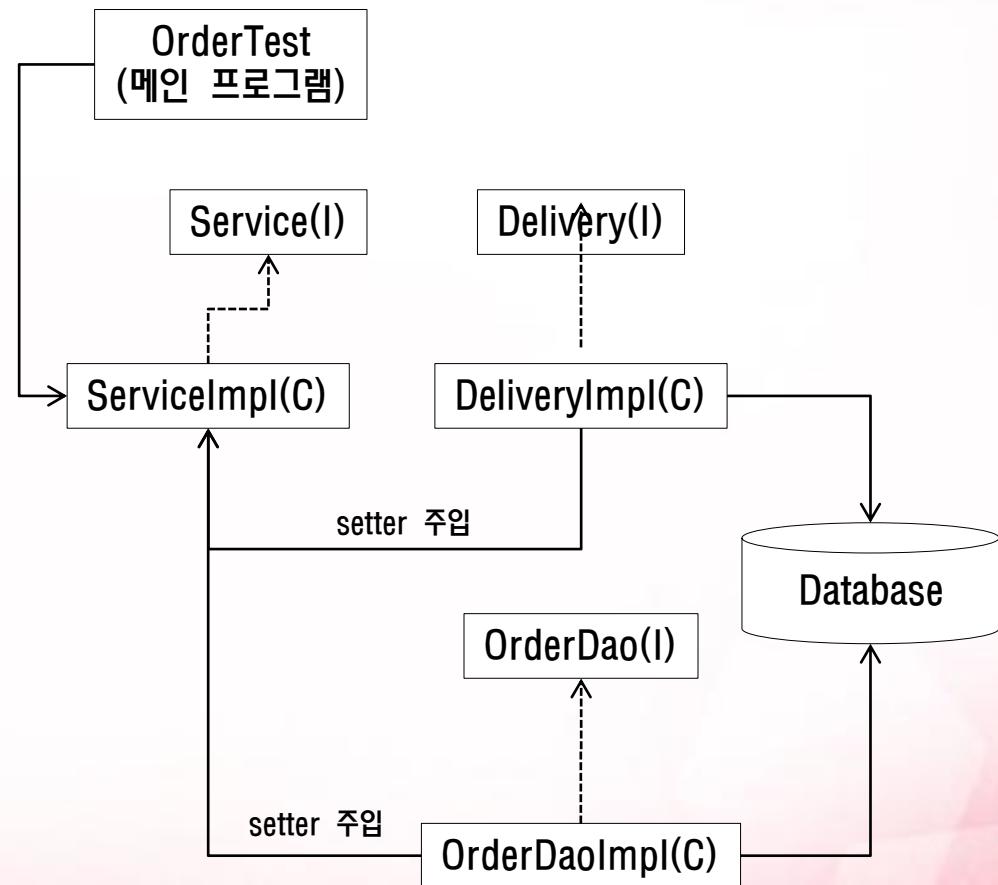
항목	처리할 내용
주문 하기	[배송지 주소 추가], [주문 정보 추가]
주문 취소	[배송지 주소 삭제], [주문 정보 삭제]

실행 결과
배송지 주소 추가 : insertAddress() 메소드
주문 정보 추가 : insertOrder() 메소드
배송지 주소 삭제 : removeAddress() 메소드
주문 정보 삭제 : removeOrder() 메소드

```
<bean id="orderDao" class="mypkg.OrderDaoImpl"/>
<bean id="deliveryDao" class="mypkg.DeliveryDaoImpl"/>

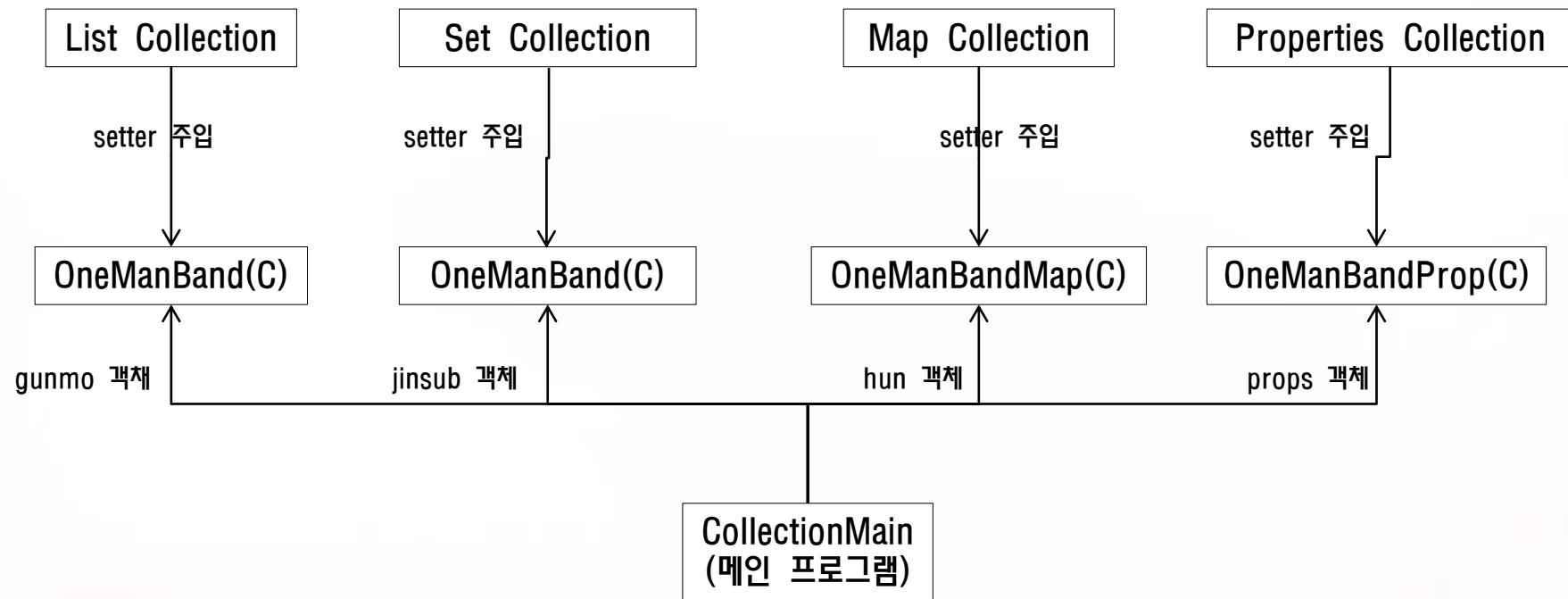
<bean id="service" class="mypkg.ServiceImpl">
    <property name="deliveryDao">
        <ref bean="deliveryDao" />
    </property>

    <property name="orderDao">
        <ref bean="orderDao" />
    </property>
</bean>
```



# 실습 : 컬렉션 다루기

- 프로젝트 : DICollection
- 자바의 컬렉션 객체들을 다루는 예시이다.



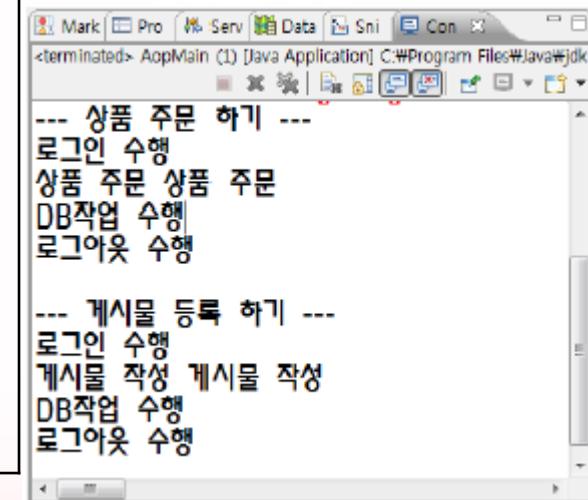
# Aop 이해하기

- 프로젝트 : AopExam

내용	설명	비고
핵심 관심 사항	상품 주문하기 , 게시물 등록	1 개의 비즈니스 로직에 해당한다. 이 예시에서는 1번째 비즈니스 로직은 [상품 주문하기]이고, 2번째 비즈니스 로직은 [게시물 등록]이다.
공통 관심 사항	로그인, DB 작업 수행, 로그 아웃	핵심 관심 사항에 공통적으로 적용될 업무

패키지	설명
sample1	핵심 기능과 공통 기능을 각각의 파일에 명시했다 단위 업무 별로 로직을 수행한다. 로그인 → 상품 주문 → 로그 아웃 로그인 → 게시물 등록 → 로그 아웃
sample2	핵심 기능과 공통 기능을 분리하여 각각의 파일에 명시했다 Order 파일과 Board 파일에서 각각 Login/Logout 파일을 이용했다 로그인과 로그 아웃은 반복이 되는 코드이므로 별도의 클래스로 구현하여 이를 호출한다.
sample3	xml을 이용한 AOP 구현 aop xml 스키마를 이용하여 구현했다.
sample4	어노테이션을 이용한 AOP 구현

구현
jar 파일 준비하기 비즈니스 규칙 Order(I), OrderImpl(C) Board(I), BoardImpl(C)
공통 업무(Aspect) Login(C) login 메소드 구현 Logout(C) logout 메소드 구현
메인 파일 AopMain.java
xml aopExam.xml



# Aop 이해하기

- 프로젝트 : AopExam sample1 패키지
- [상품 주문]과 [게시물 등록]의 공통점은 [로그인]과 [로그 아웃] 및 [DB 저장]이라는 업무를 수행 한다는 것이다.
- 그래서, sample2 패키지에서 공통 항목을 별도로 만들고 이를 호출하도록 바꿔 본다.

```
public class AopMain {  
    public static void main(String[] args) {  
        //상품 주문  
        Order myorder = new Order();  
        myorder.order();  
  
        //게시물 등록  
        Board myboard = new Board();  
        myboard.board();  
    }  
}
```

```
public class Board {  
    public void board() {  
        //로그인 수행, 게시물 등록, DB에 저장, 로그 아웃  
        String msg = "게시물 등록";  
  
        System.out.println( msg + "을 위한 로그인 수행" );//공통 기능(Aspect)  
        System.out.println( msg + " 하기" ); //핵심 기능  
        System.out.println( msg + " 을 DB에 저장" );//공통 기능(Aspect)  
        System.out.println( msg + "을 위한 로그 아웃" );//공통 기능(Aspect)  
    }  
}
```

```
public class Order {  
    public void order() {  
        //로그인 수행, 상품 주문, DB에 저장, 로그 아웃  
        String msg = "상품 주문";  
  
        System.out.println( msg + "을 위한 로그인 수행" );//공통 기능(Aspect)  
        System.out.println( msg + " 하기" ); //핵심 기능  
        System.out.println( msg + " 내역을 DB에 저장" );//공통 기능(Aspect)  
        System.out.println( msg + "을 위한 로그 아웃" );//공통 기능(Aspect)  
    }  
}
```

# Aop 이해하기

- 프로젝트 : AopExam sample2 패키지
- 코드가 별도로 분리되었지만, 여전히 호출하는 코드는 명시되어 있다.
- 이것을 sample3 패키지에서 Aop 개념을 이용하여 다시 작성해보도록 한다.

```
public class Order {  
    public void order() {  
        //로그인 수행, 상품 주문, DB에 저장, 로그 아웃  
        String msg = "상품 주문" ;  
  
        Login.login(msg) ;  
        System.out.println( msg + "상품 주문" ); //핵심 기능  
        Logout.logout(msg) ;  
    }  
}
```

```
public class Login {  
    public static void login( String msg ){  
        //공통 기능을 별도의 클래스로 만들었다.  
        System.out.println( msg + "로그인 수행" );  
    }  
}
```

```
public class Logout {  
    public static void logout(String msg){  
        //공통 기능을 별도의 클래스로 만들었다.  
        System.out.println( msg + "로그아웃 수행" );  
    }  
}
```

# Aop 이해하기

- 프로젝트 : AopExam sample3 패키지

```

<bean id="myorder" class="sample3.OrderImpl" />
<bean id="myboard" class="sample3.BoardImpl" />

<bean id="loginAdvice" class="sample3.Login" />
<bean id="logoutAdvice" class="sample3.Logout" />
<bean id="daoAdvice" class="sample3.DAO" />

<!-- sample3 패키지 내의 OrderImpl의 모든 메소드가
    수행되기 전에 login의 logging 기능이 수행되도록 설정 -->
<aop:config>
    <aop:aspect ref="loginAdvice">
        <aop:before method="login"
            pointcut="execution(* sample3.OrderImpl.*())" />
    </aop:aspect>
</aop:config>

<!-- Login : 이 클래스는 Aspect이기 때문에
    <aop:>으로 코딩하는 방식을
    aop 스키마 방식이라고 한다.
    스키마는 xml 용어이다. -->

```

```

//Login : 이 클래스는 Aspect이기 때문에
public class Login {
    public void login(){
        //공통 기능을 별도의 클래스로 만들었다.
        System.out.println( "로그인 수행" );
    }
}

```

```

public class OrderImpl implements Order {
    @Override
    public void order() {
        String msg = "상품 주문" ;
        System.out.println( msg + "상품 주문" ); //핵심 기능
    }
}

```

--- 상품 주문 하기 ---  
 로그인 수행  
 상품 주문 상품 주문  
 DB 작업 수행함  
 로그아웃 수행

--- 게시물 등록 하기 ---  
 로그인 수행  
 게시물 등록 게시물 등록  
 로그아웃 수행

```

OpMain {
    static void main(String[] args) {
        ApplicationContext context =
            new FileSystemXmlApplicationContext("src/sample3/aopExam.xml");
    }
}

```

```

System.out.println("--- 상품 주문 하기 ---");
Order myorder = (Order)context.getBean("myorder");
myorder.order();

System.out.println();
System.out.println("--- 게시물 등록 하기 ---");
Board myboard = (Board)context.getBean("myboard") ;
myboard.board();
}
}

```

# Aop 이해하기

- 프로젝트 : AopExam sample4 패키지

<aop:aspectj-autoproxy> 태그는 @Aspect 어노테이션이 적용된 클래스를 로딩한다.  
해당 클래스에 명시된 Advice 및 Pointcut 정보를 이용하여 알맞은 빈 객체에 Advice를 적용한다.

```
<aop:aspectj-autoproxy/>
```

<!-- 핵심 사항(비즈니스 규칙) -->

```
<bean id="myorder" class="sample4.OrderImpl"/>
<bean id="myboard" class="sample4.BoardImpl"/>
```

<!-- 공통 업무(Aspect) -->

```
<bean id="loginAdvice" class="sample4.Login"/>
<bean id="logoutAdvice" class="sample4.Logout"/>
<bean id="daoAdvice" class="sample4.DAO"/>
```

```
public class OrderImpl implements Order {
    @Override
    public void order() {
        String msg = "상품 주문 ";
        System.out.println( msg + "상품 주문" );
    }
}
```

@Aspect

```
public class Login {
    @Before ("execution(public void sample4.Board*.*) || execution(public void sample4.Order*.*)")
    public void login(){ // 공동기능을 별도의 클래스로 만들었다
        System.out.println("로그인 수행");
    }
}
```

--- 상품 주문 하기 ---

로그인 수행  
상품 주문 상품 주문  
DB 작업 수행함  
로그아웃 수행

--- 게시글 등록 하기 ---

로그인 수행  
게시글 등록 게시글 등록  
로그아웃 수행

```
public class AopMain {
    public static void main(String[] args) {
        ApplicationContext context
            = new FileSystemXmlApplicationContext("src/sample3/aopExam.xml");
```

```
System.out.println("--- 상품 주문 하기 ---");
Order myorder = (Order)context.getBean("myorder");
myorder.order();

System.out.println();
System.out.println("--- 게시글 등록 하기 ---");
Board myboard = (Board)context.getBean("myboard");
myboard.board();
```

# 실습 : AopSchema

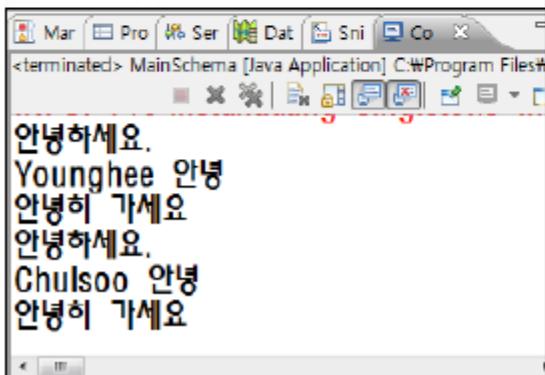
- 해결 과제
- aopSchema 패키지를 작성한다.
- 해당 패키지 내에 XML 스키마를 이용하여 다음과 같이 출력되는 프로그램 작성하시오.

- 추가 문제
- aopAnnotation 패키지 내에 Annotation을 이용하여
- 동일한 결과가 출력되도록
- 코딩하시오.

```
public static void main(String[] args) {  
    ApplicationContext context  
        = new ClassPathXmlApplicationContext("aopSchema/aopSchemaContext.xml");  
  
    Human younghee = (Human)context.getBean("younghee");  
    younghee.say();  
  
    Human chulsoo = (Human)context.getBean("chulsoo");  
    chulsoo.say();  
}
```

메인 메소드

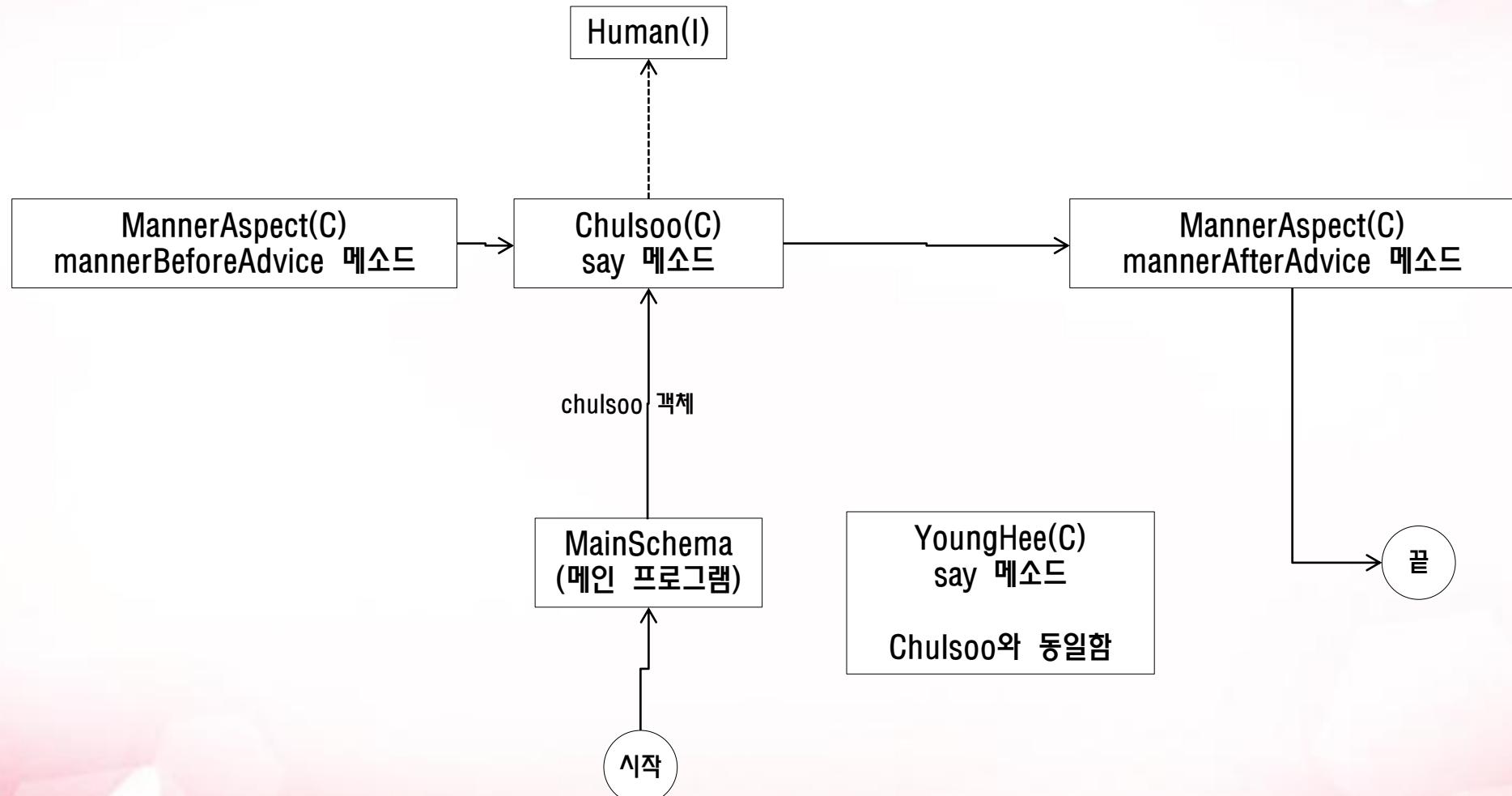
xml 파일



```
<bean id="younghee" class="aopSchema.Younghee">  
<bean id="chulsoo" class="aopSchema.Chulsoo">  
  
<bean id="mannerAspect" class="aopSchema.MannerAspect">  
  
<aop:config>  
    <aop:pointcut id="sayPointcut" expression="execution(* say(..))" />  
    <aop:aspect ref="mannerAspect">  
        <aop:before method="mannerBeforeAdvice" pointcut-ref="sayPointcut">  
        <aop:after-returning method="mannerAfterAdvice" pointcut-ref="sayPointcut">  
    </aop:aspect>  
</aop:config>
```

# 실습 : AopSchema

- 프로젝트 : AopSchema



# 실습 : AnnotationAOP

- annotation을 활용한 AOP 설정하기
- 스프링 설정 파일에 <aop:aspectj-autoproxy/>을 선언해야 한다.
- @Aspect :
  - 설정 파일에 설정하지 않고, Advice를 적용할 수 있게 하는 어노테이션
- @Before :
  - 해당 메소드가 실행되기 전에 무조건 수행하겠다
    - Before Advice로 사용할 메소드에 추가하는 어노테이션
    - annotation 패키지 하위에 BizService으로 시작하는 모든 메소드에 적용
    - @Before("execution(public void annotation.BizService.\*.\*())")
- Main 파일
  - AnnotationTest.java

## 실행 결과

dancing를 실행하기 전에 제가 먼저 실행되는 군요  
현재 시각 : Tue Jul 29 11:08:15 KST 2014  
춤을 춥니다.

====AnnotationTest 메인 메소드 호출=====

singing를 실행하기 전에 제가 먼저 실행되는 군요  
현재 시각 : Tue Jul 29 11:08:15 KST 2014  
노래를 부릅니다.

# 실습 : AnnotationAOP

```
public interface BizService {  
    public void dancing();  
    public void singing();  
}
```

```
public class BizServiceImpl implements BizService {  
    @Override //이 메소드가 실행되기 전에 Before Advice가 먼저 실행된다.  
    public void dancing() {  
        System.out.println("춤을 춥니다.");  
    }  
    @Override//method10 메소드와 동일  
    public void singing() {  
        System.out.println("노래를 부릅니다.");  
    }  
}
```

```
<!-- annotation을 활용한 AOP 설정을 위하여 반드시 다음을 선언 -->  
<!-- 모든 설정은 어노테이션을 이용하여 처리한다. -->  
<aop:aspect>-autoproxy/>  
  
<!-- 빈 생성하기 -->  
<bean id="bizService" class="annotation.BizServiceImpl"></bean>  
  
<!-- Advice 빈 생성하기 -->  
<bean id="advice" class="annotation.MyAspect"></bean>
```

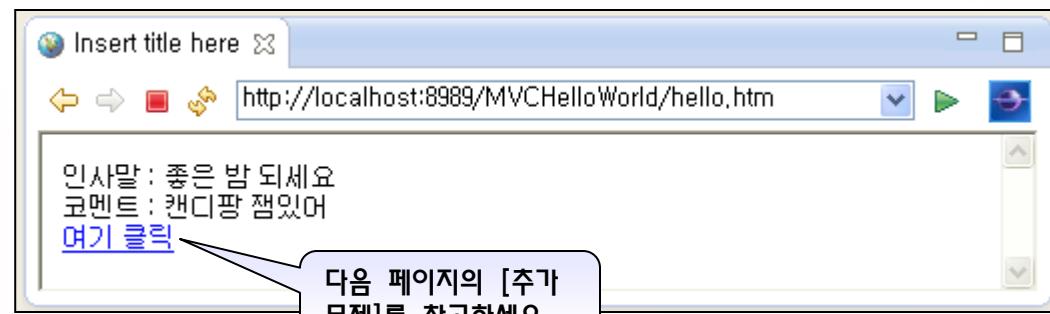
//Advice 파일 : 공통 기능의 적용 시점을 정의해 둔 파일  
//@Aspect : 설정 파일에 설정하지 않고, Advice를 적용할 수 있게 하는 어노테이션  
@Aspect  
public class MyAspect {  
 //@Before : Before Advice로 사용할 메소드에 추가하는 어노테이션  
 @Before("execution(public void annotation.BizService.\*.\*())")  
 public void beforeMethod(JoinPoint jp){  
 //Signature 인터페이스 : 호출되는 메소드와 관련된 모든 설정 정보를 담고 있다.  
 //getSignature() 메소드 : 호출되는 메소드에 대한 정보 구하기  
 Signature signature = jp.getSignature();  
 //getName() : 메소드의 이름 구하기  
 String methodName = signature.getName();  
  
 Date date = new Date();  
 System.out.println( methodName + "를 실행하기 전에 제가 먼저 실행되는 군요");  
 System.out.println("현재 시각 : " + date);  
 }  
}

```
public class AnnotationTest { //메인 파일  
    public static void main(String[] args) {  
        ApplicationContext context  
            = new FileSystemXmlApplicationContext("src/annotation/annotation.xml");  
  
        BizService biz = (BizService)context.getBean("bizService");  
  
        biz.dancing();  
        System.out.println( "====AnnotationTest 메인 메소드 호출====");  
        biz.singing();  
    }  
}
```

# 실습 : MVCHelloWorld

- 작업 순서

- web.xml 파일에 DispatcherServlet을 설정한다.(확장자 : htm)
- BeanNameUrlHandlerMapping을 설정한다.
- 컨트롤러 구현 : AbstractController를 상속 받는다
  - ModelAndView 설정하기
- 설정 파일에 컨트롤러 빈 생성하기
- ViewResolver 설정하기
- View 파일 생성하기



MVC 주요 구성 요소	설명
DispatcherServlet	사용자의 요청을 최초로 접수 받는 서블릿 클래스
HandlerMapping	요청 내용을 해석하여 어떤 Controller가 수행할지를 지정해주는 자바 클래스
Controller	비즈니스 규칙을 적절히 처리해주는 역할을 수행한다. Controller는 필요에 의하여 ModelAndView 객체를 만들어서 리턴해준다. ModelAndView : Model(어떤 정보 / 데이터) And View(어떤 UI/ 화면)
ViewResolver	어떤 뷰가 실행될 것인지를 해석(resolve)해주는 뷰-해석기

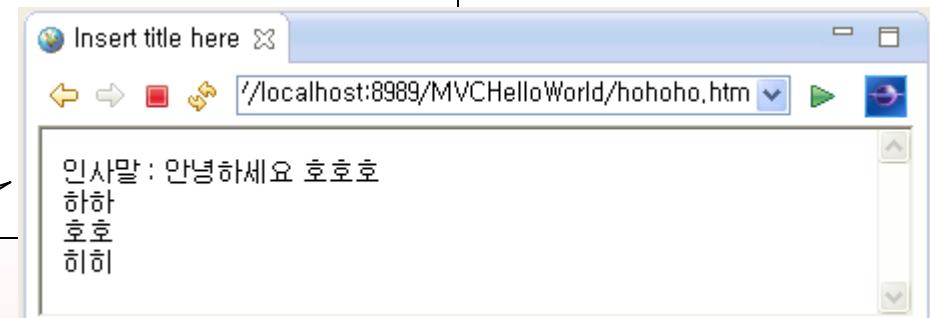
# 실습 : MVCHelloWorld

- 추가 문제

- 하단의 예제 클릭(hohoho.htm 링크)을 이용하여 다음 문제를 풀어 보세요.
- HohohoController 파일을 작성한다.
- hohoho.jsp 파일을 작성한다.

```
public class HohohoController extends AbstractController {  
    @Override  
    protected ModelAndView handleRequestInternal(HttpServletRequest arg0,  
        HttpServletResponse arg1) throws Exception {  
        ModelAndView mav = new ModelAndView("hohoho");  
  
        mav.addObject("hohoho", "안녕하세요 호호호");  
  
        //ModelAndView mav = new ModelAndView("hohoho", "hohoho", "안녕하세요 호호호");  
  
        List<String> lists = new ArrayList<String>();  
  
        lists.add("하하");  
        lists.add("호호");  
        lists.add("히히");  
        mav.addObject("lists", lists);  
  
        return mav;  
    }  
}
```

el과 jstl을 이용하여  
요소들을 출력한다.



# 실습 : MVCHelloWorldNew

- 요구 사항

- Annotation을 사용하여 프로젝트 MVCHelloWorld를 다시 작성해 보세요.

The screenshot shows a browser window on the left and a code editor window on the right. The browser window displays the URL `http://localhost:8989/MVCHelloWorldNew/hello.htm` and the content "인사말 : 안녕하세요" and "메시지 : 캔디팡 짬있어". The code editor window contains the Java code for the `HelloController`.

```
@Controller // [ 컨트롤러 클래스를 사용하겠다 ]고 선언
public class HelloController{
    //사용자가 /hello.htm라고 입력 요청을 하게 되면 hello.htm로 처리된다.
    @RequestMapping("/hello.htm")
    protected ModelAndView hello() {
        ModelAndView mav = new ModelAndView();
        //setViewName : 결과를 보여주기 위해 View를 설정
        mav.setViewName("hello");

        mav.addObject("greeting", getGreeting());
        mav.addObject("msg", "캔디팡 짬있어");

        return mav;
    }
}

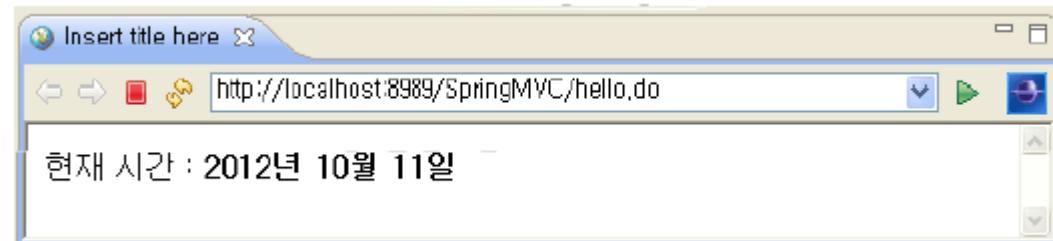
private String getGreeting() {
    int hour = Calendar.getInstance().get(Calendar.HOUR_OF_DAY);

    if(hour >= 6 && hour <= 10) {
        return "좋은 아침입니다.";
    } else if(hour >= 12 && hour <= 15) {
        return "점심 식사는 하셨나요?";
    } else if(hour >= 18 && hour <= 22) {
        return "좋은 밤 되세요";
    }
    return "안녕하세요";
}
```

# 실습 : SpringMVC

- 작업 순서

- web.xml 파일에 DispatcherServlet을 설정 ( 확장자 : do)
- Annotation을 이용한 예시
  - @Controller // [ 컨트롤러 클래스를 사용하겠다 ] 고 선언
  - @RequestMapping("/hello.do")



- 주가 문제

- 위의 문제를 xml 파일을 사용하여 다시 작성해 보세요.
- 프로젝트 명 : SpringMvcAppend

MVC 주요 구성 요소	설명
DispatcherServlet	사용자의 요청을 최초로 접수 받는 서블릿 클래스
HandlerMapping	요청 내용을 해석하여 어떤 Controller가 수행할지를 지정해주는 자바 클래스
Controller	비즈니스 규칙을 적절히 처리해주는 역할을 수행한다. Controller는 필요에 의하여 ModelAndView 객체를 만들어서 리턴해준다. ModelAndView : Model(어떤 정보 / 데이터) And View(어떤 UI/ 화면)
ViewResolver	어떤 뷰가 실행될 것인지를 해석(resolve)해주는 뷰-해석기

# 실습 : AbstractCommandEx

- AbstractCommandController(폼 데이터 처리) 사용
- 커맨드 객체 :
  - 폼 내부의 파라미터들에 대한 정보를 저장하고 있는 객체
  - 폼 데이터를 받아서 커맨드 객체를 생성하고 개발자가 작성하는 요청 처리 메소드 (handle()) 의 파라미터로 커맨드 객체를 전달해 주는 기능을 가지고 있다.

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

web.xml

사용자가 최초 요청을 하게 되면 스프링은 DispatcherServlet이라는 서블릿이 이를 접수하게된다.  
확장자 htm 요청시 dispatcher라는 서블릿이 반응을 하게 되는데, 이때 서블릿이름-servlet.xml 파일(dispatcher-servlet.xml)을 찾게 된다.

```
<center>
<br><br>
<!--
    폼 필드의 이름은 커맨드 클래스의 속성명과 일치해야
    커맨드 클래스의 속성으로 할당 된다.
-->
<h3 align="center">로그인 하기</h3>
<form action="login.htm" method="post" >
    아이디 : <input type="text" name="id" value="hong"><br><br>
    비밀 번호 : <input type="password" name="password" value="1234"><br><br>
    <input type="submit" value="로그인">
</form>
</center>
```

loginForm.jsp

사용자가 submit 버튼을 클릭하게 되면  
login.htm을 요청하게 된다.



# 실습 : AbstractCommandEx

```
public class LoginUnit {  
    private String id;  
    private String password;  
  
    //getter, setter 생략  
  
    public LoginUnit() {}  
    public LoginUnit(String id, String password) {  
        super();  
        this.id = id;  
        this.password = password;  
    }  
}
```

ViewResolver는 접두사와 접미사와 viewName을 이용하여 조립한다.  
결국 최종적으로 도착하는 페이지는 /loginResult.jsp가 된다.

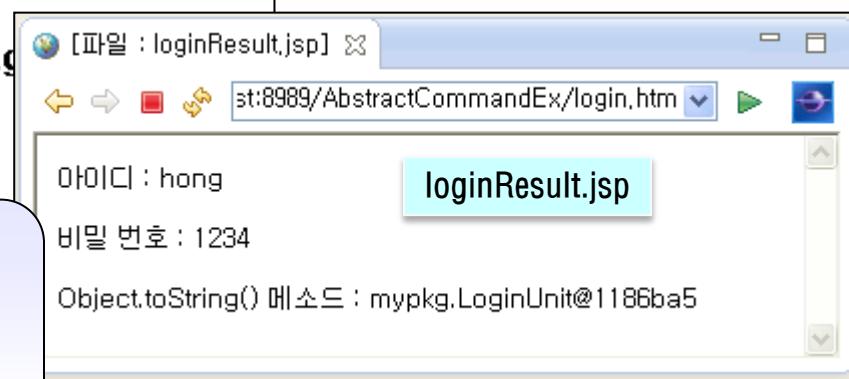
```
LoginUnit login = (LoginUnit)command;
System.out.println("id : " + login.getId() + ", pwd : " + login.getPassword());
ModelAndView mav = new ModelAndView();
mav.setViewName("loginResult"); //이동할 페이지 이름
mav.addObject("login", login);

return mav;
```

3번째 파라미터인 command의 객체에 해당 파라미터

- 3번째 파라미터인 command인 객체에 해당 파라미터들이 자동으로 들어오게 된다.  
이를 downCasting 시켜서 ModelAndView 객체에 login이라는 이름으로 Binding 시킨다.  
이때 request 내장 객체 영역에 저장된다.  
마지막으로 이동할 페이지(loginResult)도 설정한다.

```
<body>
    아이디 : ${login.id} <br><br>
    비밀 번호 : ${login.password} <br><br>
    Object.toString() 메소드 : ${login}
</body>
```



# 실습 : HumanExam

- 회원 가입을 있다고 가정한다.
- 정보를 command 객체를 이용하여 Bean 객체에 저장, jsp 페이지에서 보여주는 예시.

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
```

```
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>*.htm</url-pattern>
</servlet-mapping>
```

web.xml

사용자가 최초 요청을 하게 되면 스프링은 DispatcherServlet이라는 서블릿이 이를 접수하게 된다.

확장자 htm 요청시 dispatcher라는 서블릿이 반응을 하게 되는데, 이때 서블릿이름-servlet.xml 파일(dispatcher-servlet.xml)을 찾게 된다.

```
<form action="login.htm" method="post" >
  아이디 : <input type="text" name="id" value="hong"><br>
  이름 : <input type="text" name="name" value="홍길동"><br>
  취미 :
    <input type="radio" name="hobby" value="독서">독서
    <input type="radio" name="hobby" value="운동" checked="checked">운동
    <br>
  생일 : <input type="text" name="birth" value="70/08/28"><br>
  직업
    <select name="job">
      <option value="교수">교수
      <option value="학생" selected="selected">학생
      <option value="기타">기타
    </select>
    <br>
    <input type="submit" value="로그인">
</form>
```

start.jsp

사용자가 submit 버튼을 클릭하게 되면 login.htm을 요청하게 된다.

# 실습 : HumanExam

```

<bean id="beanNameUrlMapping"
      class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping" />

<bean name="/login.htm" class="myPkg.HumanController">
    <property name="commandName" value="human" />
    <property name="commandClass" value="myPkg.HumanBean" />
</bean>

<!-- ViewResolver -->
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewR
      <property name="prefix" value="/WEB-INF/view/"></property>
      <property name="suffix" value=".jsp"></property>
</bean>

```

dispatcher-servlet.xml

```

public class HumanBean {
    private String id ;
    private String name ;
    private String hobby ;
    private String birth ;
    private String job ;
}

```

//getter, setter 생략

넘겨진 파라미터들은 myPkg.HumanBean 클래스의 객체(human)을 이용하여 파라미터의 값을 설정한다.

```

public class HumanController extends AbstractCommandController{
    @Override
    protected ModelAndView handle(
        HttpServletRequest request,
        HttpServletResponse response,
        Object command,
        BindException be) throws Exception {
        HumanBean human = (HumanBean)command;
        ModelAndView mav = new ModelAndView();
        mav.setViewName("end");
        mav.addObject("human", human);
        return mav;
    }
}

```

```

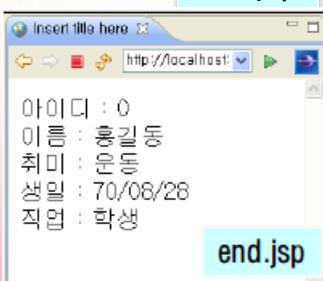
<body>
    아이디 : ${human.id} <br>
    이름 : ${human.name} <br>
    취미 : ${human.hobby} <br>
    생일 : ${human.birth} <br>
    직업 : ${human.job} <br>
</body>

```

end.jsp

ViewResolver는 접두사와 접미사와 viewName을 이용하여 조립한다.  
결국 최종적으로 도착한 페이지는 /WEB-INF/view/end.jsp가 된다.

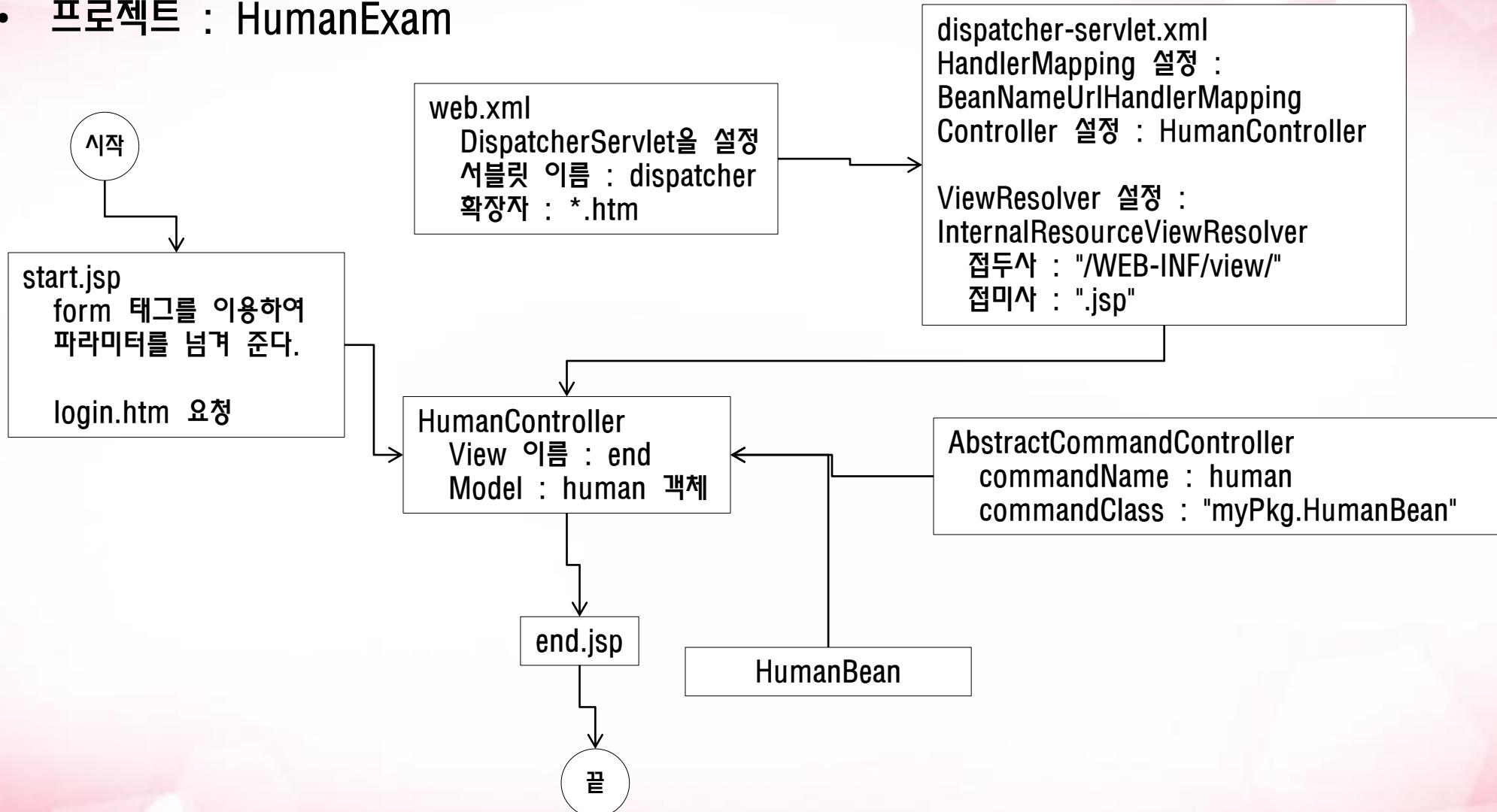
3번째 파라미터인 command인 객체에 해당 파라미터들이 자동으로 들어오게 된다.  
이를 downCasting 시켜서 ModelAndView 객체에 human이라는 이름으로 Binding 시킨다.  
이때 request 내장 객체 영역에 저장된다.  
마지막으로 이동할 페이지(end)도 설정한다.



end.jsp

# 실습 : HumanExam

- 프로젝트 : HumanExam



# 실습 : RegisterForm

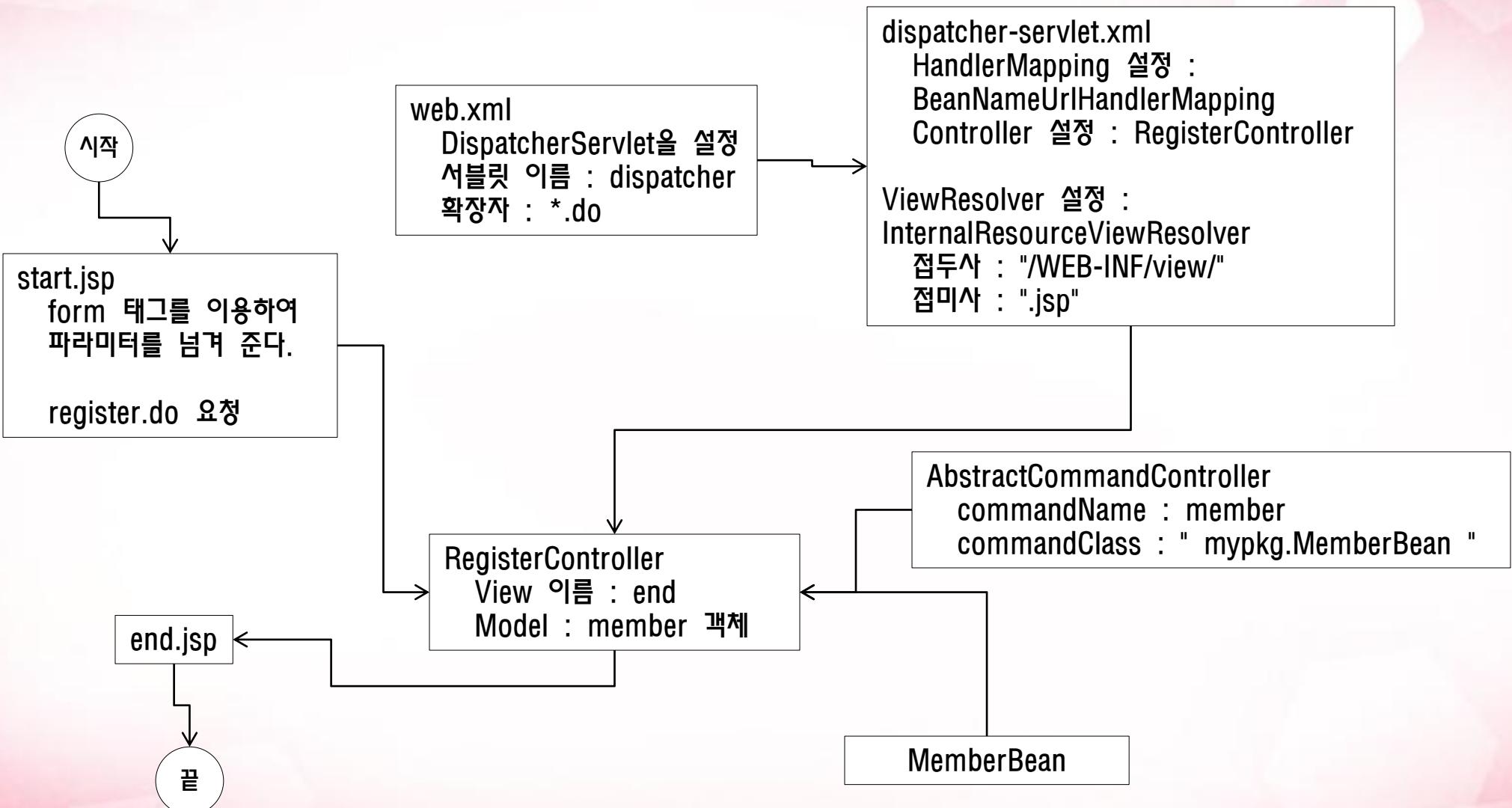
- <http://localhost:8989/RegisterForm/start.jsp>

The screenshot shows a Java development environment with the following components:

- Project Explorer:** Displays the project structure for "(RegisterForm)". It includes JAX-WS Web Services, Deployment Descriptor, Java Resources (src, mypkg), Libraries, JavaScript Resources, Deployed Resources, build, WebContent, META-INF, start.jsp, and WEB-INF (dispatcher-servlet.xml, lib, view). The lib folder contains commons-logging.jar, jstl.jar, servlet-api.jar, spring.jar, spring-webmvc.jar, and standard.jar.
- Browser Window 1:** Title: "로그인 품" (Login Form). URL: http://localhost:8989/RegisterForm/start.jsp. This window shows a registration form with fields: 아이디 (hong), 이름 (홍길동), 주소 (서울시 용산구 도원동), 세부 주소 (삼성 래미안), 선호하는 운영 체제 (winxp, win2000 checked), 직업 (교수), 개발툴 (이클립스, NetBeans), 기타 (잼있어요), and 약관 동의 (checked). Buttons at the bottom: 회원 가입 and 취소.
- Browser Window 2:** Title: "Insert title here" (Untitled). URL: http://localhost:8989/RegisterForm/register.do. This window shows the same registration form with identical values as the first window.
- Code View:** A code editor on the right side displays the MemberBean.java class. The code is annotated with Korean comments explaining its purpose.

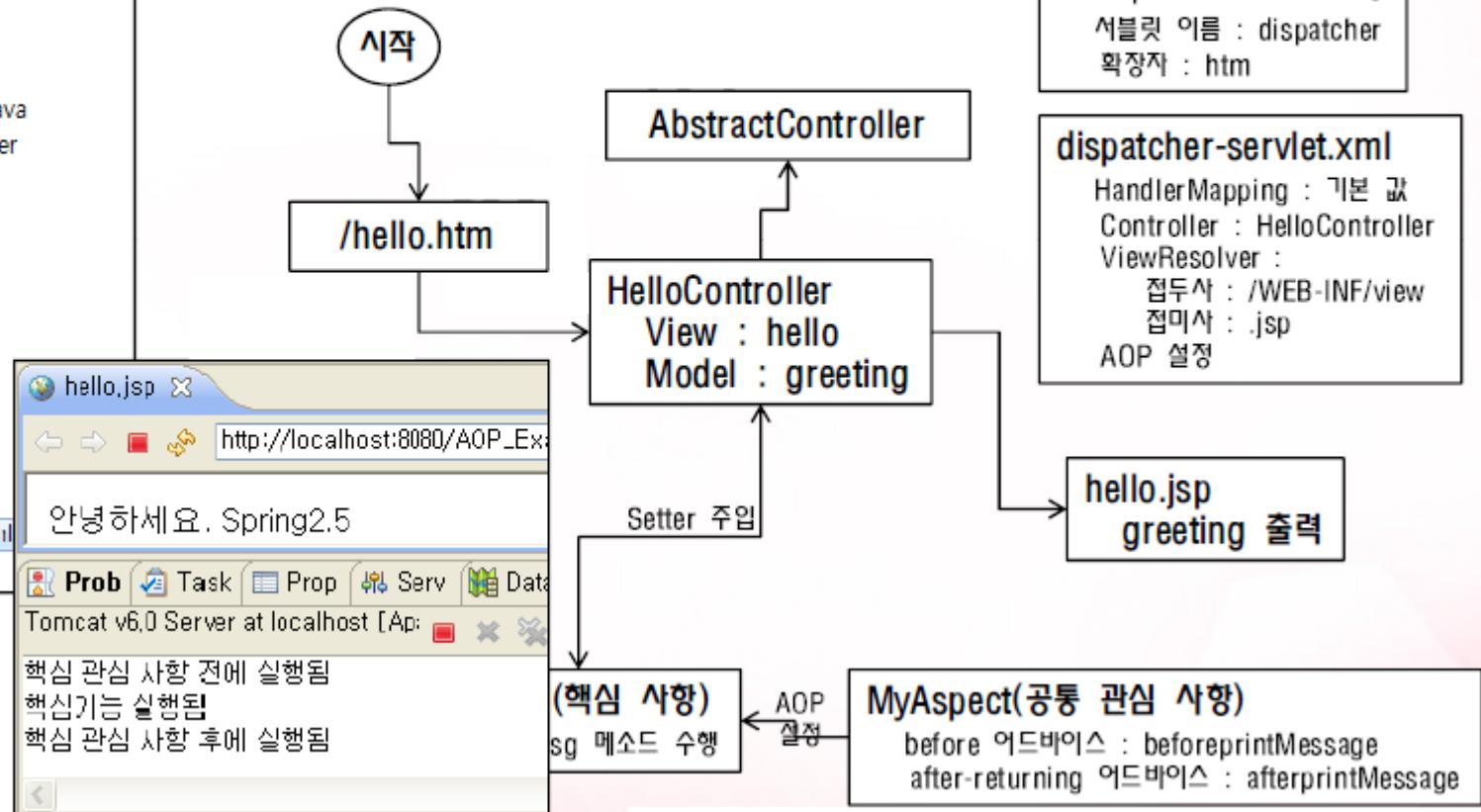
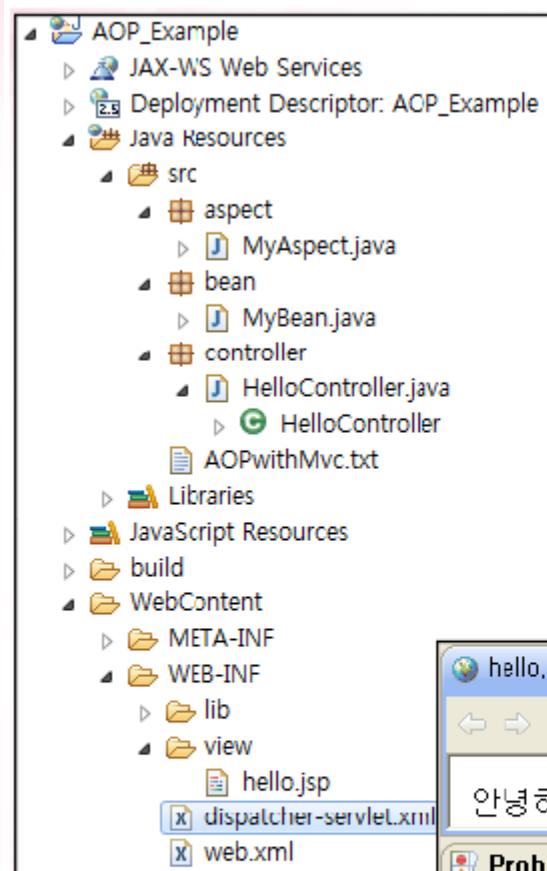
```
public class MemberBean {  
    //회원 정보를 담고 있는 빈 객체  
    private String id;  
    private String name;  
    private String address1;  
    private String address2;  
    private String[] favorites; //선호하는 운영체제  
    private String jobCode; //선호하는 직업군  
    private String tool; //주로 사용하는 개발툴  
    private String etc; //기타 사항  
    private boolean agreement; //약관 동의  
  
    //getter, setter 생략  
}
```

# 실습 : RegisterForm



# 실습 : AopExample

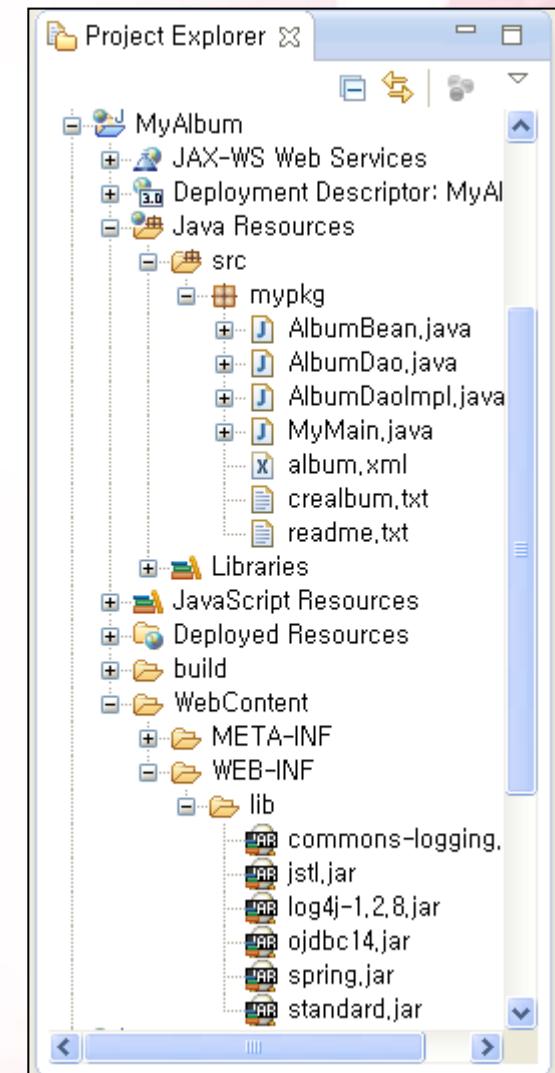
- 스프링 MVC와 AOP 기능을 이용한 예시
- 최초 실행
  - http://localhost:포트번호/AopExample/hello.htm 라고 입력한다.



# 실습 : 스프링을 이용한 JDBC 연동

- 프로젝트 : MyAlbum(readme.txt 파일 참고)
- 스프링을 이용한 JDBC 프로그램을 작성하기

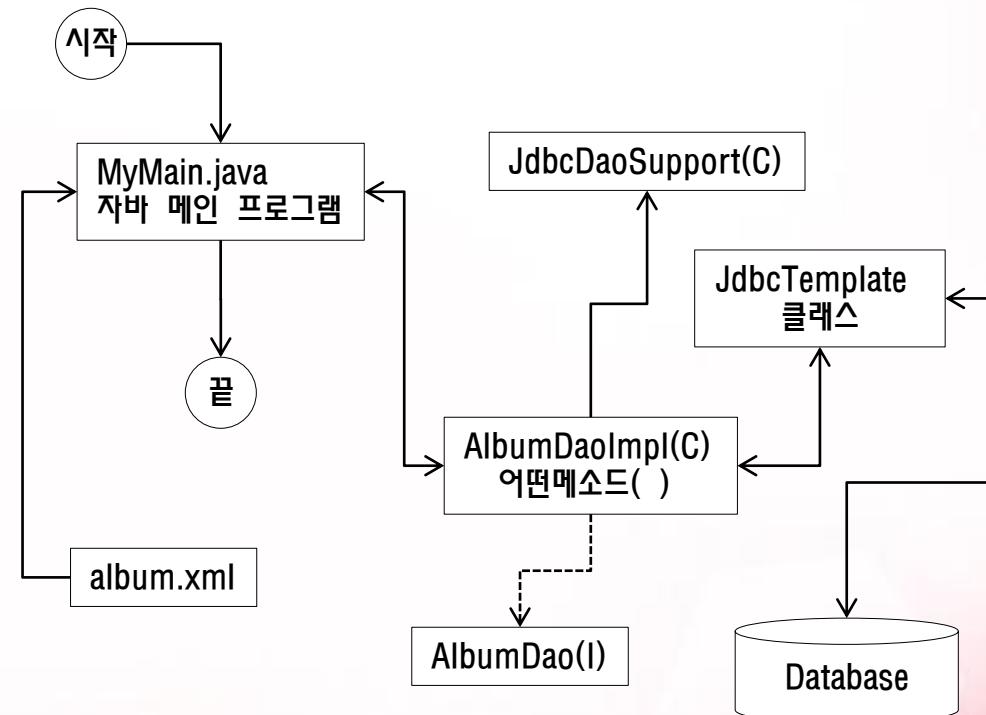
파일 리스트	설명
crealbum.txt	테이블 생성 스크립트
album.xml 파일	1) DataSource 설정 2) 사용자 정의 DAO를 위한 빈(Bean) 설정 dataSource 객체를 setter Injection 시킨다.
Bean 클래스 작성	Bean 클래스 파일 mypkg.AlbumBean.java 테이블 구조를 보고 bean 생성하길 바람
DAO 파일	Data Access Object 1) mypkg.AlbumDao.java(비즈니스 규칙용 인터페이스) 2) mypkg.AlbumDaolmpl.java(AlbumDao 구현 클래스)
메인 파일 작성	MyMain.java 파일 1) 각각의 로직에 필요한 기능을 구현한다.( switch ... case )



# 실습 : MyAlbum

- 비즈니스 규칙 정의(AlbumDao 인터페이스) 및 업무 흐름도

파일 리스트	설명
public List<AlbumBean> getAlbumList()	모든 앨범 리스트 보기
public int InsertData(AlbumBean album)	앨범 추가하기
public int updateData(AlbumBean album)	앨범 수정하기
public int deleteData(int id)	앨범 삭제하기
public AlbumBean getAlbumById(int id)	개별 앨범 정보 보기



# 실습 : MySangpum

- 프로젝트 : MySangpum
- 스프링을 이용하여 상품에 대한 기본 CRUD 작업을 테스트.



주요 파일 리스트	설명
item.sql	테이블 생성 스크립트
ItemBean.java	Bean 클래스 파일 테이블 구조를 보고 bean 생성하길 바람
ItemDao.java	Data Access Object(다오 인터페이스)
ItemDaoImpl.java	Data Access Object(다오 구현 클래스)
web.xml	배포 서술자 파일
applicationContext.xml (스프링 설정 파일)	dataSource 객체 및 ItemDao 객체 설정
mySangpum-servlet.xml (MVC 설정 파일)	MVC 설정 정보를 담고 있는 파일 HandlerMapping BeanNameUrlHandlerMapping 빈의 이름을 이용하여 Controller 설정 Controller ListController 컨트롤러 설정하기 ViewResolver : InternalResourceViewResolver 접두사 : "/WEB-INF/jsp/" 접미사 : ".jsp"
ListController	상품 목록을 구현하는 컨트롤러 파일
list.jsp	상품 목록을 보여주는 jsp 파일
shopping-1.css	스타일을 지정하는 스타일 시트

# 실습 : MySangpum

- 최초 시작 파일 : <http://localhost:포트번호/MySangpum/list.do>

시작

web.xml

- 1) ApplicationContext(스프링) 설정 파일  
contextConfigLocation 파라미터를 이용한다.  
Bootstrap listener 등록 : ContextLoaderListener
- 2) 문자열 인코딩 설정하기
- 3) DispatcherServlet 설정  
예제에서는 확장자 do 에 대한 맵핑을 수행한다.  
서블릿 이름 : mySangpum

Database

ApplicationContext 설정 파일 작성(applicationContext.xml)

- 1) DBCP 를 이용한 DataSource 설정
- 2) DAO 빈 설정  
dataSource 객체를 setter Injection 시킨다.

Dao 객체  
ItemDaolmpl

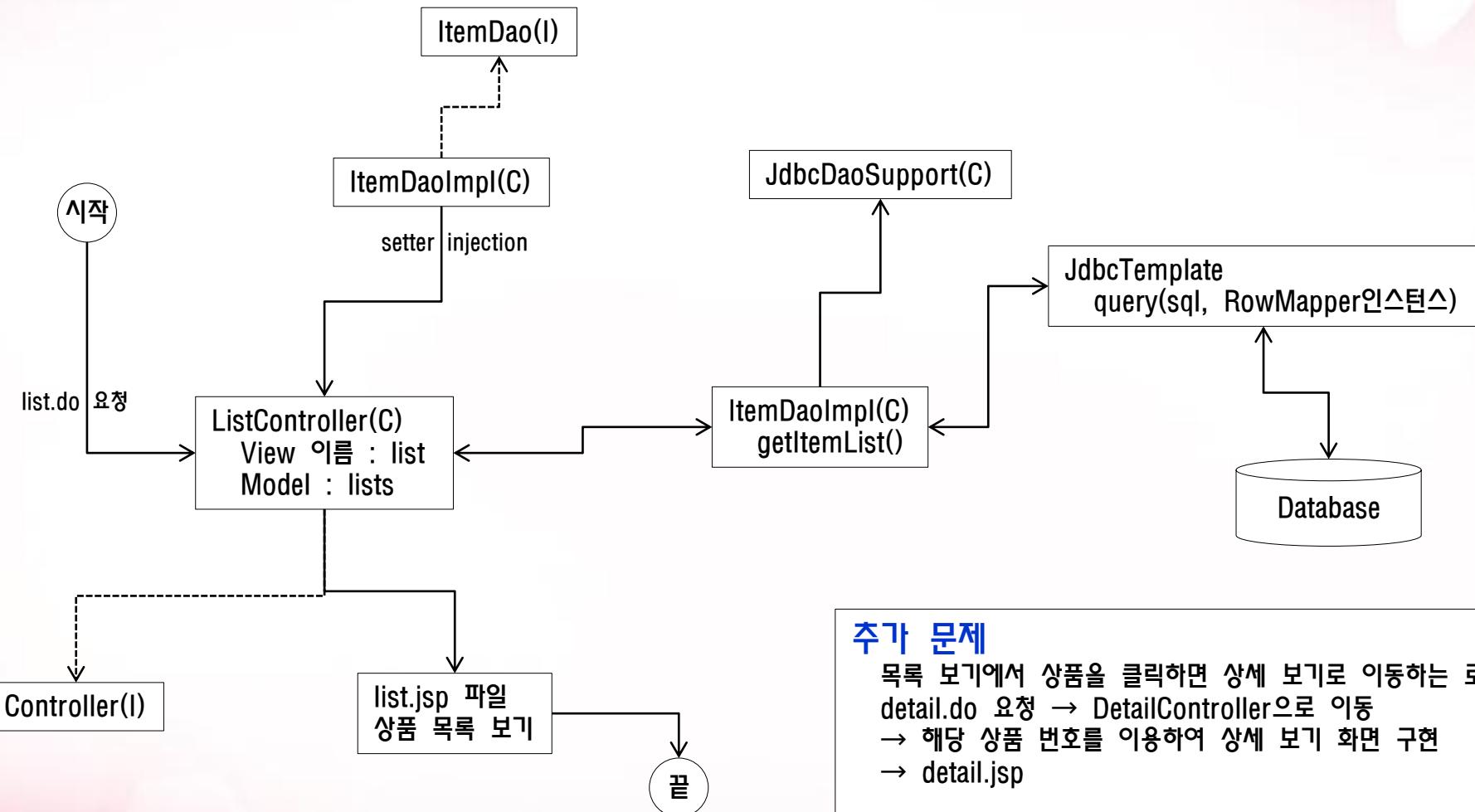
MVC용 설정 파일 작성(mySangpum-servlet.xml)

- 1) HandlerMapping 설정 : BeanNameUrlHandlerMapping
- 2) 컨트롤러 설정하기  
(1) ListController(목록 보기) 설정하기
- 3) ViewResolver 설정 : InternalResourceViewResolver  
(1) 접두사 : "/WEB-INF/jsp/"  
(2) 접미사 : ".jsp"

끝

# 실습 : MySangpum

## • 상품 목록 보기



# 실습 : MyBatisEx01

- 프로젝트 : MyBatisEx01
- MyBatis를 이용하여 회원들에 대한 추가/수정/삭제/조회 기능을 구현한다.
- [MyMain.java](#) 파일을 이용하여 자바 콘솔에서 시작한다.

주요 파일 리스트	설명
MyMain.java	최초 실행 파일(자바의 main 메소드가 존재하는 파일)
dbscript.txt	테이블 생성 스크립트
mypkg.MemberBean.java	회원에 대한 Bean 클래스 파일
mypkg.MemberDao.java	Data Access Object
/src/mybatis-config.xml	SQL 맵 설정 파일
/src/MyMapper.xml	맵핑 파일

## 최초 실행 예시

하시고자 하는 업무 번호 입력

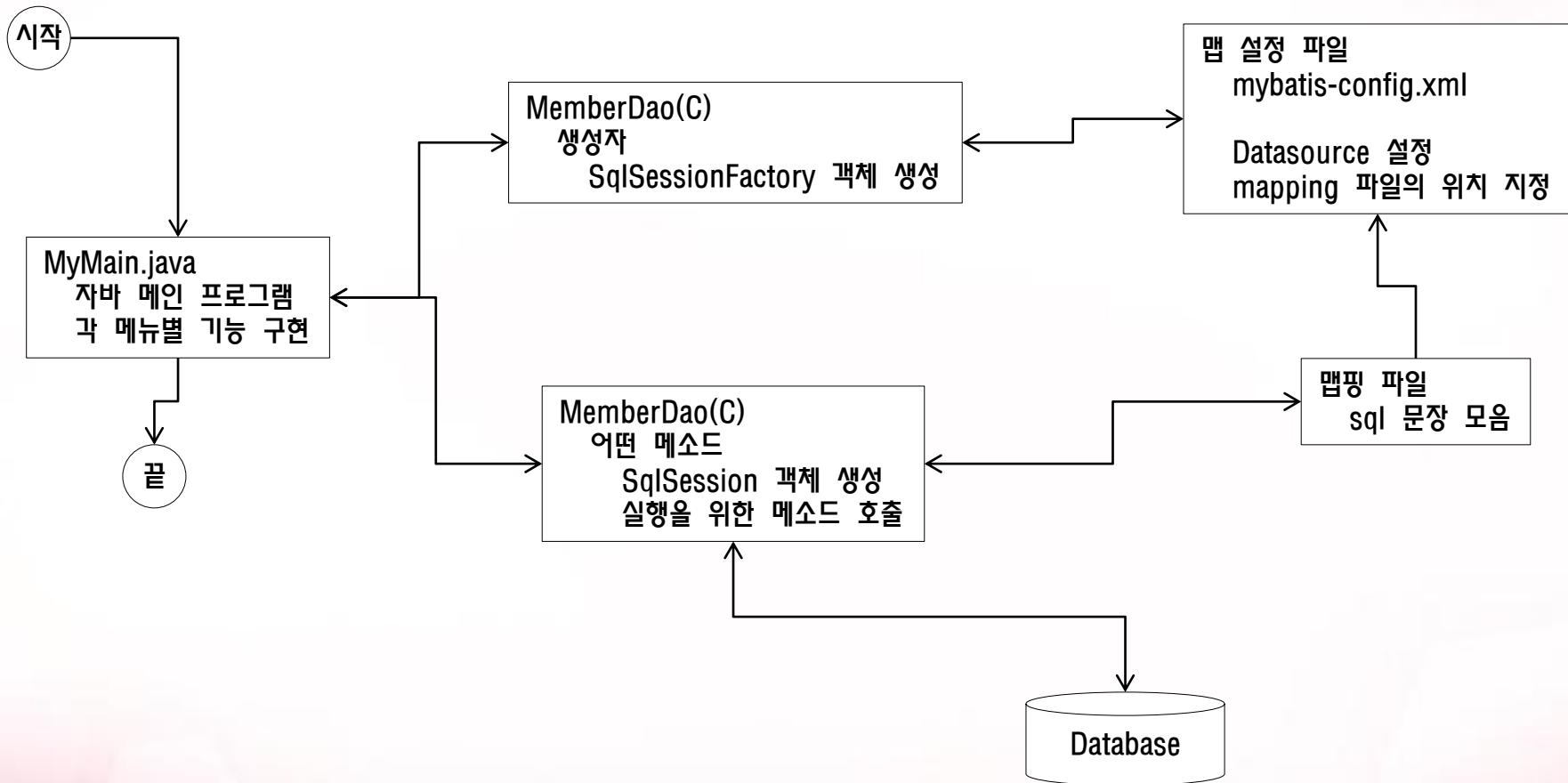
0.프로그램 종료, 1. 모든 회원 조회, 2.이름을 이용한 조회  
3.회원 추가, 4. 회원 수정, 5.회원 삭제

1

이름	나이	성별
박영희	20	여자
홍길동	30	남자

# 실습 : MyBatisEx01

- 프로젝트 : MyBatisEx01



# 실습 : MyBatisEx02

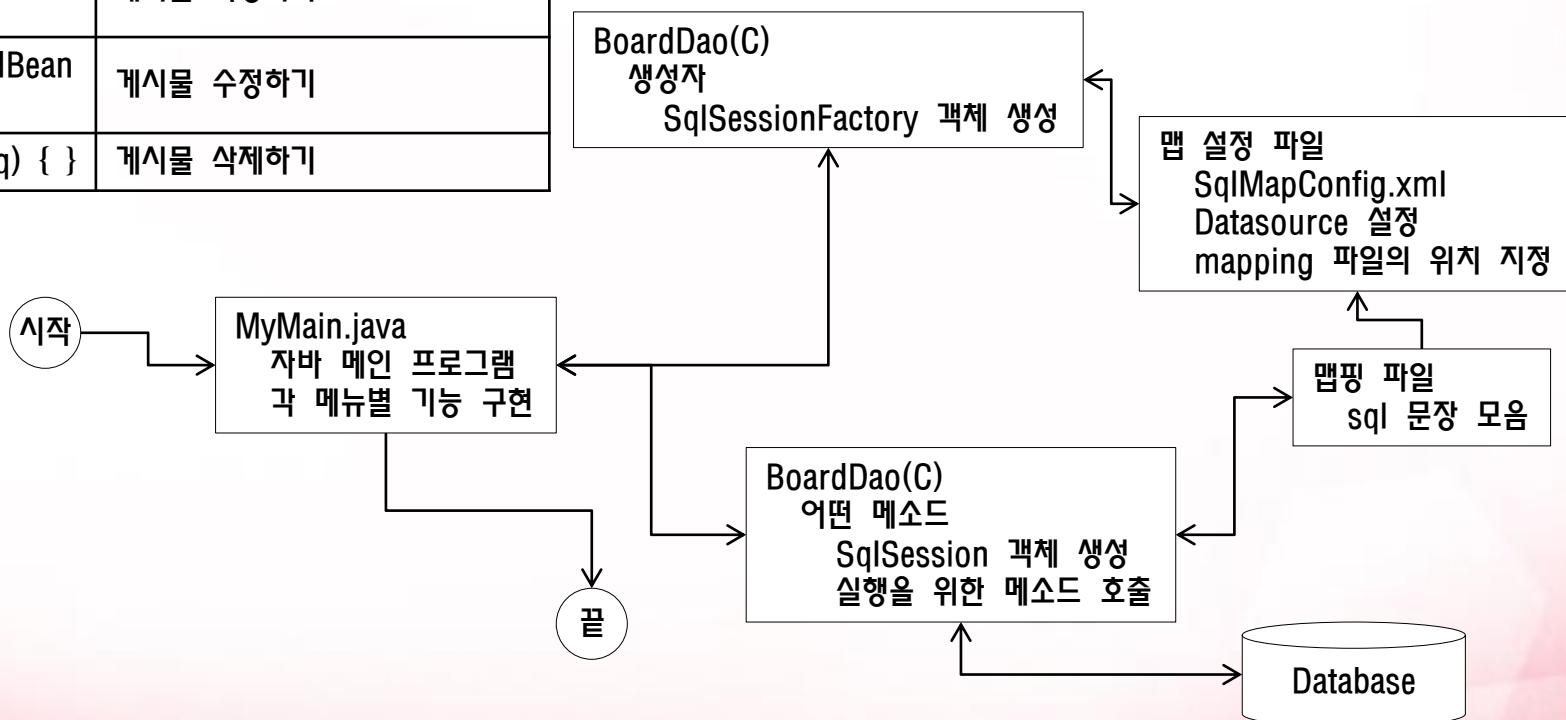
- 프로젝트 : MyBatisEx02
- MyBatis를 이용하여 게시물들에 대한 추가/수정/삭제/조회 기능을 구현한다.
- [MyMain.java](#) 파일을 이용하여 자바 콘솔에서 시작한다.

주요 파일 리스트	설명
MyMain.java	최초 실행 파일(자바의 main 메소드가 존재하는 파일)
dbscript.txt	테이블 생성 스크립트
mybatis.board.BoardBean.java	게시물에 대한 Bean 클래스 파일
mybatis.board.BoardDao.java	Data Access Object
mybatis/board/SqIMapConfig.xml	SQL 맵 설정 파일
/mybatis/board/SqIMap.xml	맵 핑 파일
db.properties	데이터 베이스 출처 정보를 저장하고 있는 properties 파일

# 실습 : MyBatisEx02

- 비즈니스 규칙 정의(BoardDao 클래스) 및 업무 흐름도

파일 리스트	설명
public List<BoardBean> getList() { }	모든 게시물 리스트 보기
public BoardBean getSelectBySeq(int seq) { }	기본 키로 게시물 조회하기
public int insertBoard(BoardBean bean) { }	게시물 작성하기
public int updateBoard(BoardBean bean){ }	게시물 수정하기
public int deleteBoard(int seq) { }	게시물 삭제하기



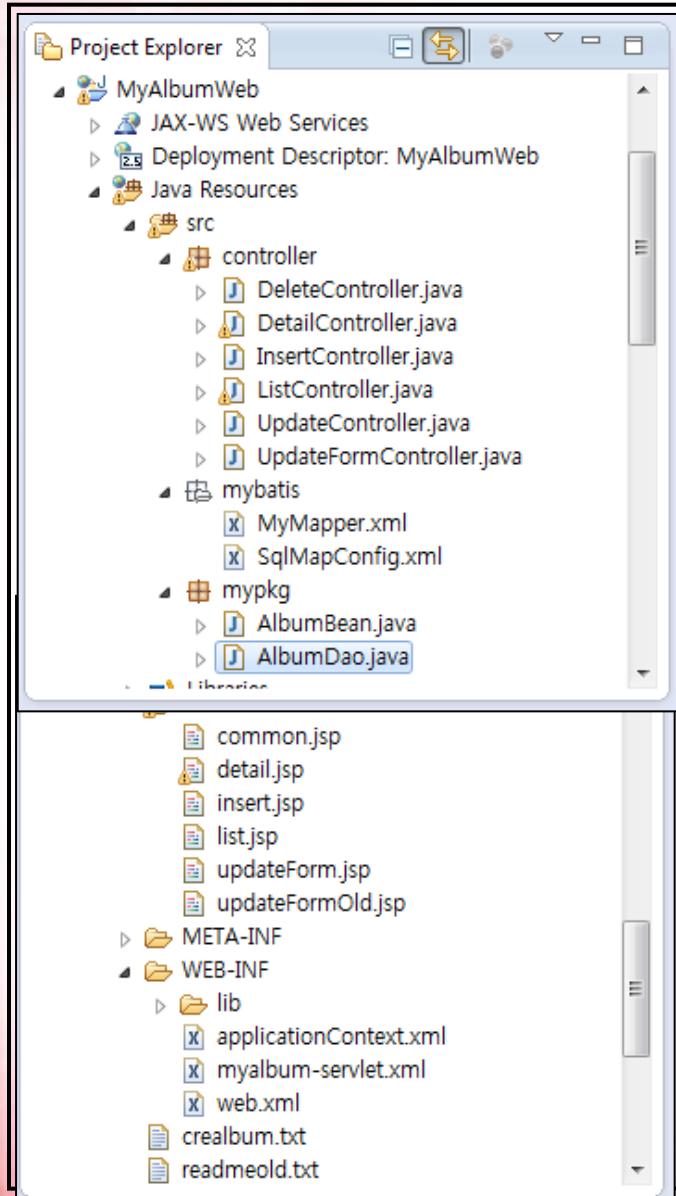
# 실습 : MyBatisEx03

- 프로젝트 : MyBatisEx03
- MyBatis를 이용하여 추가/수정/삭제/조회 기능을 구현한다.
- [MyMain.java](#) 파일을 이용하여 자바 콘솔에서 시작한다.

주요 파일 리스트	설명
MyMain.java	최초 실행 파일(자바의 main 메소드가 존재하는 파일)
dbscript.txt	테이블 생성 스크립트
mybatis.board.CommentBean.java	Bean 클래스 파일
mybatis.board.CommentDao.java	Data Access Object
mybatis/board/SqIMapConfig.xml	SQL 맵 설정 파일
/mybatis/board/SqIMap.xml	맵 핑 파일
db.properties	데이터 베이스 출처 정보를 저장하고 있는 properties 파일

# 실습 : MyAlbumWeb

- 프로젝트 : MyAlbumWeb
- 스프링과 MyBatis를 이용하여 앨범에 대한 기본 CRUD 작업을 테스트 해본다.



주요 파일 리스트	설명
insert.jsp	최초 실행 파일(실행하게 되면 최초 /insert.do를 요청한다.)
crealbum.txt	테이블 생성 스크립트
AlbumBean.java	Bean 클래스 파일 myPkg.bean. AlbumBean.java 파일 생성 테이블 구조를 보고 bean 생성하길 바람
AlbumDao.java	Data Access Object
web.xml	배포 서술자 파일
applicationContext.xml (스프링 설정 파일)	dataSource / sqlSessionFactoryBean / sqlSessionTemplate / albumDao 객체 설정
myalbum-servlet.xml (MVC 설정 파일)	MVC 설정 정보를 담고 있는 파일 HandlerMapping : BeanNameUrlHandlerMapping 빈의 이름을 이용하여 Controller 설정 Controller : 단위 업무마다 별도의 컨트롤러를 설정하고 있다. ViewResolver : InternalResourceViewResolver /album/viewname.jsp 파일을 지정하고 있다.
SqlMapConfig.xml	맵 설정 파일
MyMapper.xml	맵핑 파일
컨트롤러 파일	controller 패키지 내에 xxxController 파일을 생성하도록 한다.
jsp 파일	common.jsp, detail.jsp, insert.jsp, list.jsp, updateForm.jsp

# 실습 : MyAlbumWeb

- 최초 시작 파일 : <http://localhost:포트번호/MyAlbumWeb/album/insert.jsp>

시작

web.xml

- 1) ApplicationContext(스프링) 설정 파일  
contextConfigLocation 파라미터를 이용한다.  
Bootstrap listener 등록 : ContextLoaderListener
- 2) 문자열 인코딩 설정하기
- 3) DispatcherServlet 설정  
예제에서는 확장자 do 에 대한 맵핑을 수행한다  
서블릿 이름 : withmybatis

Database

ApplicationContext 설정 파일 작성(applicationContext.xml)

- 1) DBCP 를 이용한 DataSource 설정
- 2) sqlSessionFactoryBean 객체 설정  
(1) setter Injection  
dataSource  
configLocation : MyBatis 맵 설정 파일의 이름과 경로를 의미.  
mapperLocations : 맵핑 파일의 이름과 경로를 의미.
- 3) sqlSessionTemplate 객체 생성  
sqlSessionFactoryBean 객체를 생성자 주입
- 4) DAO 빈 설정  
sqlSessionTemplate 객체를 setter Injection 시킨다.

맵 설정 파일

맵핑 파일  
네임 스페이스 명  
MyNamespace

Dao 객체  
AlbumDao

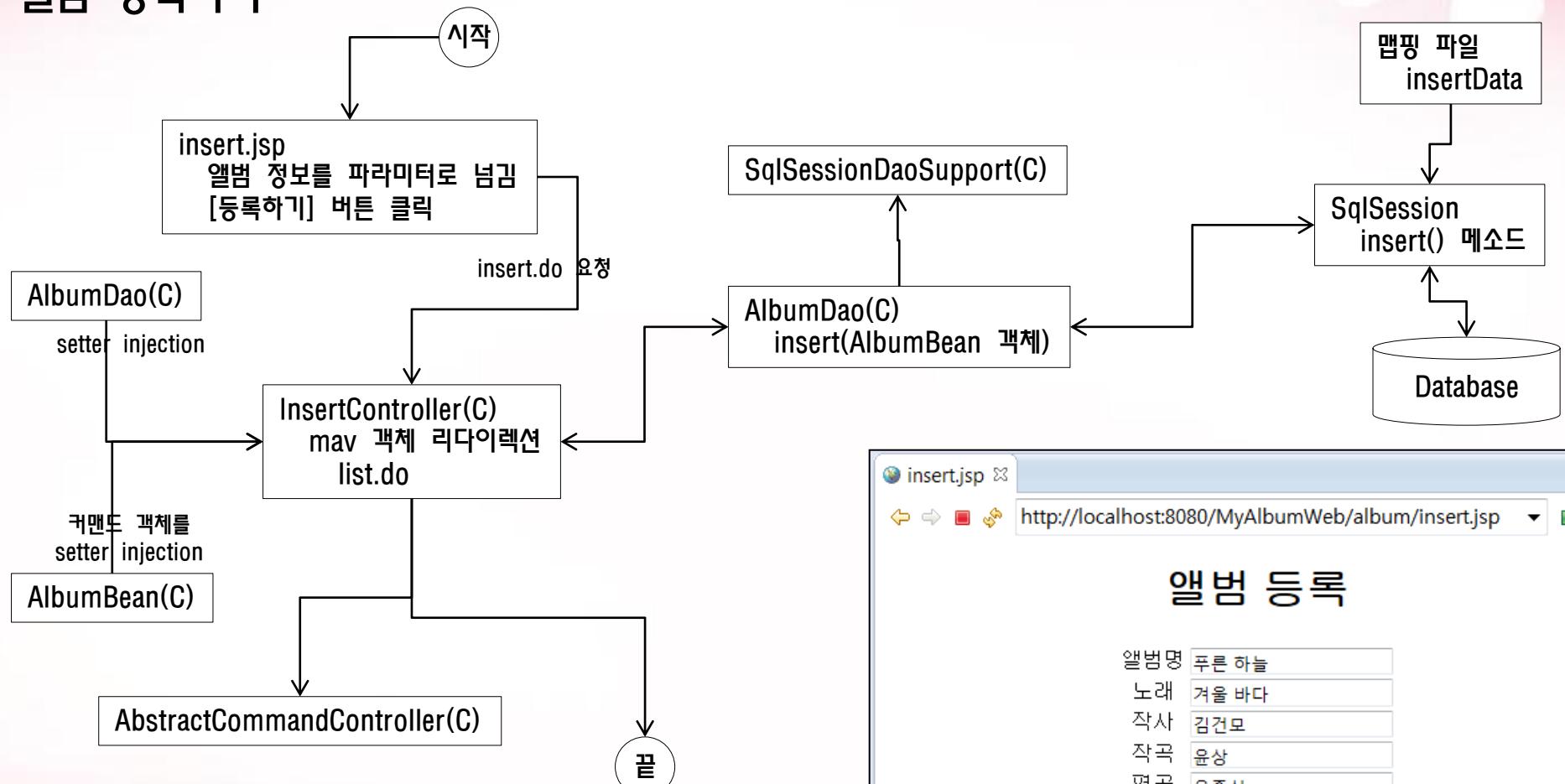
MVC용 설정 파일 작성(withmybatis-servlet.xml)

- 1) HandlerMapping 설정 : BeanNameUrlHandlerMapping
- 2) 컨트롤러 설정하기  
(1) 업무별 Controller 설정하기
- 3) ViewResolver 설정 : InternalResourceViewResolver  
(1) 접두사 : "/album/"  
(2) 접미사 : ".jsp"

끝

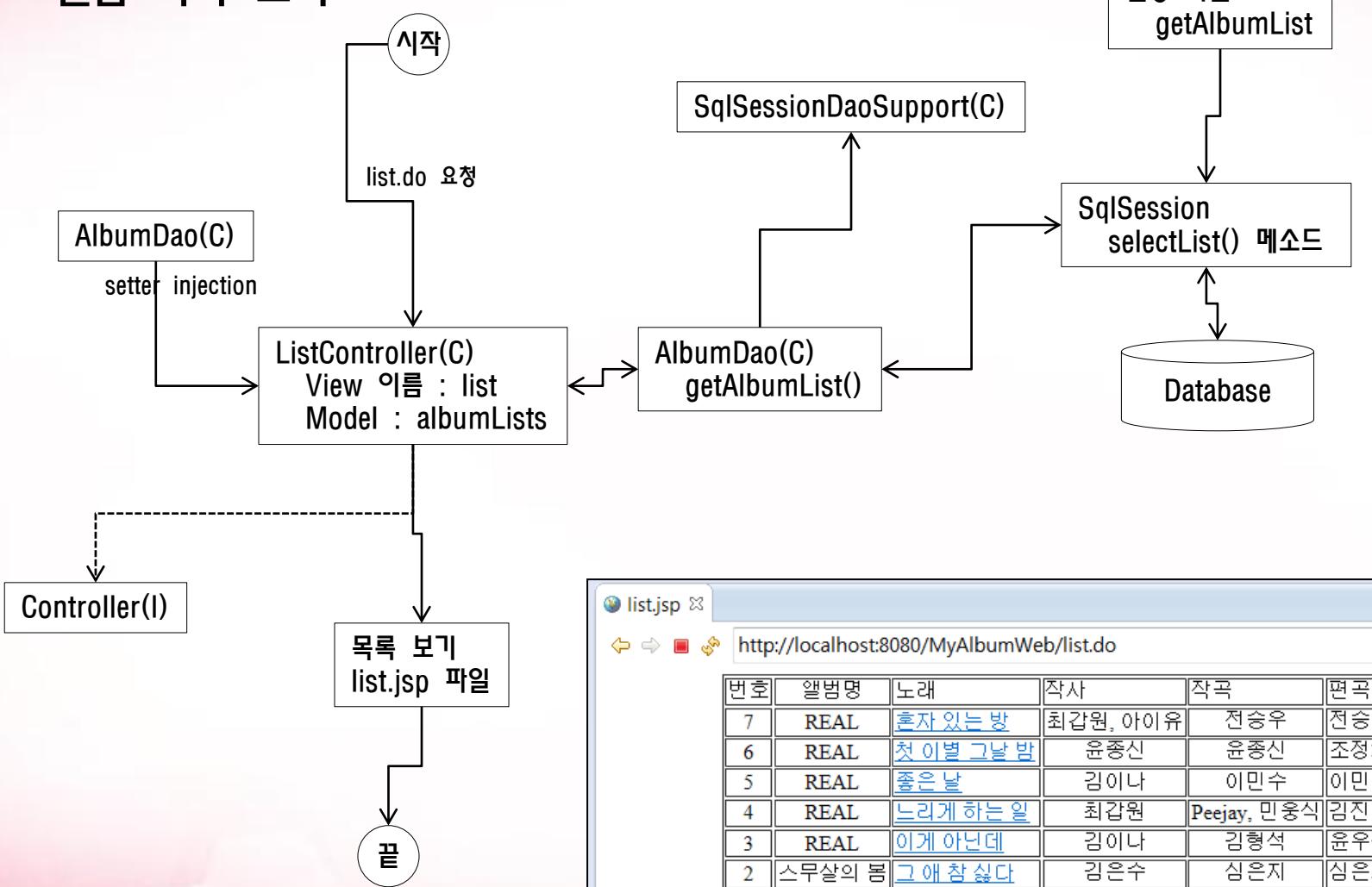
# 실습 : MyAlbumWeb

- 앨범 등록하기



# 실습 : MyAlbumWeb

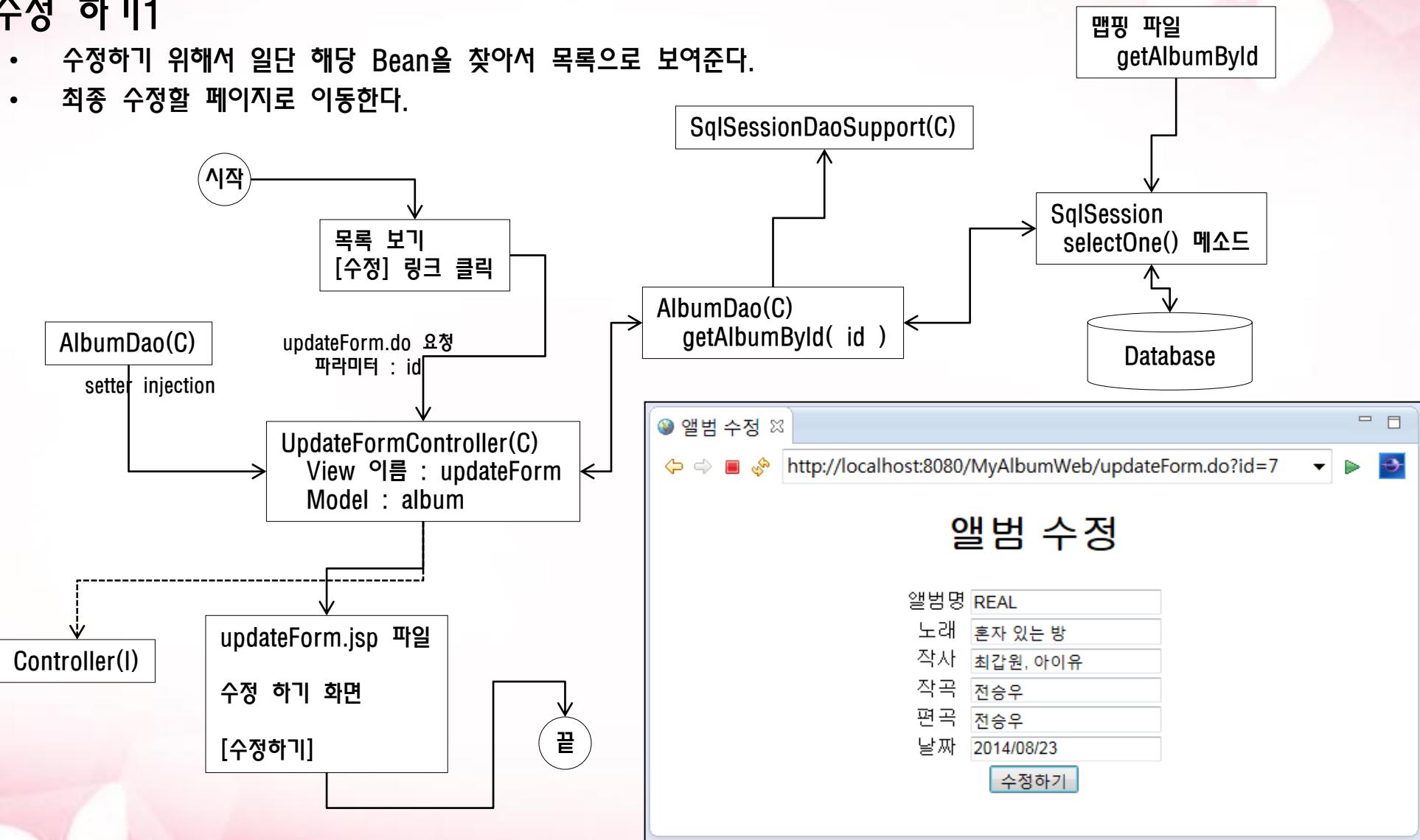
- 앨범 목록 보기



# 실습 : MyAlbumWeb

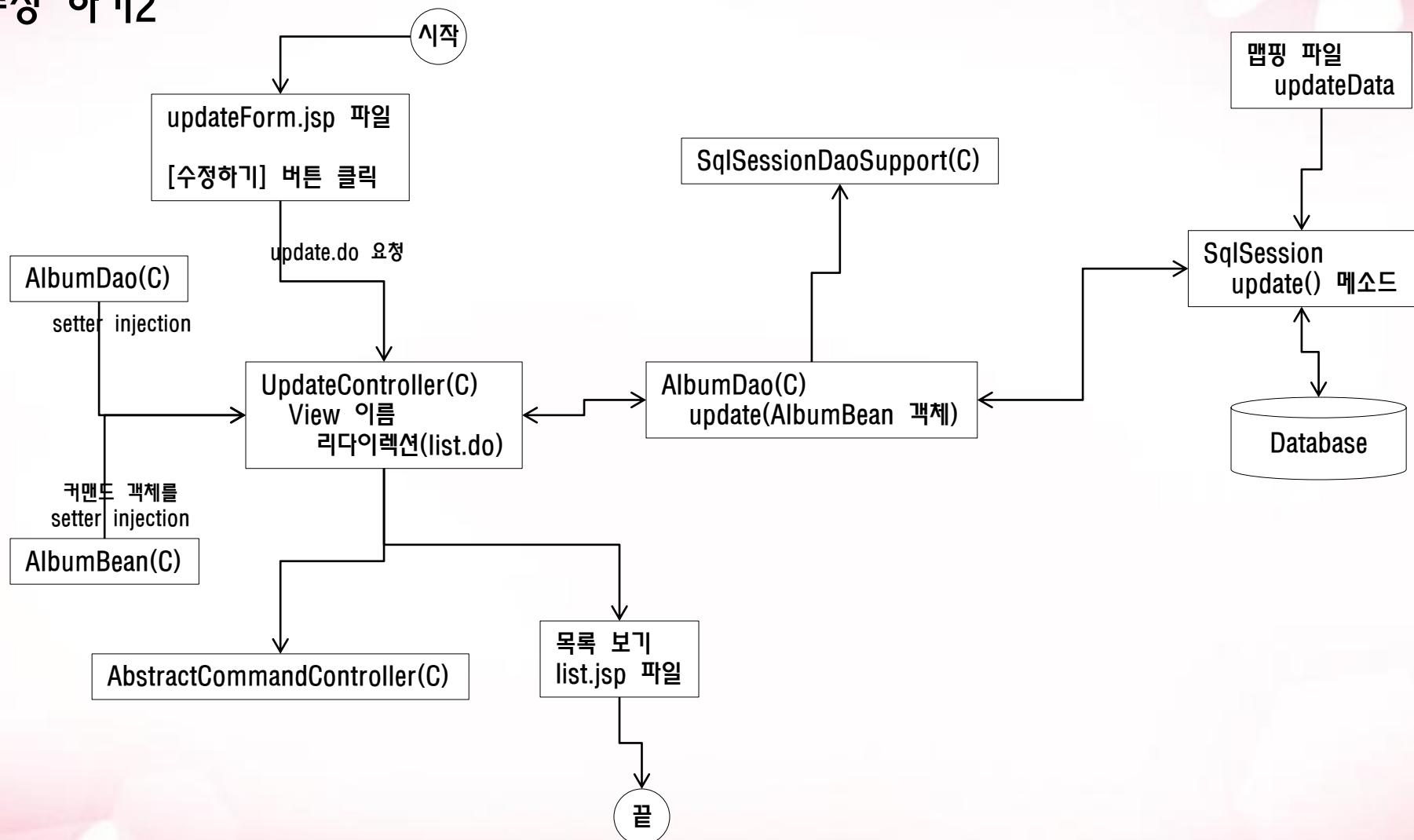
## • 수정하기1

- 수정하기 위해서 일단 해당 Bean을 찾아서 목록으로 보여준다.
- 최종 수정할 페이지로 이동한다.



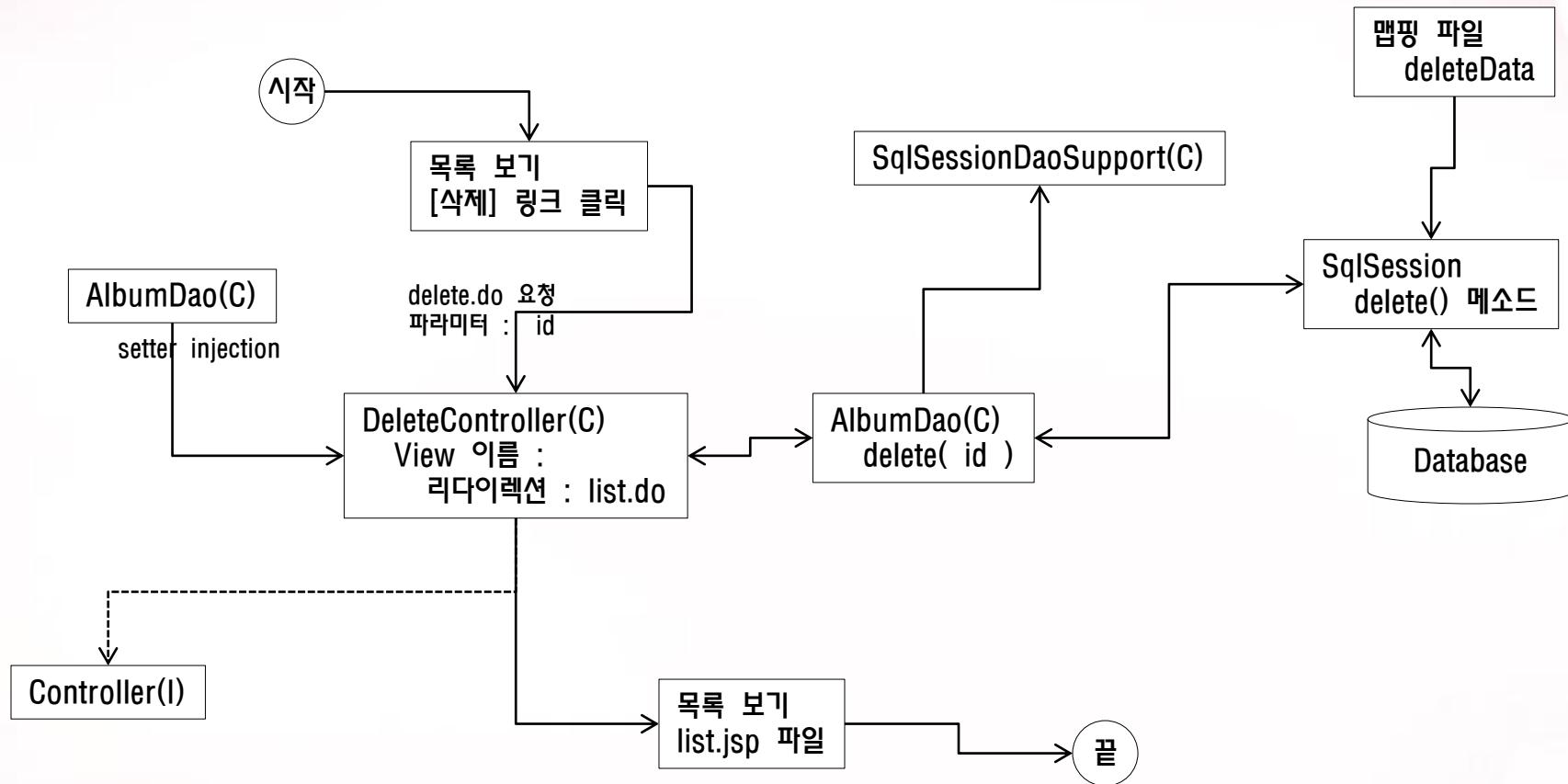
# 실습 : MyAlbumWeb

## • 수정 하기2



# 실습 : MyAlbumWeb

- 삭제하기



# 실습 : SangpumDetail

- XXX

The screenshot displays two browser windows side-by-side, illustrating a web application's product management feature.

**Left Window (상품 리스트 화면):** This window shows a table of products. The columns are "상품ID" and "상품명". The data is as follows:

상품ID	상품명
9	파인애플
8	수박
7	메론
6	체리
5	블루베리
4	파란사과
3	키위
2	오렌지
1	레몬

**Right Window (상품 상세 화면):** This window shows a detailed view of a product. The URL in the address bar is `http://localhost:8080/SangpumDetail/detail.html?itemId=9`. The product details are:

상품명	파인애플
가격	2000 원
설명	비타민B1 비타민B2 가 풍부합니다. 요리에도 사용할 수 있습니다.

A large image of a pineapple is displayed on the left side of the right window. At the bottom right of the right window, there is a link: "상품 리스트 화면으로 돌아갈".

# 실습 : SangpumDetail

- XXX

The image displays two adjacent browser windows side-by-side, illustrating a web application's product management interface.

**Left Window (상품 추가 화면):** This window shows a form for adding a new product. The URL in the address bar is `http://localhost:8080/SangpumDetail`. The form fields are:

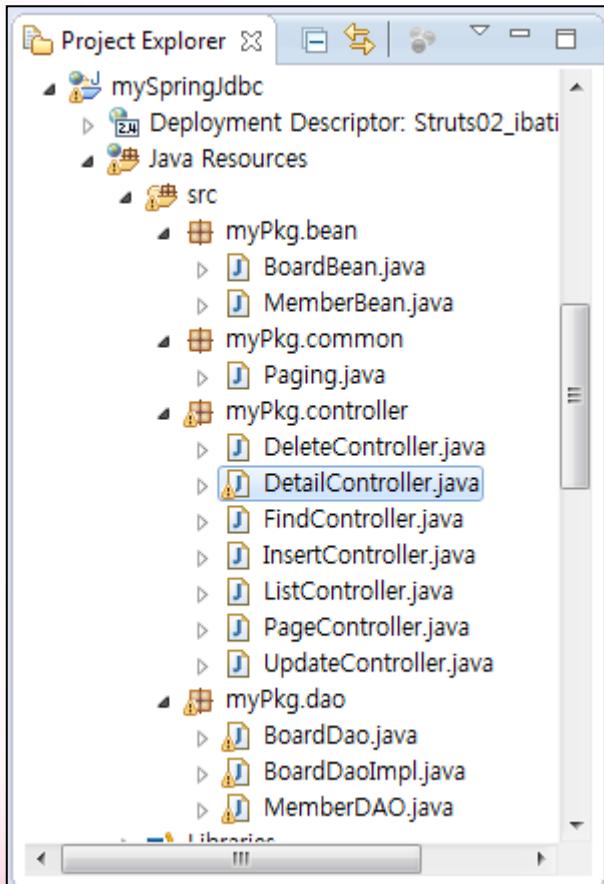
상품명	참외
가격	1234
설명	이건 설명
그림 파일	abc.jpg
<input type="button" value="글쓰기"/>	

**Right Window (상품 수정 화면):** This window shows a form for modifying an existing product. The URL in the address bar is `http://localhost:8080/SangpumDetail`. The form fields are:

상품명	파인애플
가격	2000
설명	비타민B1 비타민B2가 풍부
그림 파일	pine.jpg
<input type="button" value="글쓰기"/>	

# 실습 : mySpringJdbc

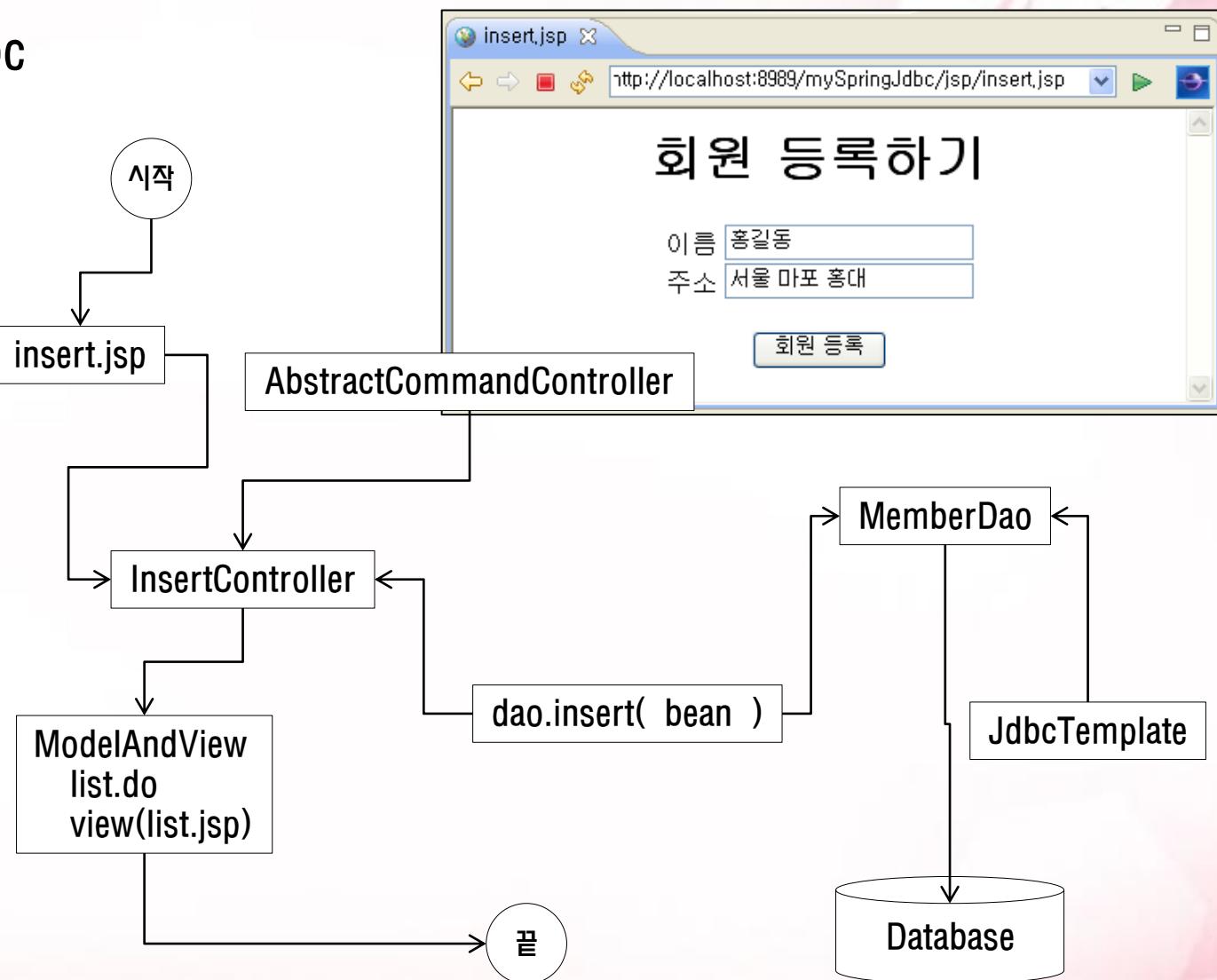
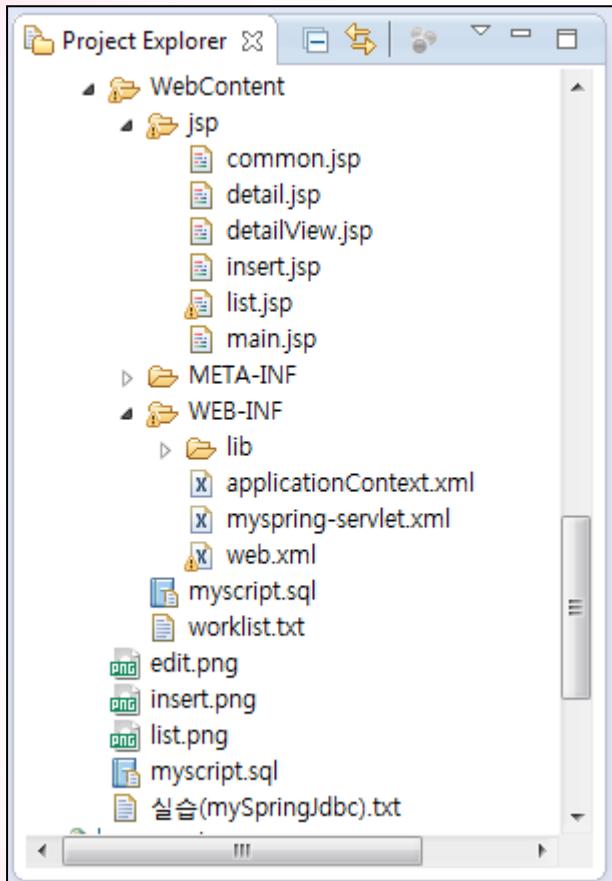
- 프로젝트 : mySpringJdbc
- 스프링을 이용하여 회원에 대한 기본 CRUD 작업을 테스트 해본다.



주요 파일 리스트	설명
insert.jsp	최초 실행 파일(실행하게 되면 최초 /insert.do를 요청한다.)
myscript.sql	테이블 생성 스크립트
MemberBean.java	회원에 대한 Bean 클래스 파일 myPkg.bean.MemberBean.java 파일 생성 테이블 구조를 보고 bean 생성하길 바람
MemberDao.java	Data Access Object
web.xml	배포 서술자 파일
applicationContext.xml	스프링 설정 파일 dataSource / JdbcTemplate 객체 설정 / DAO 빈 객체 설정
myspring-servlet.xml	MVC 설정 정보를 담고 있는 파일 HandlerMapping : BeanNameUrlHandlerMapping 빈의 이름을 이용하여 Controller 설정 Controller : 단위 업무마다 별도의 컨트롤러를 설정하고 있다. ViewResolver : InternalResourceViewResolver /jsp/viewname.jsp 파일을 지정하고 있다.
컨트롤러 파일	myPkg.controller 패키지 내에 xxxController 파일을 생성하도록 한다.

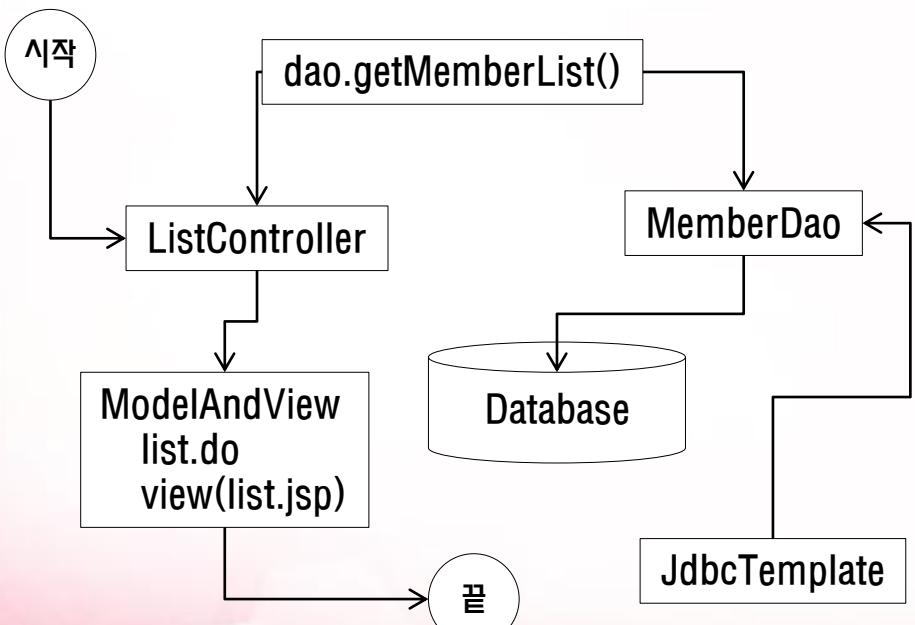
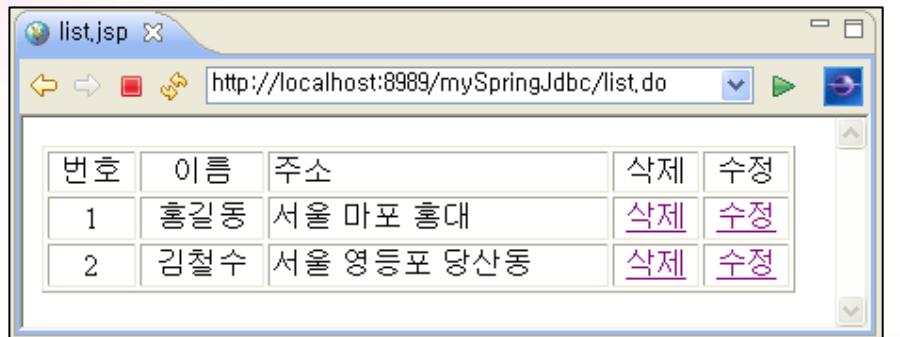
# 실습 : mySpringJdbc

- 프로젝트 : mySpringJdbc

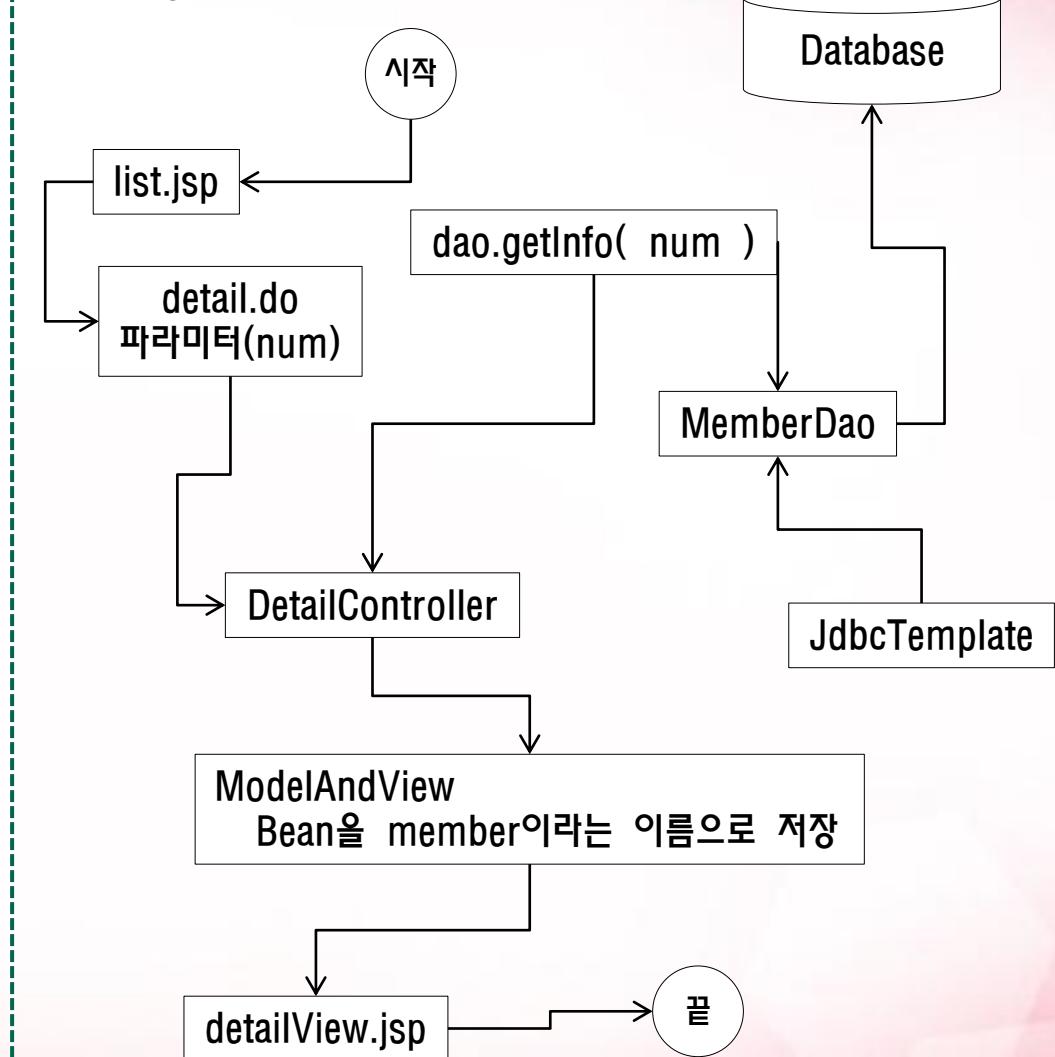


# 실습 : mySpringJdbc

## • 목록 조회하기

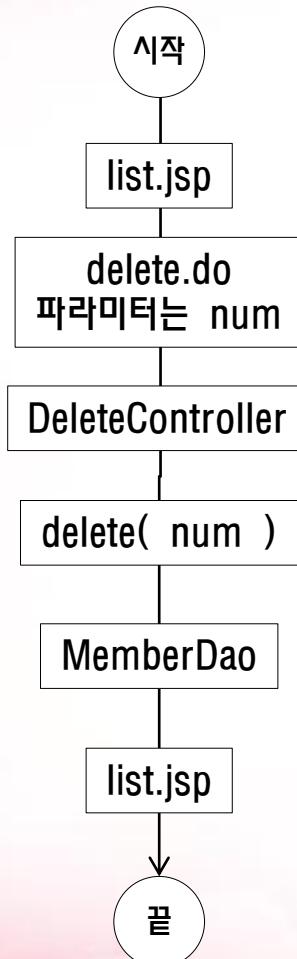


## • 상세 보기

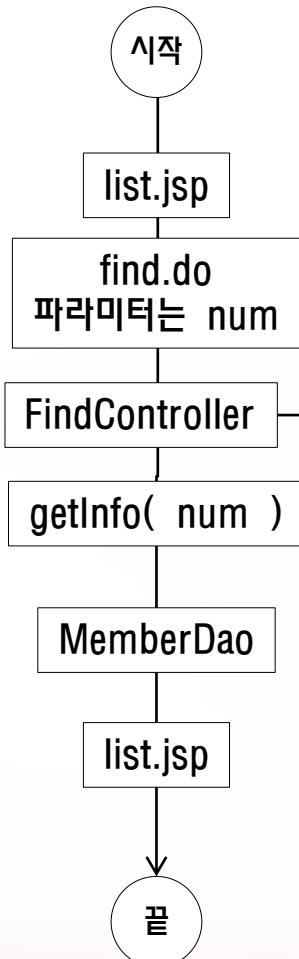


# 실습 : mySpringJdbc

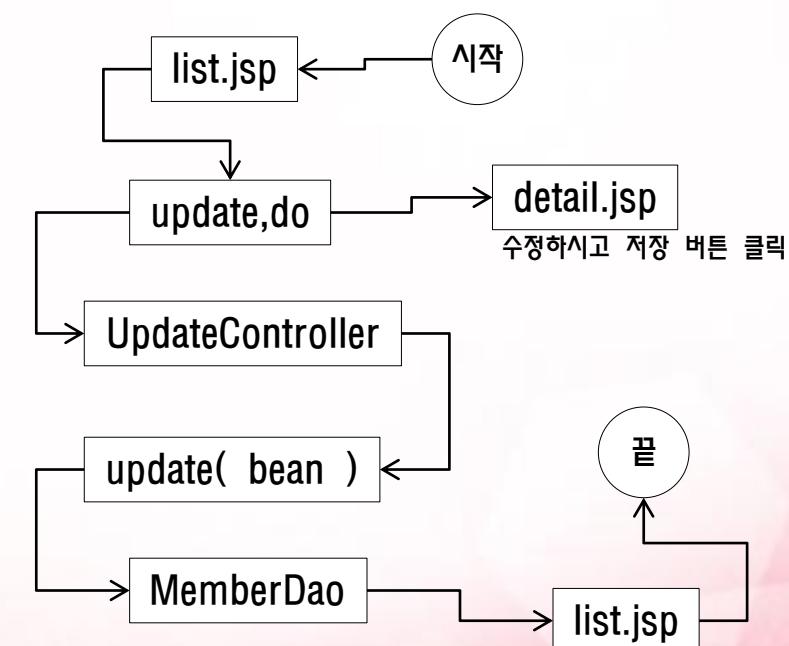
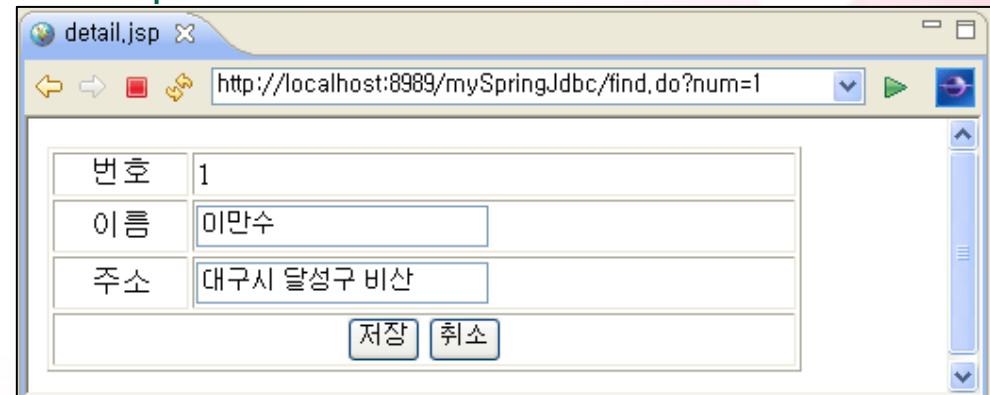
- 삭제하기



- 찾기



- 수정하기

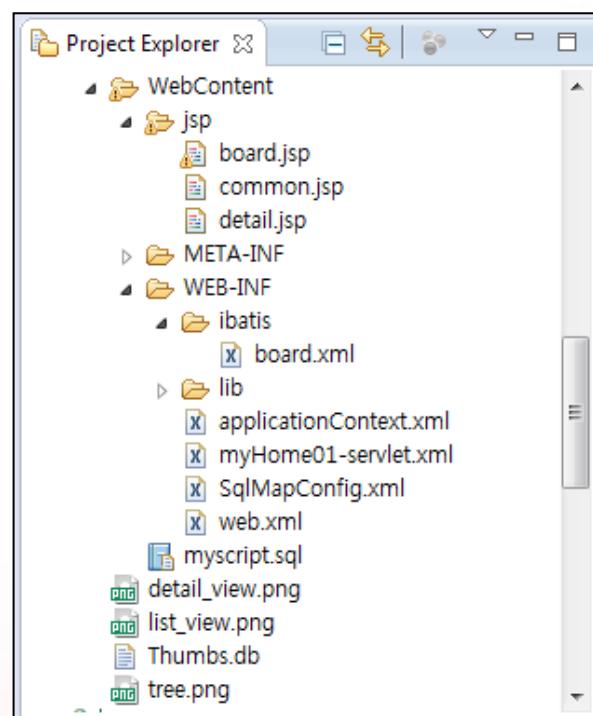
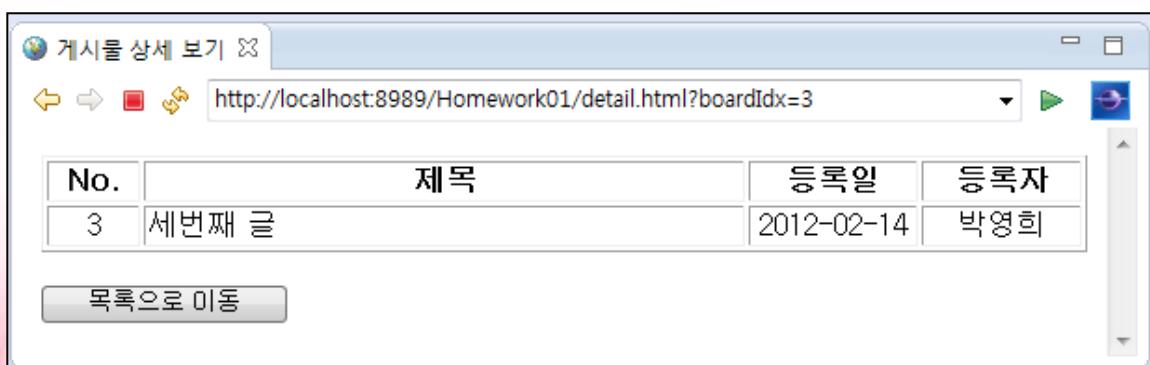
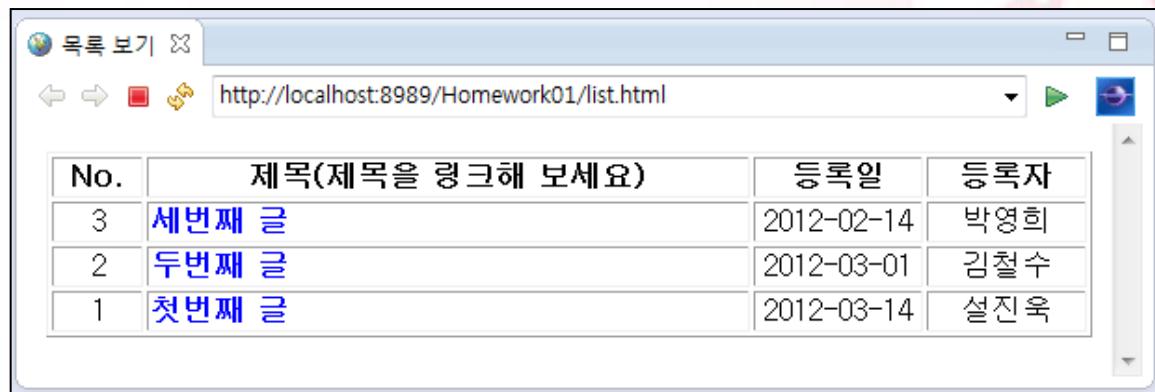
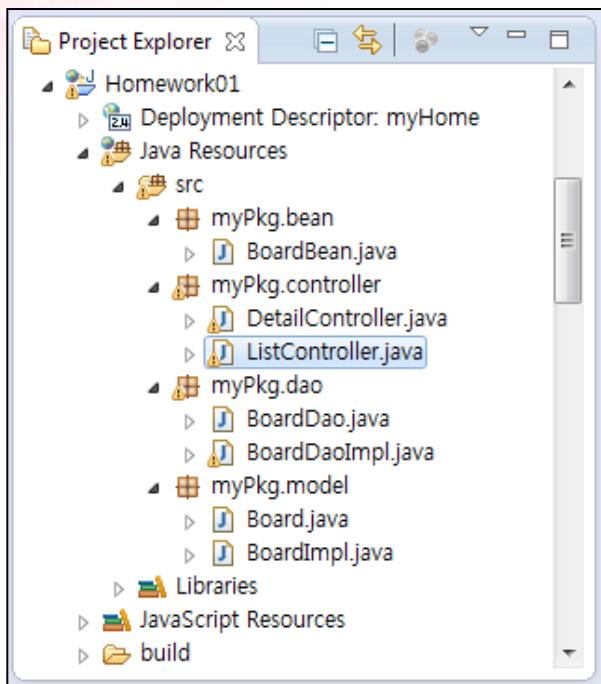


# 실습 : Homework01

- 프로젝트 : Homework01
- 이 예제는 iBatis를 이용한 회원을 조회하는 예시이다.

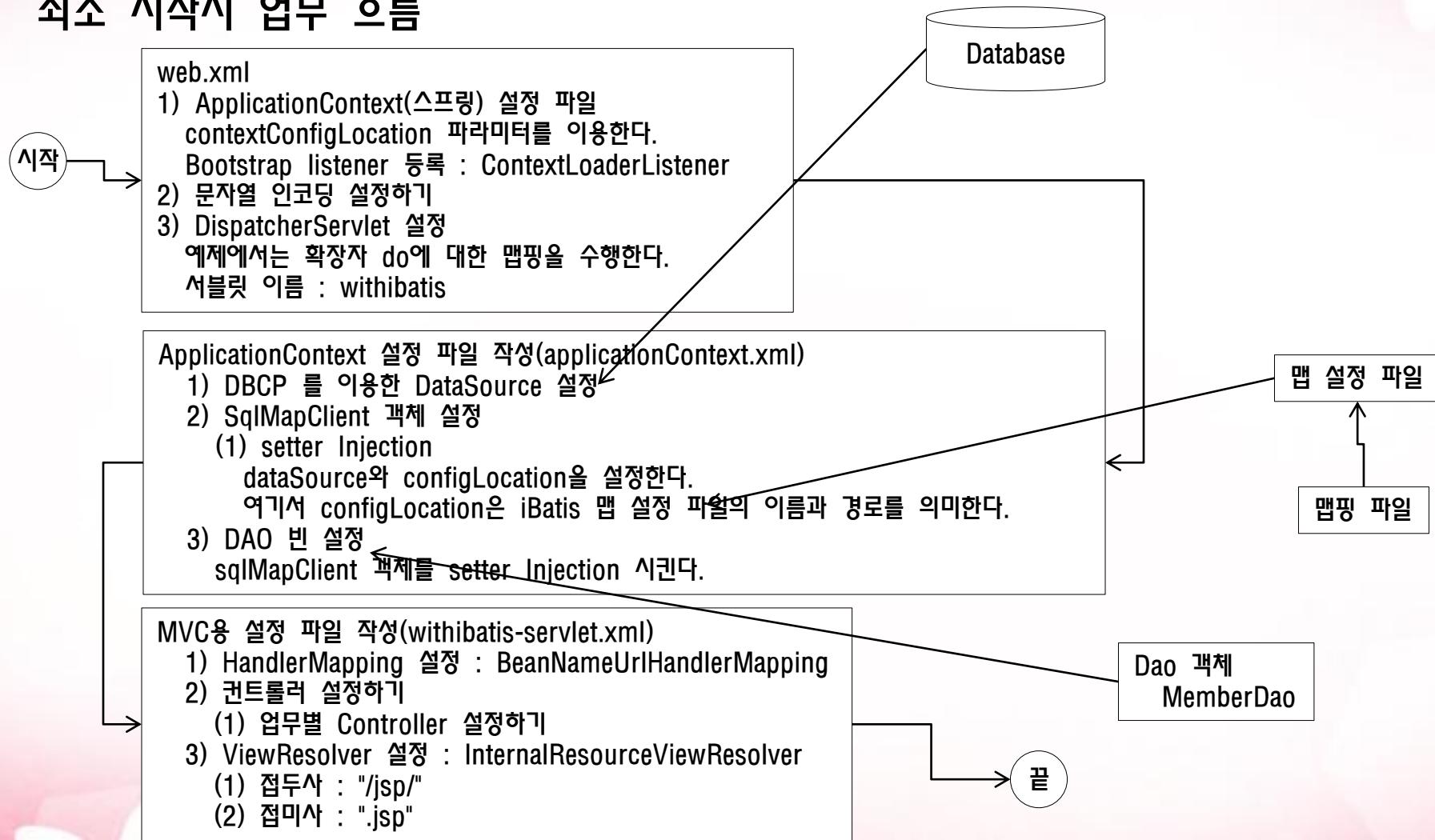
주요 파일 리스트	설명
최초 실행	<a href="http://localhost:8989/Homework01/list.html">http://localhost:8989/Homework01/list.html</a>
myscript.sql	테이블 생성 스크립트
BoardBean.java	게시물에 대한 Bean 클래스 파일
BoardDaoImpl.java	Data Access Object SqlMapClientDaoSupport 클래스 및 BoardDao 인터페이스를 상속 받고 있다.
web.xml	배포 서술자 파일
applicationContext.xml	스프링 설정 파일(dataSource 및 iBatis 설정 파일)
myHome01-servlet.xml	MVC 설정 정보를 담고 있는 파일
SqlMapConfig.xml	SQL 맵 설정 파일
board.xml	맵핑 파일
XXXController.java	게시물 관련한 컨트롤러 파일들

# 실습 : Homework01



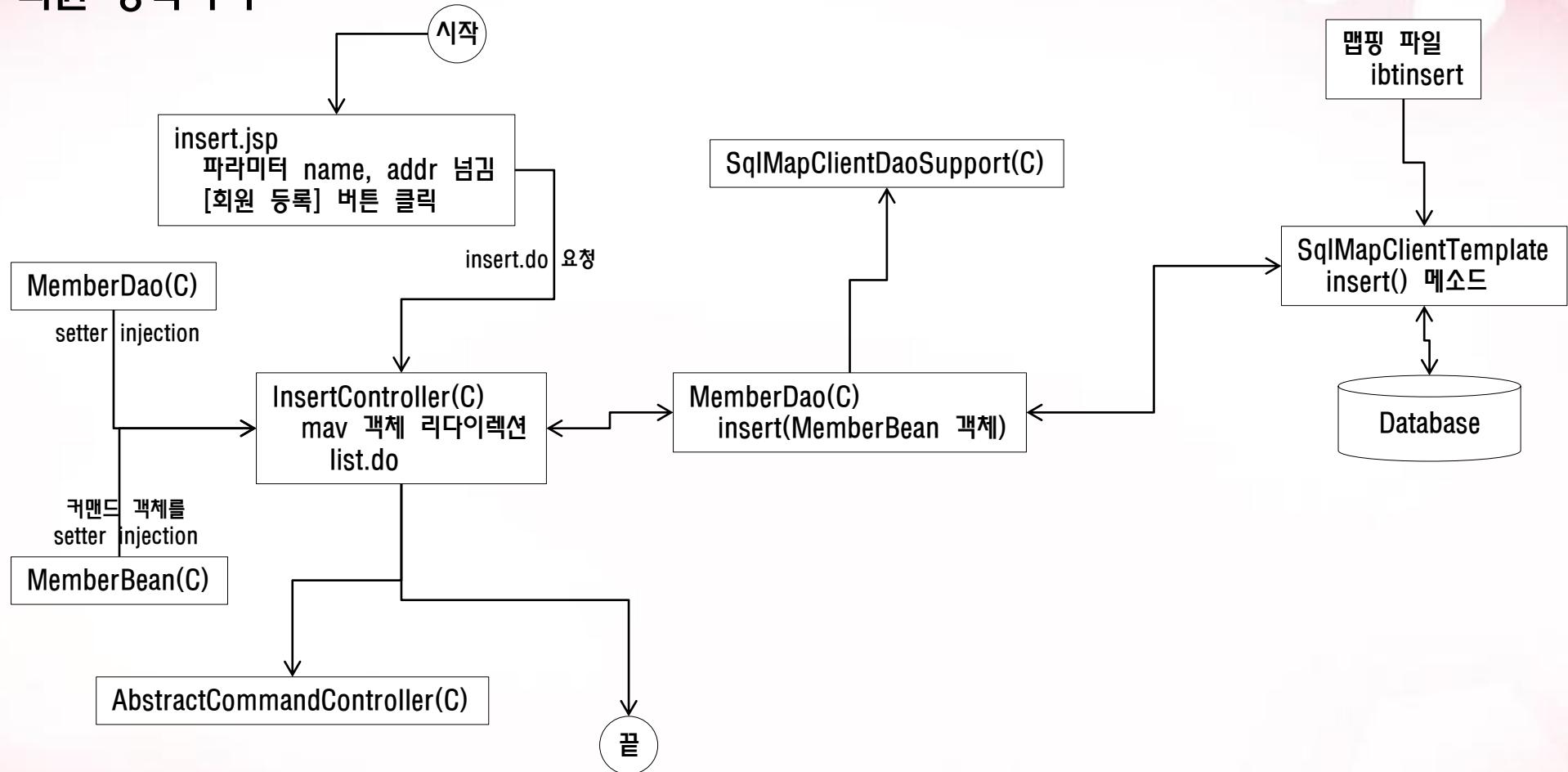
# 실습 : MySpringIbatis

- 최초 시작 파일 : `http://localhost:포트번호/MySpringIbatis/jsp/insert.jsp`
- 최초 시작시 업무 흐름



# 실습 : MySpringlbatis

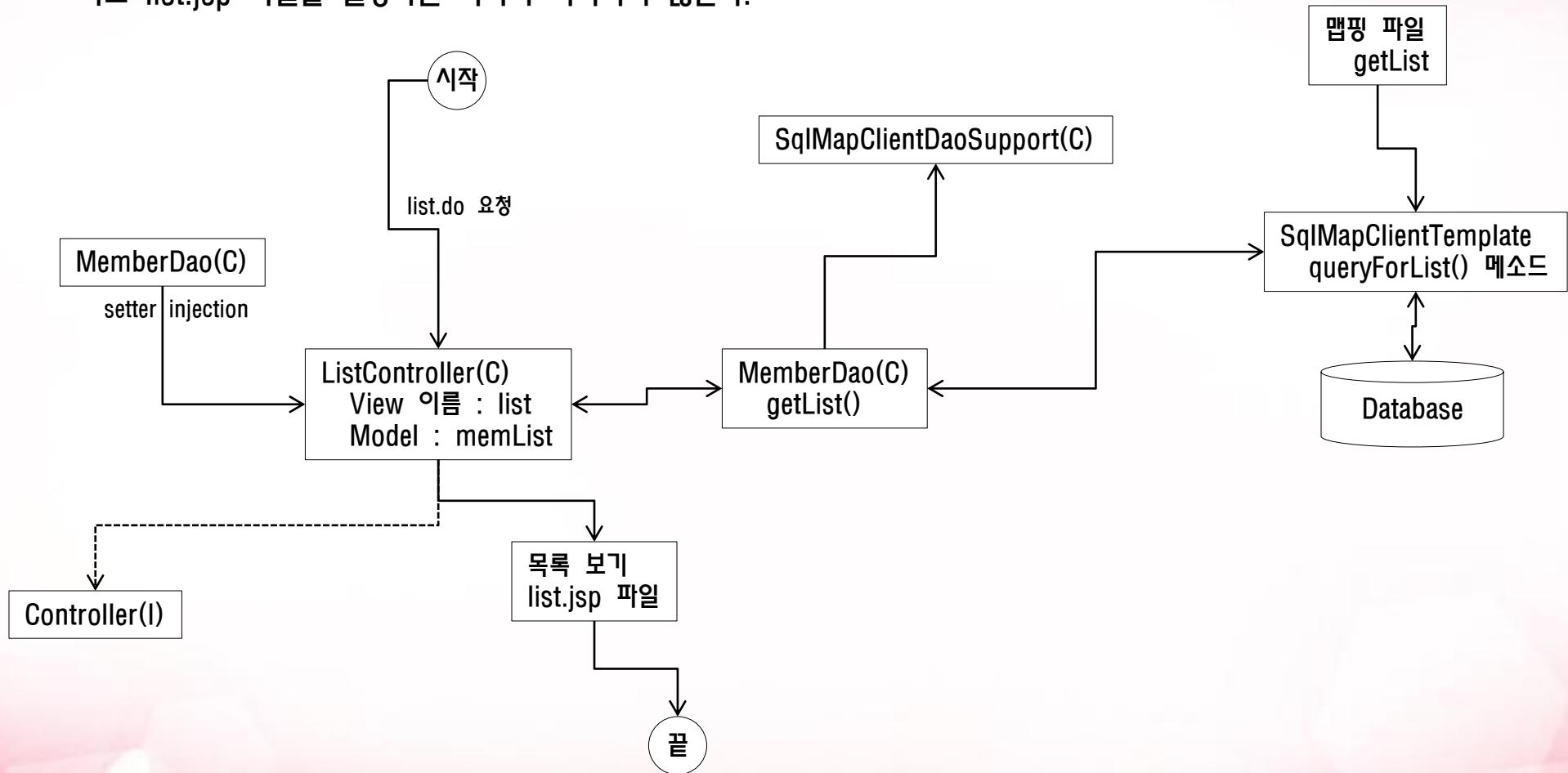
- 회원 등록하기



# 실습 : MySpringlbatis

## • 목록 보기

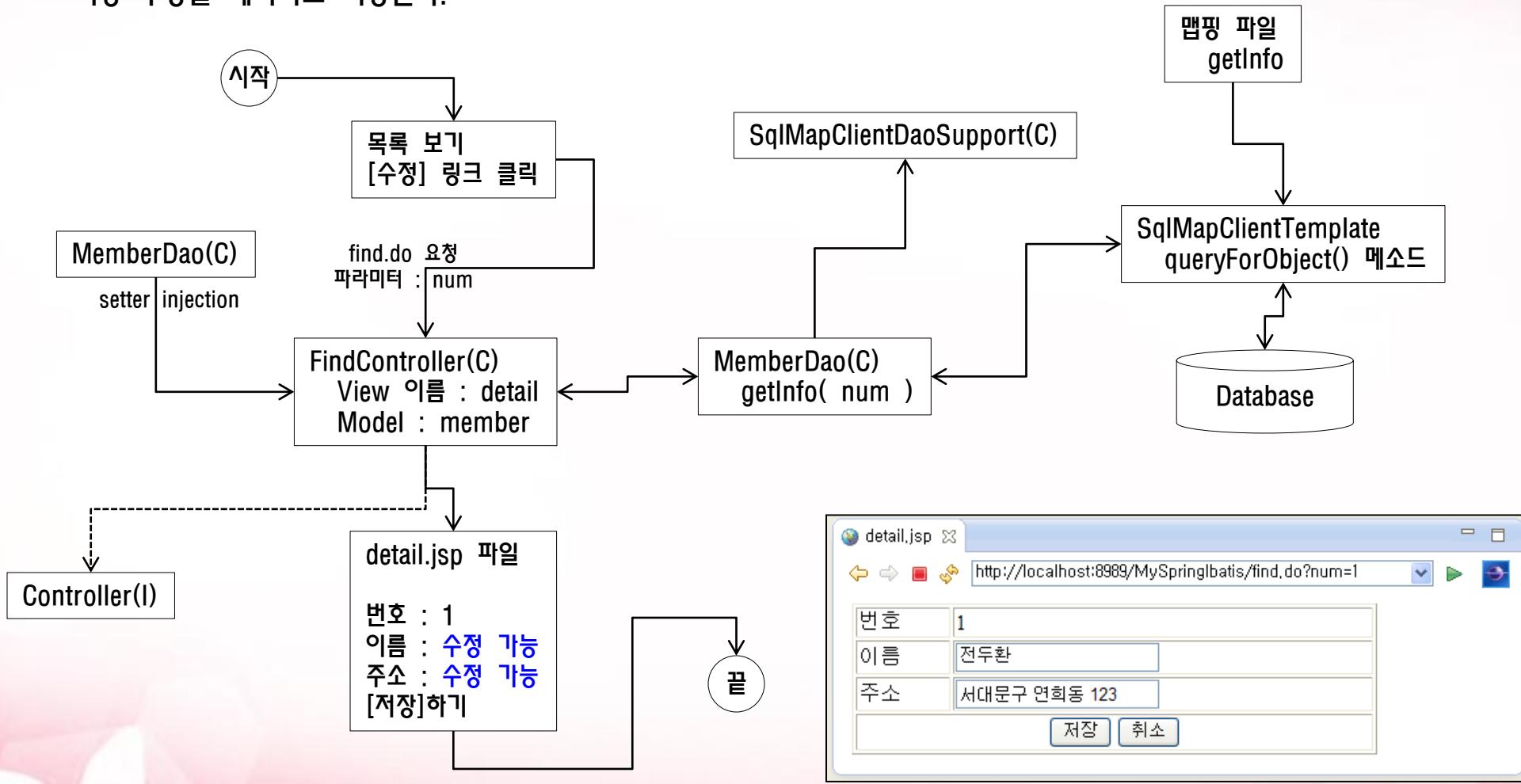
- 주의 : 회원 등록을 하고 나서, 목록 보기 페이지로 데이터가 넘어온다.
- 최초 list.jsp 파일을 실행하면 목록이 나타나지 않는다.



# 실습 : MySpringlbatis

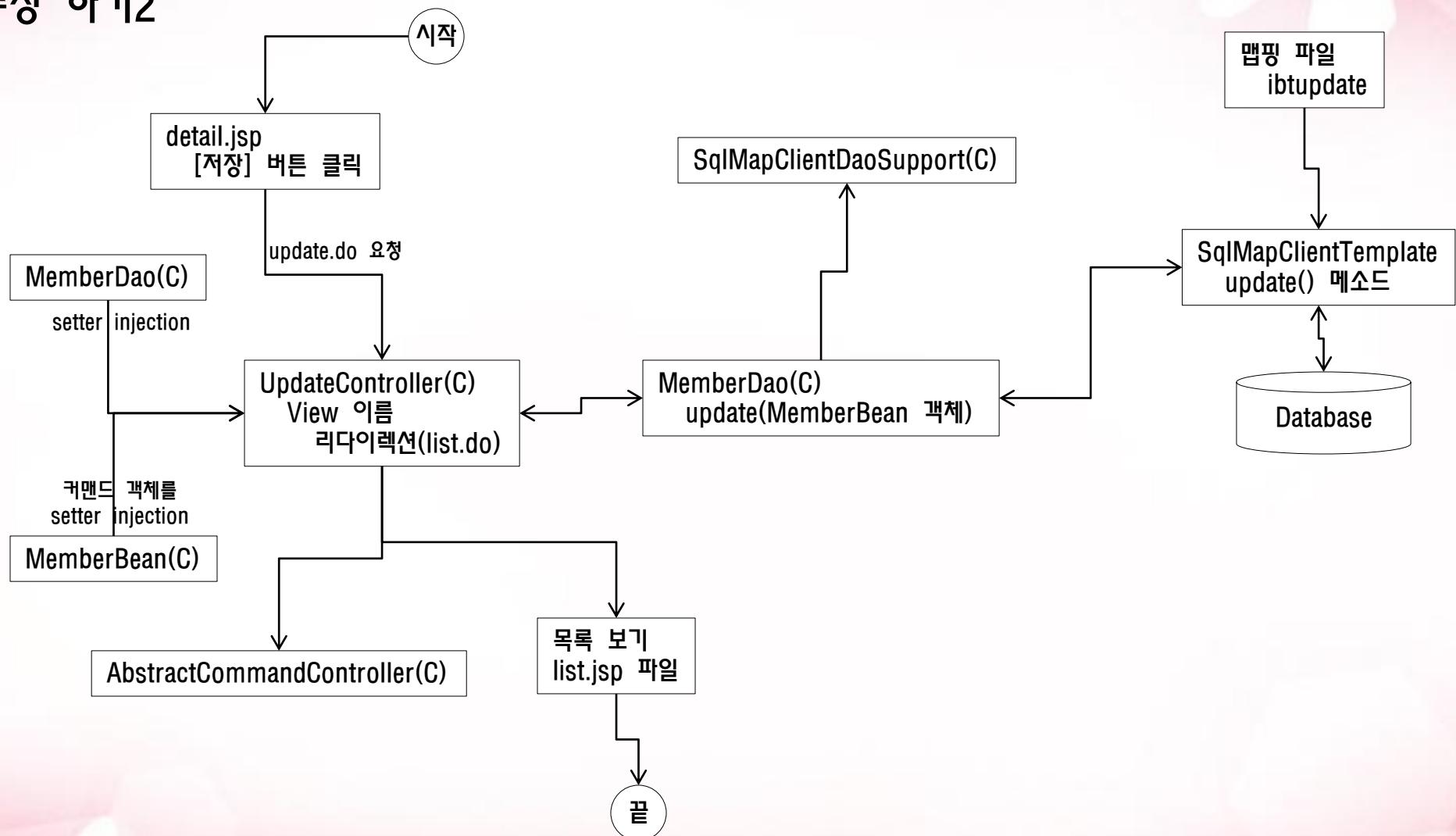
## • 수정 하기1

- 수정하기 위해서 일단 해당 Bean을 찾아서 목록으로 보여준다.
- 최종 수정할 페이지로 이동한다.



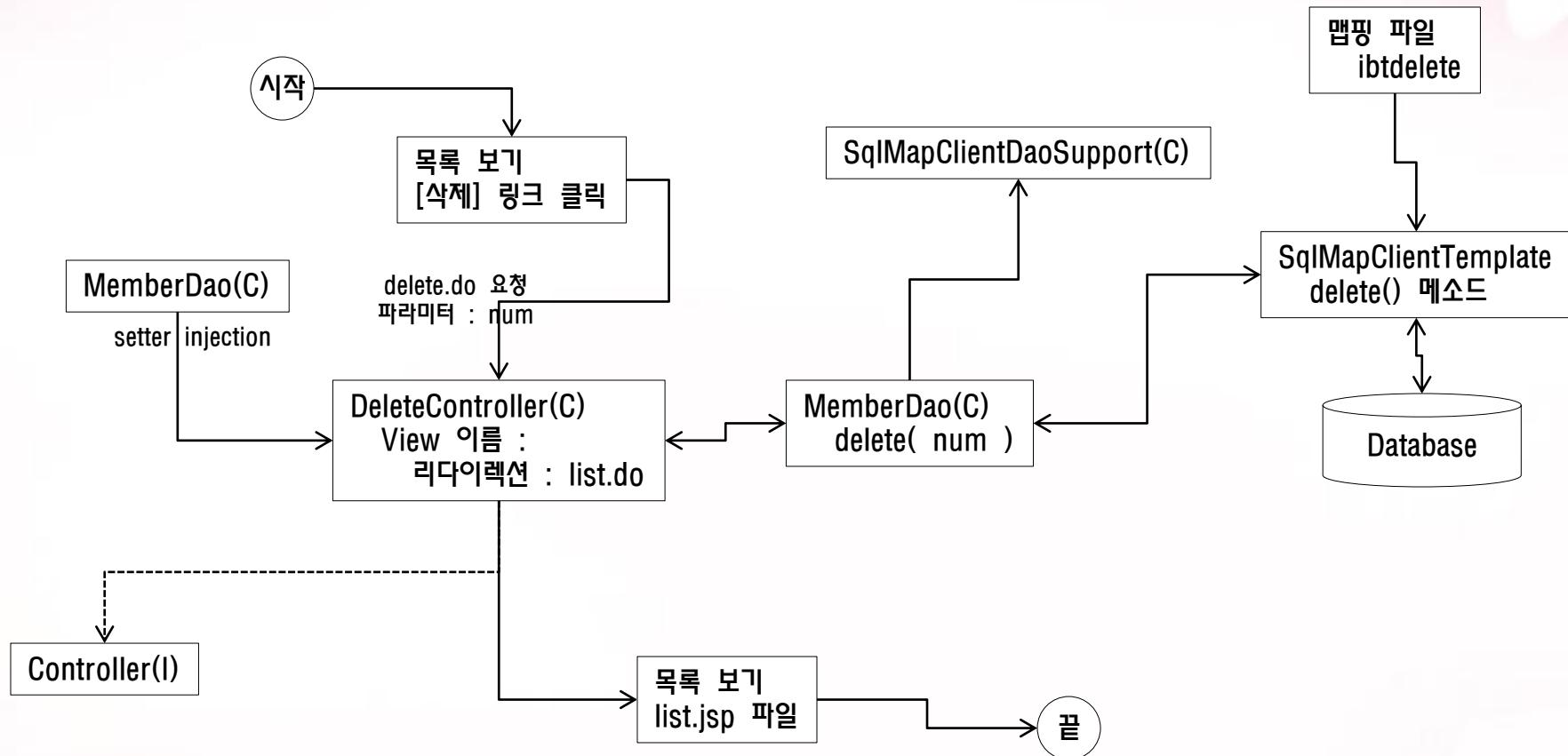
# 실습 : MySpringlbatis

- 수정 하기2



# 실습 : MySpringlbatis

- 삭제하기

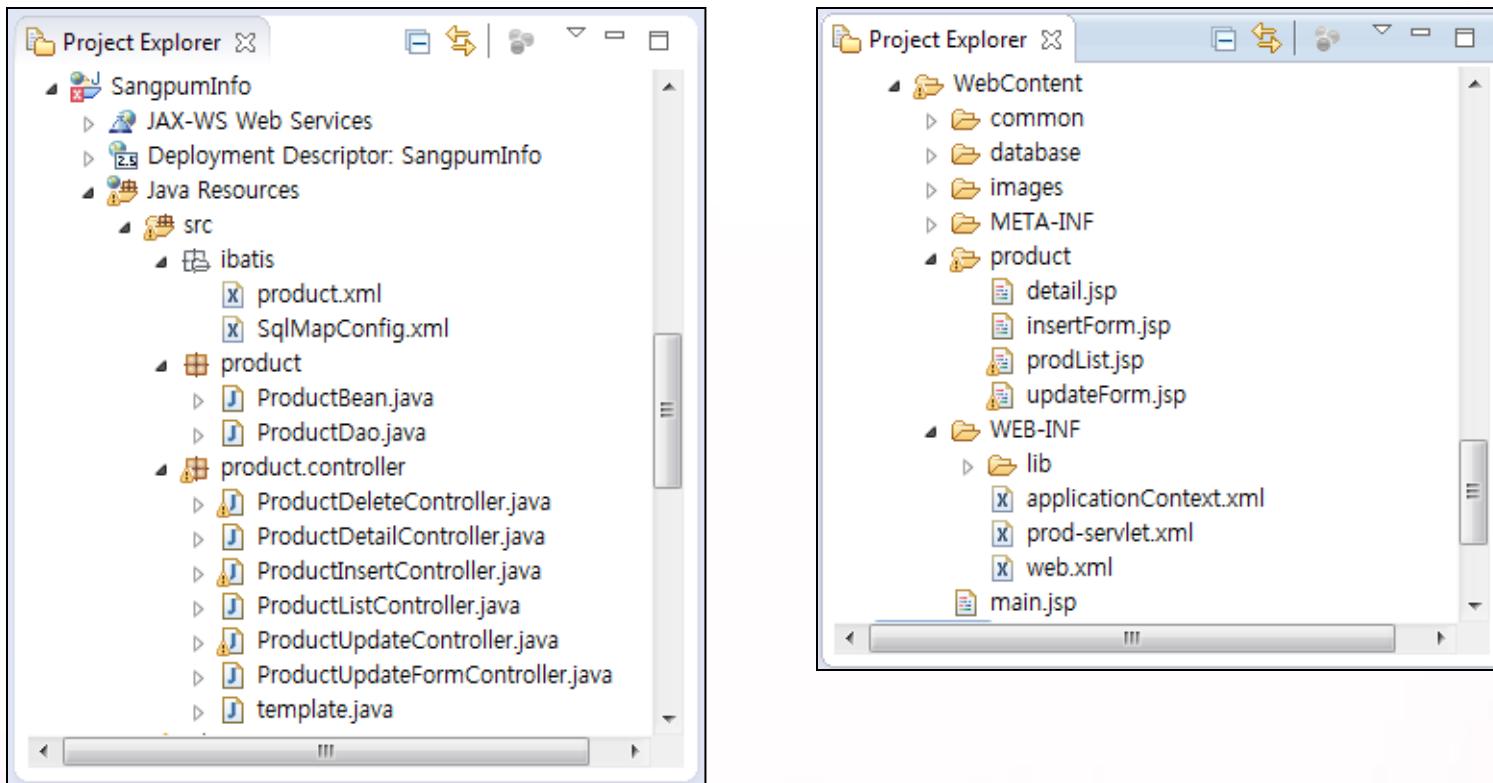


# 실습 : SangpumInfo

- 프로젝트 : SangpumInfo
- 스프링과 iBatis를 이용하여 상품에 대한 정보 보기 페이지를 구축한다.
- main.jsp 파일을 실행하여 프로그램을 시작한다.

주요 파일 리스트	설명
main.jsp	최초 실행 파일(실행하게 되면 /list.prd으로 리다이렉션된다.)
dbscript.sql	테이블 생성 스크립트
ProductBean.java	상품에 대한 Bean 클래스 파일
ProductDao.java	Data Access Object
web.xml	배포 서술자 파일
applicationContext.xml	스프링 설정 파일(dataSource 및 iBatis 설정 파일)
prod-servlet.xml	MVC 설정 정보를 담고 있는 파일
SqlMapConfig.xml	SQL 맵 설정 파일
product.xml	맵핑 파일
ProductXXXController.java	상품 관련한 컨트롤러 파일들

# 실습 : SangpumInfo



# 실습 : SangpumInfo

상품 리스트 화면

상품ID	상품명	가격	삭제	수정
1	레몬	300원	<a href="#">삭제</a>	<a href="#">수정</a>
2	오렌지	2000원	<a href="#">삭제</a>	<a href="#">수정</a>
3	키위	300원	<a href="#">삭제</a>	<a href="#">수정</a>
4	파란사과	500원	<a href="#">삭제</a>	<a href="#">수정</a>
5	블루베리	500원	<a href="#">삭제</a>	<a href="#">수정</a>
6	체리	1000원	<a href="#">삭제</a>	<a href="#">수정</a>
7	메론	1000원	<a href="#">삭제</a>	<a href="#">수정</a>
8	수박	2000원	<a href="#">삭제</a>	<a href="#">수정</a>

[추가하기](#)

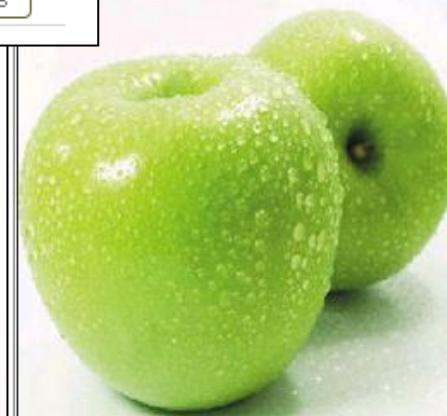


상품 추가 화면

상품명	참외
가격	1000
설명	이건 설말
그림 파일	lemon.jp

[추가하기](#)

상품 상세 화면



상품명	파란사과
가격	500 원
설명	단맛이 강한 향기가 상큼한 파란 사과입니다.

[상품 리스트 화면으로 돌아감](#)

# 실습 : SangpumInfo

- 최초 시작 파일 : <http://localhost:포트번호/SangpumInfo/main.jsp>

시작

web.xml

1) ApplicationContext(스프링) 설정 파일  
contextConfigLocation 파라미터를 이용한다.  
Bootstrap listener 등록 : ContextLoaderListener

Database

2) 문자열 인코딩 설정하기

3) DispatcherServlet 설정  
예제에서는 확장자 prd에 대한 맵핑을 수행한다.  
서블릿 이름 : product  
개발자가 직접 <init-param> 태그를 이용하여 실제 파일의 위치를 지정한다.

ApplicationContext 설정 파일 작성(applicationContext.xml)

1) DBCP 를 이용한 DataSource 설정  
2) SqlMapClient 객체 설정  
(1) setter Injection  
dataSource  
configLocation : iBatis 맵 설정 파일의 이름과 경로를 의미.

맵 설정 파일

맵핑 파일

MVC용 설정 파일 작성(prod-servlet.xml)

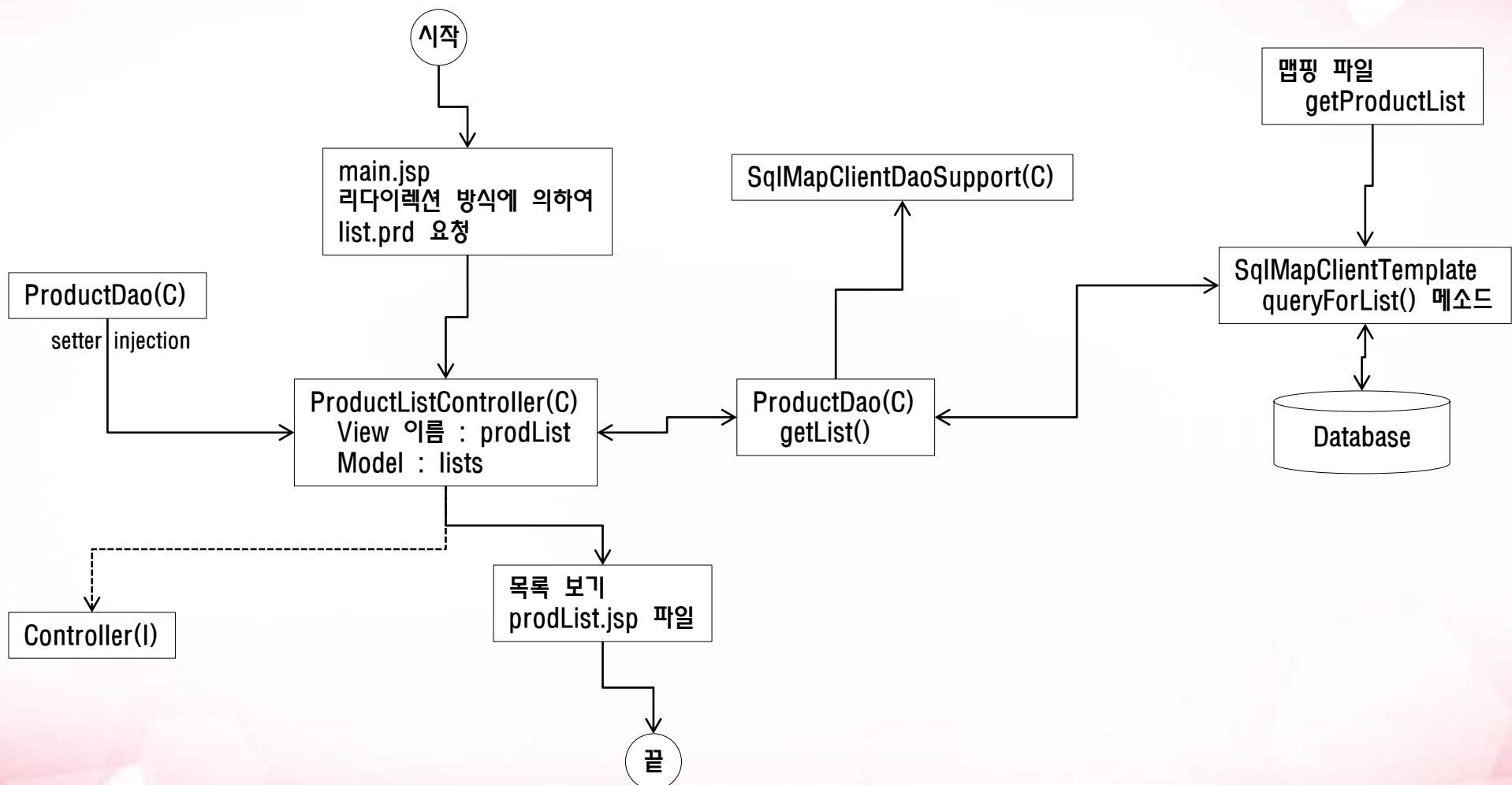
1) DAO 설정하기  
sqlMapClient 객체를 setter Injection 시킨다.  
2) HandlerMapping 설정 : BeanNameUrlHandlerMapping  
3) 컨트롤러 설정하기  
(1) 업무별 Controller 설정하기  
4) ViewResolver 설정 : InternalResourceViewResolver  
(1) 접두사 : "/product/"  
(2) 접미사 : ".jsp"

Dao 객체  
ProductDao

끝

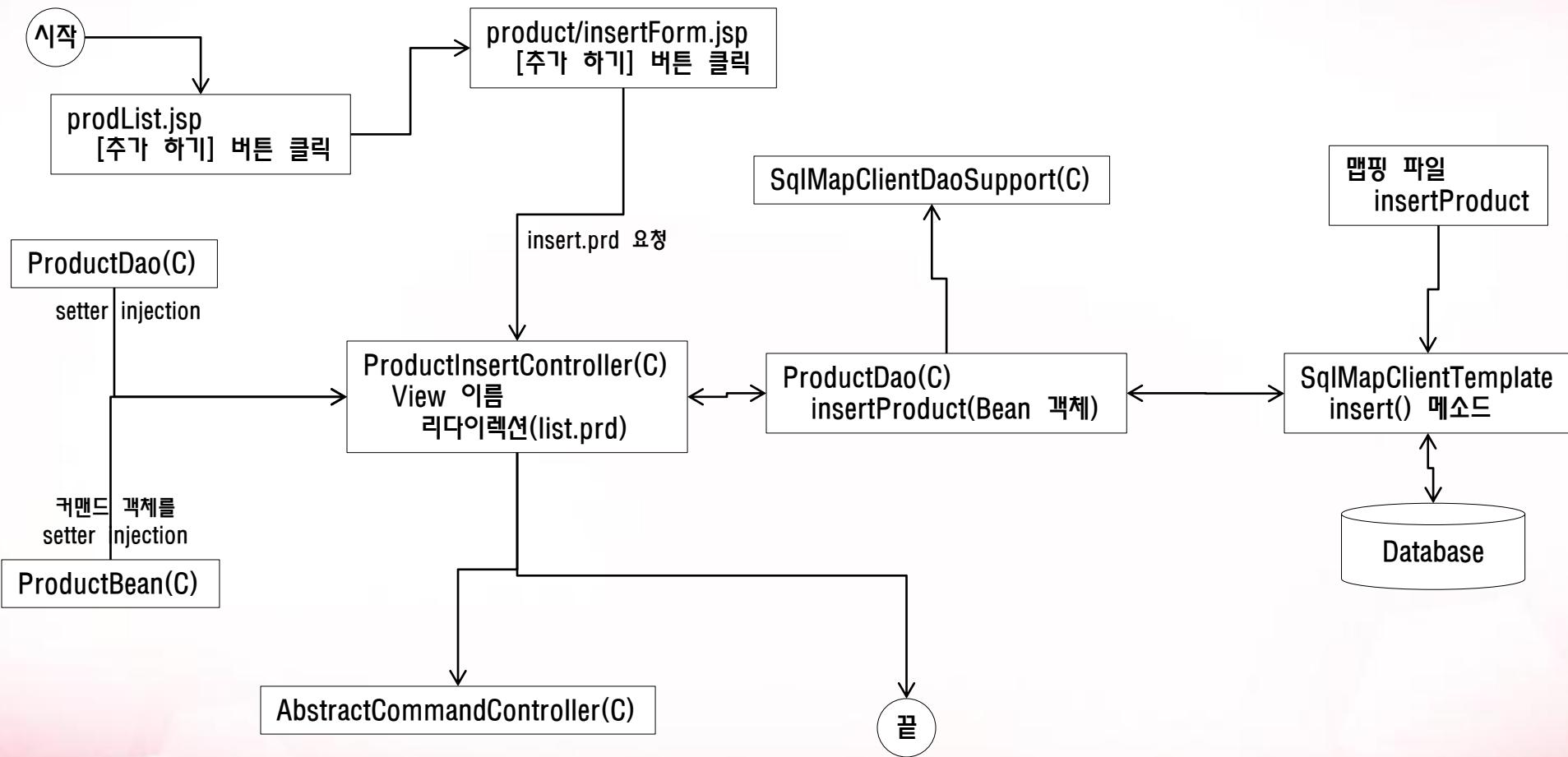
# 실습 : SangpumInfo

- 최초 시작 : 목록 보기 페이지로 이동한다..



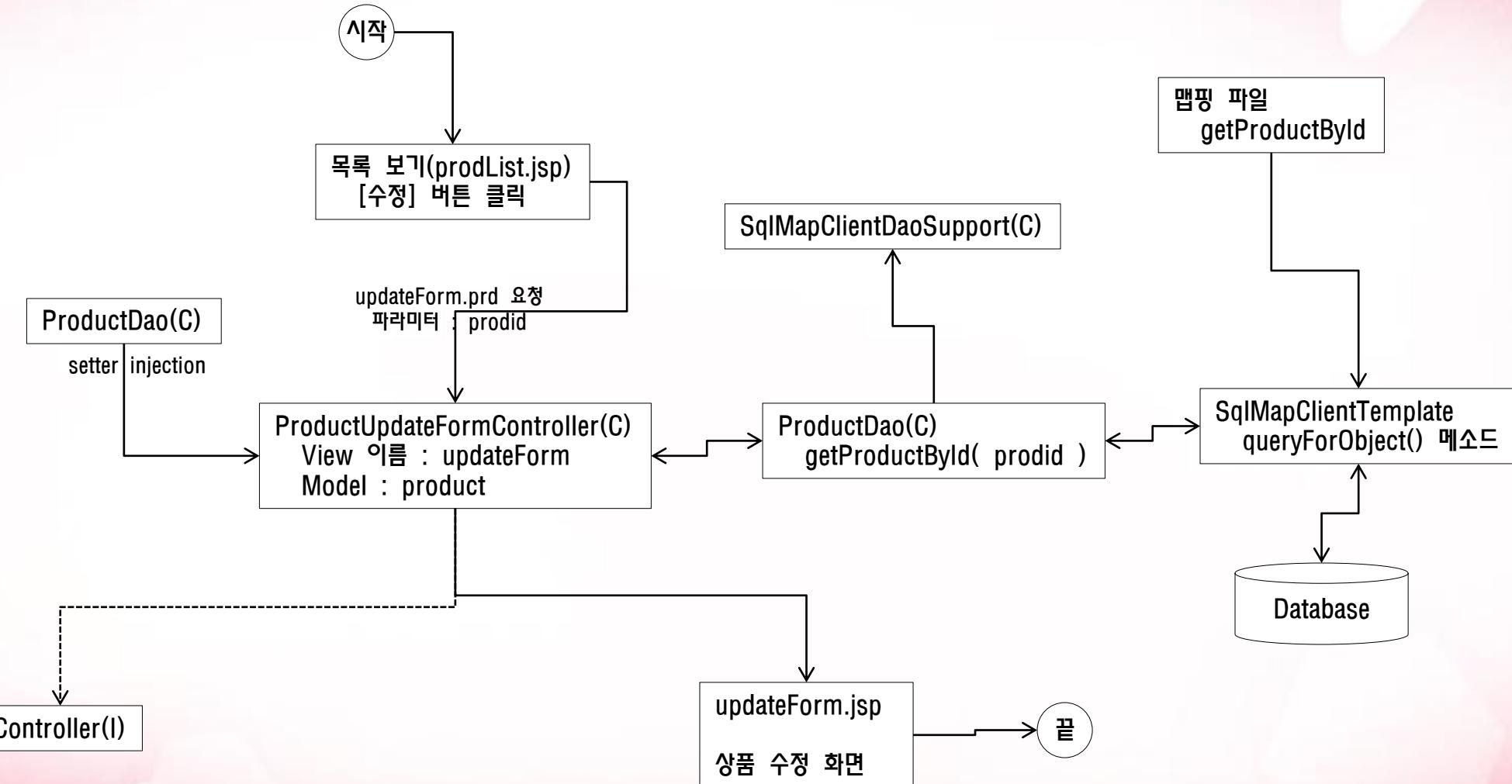
# 실습 : SangpumInfo

- 상품 추가하기



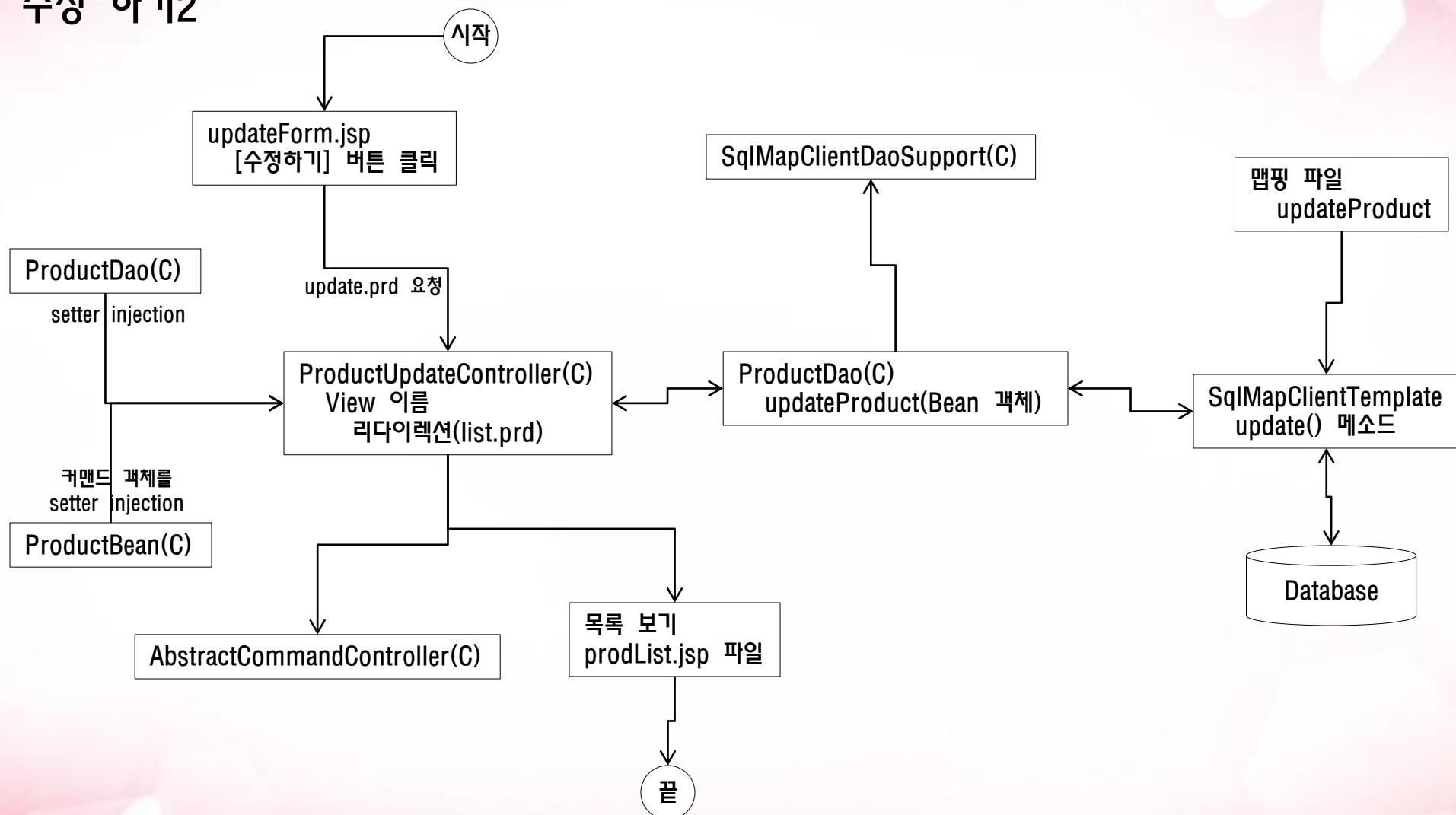
# 실습 : SangpumInfo

- 수정 하기1



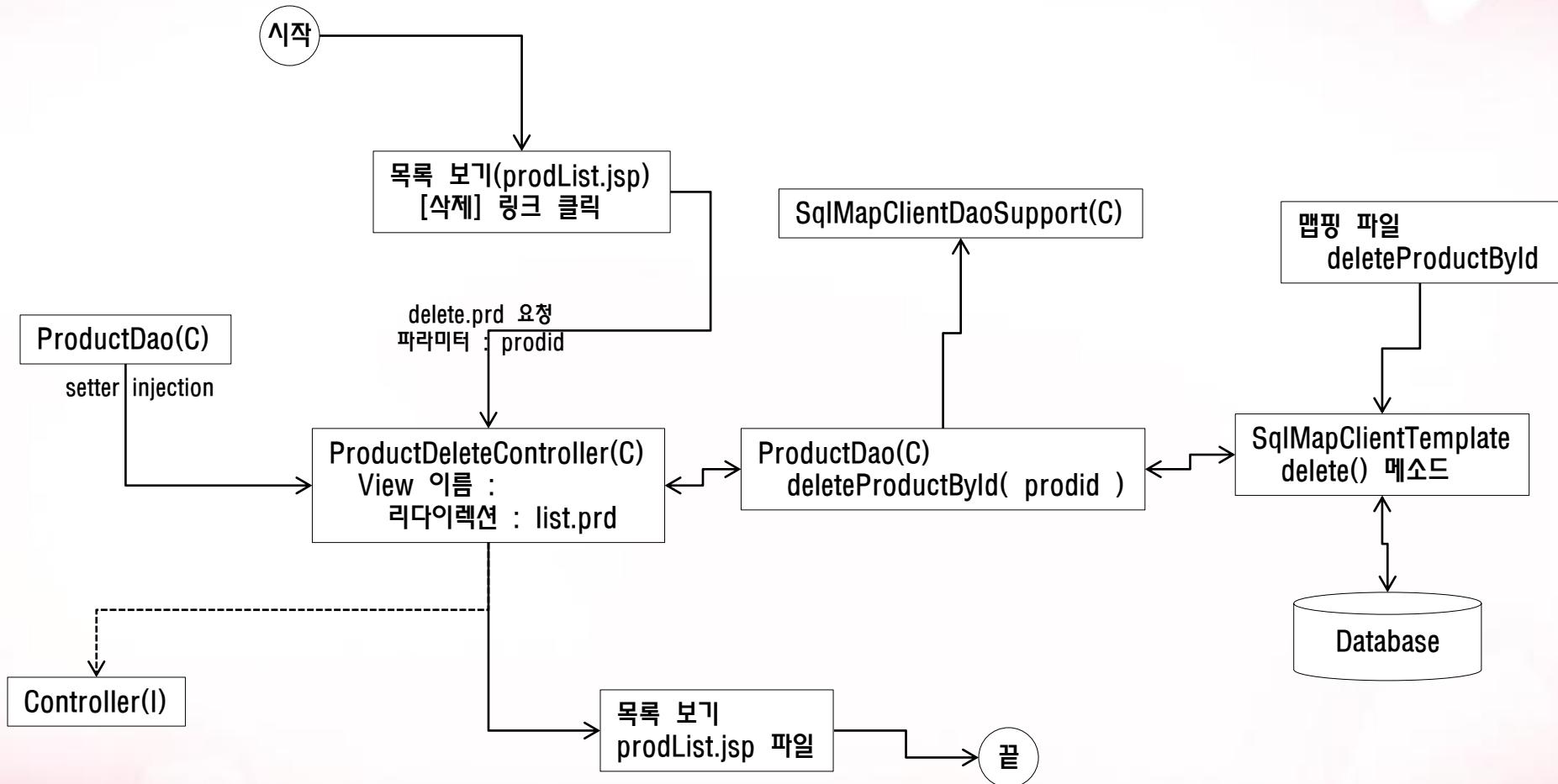
# 실습 : SangpumInfo

- 수정 하기2



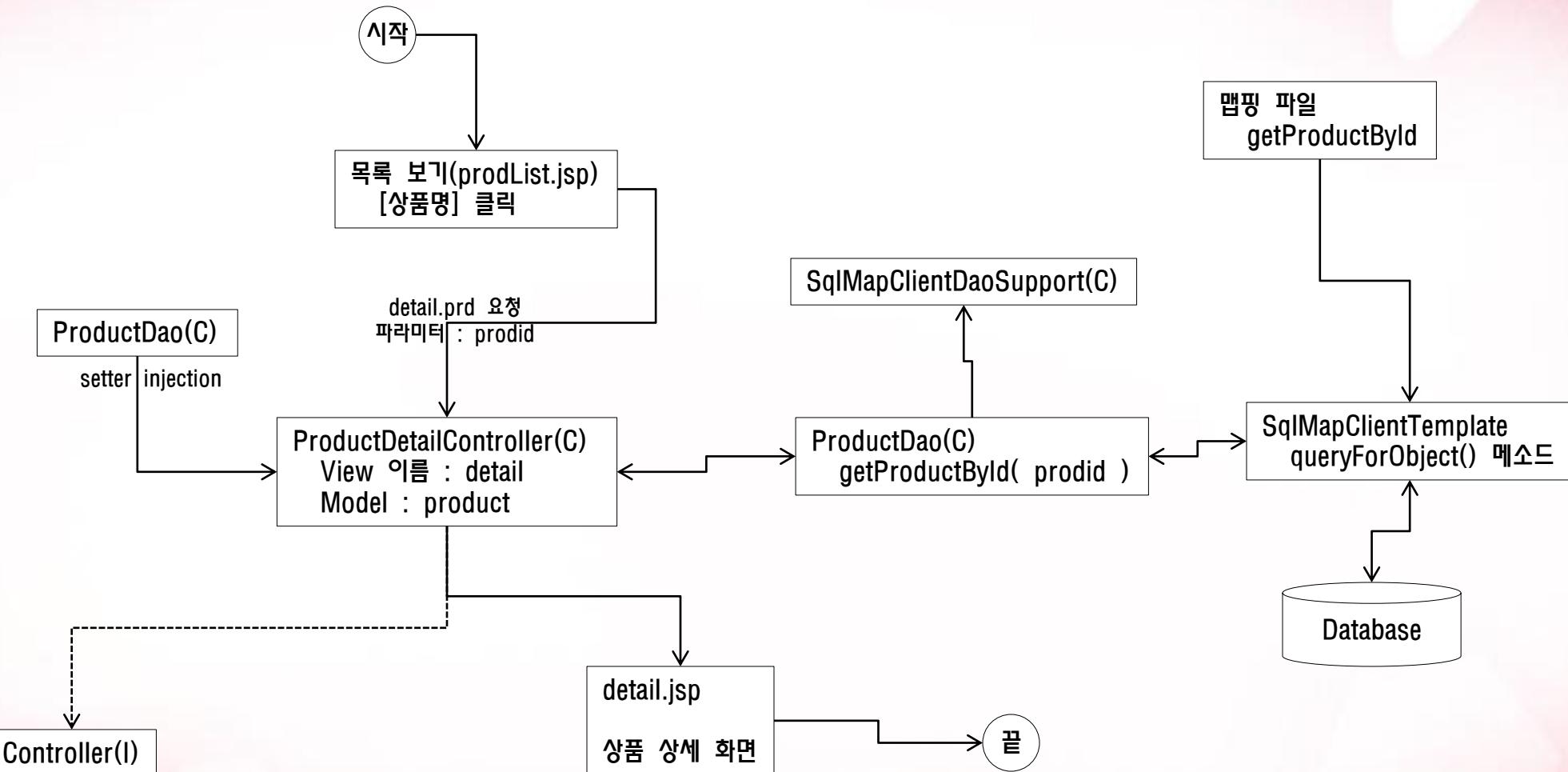
# 실습 : SangpumInfo

- 삭제하기



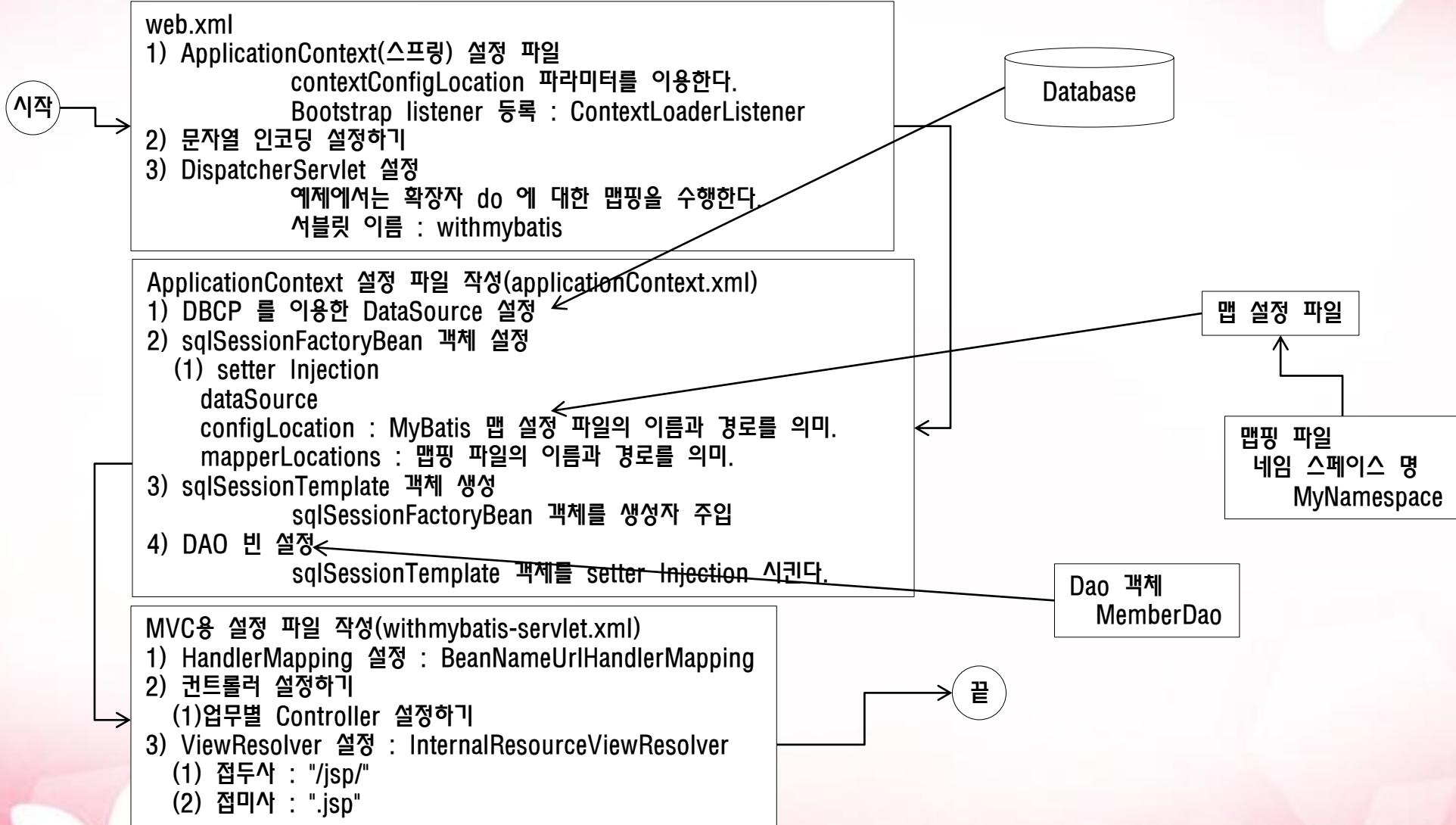
# 실습 : SangpumInfo

- 상품 상세 보기



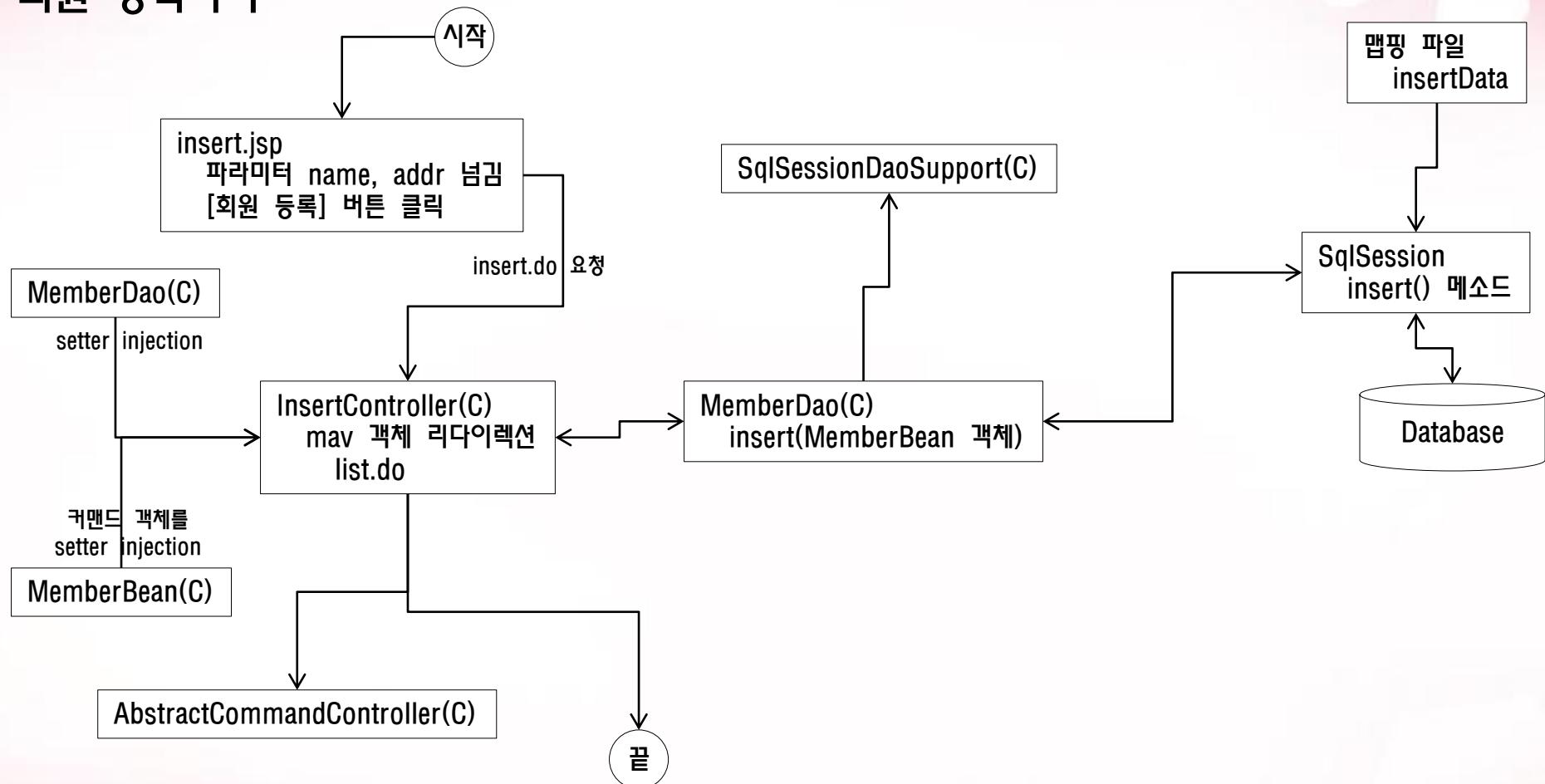
# 실습 : MySpringMybatis

- 최초 시작 파일 : <http://localhost:포트번호/MySpringMybatis/jsp/insert.jsp>



# 실습 : MySpringMybatis

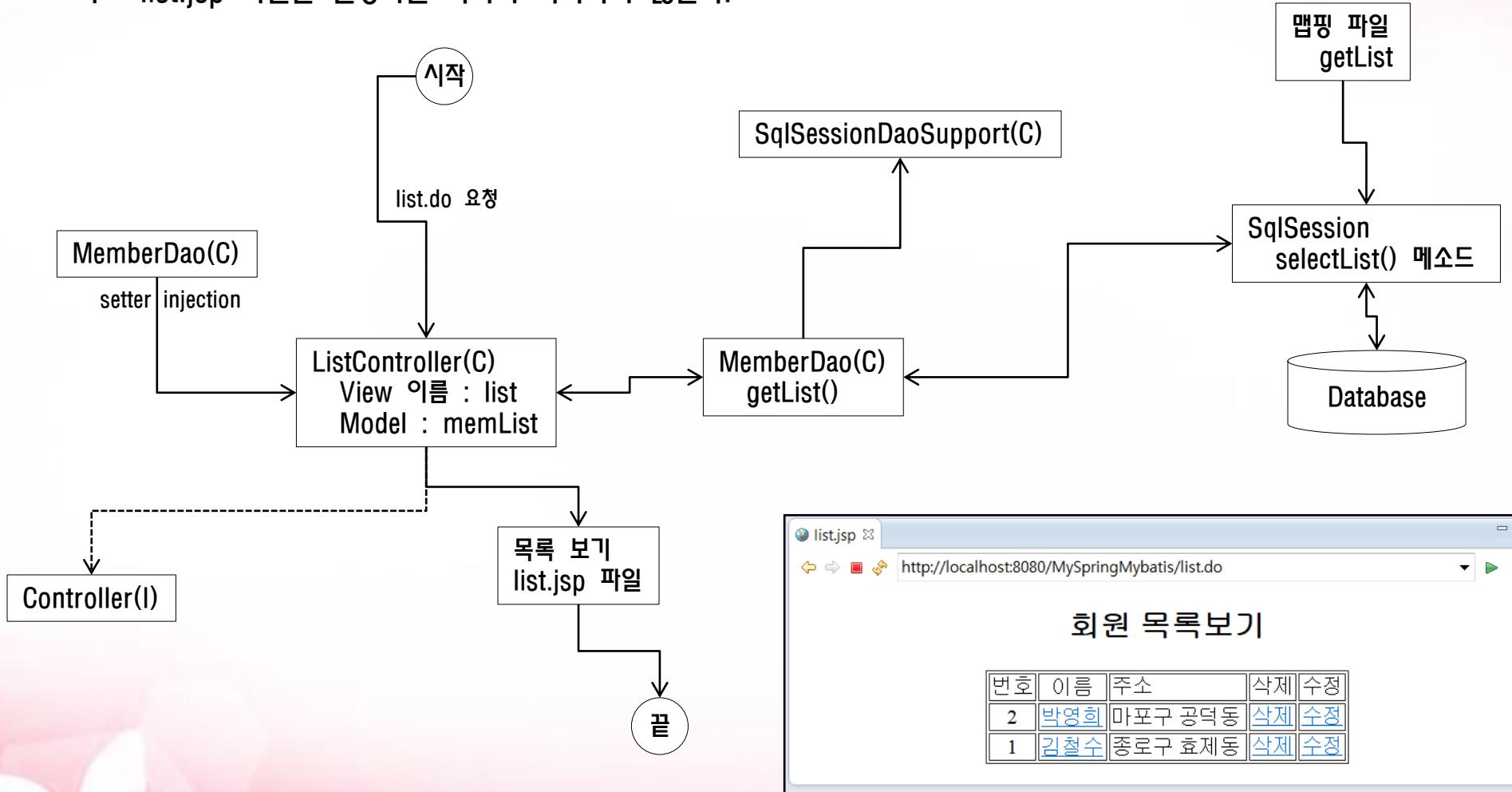
- 회원 등록하기



# 실습 : MySpringMybatis

## • 목록 보기

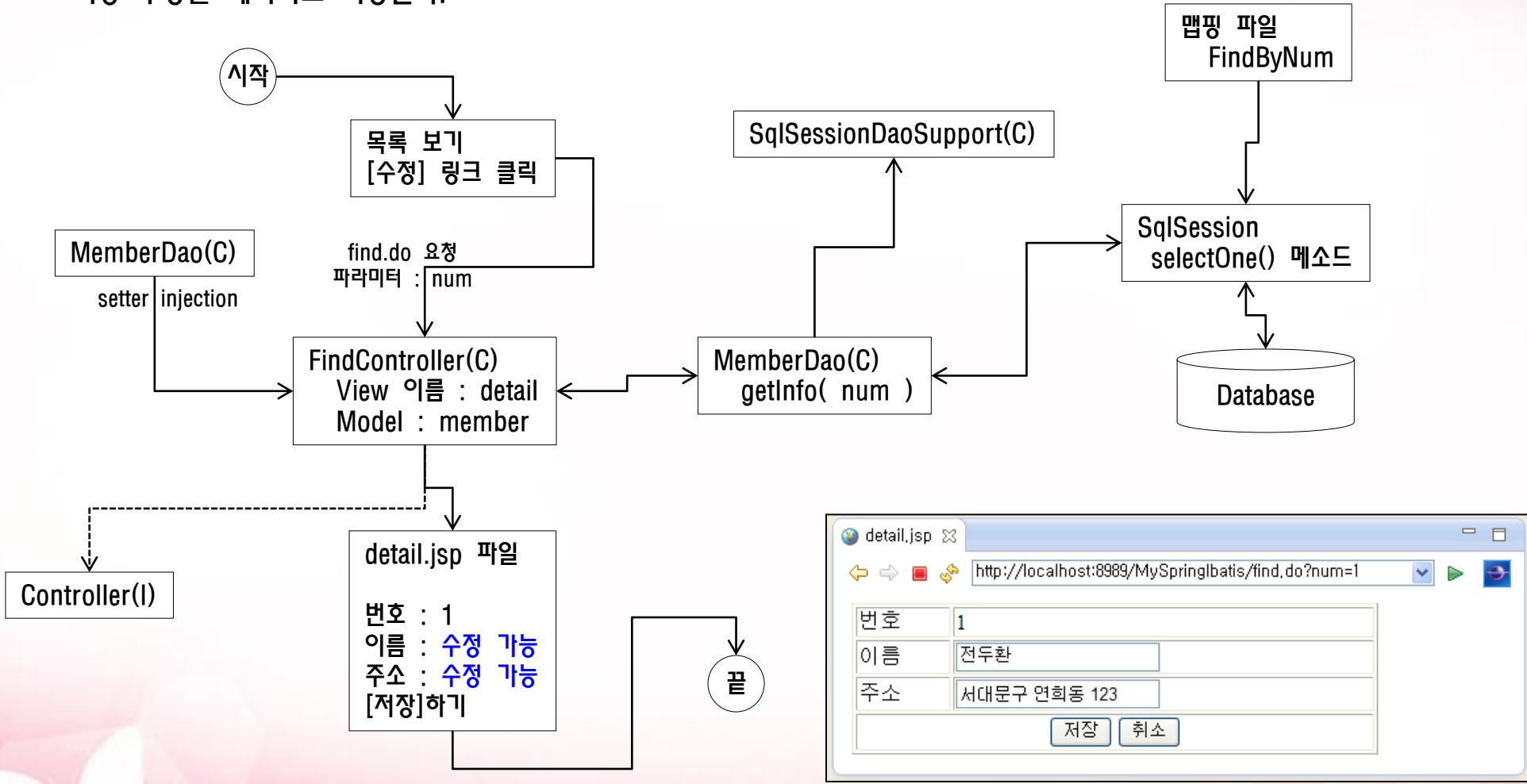
- 주의 : 회원 등록을 하고 나서, 목록 보기 페이지로 데이터가 넘어온다.
- 최초 list.jsp 파일을 실행하면 목록이 나타나지 않는다.



# 실습 : MySpringMybatis

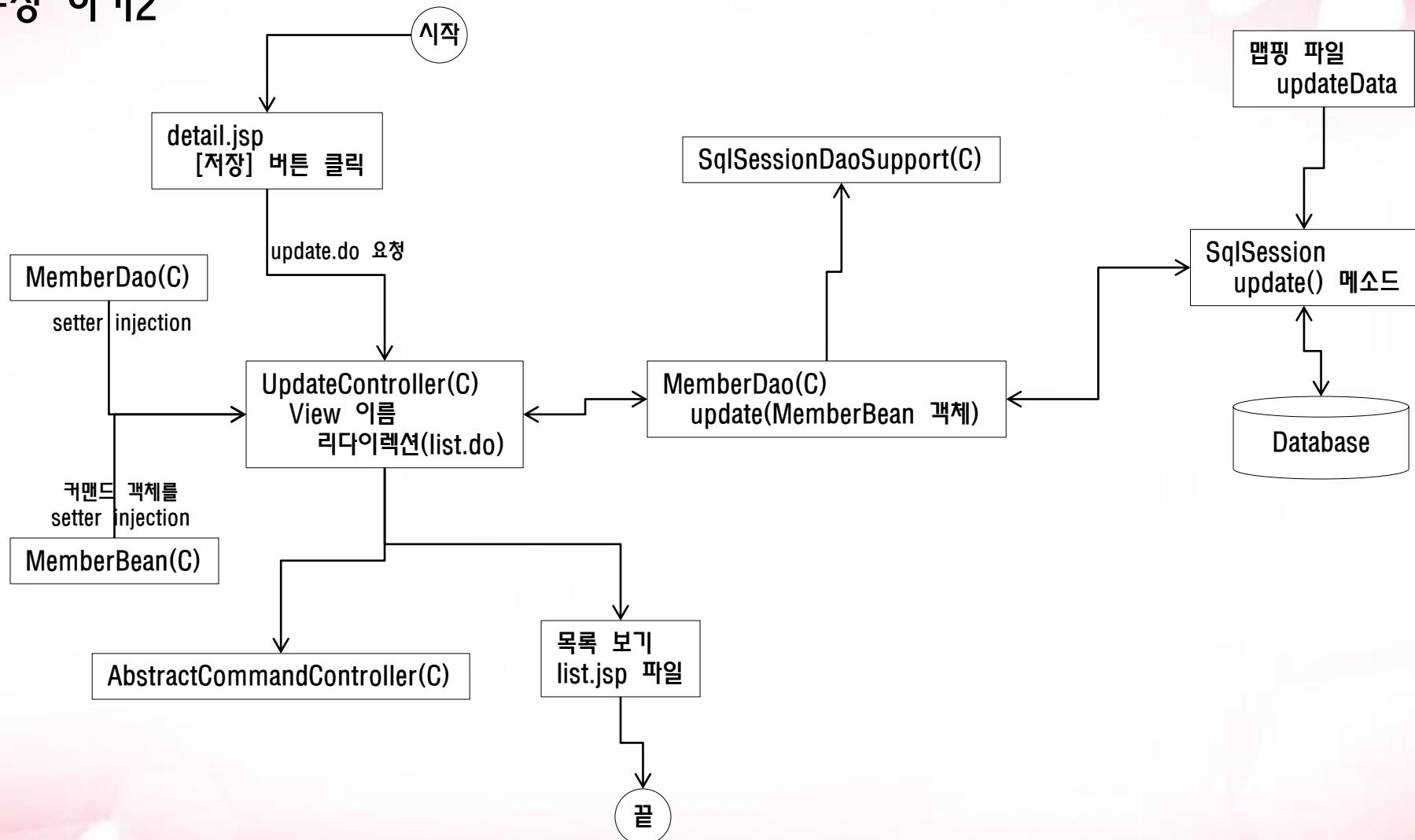
## • 수정 하기1

- 수정하기 위해서 일단 해당 Bean을 찾아서 목록으로 보여준다.
- 최종 수정할 페이지로 이동한다.



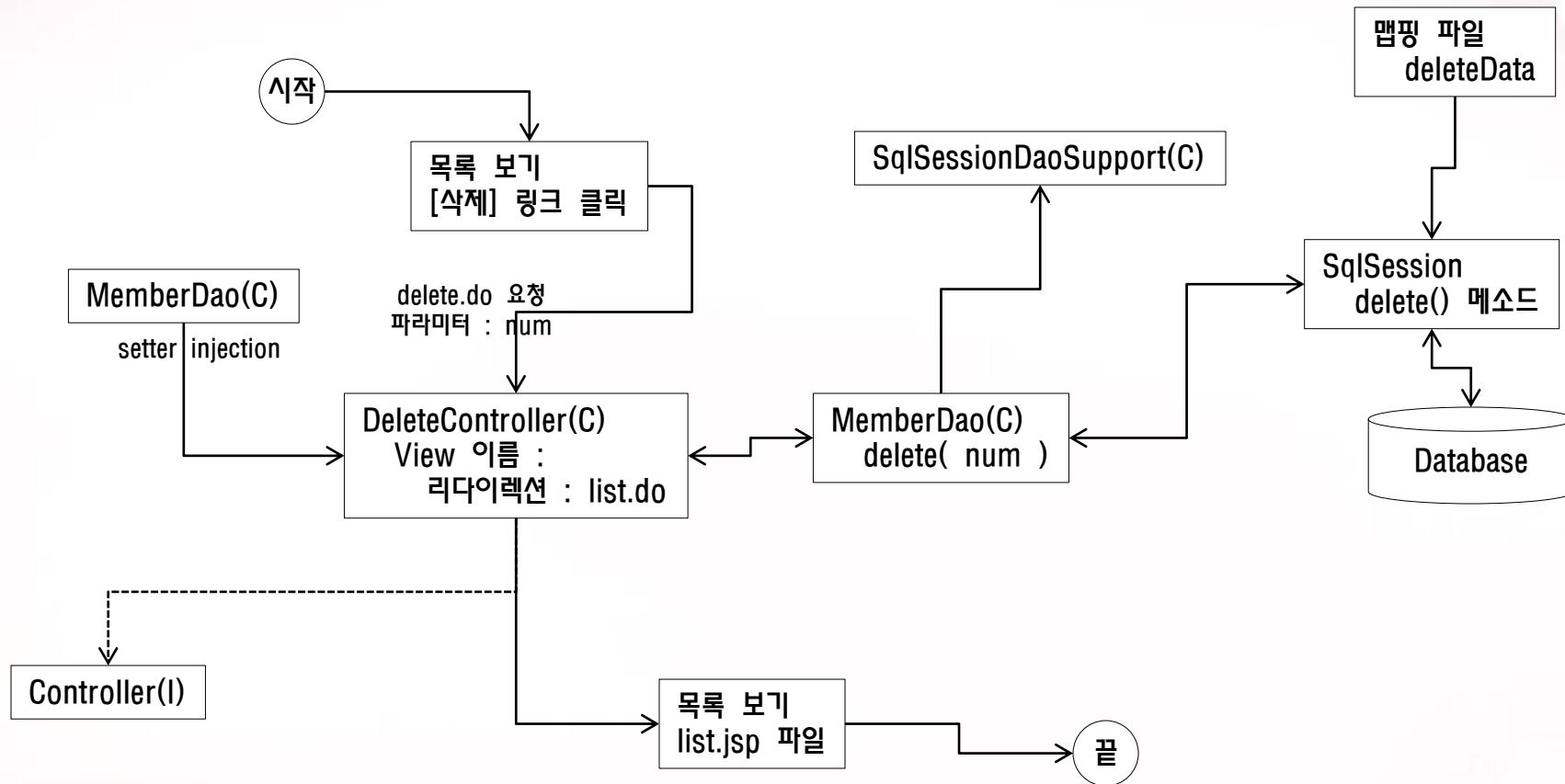
# 실습 : MySpringMybatis

## • 수정 하기2



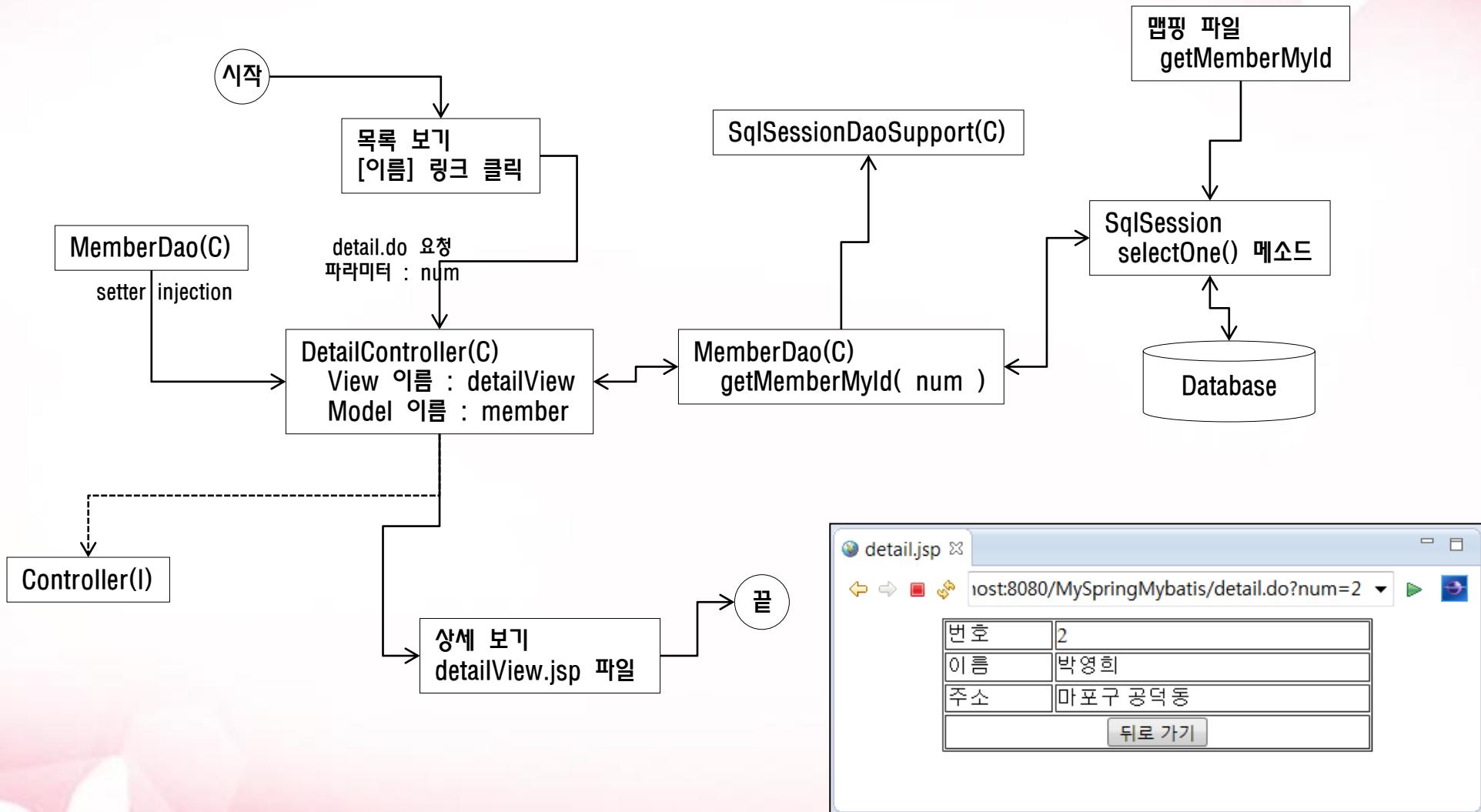
# 실습 : MySpringMybatis

- 삭제 하기



# 실습 : MySpringMybatis

- 상세 보기

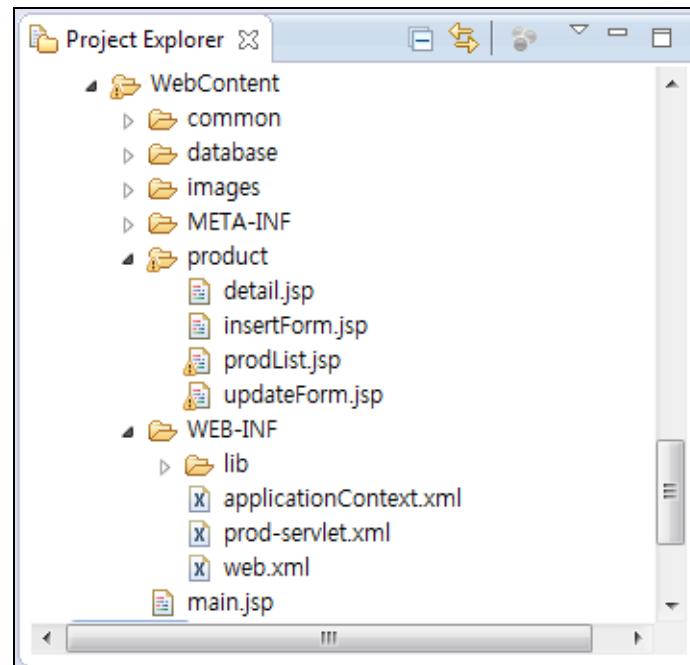
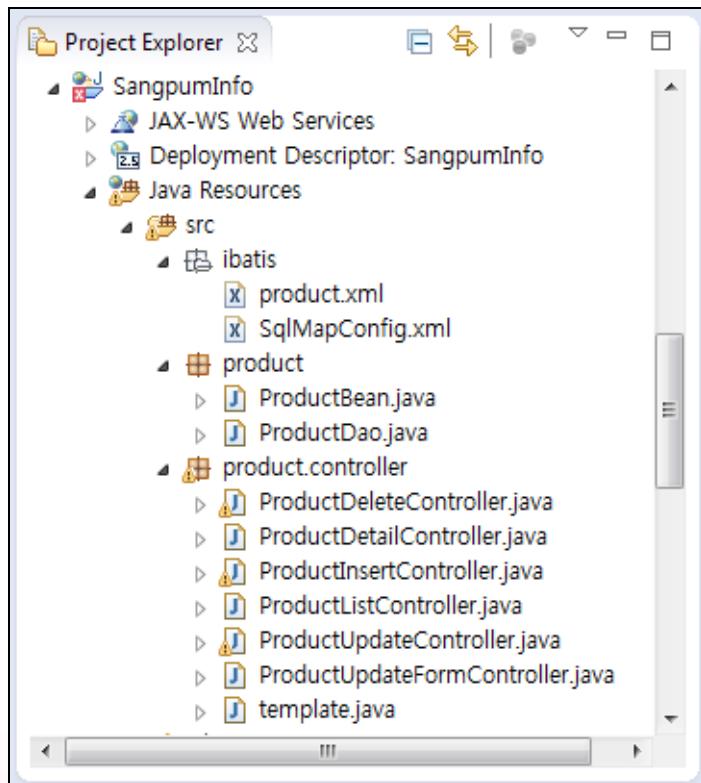


# 실습 : ProductInfo

- 프로젝트 : ProductInfo
- 스프링과 MyBatis를 이용하여 상품에 대한 정보 보기 페이지를 구축한다.
- main.jsp 파일을 실행하여 프로그램을 시작한다.

주요 파일 리스트	설명
main.jsp	최초 실행 파일(실행하게 되면 /list.prd으로 리다이렉션된다.)
creatable.txt	테이블 생성 스크립트
ProductBean.java	상품에 대한 Bean 클래스 파일
ProductDao.java	Data Access Object
web.xml	배포 서술자 파일
applicationContext.xml	스프링 설정 파일(dataSource 및 MyBatis 설정 파일)
prod-servlet.xml	MVC 설정 정보를 담고 있는 파일
SqlMapConfig.xml	SQL 맵 설정 파일
product.xml	맵핑 파일
ProductXXXController.java	상품 관련한 컨트롤러 파일들

# 실습 : ProductInfo



# 실습 : ProductInfo

- 최초 시작 파일 : <http://localhost:포트번호/ProductInfo/main.jsp>

시작

web.xml

- 1) ApplicationContext(스프링) 설정 파일  
contextConfigLocation 파라미터를 이용한다.  
Bootstrap listener 등록 : ContextLoaderListener

Database

- 2) 문자열 인코딩 설정하기
- 3) DispatcherServlet 설정  
예제에서는 확장자 prd에 대한 맵핑을 수행한다.  
서블릿 이름 : product  
개발자가 직접 <init-param> 태그를 이용하여 실제 파일의 위치를 지정한다.

ApplicationContext 설정 파일 작성(applicationContext.xml)

- 1) DBCP 를 이용한 DataSource 설정
- 2) sqlSessionFactoryBean 객체 설정  
(1) setter Injection  
dataSource  
configLocation : MyBatis 맵 설정 파일의 이름과 경로를 의미.  
mapperLocations : 맵핑 파일의 이름과 경로를 의미.
- 3) sqlSessionTemplate 객체 생성  
sqlSessionFactoryBean 객체를 생성자 주입
- 4) DAO 빈 설정  
sqlSessionTemplate 객체를 setter Injection 시킨다.

맵 설정 파일

맵핑 파일

Dao 객체  
ProductDao

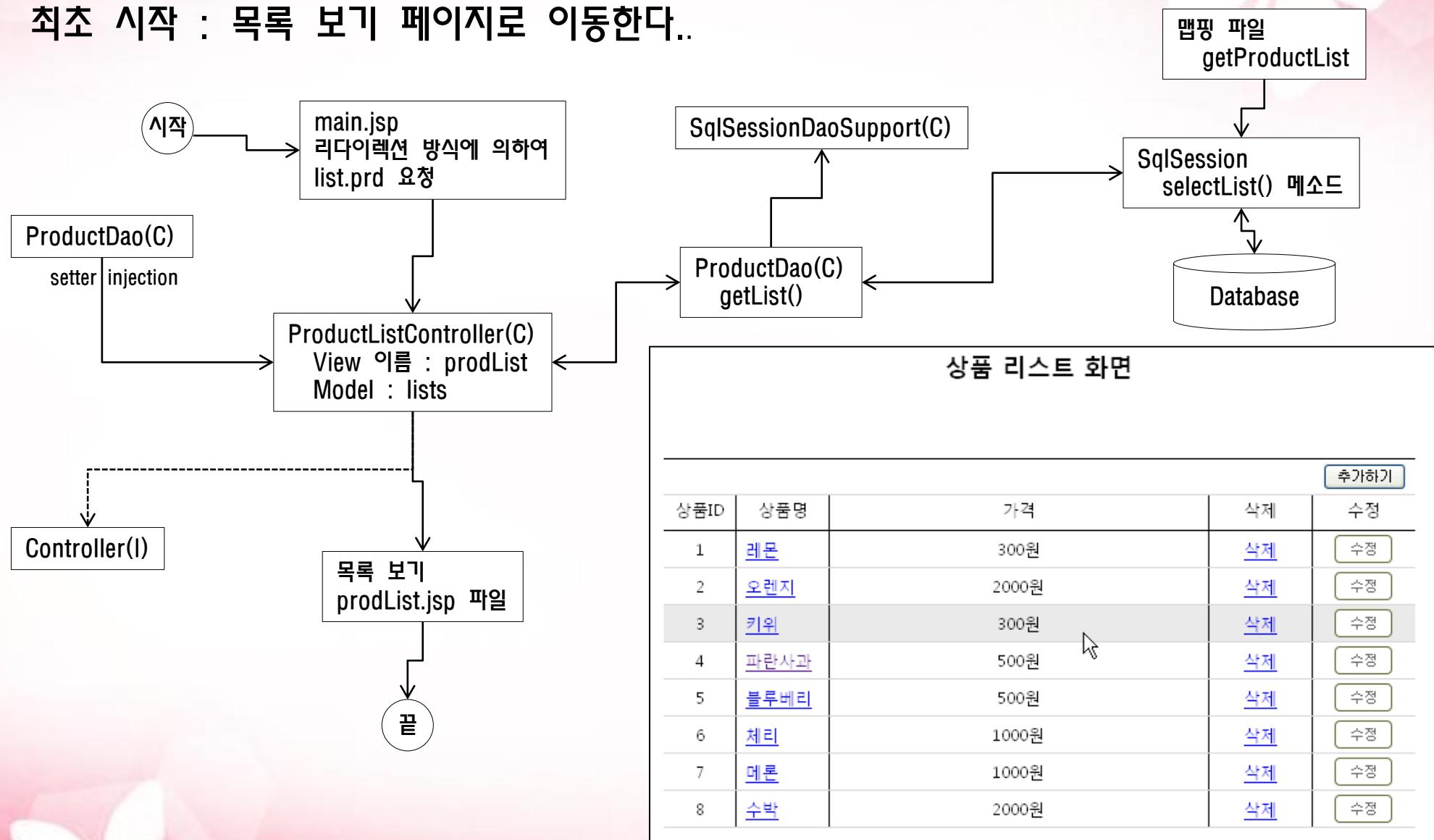
MVC용 설정 파일 작성(prod-servlet.xml)

- 1) HandlerMapping 설정 : BeanNameUrlHandlerMapping
- 2) 컨트롤러 설정하기  
(1) 업무별 Controller 설정하기
- 3) ViewResolver 설정 : InternalResourceViewResolver  
(1) 접두사 : "/product/"  
(2) 접미사 : ".jsp"

끝

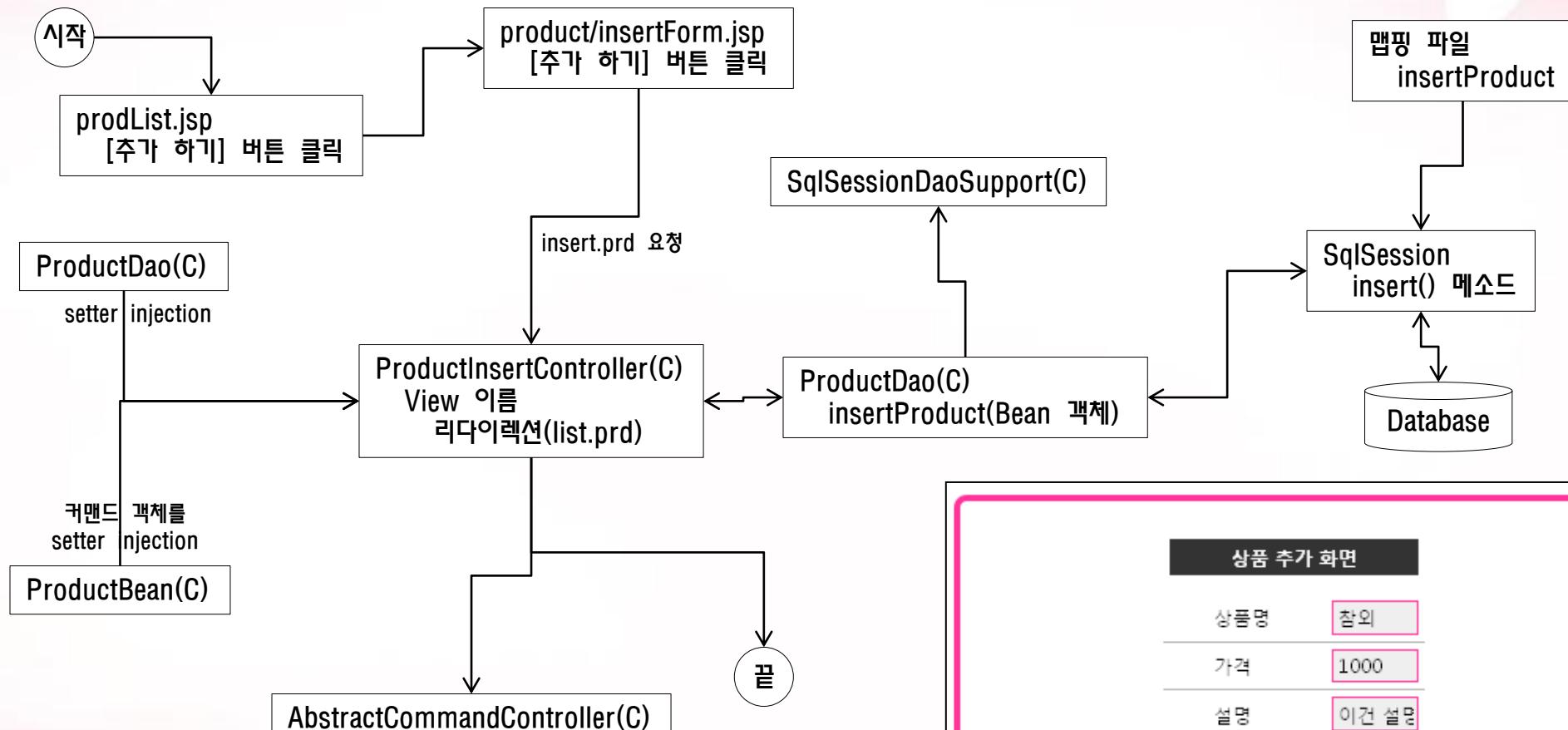
# 실습 : ProductInfo

- 최초 시작 : 목록 보기 페이지로 이동한다..



# 실습 : ProductInfo

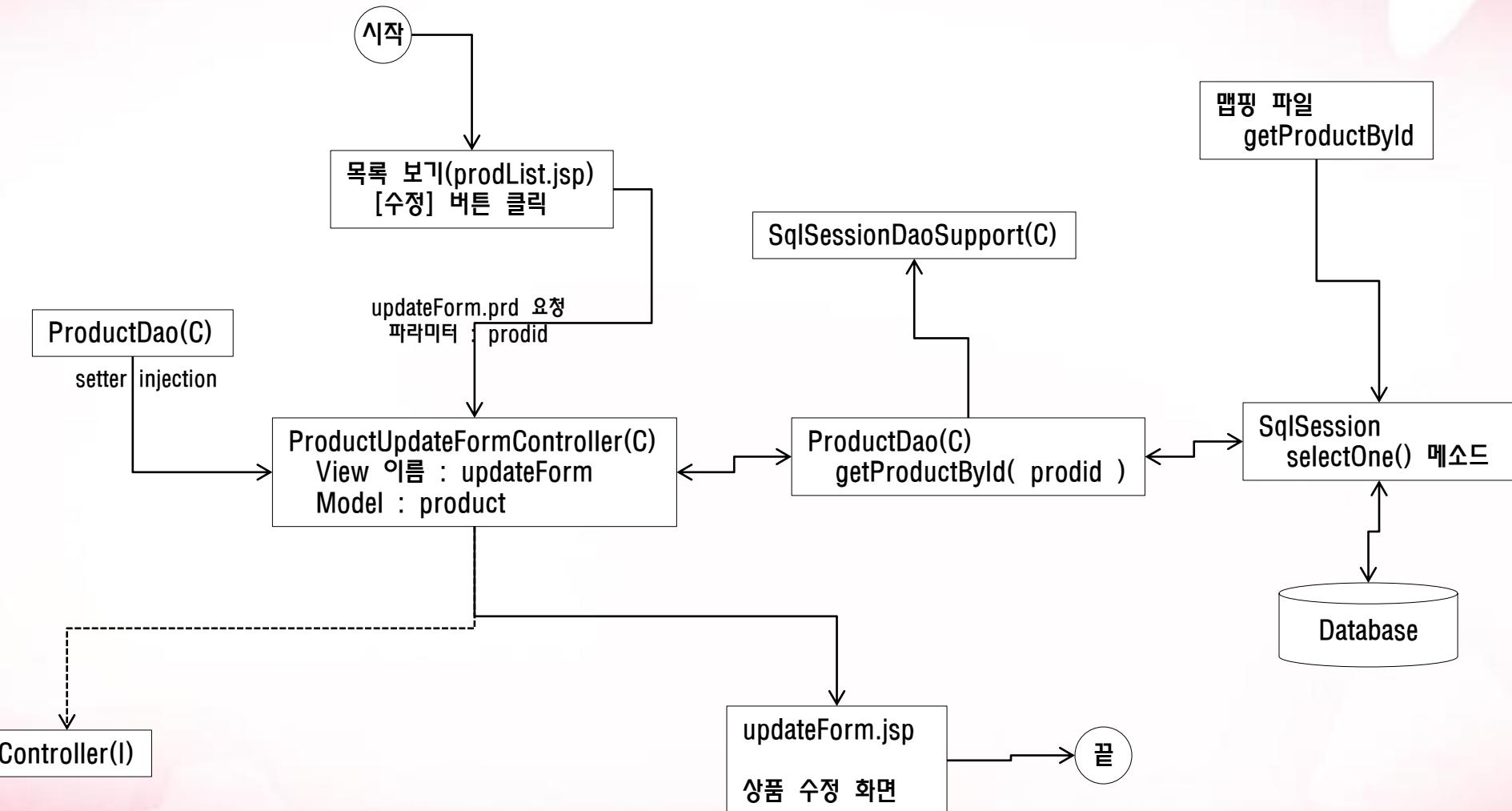
- 상품 추가하기



상품 추가 화면	
상품명	참외
가격	1000
설명	이건 설명
그림 파일	lemon.jp
추가하기	

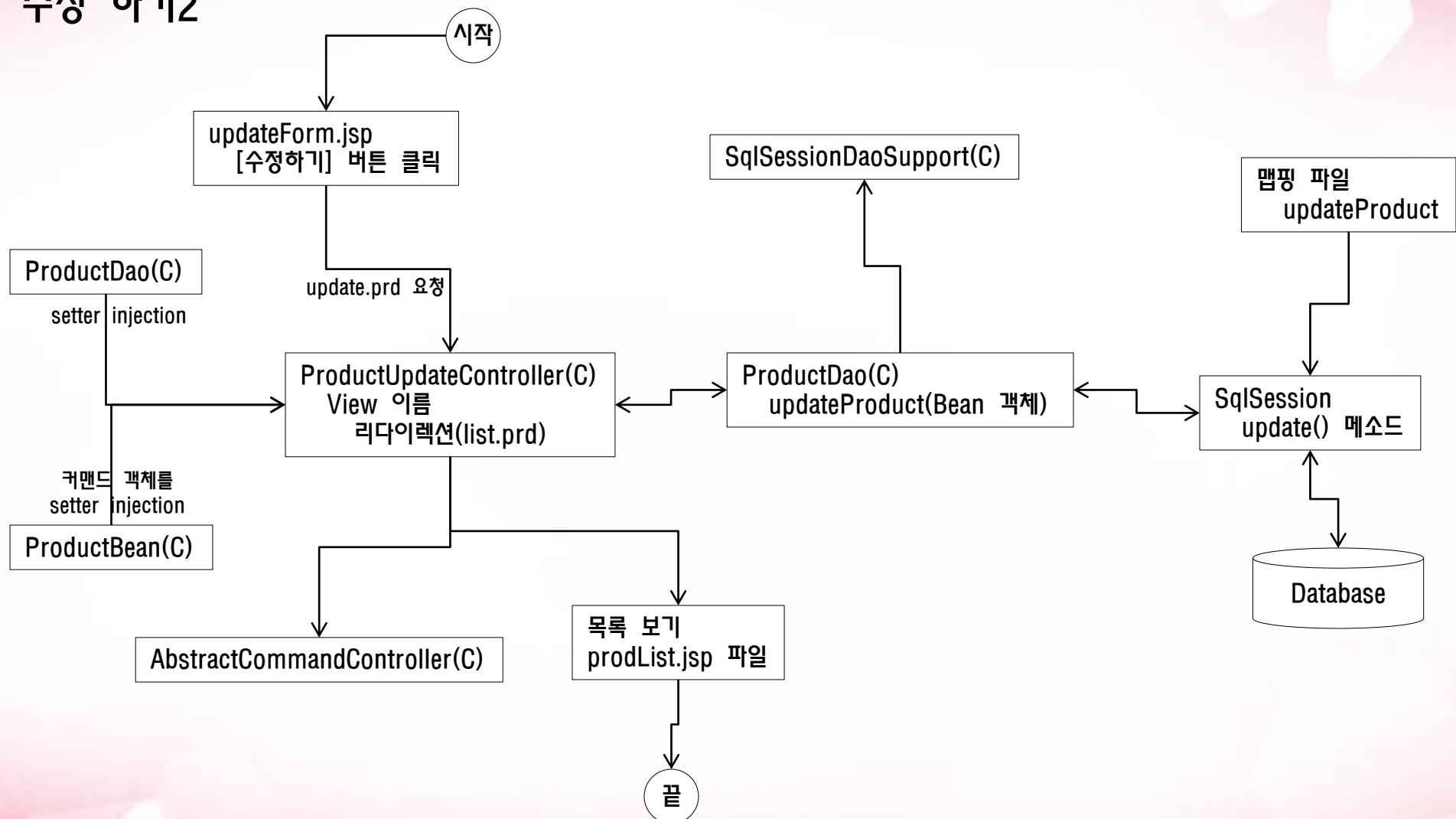
# 실습 : ProductInfo

- 수정 하기1



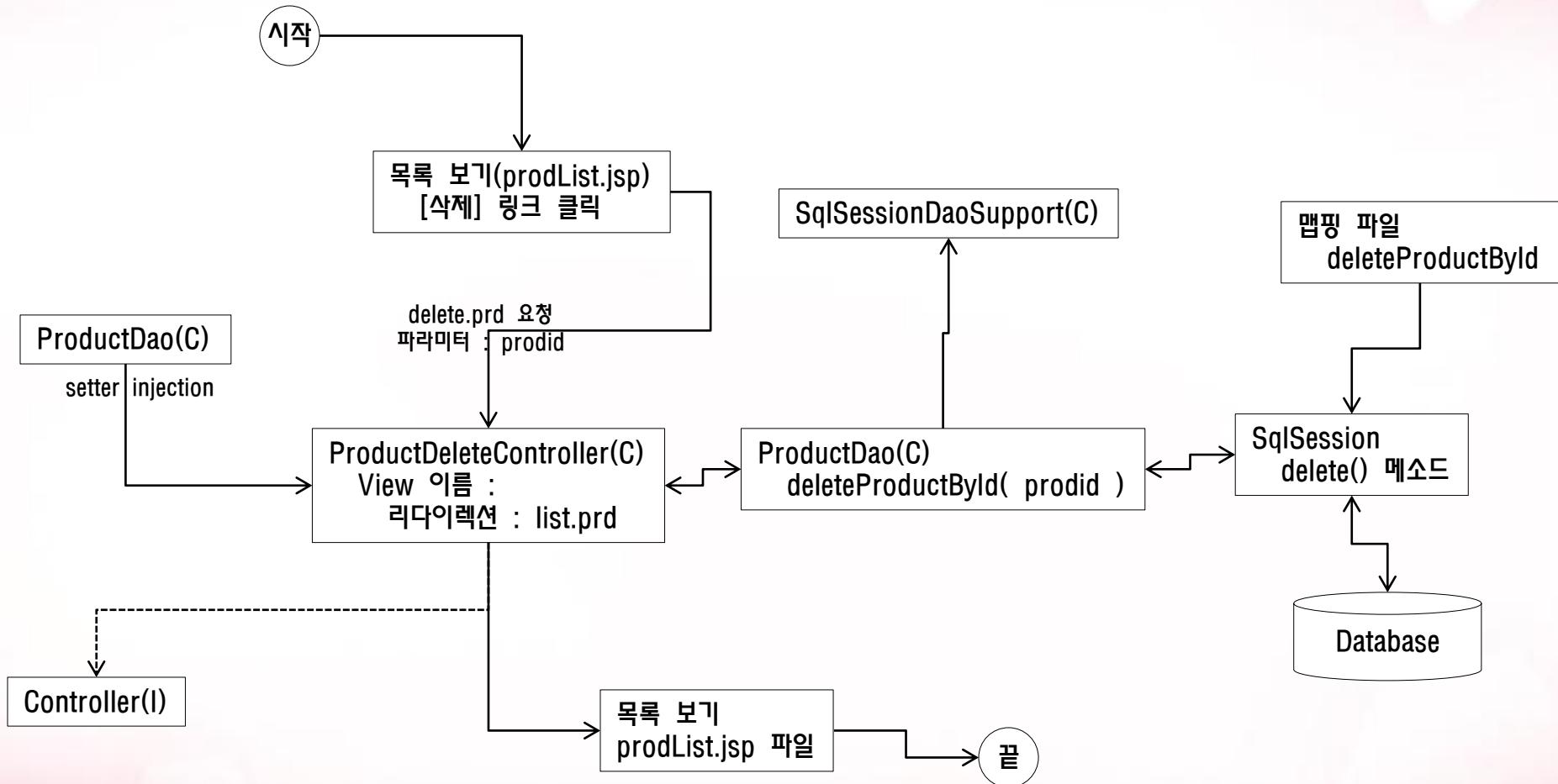
# 실습 : ProductInfo

- 수정 하기2



# 실습 : ProductInfo

- 삭제하기



# 실습 : ProductInfo

- 상품 상세 보기

