

스프링 MVC

강사 : 강병준

Architecture

View - Controller - Service - Dao - Model - DB 이런 구조로 데이터가 흘러다닌다.

DAO(Data Access Object) 와 Service 의 차이점은

DAO :

단일 데이터 접근

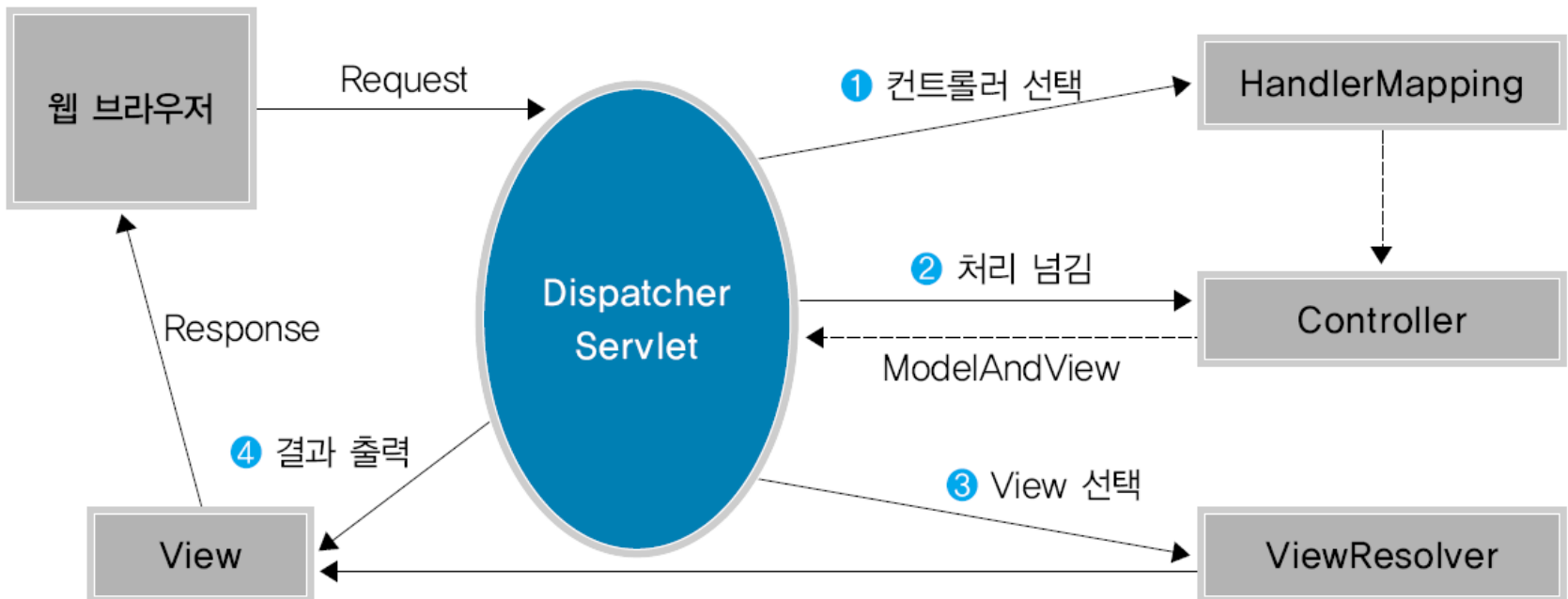
Service :

여러 DAO를 호출하여 여러 번의 데이터 접근/갱신을 하며
이렇게 읽은 데이터에 대해 비즈니스 로직을 수행하고
하나의 트랜잭션으로 묶는다.

Service 와 DAO 가 동일해지는 경우도 있는데,
이때는 비즈니스 로직이 단일 DB접근으로 끝나기 때문이다.

스프링 MVC 패턴 애플리케이션 개요

브라우저로부터 송신된 요청은 모두 스프링 MVC에 의해 제공되는 DispatcherServlet 클래스에 의해 관리되고 있다.



▲ 스프링 MVC 처리 흐름

스프링 MVC 패턴 애플리케이션 개요

스프링 MVC는 `org.springframework.web` 패키지와 `org.springframework.web.servlet` 패키지에 포함된 클래스를 사용

구성요소	개요
DispatcherServlet	브라우저로부터 송신된 Request를 일괄적으로 관리한다.
HandlerMapping	RequestURL과 Controller 클래스의 맵핑을 관리한다. [묵시적 사용]
Controller	비즈니스 로직을 호출하여 처리 결과의 ModelAndView 인스턴스를 반환한다.
ViewResolver	Controller 클래스로부터 반환된 View명을 기본으로 이동처가 되는 View 인스턴스를 해결한다. [묵시적 사용]
View	프레젠테이션층으로의 출력 데이터를 설정한다. [묵시적 사용]

1. 처리 흐름

- 1) 클라이언트의 요청이 DispatcherServlet에 전달되어서 HandlerMapping을 사용하여 클라이언트의 요청을 처리할 Controller 객체를 선택합니다.
- 2) Controller 객체는 비즈니스 로직을 호출해서 처리하고 요청 처리 결과 정보를 ModelAndView 객체를 리턴합니다.
- 3) DispatcherServlet은 넘겨받은 ModelAndView 객체를 이용해서 ViewResolver로부터 응답 결과를 출력할 뷰 객체를 구해서 출력합니다.

2. 스프링 MVC 웹 애플리케이션을 개발 과정

- 1) 클라이언트의 요청을 받을 DispatcherServlet을 web.xml에 등록
- 2) 클라이언트의 요청을 처리할 Controller 클래스를 생성
- 3) ViewResolver를 설정
- 4) JSP나 HTML 또는 Velocity를 이용해서 출력 코드 작성

스프링 MVC

DispatcherServlet 설정은 web.xml 파일에 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns=http://java.sun.com/xml/ns/javaee
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" version="2.5">
  <servlet>
    <servlet-name>서블릿 이름</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>서블릿 이름</servlet-name>
    <url-pattern>주소패턴</url-pattern>
  </servlet-mapping>
</web-app>
```

위처럼 설정하게 되면 주소패턴에 맞는 요청이 오면 WEB-INF에 있는 [서블릿이름]-servlet.xml 파일을 설정 파일로 사용하여 읽어내게 됩니다.

설정 예

```
<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>*.do</url-pattern>
</servlet-mapping>
```

do로 끝나는 요청이 오면 WEB-INF 폴더에 있는 dispatcher-servlet.xml 파일에 있는 Controller 객체가 요청을 처리하도록 설정한 것입니다.

스프링 MVC

1. DispatcherServlet 설정은 내부적으로 스프링 컨테이너를 생성해서 내부에 설정된 bean 태그의 객체들을 생성합니다.
2. DispatcherServlet 설정에는 아래 3개의 객체를 등록해주어야 하는데 경우에 따라서는 생략이 가능합니다.
 - 1) HandlerMapping 객체
 - 2) HandlerAdapter 객체
 - 3) ViewResolver 객체
3. DispatcherServlet 설정 파일에 mvc 네임 스페이스를 추가하고 <mvc:annotation-driven /> 태그를 추가하는 경우가 있습니다.
4. 위 태그를 추가하게 되면 HandlerMapping과 HandlerAdapter 객체를 빈으로 등록해 줍니다.
5. 또한 JSON이나 XML 요청/응답 처리를 위한 ConversionService 객체를 빈으로 등록해 줍니다.

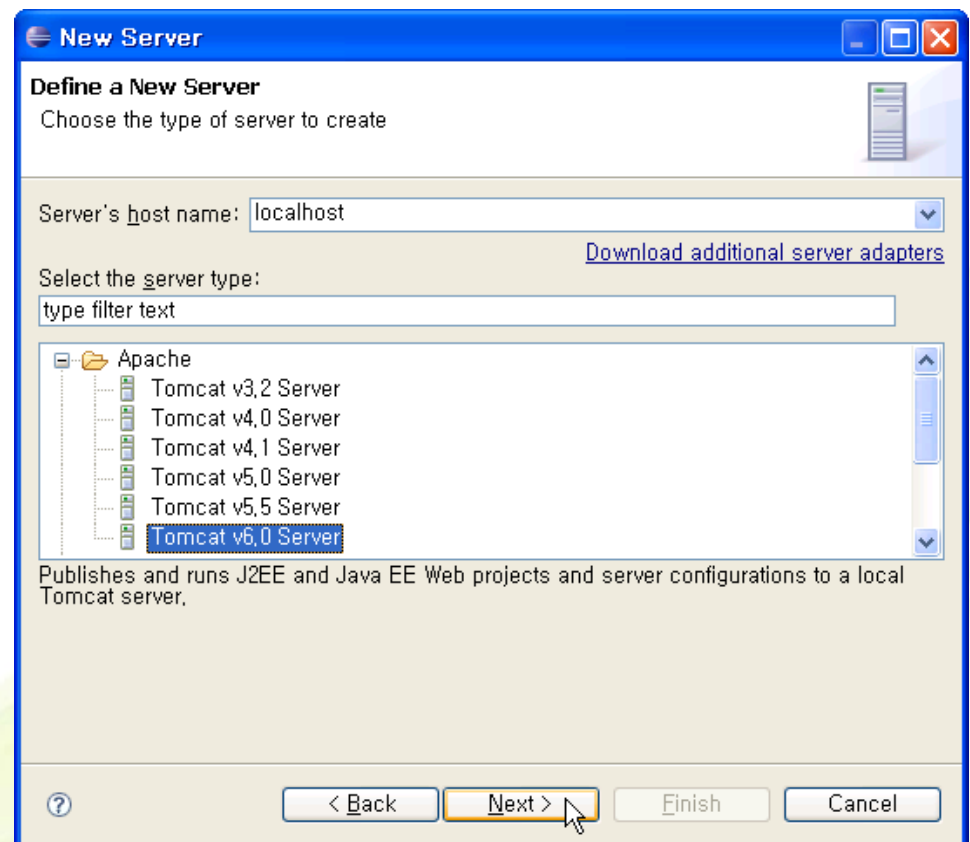
상품 목록 화면 만들기

이클립스에 동적 웹 프로젝트 설정

서버 설정을 하기 위해서 **[File]-[New]-[Other...]** 메뉴를 선택한다.

서버 설정을 하기 위해서 **[File]-[New]-[Other...]** 메뉴를 선택한다.

[Define a New Server]
화면이 나타나면 설정할 서버의 종류를 선택한다.



목차

1. 상품 리스트 화면 작성하기
2. 상품 정보 취득 로직
3. 상품 리스트를 콘솔에 출력하기

스프링MVC

스프링이 제공하는 웹 애플리케이션 구축을 위한 프레임 워크
스프링 웹 애플리케이션을 이용하게되면 모델, 뷰, 컨트롤러 사이에 있는 의존관계를 의존
관계 주입컨테이너인 스프링에서 관리

Org.springframework.web패키지와 **org.springframework.web.servlet** 패키지에
포함된 클래스 사용

애플리케이션 아키텍처

Client

JSP
뷰 계층

스프링
MVC
Presentation

비즈니스
로직층

스프링
JDBC
영속화

RDB

상품 리스트 화면 작성하기

쇼핑 사이트 구축에 첫걸음으로서 상품 정보를 표시하는 화면을 작성한다.



상품 리스트 화면

상품 ID	상품 명	가격
1	레몬	300원
2	오렌지	2000원
3	키위	300원
4	파란사과	500원
5	블루베리	500원
6	체리	1000원
7	메론	1000원
8	수박	2000원
9	파인애플	2000원

스프링 MVC 패턴 애플리케이션 개요

item



itemid:INTEGER(5)

itemname:VARCHAR(20)

price:INTEGER(20)

description:VARCHAR(255)

pictureurl:VARCHAR(20)

▲ 상품 테이블(Item)의 ER 그림

스프링 MVC 패턴 애플리케이션 개요

Oracle

```
create sequence item_seq start with 1 increment by 1  
nocycle nocache;
```

```
create table item(  
    itemId number(5) ,  
    itemName varchar2(20),  
    price number(6),  
    description varchar2(100),  
    pictureUrl varchar2(20),  
    primary key (itemId)  
);
```

```
insert into item values(item_seq.nextval,'레몬',50,'피로회복에 좋고 비타민 C도  
풍부','lemon.jpg');
```

```
insert into item values(item_seq.nextval,'오렌지',100,'비타민 C가 풍부하고 생  
과일 주스로 마심.','orange.jpg');
```

```
insert into item values(item_seq.nextval,'키위',200,'비타민 C가 풍부하여 다이  
어트나 미용','kiui.jpg');
```

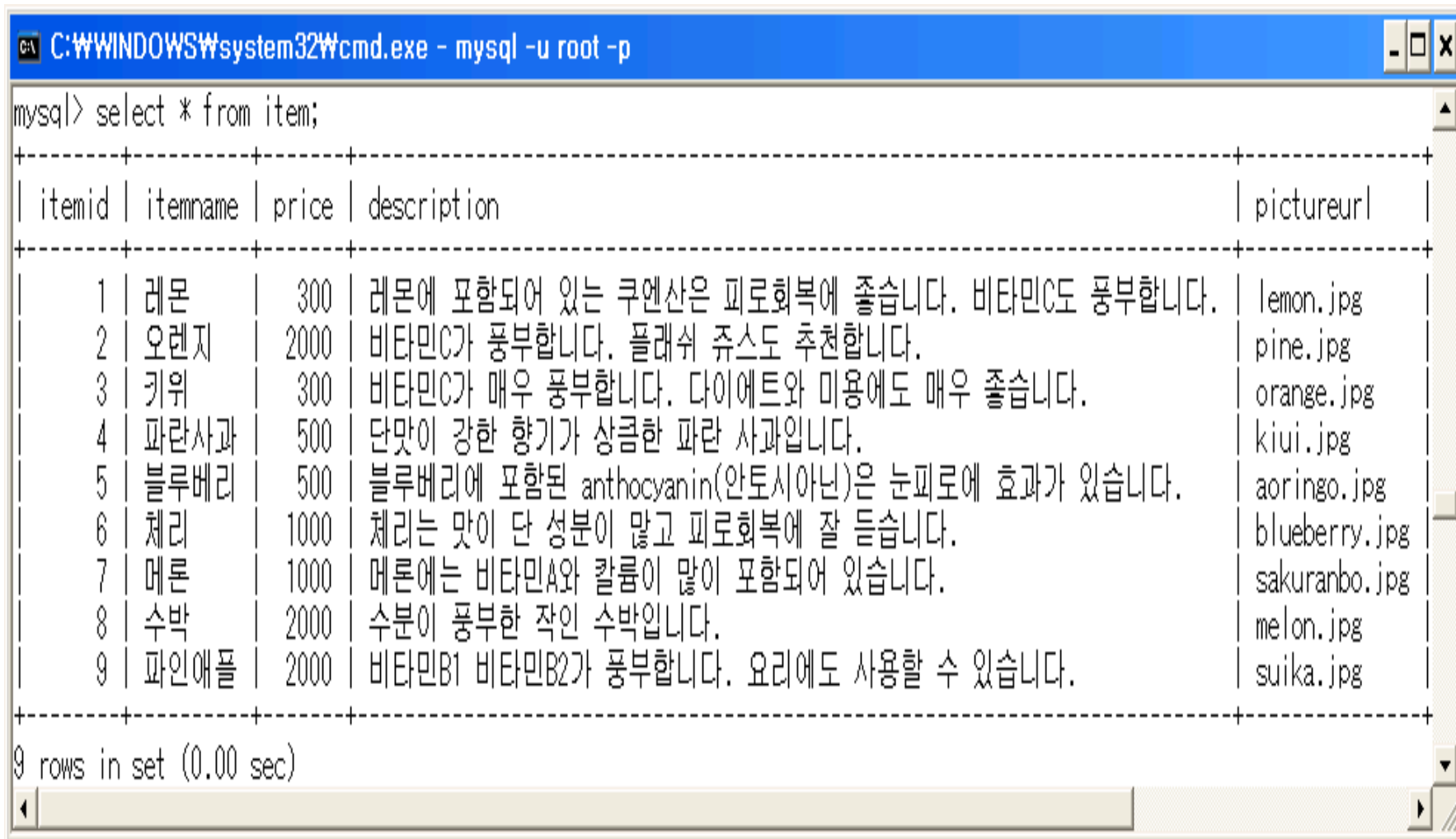
```
insert into item values(item_seq.nextval,'딸기',300,'폴리페놀을 다량 함유하고  
있어 항산화 작용','ichigo.jpg');
```

```
insert into item values(item_seq.nextval,'포도',800,'비타민 C나 플라보노이드  
를 다량 함유','budou.jpg');
```

```
insert into item values(item_seq.nextval,'귤',1000,'시네피린을 함유하고 있어  
감기 예방','mikan.jpg');
```

스프링 MVC 패턴 애플리케이션 개요

상품 테이블 데이터



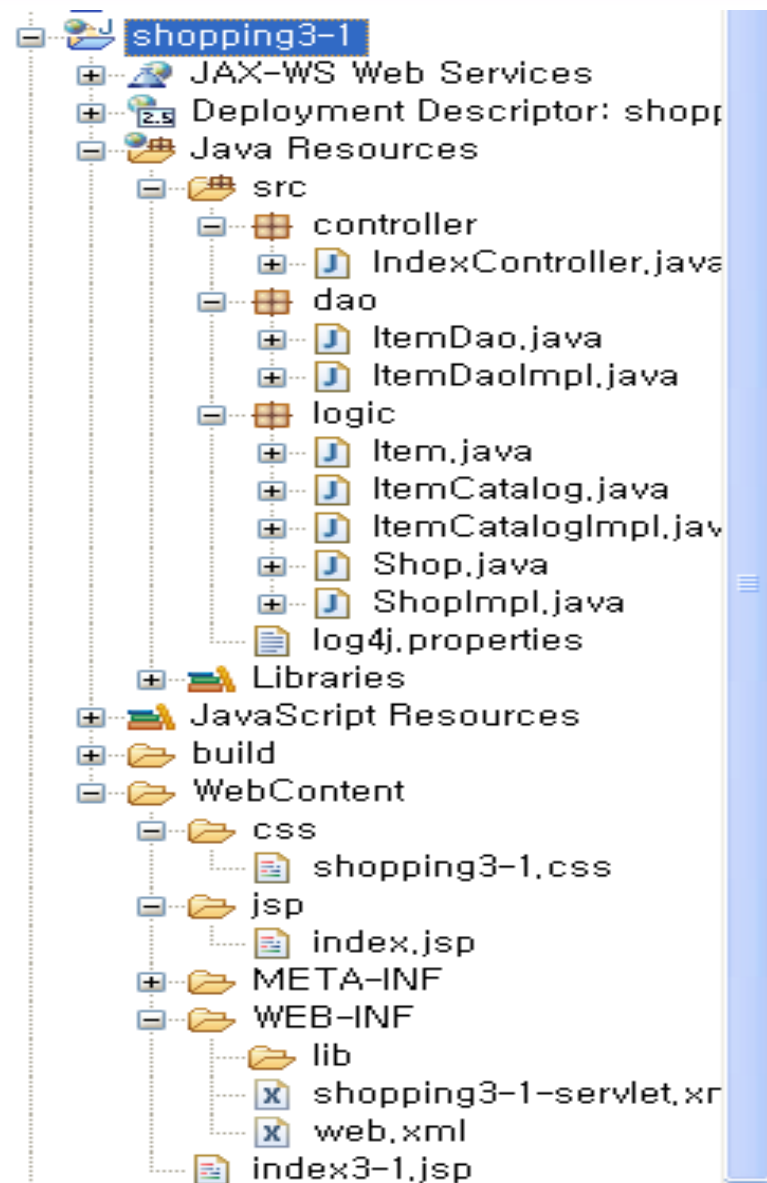
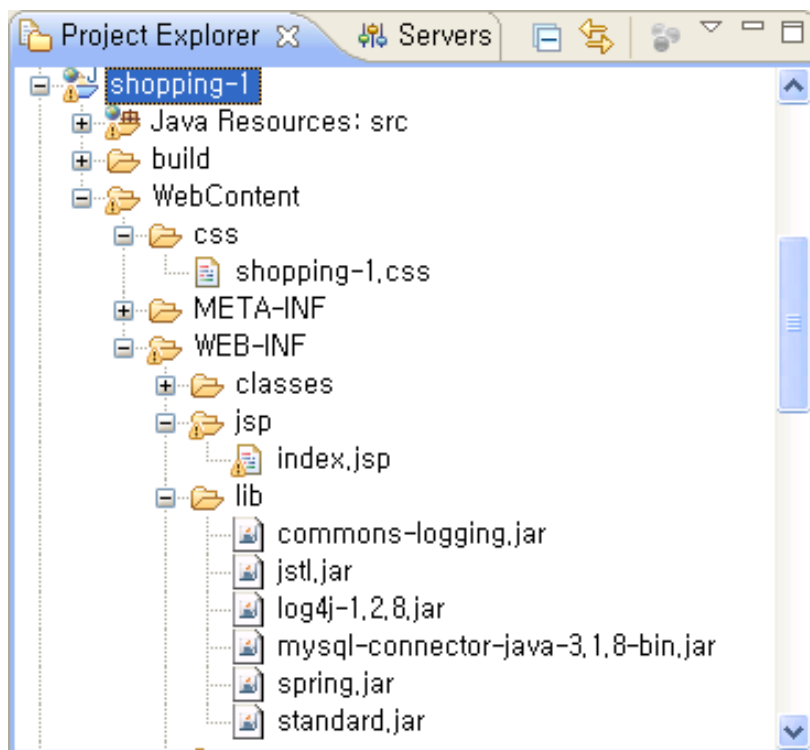
The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe - mysql -u root -p". The user has entered the command "mysql> select * from item;". The output is a table with 5 columns: itemid, itemname, price, description, and pictureurl. There are 9 rows of data, each representing a fruit item with its ID, name, price, a detailed description in Korean, and a corresponding image file name.

itemid	itemname	price	description	pictureurl
1	레몬	300	레몬에 포함되어 있는 쿠엔산은 피로회복에 좋습니다. 비타민C도 풍부합니다.	lemon.jpg
2	오렌지	2000	비타민C가 풍부합니다. 플레쉬 주스도 추천합니다.	pine.jpg
3	키위	300	비타민C가 매우 풍부합니다. 다이어트와 미용에도 매우 좋습니다.	orange.jpg
4	파란사과	500	단맛이 강한 향기가 상큼한 파란 사과입니다.	kiui.jpg
5	블루베리	500	블루베리에 포함된 anthocyanin(안토시아닌)은 눈피로에 효과가 있습니다.	aoringo.jpg
6	체리	1000	체리는 맛이 단 성분이 많고 피로회복에 잘 들습니다.	blueberry.jpg
7	메론	1000	메론에는 비타민A와 칼륨이 많이 포함되어 있습니다.	sakuranbo.jpg
8	수박	2000	수분이 풍부한 작인 수박입니다.	melon.jpg
9	파인애플	2000	비타민B1 비타민B2가 풍부합니다. 요리에도 사용할 수 있습니다.	suika.jpg

9 rows in set (0.00 sec)

스프링 MVC 패턴 애플리케이션 개요

파일구성



ModelAndView와 ViewResolver

- ① Controller 처리 결과 후 응답할 view와 view에 전달할 값을 저장
- ② 생성자
 - ModelAndView(String viewName) : 응답할 view 설정
 - ModelAndView(String viewName, Map values) : 응답할 view와 view로 전달할 값들을 저장 한 Map 객체
 - ModelAndView(String viewName, String name, Object value) : 응답할 view이름, view로 넘길 객체의 name-value
- ③ 보시 주요 메소드
 - setViewName(String view) : 응답할 view이름을 설정
 - addObject(String name, Object value) : view에 전달할 값을 설정
 - requestScope에 설정됨
 - addAllObject(Map values) : view에 전달할 값을 Map에 name-value로 저장하여 한번에 설정
 - requestScope에 설정됨
- ④ Redirect 방식 전송
 - view이름에 redirect: 접두어 붙인다.
 - ex) mv.setViewName("redirect:/welcome.html");

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.test</groupId>
  <artifactId>board</artifactId>
  <name>abc</name>
  <packaging>war</packaging>
  <version>1.0.0-BUILD-SNAPSHOT</version>
  <properties>
    <java-version>1.6</java-version>
    <org.springframework-version>3.0.6.RELEASE</org.springframework-
version>
    <org.aspectj-version>1.6.9</org.aspectj-version>
    <org.slf4j-version>1.5.10</org.slf4j-version>
  </properties>
  <repositories>
    <repository>
      <id>codeIds</id>
      <url>https://code.lds.org/nexus/content/groups/main-repo</url>
    </repository>
  </repositories>
```

<dependencies>

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId><version>1.0.2</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId><artifactId>spring-orm</artifactId>

<version>\${org.springframework-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>\${org.springframework-version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>\${org.springframework-version}</version>

</dependency>

<!-- oracle -->

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc6</artifactId>

<version>11.2.0.3</version>

<scope>compile</scope>

</dependency>

<!-- dbcp jdbc connection pool -->

<dependency>

**● <groupId>commons-dbcp</groupId><artifactId>commons-dbcp</artifactId>
<version>1.2.2</version> </dependency>**

<!-- ibatis -->

<dependency>

**<groupId>org.apache.ibatis</groupId><artifactId>ibatis-sqlmap</artifactId>
<version>2.3.4.726</version> </dependency>**

<!-- cglib ibatis를 위한 프록시 라이브러리 -->

<dependency>

**<groupId>cglib</groupId> <artifactId>cglib</artifactId>
<version>2.2</version>**

</dependency>

<!-- Spring -->

<dependency>

**<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>\${org.springframework-version}</version>
<exclusions>**

<!-- Exclude Commons Logging in favor of SLF4j -->

<exclusion>

**<groupId>commons-logging</groupId>
<artifactId>commons-logging</artifactId>**

</exclusion>

</exclusions>

</dependency>

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${org.springframework-version}</version>
</dependency>
<!-- AspectJ -->
<dependency>
  <groupId>org.aspectj</groupId><artifactId>aspectjrt</artifactId>
  <version>${org.aspectj-version}</version>
</dependency>
<!-- Logging -->
<dependency>
  <groupId>org.slf4j</groupId><artifactId>slf4j-api</artifactId>
  <version>${org.slf4j-version}</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId><artifactId>jcl-over-slf4j</artifactId>
  <version>${org.slf4j-version}</version><scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>${org.slf4j-version}</version>
  <scope>runtime</scope>
</dependency>
```

```
<dependency>
  <groupId>log4j</groupId><artifactId>log4j</artifactId>
  <version>1.2.15</version>
  <exclusions>
    <exclusion>
      <groupId>javax.mail</groupId><artifactId>mail</artifactId>
    </exclusion>
    <exclusion>
      <groupId>javax.jms</groupId><artifactId>jms</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.sun.jdmk</groupId><artifactId>jmxtools</artifactId>
    </exclusion>
    <exclusion>
      <groupId>com.sun.jmx</groupId>
      <artifactId>jmxri</artifactId>
    </exclusion>
  </exclusions>
  <scope>runtime</scope>
</dependency>
<!-- @Inject -->
<dependency>
  <groupId>javax.inject</groupId>
  <artifactId>javax.inject</artifactId>
  <version>1</version>
</dependency>
```


<!-- Servlet -->

<dependency>

**<groupId>javax.servlet</groupId>
<artifactId>servlet-api</artifactId>
<version>2.5</version>
<scope>provided</scope>**

</dependency>

<dependency>

**<groupId>javax.servlet.jsp</groupId>
<artifactId>jsp-api</artifactId>
<version>2.1</version>
<scope>provided</scope>**

</dependency>

<dependency>

**<groupId>javax.servlet</groupId>
<artifactId>jstl</artifactId>
<version>1.2</version>
</dependency>**

<!-- Test -->

<dependency>

**<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.7</version>
<scope>test</scope>**

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<source>\${java-version}</source><target>\${java-version}</target>

</configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<configuration><warName>abc</warName></configuration>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-dependency-plugin</artifactId>

<executions>

<execution>

<id>install</id><phase>install</phase>

<goals><goal>sources</goal></goals>

</execution>

</executions>

</plugin>


```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-resources-plugin</artifactId>
  <version>2.5</version>
  <configuration>
    <encoding>UTF-8</encoding>
  </configuration>
</plugin>
</plugins>
</build>
</project>
```

log4j.properties

```
log4j.rootLogger=INFO, stdout
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d %p [%c] - <%m>%n
```

JAVA9 추가

```
<!-- https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api -->  
<dependency>  
  <groupId>javax.xml.bind</groupId>  
  <artifactId>jaxb-api</artifactId>  
  <version>2.3.0</version>  
</dependency>
```

Item.java

```
package logic;
import java.io.Serializable;
public class Item implements Serializable {
    private static final long serialVersionUID = 1L;
    private Integer itemId;
    private String itemName;
    private Integer price;
    private String description;
    private String pictureUrl;
    public String getDescription() {return this.description;}
    public void setDescription(String description) {
        this.description = description;
    }
    public Integer getItemId() {return this.itemId;}
    public void setItemId(Integer itemId) {this.itemId = itemId;}
    public String getItemName() {return this.itemName;}
    public void setItemName(String itemName) {this.itemName = itemName;}
    public String getPictureUrl() {return this.pictureUrl;}
    public void setPictureUrl(String pictureUrl) {
        this.pictureUrl = pictureUrl;
    }
    public Integer getPrice() {return this.price;}
    public void setPrice(Integer price) {this.price = price;}
}
```

상품 리스트 화면 작성하기

```
package dao;  
import java.util.List; import logic.*;  
public interface ItemDao {  
    List<Item> findAll();  
}
```

```
package dao;  
@Repository  
public class ItemDaoImpl implements ItemDao{  
    @Autowired  
    private JdbcTemplate jt;  
    public List<Item> list() {  
        List<Item> list = jt.query("select * from item",  
            new BeanPropertyRowMapper<Item>(Item.class));  
        return list;  
    }  
}
```

Shop.java

```
package logic;  
import java.util.List;  
public interface Shop {  
    List<Item> getItemList();  
}
```

ShopImpl.java

```
package logic;  
import java.util.List;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Component;  
import org.springframework.stereotype.Service;  
import dao.ItemDao;  
@Service  
public class ShopImpl implements Shop {  
    @Autowired  
    private ItemDao iDao;  
    public List<Item> getItemList() {  
        return iDao.getItemList();  
    }  
}
```

IndexController.java

```
package controller;
import java.util.HashMap;   import java.util.List; import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import logic.Item; import logic.Shop;
import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.Controller;
public class IndexController implements Controller {
    @Autowired
    private Shop shop;
    // public void setShop(Shop shop) { this.shop = shop; }
    public ModelAndView handleRequest(HttpServletRequest request,
        HttpServletResponse response) throws Exception {
        List<Item> itemList = shop.getItemList();
        //Map<String, Object> model = new HashMap<String, Object>();
        //model.put("itemList", itemList);
        ModelAndView mav = new ModelAndView();
        //mav.addAllObjects(model);
        mav.addObject("itemList", itemList);
        mav.setViewName("/WEB-INF/views/list.jsp");
        return mav;
    }
}
```

Index.jsp

```
<body>  
  <script type= "text/javascript">  
    location.href="index.do";  
  </script>  
</body>
```

header.jsp

```
<meta http-equiv= "X-UA-Compatible" content="IE=edge">  
<meta name= "viewport" content="width=device-width, initial-  
    scale=1">  
<link href= "css/bootstrap.min.css" rel="stylesheet">  
<script src= "js/jquery.js"></script>  
<script src= "js/bootstrap.min.js"></script>  
<%@ taglib prefix= "c" uri="http://java.sun.com/jsp/jstl/core" %>
```


list.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html><head><meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
<title>Insert title here</title></head>
<body><div class="container">
    <table class="table table-hover">
        <caption class="text-primary">상품 리스트</caption>
        <tr><th>아이디</th><th>이름</th><th>가격</th><th>설명</th></tr>
        <c:forEach var="item" items="${list}">
            <tr><td>${item.itemId}</td><td>${item.itemName}</td>
            <td>${item.price}</td><td>${item.description}</td></tr>
        </c:forEach>
    </table>
</div>
</body>
</html>
```

설정 파일(web.xml)

web.xml 파일은 J2EE 웹 애플리케이션의 기본이 되는 설정 파일이다.

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns="http://java.sun.com/xml/ns/javaee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"  
  version="2.5">
```

```
<servlet>  
  <servlet-name>shopping3-1</servlet-name>  
  <servlet-  
class>org.springframework.web.servlet.DispatcherServlet</servlet-  
class>  
  <load-on-startup>1</load-on-startup>  
</servlet>
```

```
<servlet-mapping>  
  <servlet-name>shopping3-1</servlet-name>  
  <url-pattern>*.html</url-pattern>  
</servlet-mapping>  
</web-app>
```

설정 파일(web.xml)

- context root 의 확장자 .html 파일로 요청을 하면 DispatcherServlet 클래스로 맵핑하도록 정의하고 있다(❶).
- 이 정의에 의해 .html의 확장자가 붙은 파일로의 액세스는 모두 DispatcherServlet로 송신된다(❸).
- DispatcherServlet 클래스에는 <servlet-name> 태그에 의해 'shopping3-1' 이라고 서블릿 이름을 지정하여 정의하고 있다(❷).
- 이 서블릿 이름 'shopping3-1' 에 '-servlet.xml' 라는 문자열을 부가함으로써 컨테이너 상에 로드된 스프링 설정 파일명이 결정된다. 예제의 경우 shopping3-1-servlet.xml 파일이 로드되게 된다.

설정 파일(shopping1-servlet.xml)

- 스프링 MVC용 설정 파일이다.

shopping-1-servlet.xml은 크게 나누면 web층에 있는 스프링 MVC의 정의(❶)와 비즈니스층 이후의 정의(❷) 둘로 구성된다.

1)

◆ 스프링 설정 파일에서의 BeanNameUrlHandlerMapping의 정의

```
<!-- Controller -->
```

```
<bean id = "indexController" name = "/index.html" class = "controller.IndexController" >
```

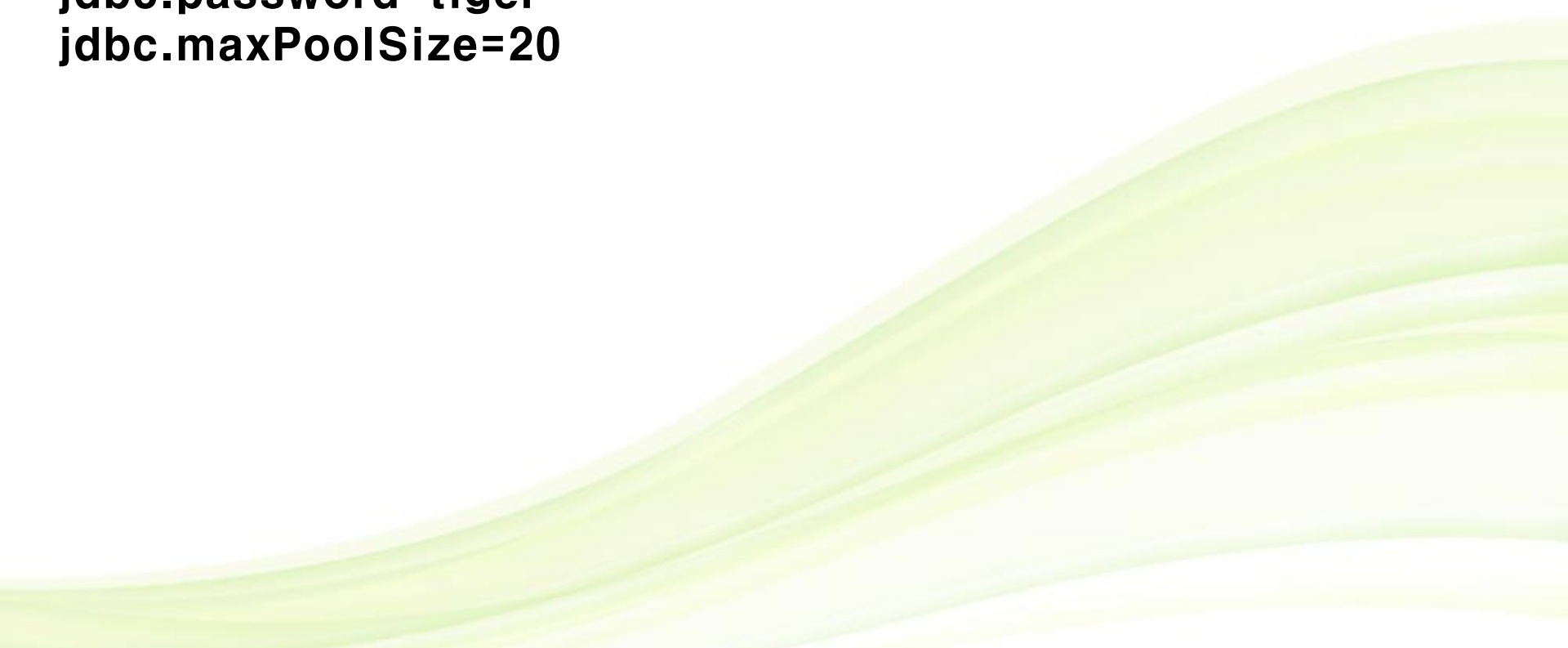
```
    <property name = "shopService" > <ref bean = "shopService" /> </property>
```

```
</bean>
```



jdbc.properties

```
jdbc.driverClassName=oracle.jdbc.driver.OracleDriver  
jdbc.url=jdbc:oracle:thin:@127.0.0.1:1521:xe  
jdbc.username=scott  
jdbc.password=tiger  
jdbc.maxPoolSize=20
```



```
<!-- Handler -->
<!-- Controller -->
<context:property-placeholder location="classpath:jdbc.properties"/>
<bean id="indexController" name="/index.html"
      class="controller.IndexController" />
<!-- Resolver --> <!-- Data Source -->
<bean id="template" class="org.springframework.jdbc.core.JdbcTemplate">
<constructor-arg ref="dataSource" />
</bean>
<bean id="dataSource"
      class="com.mchange.v2.c3p0.ComboPooledDataSource"
      destroy-method="close">
    <property name="driverClass" value="${jdbc.driverClassName}" />
    <property name="jdbcUrl" value="${jdbc.url}" />
    <property name="user" value="${jdbc.username}" />
    <property name="password" value="${jdbc.password}" />
    <property name="maxPoolSize" value="${jdbc.maxPoolSize}" />
</bean>
<context:component-scan base-package="dao,logic"/>
</beans>
```

Shopping1-servlet.xml

웹계층인 스프링 MVC정의

브라우저가 요청을 보내면 **DispatcherServlet** 인스턴스가 요청을 받아서 **HandlerMapping** 인스턴스를 참조해서 웹 요청 **URL**과 컨트롤러를 매핑해서 처리를 넘길 곳 지정
ViewResolver 인터페이스도 설정파일에 구현 클래스를 정의하지 않으면 기본인 **InternalResourceViewResolver**를 사용

BeanNameUrlHandlerMapping 클래스

스프링 설정 파일에 컨트롤러를 정의하면서 지정한 **name** 속성의 값과 웹 요청 **URL**을 매핑하는 **HandlerMapping** 구현 클래스. 여기서는 스프링 설정파일의 **<bean>** 태그 **name** 속성에

‘/list.html’ 이라고 기술

이번 예제는 설정 파일에 **ViewResolver** 클래스를 지정하고 있지 않기 때문에 스프링 **MVC** 기본 **ViewResolver**인 **InternalResourceViewResolver** 클래스를 암묵적 사용.

명시적으로 정의하면 뷰정보에 **prefix** 프로퍼티나 **suffix**를 추가하거나 뷰의 구현 클래스 지정

스프링 설정파일 상 BeanNameUrlHandlerMapping 예

```
<!-- ViewResolver -->
```

```
<bean id= "InternalResourceViewResolver " class
```

```
= "org.springframework.web.servlet.view.InternalResourceViewResolver ">
```

```
<property
```

```
name= "viewClass" ><value>org.springframework.web.servlet.view.JstlView</value></property>
```

```
<property name= "suffix" ><value>.jsp</value></property></bean>
```


web.xml 서블릿 설정에서 <load-on-startup>

<load-on-startup> 엘리먼트의 사용의 목적

웹 어플리케이션 구동시 자동으로 서블릿 클래스를 초기화(**init** 메서드 호출) 시키기 위한 목적

<load-on-startup> 값이 0이거나 양수인 경우

해당 서블릿이 속한 웹 어플리케이션이 실행될 때, 초기화(**init** 메서드 호출) 한다.

즉, **WAS**(컨테이너)가 실행되는 시점에 서블릿이 초기화된다.

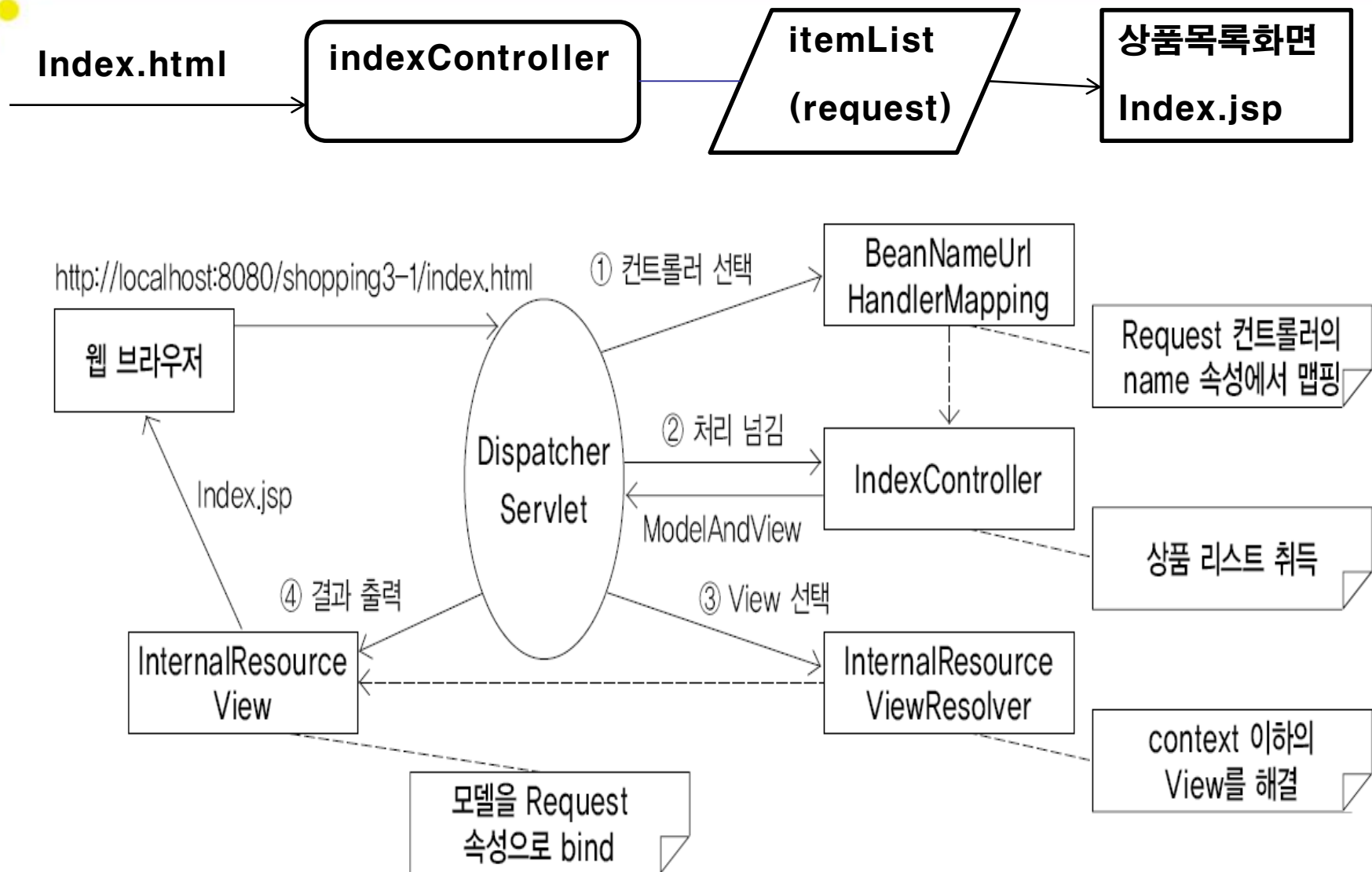
가지고 있는 값에 따라 초기화 순서가 결정되며, 값이 같을 경우 **WAS** 임의로 순서를 정해서 초기화한다.

<load-on-startup> 값이 음수이거나 해당 엘리먼트가 없는 경우

해당 서블릿이 실행되는 시점에 초기화(**init** 메서드 호출) 한다.

웹 어플리케이션이 시작되었더라도, 해당 서블릿에 대한 요청이 없다면 서블릿의 초기화(**init** 메서드의 호출)가 되지 않을 수도 있음 (**WAS**가 임의의 시점에 호출할 수도 있음)

예제의 화면 이동과 화면 정보의 입출력



문제

아래와 같이 출력할 수 있도록 프로그램을 추가하거나 변경

1. 추가

IndexController2.java, Dept.java, list2.jsp

2. 변경

**ItemDao.java, ItemDaoImpl.java,
Shop.java, ShopImpl.java
shopping31-servlet.xml**

부서 정보

부서코드	부서명	근무지
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

상품 상세 화면 만들기

상품 상세

상품 리스트 화면

상품ID	상품명	가격
1	레몬	300원
2	오렌지	2000원
3	키위	300원
4	파란사과	500원
5	블루베리	500원
6	체리	1000원
7	메론	1000원
8	수박	2000원
9	파인애플	2000원

상품 상세



상품목록화면

상품상세화면

webapp에 img폴더와 그림 추가

설정 파일

파일	설명
web.xml	웹 어플리케이션 기본 설정 파일
shopping-2-servlet.xml	스프링 MVC 용의 설정 파일
applicationContext.xml	비즈니스로직 정의용 스프링 설정 파일

사용하는 MVC 클래스

인터페이스	구현 클래스
HandlerMapping	SimpleUrlHandlerMapping
ViewResolver	InternalResourceViewResolver
View	JstlView

상품 상세

```
package shopping2.dao;  
import java.util.List;  
import shopping2.model.Item;  
public interface ItemDao {  
    List<Item> list();  
    Item select(int itemId);  
}
```

ItemDaoImpl에 sql과 method추가

```
public Item select(int itemId) {  
    Item item = jt.queryForObject(  
        "select * from item where itemId=?",  
        new BeanPropertyRowMapper<Item>(Item.class), itemId);  
    return item;  
}
```

상품 상세

```
package logic;  
import java.util.List;  
public interface Shop {  
    List<Item> list();  
    Item select(int itemId);  
}
```

ShopImpl에 다음 메소드 추가

```
public Item select(int itemId) {  
    return id.select(itemId);  
}
```


상품 상세

```
package shopping2.controller;
@Controller
public class ItemController {
    @Autowired
    private ItemService is;
    @RequestMapping("index1")
    public String itemList(Model model) {
        List<Item> list = is.list();
        model.addAttribute("list", list);
        return "list";
    }
    @RequestMapping("detail")
    public String itemDetail(int itemId, Model model) {
        Item item = is.select(itemId);
        model.addAttribute("item", item);
        return "detail";
    }
}
```

list.jsp

```
<div class= "container" align="center">
<h2>과일 목록</h2>
<table class= "table table-bordered">
  <tr><th>아이디</th><th>이름</th><th>가격</th><th>설명</th></tr>
  <c:forEach var= "item" items= "${list }">
    <tr><td>${item.itemId}</td>
      <td><a href= "detail.do?itemId=${item.itemId}">
        ${item.itemName}</a></td>
      <td>${item.price}</td><td>${item.description}</td></tr>
    </c:forEach>
  </table>
</div>
```

detail.jsp

```
<div class= "container" align="center">
  <h1>과일 상세 정보</h1>
  <table class= "table table-bordered">
    <tr><td rowspan= "4">
      <img alt= "" src= "img/${item.pictureUrl }"></td>
    <td>아이디</td><td>${item.itemId}</td></tr>
    <tr><td>과일명</td><td>${item.itemName}</td></tr>
    <tr><td>가격</td><td>${item.price}</td></tr>
    <tr><td>설명</td><td>${item.description}</td></tr>
  </table>
  <a href= "index1.do">목록보기</a>
</div>
```

설정파일에서 SimpleUrlHandlerMpping

아래 문장을 **shopping3-2-servlet.xml** 추가

<!-- HandlerMapping -->

```
<bean id= "handlerMapping" class=  
  "org.springframework.web.servlet.handler.SimpleUrlHandlerMapping" >  
  <property name= "mappings" >  
    <value>  
      /index.html=indexController  
      /detail.html=detailController  
    </value>  
  </property>  
</bean>
```

<!-- HandlerMapping -->

```
<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>
```

<!-- Controller -->변경

```
<bean id= "indexController" class= "controller.IndexController" / >
```

```
<bean id= "detailController" class= "controller.DetailController" / >
```

SimpleUrlHandlerMapping 클래스에는 **Properties** 타입 **mappings** 프로퍼티가 있음
이 **mappings** 프로퍼티에 웹 요청 **URL**과 컨트롤러 지정

<value>요소 안에 ‘키=값’ 형식으로 기술

InternalResourceViewResolver 클래스

ViewResolver구현하는 클래스

InternalResourceViewResolver 클래스는 ‘viewClass’ 나 ‘prefix’ , ‘suffix’ 등
프로퍼티를 설정 아래내용 추가

```
<!-- ViewResolver -->
```

```
<bean id= "internalResourceViewResolver"
```

```
class= "org.springframework.web.servlet.view.InternalResourceViewResolver">
```

```
<property name= "viewClass">
```

```
<value>org.springframework.web.servlet.view.JstlView</value>
```

```
</property>
```

```
<property name= "prefix"><value>WEB-INF/jsp/</value> </property>
```

```
<property name= "suffix">
```

```
<value>.jsp</value>
```

```
</property>
```

```
</bean>
```

web.xml에 listener추가

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

applicationContext.xml에 아래 내용 추가

```
<bean id="jdbcTemplate"
class="org.springframework.jdbc.core.JdbcTemplate">
  <constructor-arg ref="dataSource" /> </bean>
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
  <property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:xe"/>
  <property name="username" value="scott"/>
  <property name="password" value="tiger"/>
</bean>
</beans>
```

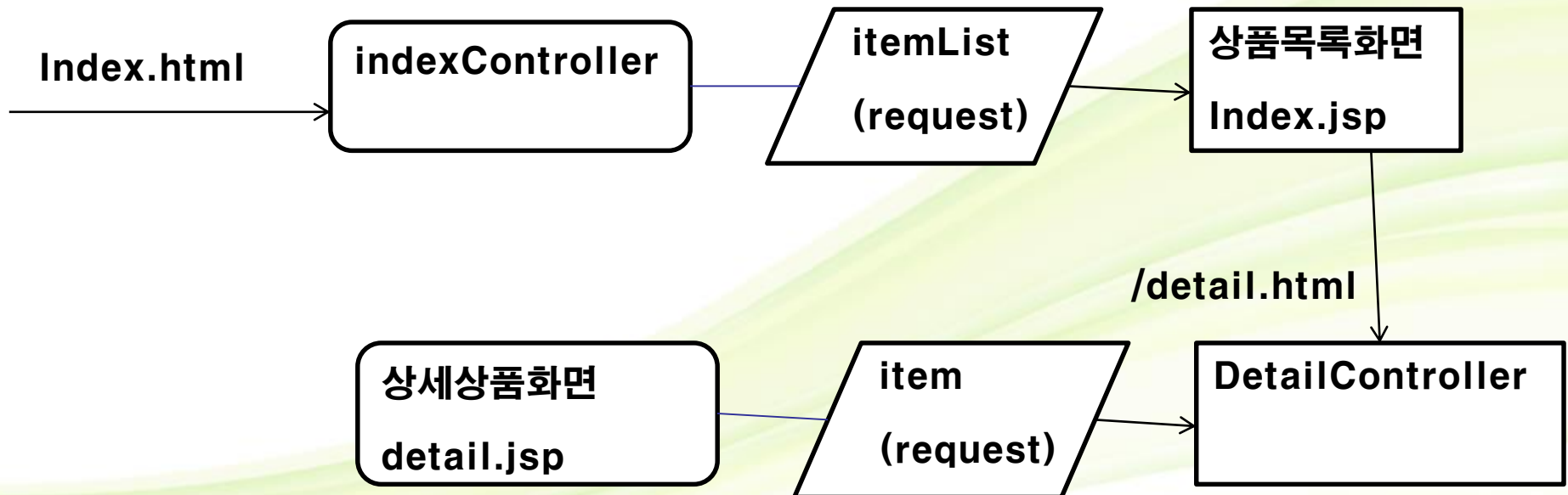
JstlView의 핵심

JSTL을 사용해서 **JSP** 만드는 것을 지원하는 클래스, **View** 인터페이스를 구현
JstlView 클래스를 사용하면 **JSTL** 포맷태그에서 스프링의 메시지 자원에 접근

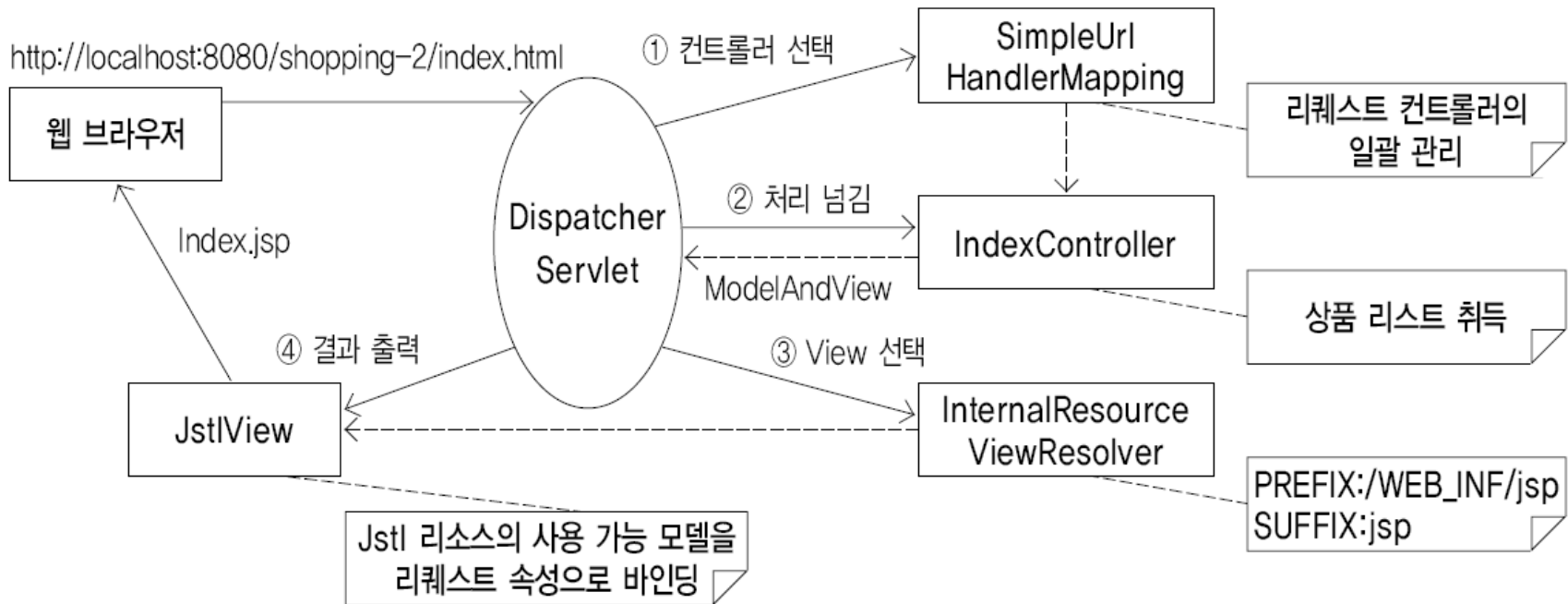
설정파일(web.xml)의 핵심

- 1) **DispatcherServlet**을 매핑, 리스너로 **ContextLoaderListener** 클래스 정의
- 2) 서렛ㅇ파일을 복수로 사용할때는 읽는 순서 중요
- 3) **ContextLoaderListener** 클래스는 **ServletContext** 인스턴스 생성할 때 호출하는 것으로 **DispatcherServlet**일기 전에 **ContextLoaderListener**가 **applicationContext.xml**을 읽음

화면 이동과 화면 정보의 입출력

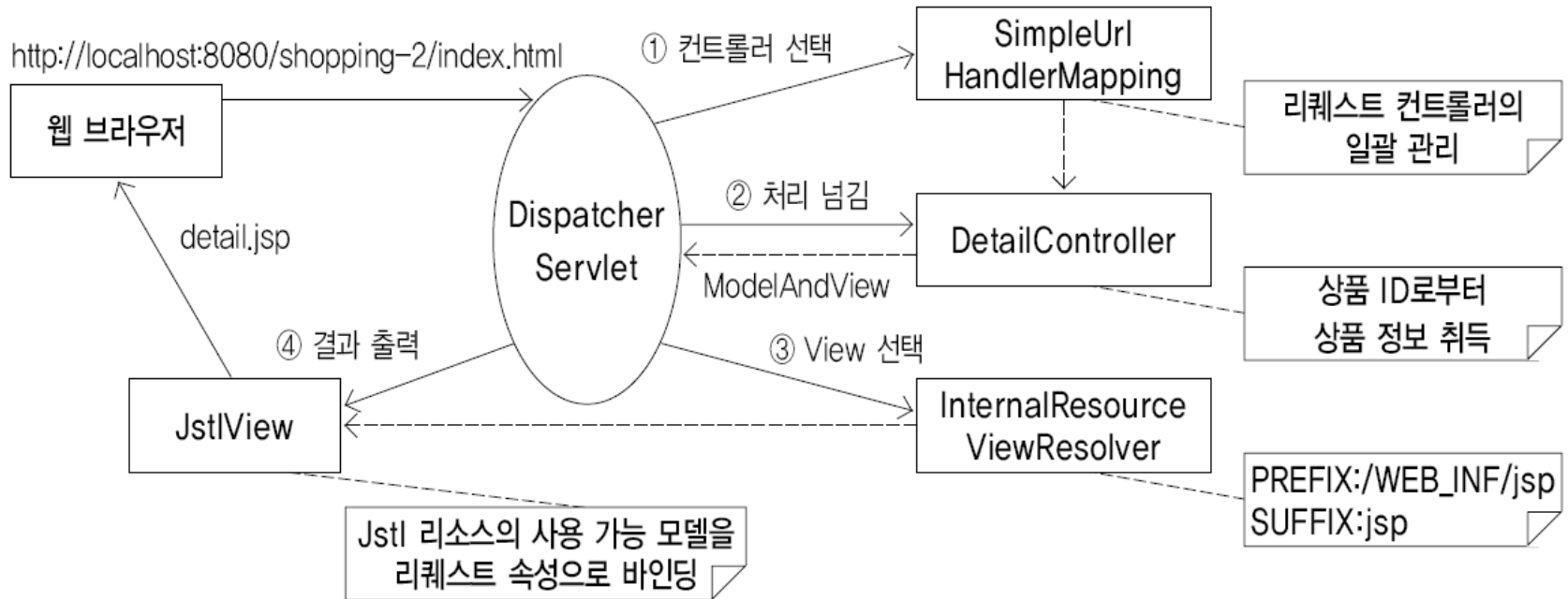


예제의 화면 이동과 화면 정보 입출력



▲ 상품 리스트 화면의 처리 흐름

예제의 화면 이동과 화면 정보 입출력



▲ 상품 상세 화면의 처리 흐름

문제

부서정보 리스트에서 부서코드를 클릭하면
한 개의 부서정보를 출력하는 프로그램을 작성하시오

힌트

/index.html=indexController

/index2.html=indexController2

/detail.html=detailController

/detail2.html=detailController

@RequestMapping(value="detail2")

로그인 화면 만들기

강사 : 강병준

이번 장의 목차

1. 개요
2. 화면에 표시하기

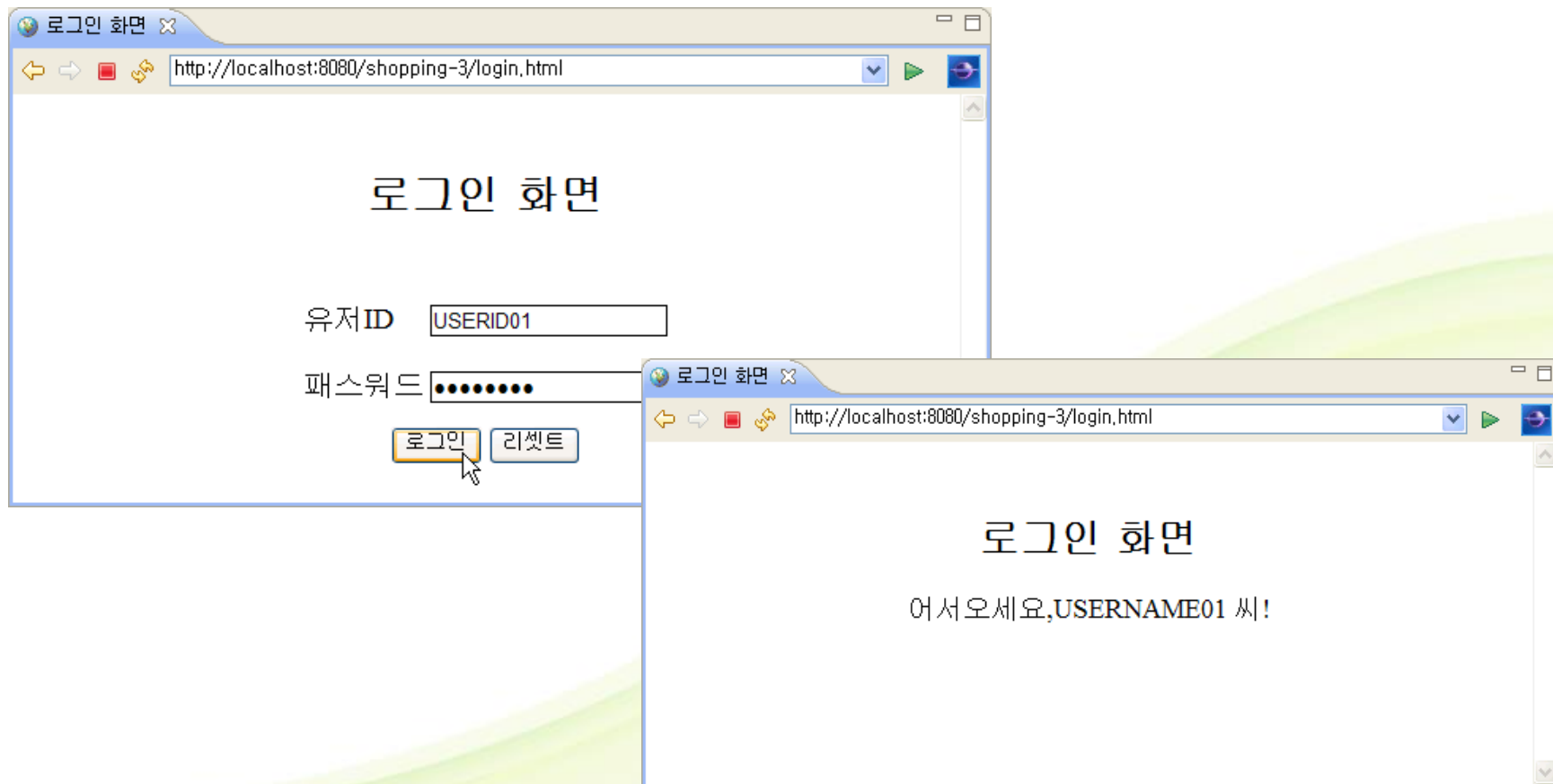
이 장에서는 로그인 화면에서 인증 처리를 하고 로그인 완료 화면을 표시하기까지의 샘플 애플리케이션을 작성한다.

일반적으로 애플리케이션이라면 로그인 후에 여러 기능을 이용할 수 있게 되어 있지만 여기에서는 로그인 후에 완료 화면을 출력하는 것으로 간단하게 구성되어 있다.

그런 후에 다음 장에서부터 로그인 화면에 다른 샘플 애플리케이션을 통합하여 하나의 애플리케이션으로 조합하도록 하겠다.

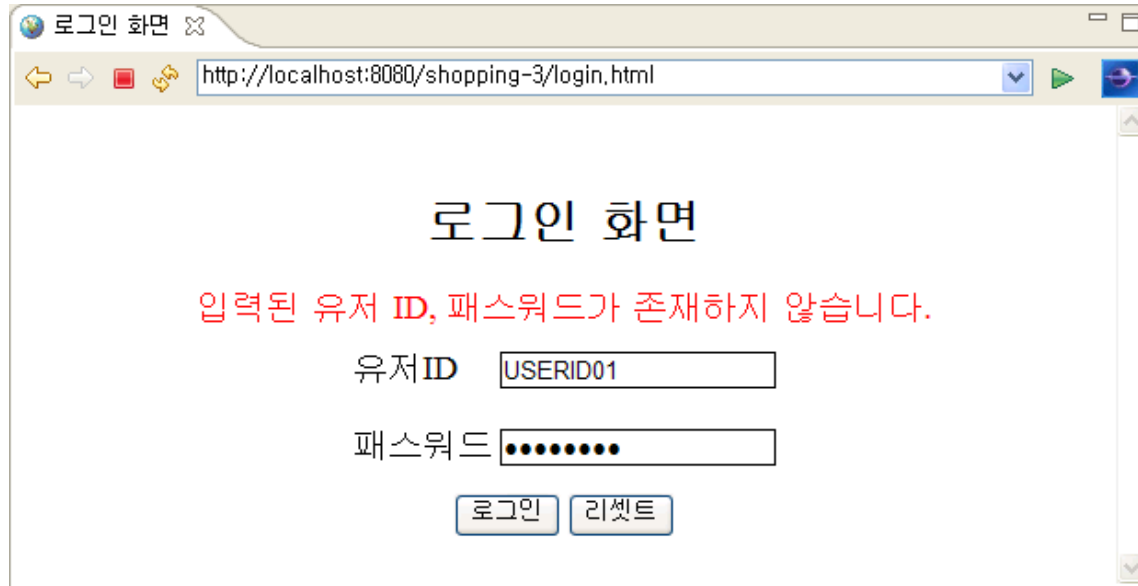
개요

서버 측에서는 입력된 데이터가 유효한 값인지 살펴보고 정확한 값이라면 데이터베이스에서 유저 테이블(Usertb)에 해당 유저가 존재하는지 검색한다. 유저가 존재하는 경우, 다음 로그인 완료 화면을 표시한다.



개요

입력된 데이터가 잘못된 값이거나 데이터베이스에 해당하는 유저를 찾지 못할 경우는 로그인 에러로 처리되어 에러 메시지를 표시한다.



샘플에서 사용하는 스프링 **MVC** 클래스

인터페이스	구현 클래스
HandlerMapping	SimpleUrlHandlerMapping
ViewResolver	InternalResourceViewResolver
View	JstlView

```
create table user (  
    userId varchar2(20) primary key,  
    userName varchar2(20),  
    password varchar2(20),  
    postCode varchar2(8),  
    address VARCHAR2(50),  
    email VARCHAR2(50),  
    job VARCHAR2(30),  
    birthday DATE  
);
```

```
insert into user values  
('kk1','길동1','ps1','111-111','서울시 서초구','kk1@korea.net','사회인','1979-01-  
01');
```

```
insert into user values  
('kk2','길동2','ps2','222-222','부산시 남구','kk2@korea.net','사회인','1979-06-  
30');
```

```
insert into user values  
('kk3','길동3','ps3','333-333','인천시 북구','kk3@korea.net','학생','1985-12-31');
```

설정 파일(web.xml)

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns="http://java.sun.com/xml/ns/javaee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee  
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"  
  version="2.5">
```

```
<!-- 한글 입력 -->
```

```
<filter>
```

```
  <filter-name>CharacterEncodingFilter</filter-name>
```

```
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter
```

```
  </filter-class>
```

```
  <init-param>
```

```
    <param-name>encoding</param-name>
```

```
    <param-value>UTF-8</param-value>
```

```
  </init-param>
```

```
  <init-param>
```

```
    <param-name>forceEncoding</param-name>
```

```
    <param-value>true</param-value>
```

```
  </init-param>
```

```
</filter>
```


<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/*</url-pattern>

</filter-mapping>

<listener>

<listener-

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

</listener>

<servlet>

<servlet-name>shopping3-3</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>shopping3-3</servlet-name>

<url-pattern>*.html</url-pattern>

</servlet-mapping>

</web-app>

User.java

```
package logic;      import java.io.Serializable;  import java.util.Date;
public class User implements Serializable{
    private String userId;      private String password;
    private String userName;    private String postCode;
    private String address;     private String email;
    private String job;         private Date birthDay;

    public String getAddress() {return this.address;}
    public void setAddress(String address) {this.address = address;}
    public Date getBirthDay() {return this.birthDay;}
    public void setBirthDay(Date birthDay) { this.birthDay = birthDay;}
    public String getEmail() {return this.email;}
    public void setEmail(String email) {this.email = email;}
    public String getJob() {return this.job;}
    public void setJob(String job) {this.job = job; }
    public String getPassword() {return this.password;}
    public void setPassword(String password) {this.password = password; }
    public String getPostCode() {return this.postCode;}
    public void setPostCode(String postCode) {this.postCode = postCode;}
    public String getUserId() {return this.userId;}
    public void setUserId(String userId) {this.userId = userId;}
    public String getUserName() {return this.userName;}
    public void setUserName(String userName) {this.userName = userName;}
}
```

UserDao.java

```
package dao;  
import logic.User;  
public interface UserDao {  
    User findByUserIdAndPassword(String userId, String password);  
}
```

UserDaoImpl.java

```
package dao;    import javax.sql.DataSource; import logic.User;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.simple.SimpleJdbcTemplate;
@Repository
public class UserDaoImpl implements UserDao {
    private static final String SELECT_BY_USERID_PASSWORD =
        "SELECT userId, password, userName, postcode,"
        + " address, email, job, birthday FROM userAccount
        WHERE userId = ? AND password = ?";
    @Autowired
    private JdbcTemplate jdbcTemplate;
    public User findByUserIdAndPassword(String userId,
        String password) {
        RowMapper<User> mapper =
            new BeanPropertyRowMapper<User>(User.class);
        return this.template.queryForObject(
            SELECT_BY_USERID_PASSWORD, mapper, userId, password);
    }
}
```

LoginValidator.java

```
package utils;    import logic.User;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
public class LoginValidator implements Validator {
    public boolean supports(Class<?> clazz) {
        return User.class.isAssignableFrom(clazz);
    }
    public void validate(Object command, Errors errors) {
        User user = (User) command;
        if (!StringUtils.hasLength(user.getUserId())) {
            errors.rejectValue("userId", "error.required");
        }
        if (!StringUtils.hasLength(user.getPassword())) {
            errors.rejectValue("password", "error.required");
        }
        if (errors.hasErrors()) {
            errors.reject("error.input.user");
        }
    }
}
```

@ModelAttribute

@ModelAttribute는 모델 맵에 담겨서 뷰에 전달되는 모델 오브젝트의 한가지라고도 볼 수 있다. 기본적으로 모든 **@ModelAttribute**는 별도의 설정 없이도 자동으로 뷰에 전달 된다. 그렇다면 **@ModelAttribute**가 붙은 모델은 그 밖의 모델과 어떤 차이점이 있을까?

컨트롤러가 사용하는 모델 중에는 클라이언트로부터 받는 **HTTP** 요청 정보를 이용해 생성되는 것이 있다. 단순히 검색을 위한 파라미터 처럼 컨트롤러의 로직에서 사용하고 버리는 요청 정보도 있지만, 웹 페이지의 폼 정보처럼 일단 컨트롤러가 전달 받아서 내부 로직에 사용하고 필요에 따라 다시 화면에 출력하기도 하는 요청 정보도 있다.

이렇게 클라이언트로부터 컨트롤러가 받는 요청 정보 중에서, 하나 이상의 값을 가진 오브젝트 형태로 만들 수 있는 구조적인 정보를 **@ModelAttribute**라고 부른다.

@ModelAttribute는 이렇게 컨트롤러가 전달 받는 오브젝트 형태의 정보를 가리키는 말이다.

컨트롤러가 클라이언트에게 전달받는 정보 중 가장 단순한 형태는 요청 파라미터다.

GET 매서드라면 **URL**에 **name=Spring** 같은 쿼리 스트링을 통해서 전달될 것이고, **POST**라면 **<input type=**과 같이 폼의 필드로 전달될 것이다.

이런 **name** 같은 파라미터는 **@RequestParam** 어노테이션으로 받으면 된다.

그렇다면 단순히 **@RequestParam**이 아니라 **@ModelAttribute**를 사용해서 모델로 받는 것은 어떤 것이 있을까?

사실 정보의 종류가 다른 건 아니다. 단지 요청 파라미터를 매서드 파라미터에서 **1:1**로 받으면 **@RequestParam**이고, 도메인 오브젝트나 **DTO**의 프로퍼티에 요청 파라미터를 바인딩 해서 한번에 받으면 **@ModelAttribute**라고 볼 수 있다.

하나의 오브젝트에 클라이언트의 요청 정보를 담아서 한 번에 전달되는 것이기 때문에 이를 커맨드 패턴에서 말하는 커맨드 오브젝트라고 부르기도 한다.


```
@RequestMapping("/user/search")
public String search(@ModelAttribute UserSearch usersearch) {
    List<User> list = userService.search(userSearch);
    model.addAttribute("userList" , list);
}
```

스프링은 **@ModelAttribute**가 붙은 파라미터 타입의 오브젝트를 만들고 프로퍼티를 통해 요청 파라미터를 넣어 준다.

주로 **URL**의 쿼리스트링으로 들어오는, 검색조건과 같은 정보를 **@ModelAttribute**가 붙은 파라미터 타입의 오브젝트에 모두 담아서 전달해주는 것은 커맨드라고 부른다.

UserSearch에 이 정보를 활용하는 부가기능까지 넣는다면 본격적인 커맨드 패턴의 오브젝트처럼 사용할 수 있기 때문이다.

@ModelAttribute 어노테이션도 생략 가능하다.

매서드를 다음과 같이 정의해도 **@ModelAttribute**가 있을 때와 동일 하게 동작한다.

```
public String add(User user) { }
```

@RequestParam도 생략이 가능 한데 , 스프링은 몇 가지 단순 타입(**String** , **int**)등은 **@RequestParam**으로 보고 그외의 복잡한 오브젝트는 모두 **@ModelAttribute**가 생략 됐다고 간주 한다.

스프링은 간단한 숫자나 문자로 전달된 요청 파라미터를 제법 복잡한 오브젝트로 변환할 수도 있다.

따라서 단순 타입이 아니라고 해서 꼭 **@ModelAttribute**가 생략됐다고 볼 수는 없다.

그래서 가능하면 **@RequestParam** , **@Modelattribute** 어노테이션을 붙이는 것이 좋다.

LoginFormController.java

```
package controller;
import logic.Shop;
import logic.User;
import org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.stereotype.Controller;
import org.springframework.validation.BindingResult;
import org.springframework.validation.Validator;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;
@Controller
public class LoginFormController {
    @Autowired
    private Shop shopService;
    @Autowired
    private Validator loginValidator;
    @RequestMapping(method = RequestMethod.GET)
    public String toLoginView() {        return "login";    }
}
```


@ModelAttribute

public User setUpForm() { return new User(); }

@RequestMapping(method = RequestMethod.POST)

public ModelAndView onSubmit(User user, BindingResult bindingResult) {

loginValidator.validate(user, bindingResult);

ModelAndView modelAndView = new ModelAndView();

if (bindingResult.hasErrors()) {

modelAndView.getModel().putAll(bindingResult.getModel());

return modelAndView;

}

try {

// 유저 정보 검색

User loginUser =

this.shopService.getUserByIdAndPassword(

user.getId(), user.getPassword());

modelAndView.setViewName("loginSuccess");

modelAndView.addObject("loginUser", loginUser);

return modelAndView;

} catch (EmptyResultDataAccessException e) {

bindingResult.reject("error.login.user");

modelAndView.getModel().putAll(bindingResult.getModel());

return modelAndView;

}

}

}

에러코드와 메시지

글로벌 에러코드

1. 에러코드 + " " + 커맨드 객체이름
2. 에러코드

Errors.rejectValues()를 이용하여 생성한 에러코드

1. 에러코드 + " " + 커맨드객체이름 + " " + 필드명
2. 에러코드 + " " + 필드명
3. 에러코드 + " " + 필드타입
4. 에러코드

필드가 List나 목록인 경우

1. 에러코드 + " " + 커맨드객체이름 + " " + 필드명[인덱스].중첩필드명
2. 에러코드 + " " + 커맨드객체이름 + " " + 필드명.중첩필드명
3. 에러코드 + " " + 필드명[인덱스].중첩필드명
4. 에러코드 + " " + 필드명.중첩필드명
5. 에러코드 + " " + 중첩필드명
6. 에러코드 + " " + 필드타입
7. 에러코드

에러코드와 메시지

메시지 파일

1. **requireds** : 필수 항목
2. **minlength** : 최소글자 수
3. **maxlength** : 최대 글자수
4. **unsafe.password** : 암호는 알파벳과 숫자포함
5. **invalidOrPassword** : id와 암호가 일치하지 않음

<form:form>태그

 <form:errors path= “email” />

</form:form>

<form>태그

 <spring:hasBindErrors name= “email” />

</form>

messages.properties

입력정보에 문제가 있습니다

**error.input.user= \uc785\ub825 \uc815\ubcf4\uc5d0
\ubb38\uc81c\uac00 \uc788\uc2b5\ub2c8\ub2e4.**

유저 ID를 입력하십시오.

**error.required.userId= \uc720\uc800ID\ub97c \uc785\ub825\ud574
\uc8fc\uc138\uc694.**

패스워드를 입력해 주세요.

**error.required.password= \ud328\uc2a4\uc6cc\ub4dc\ub97c
\uc785\ub825\ud574 \uc8fc\uc138\uc694.**

입력된 유저ID 비밀번호가 맞지 않습니다.

**error.login.user= \uc785\ub825\ub41c \uc720\uc800ID,
\ud328\uc2a4\uc6cc\ub4dc\uac00 \ub9de\uc9c0
\uc54a\uc2b5\ub2c8\ub2e4.**

header.jsp에 빨간부분 추가

```
<%@ page session= "false"%>
<%@ taglib prefix= "c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix= "spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix= "form" uri="http://www.springframework.org/tags/form"%>
<link rel= "stylesheet" type="text/css" href="css/shopping3-3.css">
```

loginsuccess.jsp

```
<%@ page contentType= "text/html; charset=UTF-8" %>
<%@ include file= "jsp_header.jsp" %>
<html>
<head>
<title>로그인화면</title>
</head>
<body>
<div align= "center" class="body">
<h2>로그인화면</h2>
환영해요, ${loginUser.userName} 씨 !
</div>
</body>
</html>
```

form:form태그

```
<form:form>
```

```
<form:input path="business.title" cssStyle="width:99%;" />
```

```
</form:form>
```

다음은 스프링의 form tag와 html의 form tag와의 비교

```
<form:input path="business.title" cssStyle="width:99%;" />
```

```
<input id="title" name="title" style="width:99%;" type="text" value=""/>
```

```
<form:form commandName= "member" ><form:input path= "userId" />
```

```
<input id= "userId" name= "userId" type= "text"  
value= "${member.userId}" />
```

form:form

form:input

form:checkbox

form:radiobutton

form:password

form:select

form:option

form:options

form:textarea

form:errors

form

text

checkbox

radio

password

select

option

option

textarea

span

1. <form:form>

1) **spring** 폼 태그를 사용하기 위해서는 **spring-form.tld**파일이 필요하고 이는 **spring-webmvc-2.5.2.jar** 파일에 포함되어 있다. 이 폼 태그를 사용하기 위해서는 **JSP** 페이지에 **taglib**을 추가해줘야한다.

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
```

2) '**form**'태그는 데이터 바인딩을 위해 태그 안에 바인딩 **path**를 지정해 줄 수 있다. 이 패스를 처음 보면 많이 이상한데 사용하다 보면 상당히 편한 기능이다. **path**에 해당되는 값은 도메인 모델의 **Bean** 객체를 의미한다. 사용 예는 다음과 같다.

```
<form:form commandName="user">
    userId : <form:input path="userId" />
</form:form>
```

spring form 태그를 이용하기 위해서는 각각의 입력 **path**값에 매칭될 도메인 객체를 지정해 줘야 하는데 **form**태그 안에 **commandName** 속성으로 다음과 같이 지정해 줄 수 있다.

```
<% request.setAttribute("user", sample.services.UserVO())%>
```

commandName의 기본값은 "**command**"이며 **input**값들과 매칭될 도메인 객체를 **request**값으로 세팅해 줘야한다. 이 값은 **SimpleFormController**를 사용할 경우 **FormBackingObject()** 메소드에서 지정해 줄 수도있다.

```
protected Object formBackingObject(HttpServletRequest request)
```

```
    throws Exception {
        UserVO vo=new UserVO();
        request.setAttribute("user",vo);
        return new UserVO();
```

```
}
```


2. <form:select> 태그

select tag도 위의 **checkboxes tag**나 **radiobuttons tag** 처럼 **items** 속성을 이용하여 **formBackingObject**에서 넘겨주는 값으로 자동 매핑 시켜줄 수 있다.

protected Object formBackingObject(HttpServletRequest request)

throws Exception {

```
UserVO vo=new UserVO();      Map address =new HashMap();
address.put("seoul","서울");  address.put("daegu","대구");
address.put("busan","부산");  request.setAttribute("address",address);
request.setAttribute("user",vo);
return new UserVO();
```

}

HTML에서는 다음과 같이 사용한다.

<tr> <td>주소</td>

<td><form:select path="address" items="\${address}" /></td></tr>

*** <form:option> 태그 이용 --> 일반적인 경우에 해당**

<tr><td>주소</td>

```
<td><form:select path="address"><form:option value="seoul" label="서울" />
<form:option value="daegu" label="대구" />
<form:option value="busan" label="부산" /></form:select></td></tr>
```

*** <form:options> 태그**

<tr><td>주소</td>

<td><form:select path="address">

<form:options items="\${address}" /></form:select></td>

</tr>

login.jsp

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ include file="jsp_header.jsp"%>
<html>
  <head><title>로그인 화면</title></head>
<body>
  <div align="center" class="body">
    <h2>로그인 화면</h2>
    <form:form modelAttribute="user" method="post"
      action="login.html">
      <spring:hasBindErrors name="user">
        <font color="red"><c:forEach items="${errors.globalErrors}"
          var="error">
            <spring:message code="${error.code}" />
          </c:forEach> </font>
      </spring:hasBindErrors>
    <table>
      <tr height="40px">
        <td>유저ID</td><td><form:input path="userId" cssClass="userId" />
          <font color="red"><form:errors path="userId" /></font></td>
      </tr>
```

```
<tr height= "40px">
<td>패스워드</td>
<td><form:password path= "password" cssClass="password" />
<font color= "red"><form:errors
path= "password" /></font></td>
</tr>
</table>
<table>
<tr>
<td align= "center"><input type="submit" value="로그인"></td>
<td align= "center"><input type="reset" value="리셋"></td>
</tr>
</table>
</form:form></div>
</body></html>
```

설정 파일(shopping33-servlet.xml)

LogFormController클래스는 요청 파라미터에서 받은 유저 ID와
패스워드를 검증하기 위해 LoginValidator를 이용

LoginValidator클래스는 또하나의 설정 파일인
applicationContext.xml에 정의

Native2ascii

native2ascii 변환시킬 파일명 변환후 파일명

```
<!-- HandlerMapping -->
<bean id="handlerMapping"
      class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <value>
      /login.html=loginFormController
    </value>
  </property>
</bean>
<!-- Controller -->
<bean id="loginFormController"
      class="controller.LoginFormController" />
<!-- ViewResolver -->
<bean id="internalResourceViewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="viewClass">
    <value>org.springframework.web.servlet.view.JstlView</value>
  </property>
  <property name="prefix" value="/WEB-INF/views/" />
  <property name="suffix" value=".jsp" />
</bean>
<context:component-scan base-package="dao,logic"/>
</beans>
```

applicationContext.xml

<!-- DataSource 생략 -->

<!-- Validator -->

<bean id= "*loginValidator*" class= "*logic.LoginValidator*" />

<!-- MessageSource -->

<bean id= "*messageSource*"

class= "*org.springframework.context.support.ResourceBundleMessageSource*"
>

<property name= "*basenames*">

<list>

<value>messages</value>

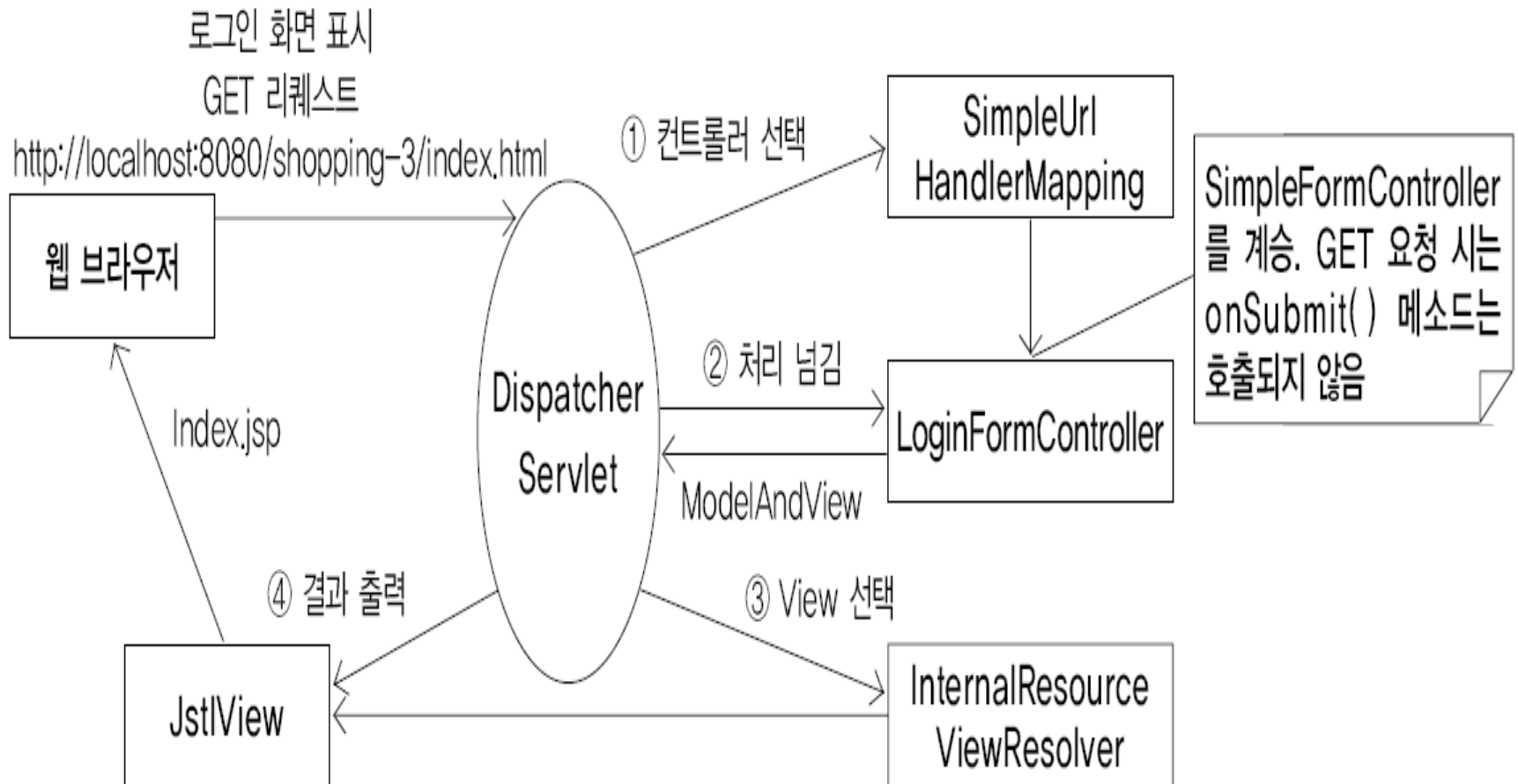
</list>

</property>

</bean>

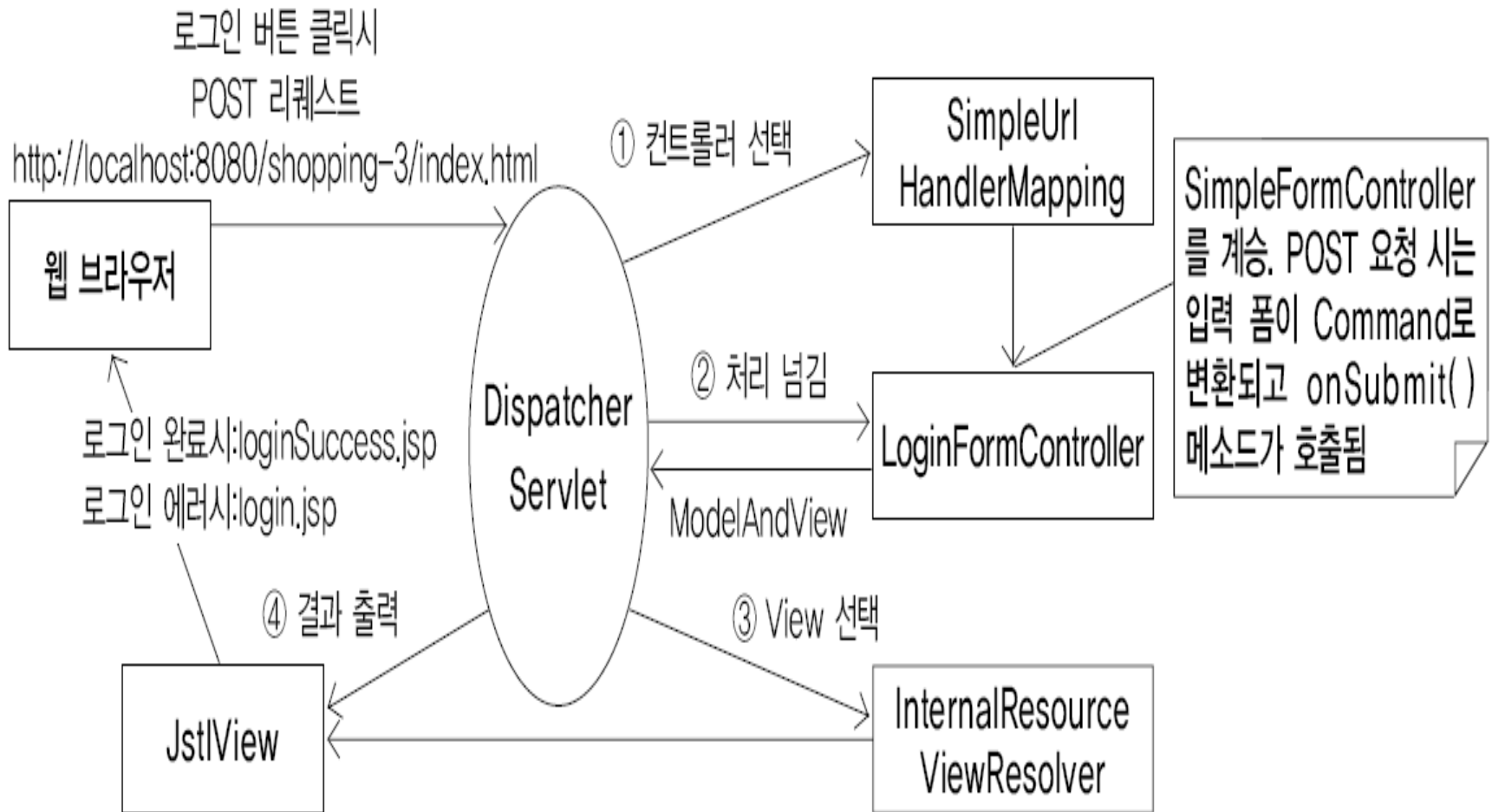
</beans>

예제의 화면이동과 화면정보 입출력



▲ 로그인 화면 표시(1)

예제의 화면이동과 화면정보 입출력



▲ 로그인 버튼 클릭(2)

```
<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<%@ include file="header.jsp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html><head><meta http-equiv="Content-Type" content="text/html;
charset=EUC-KR"><title>Insert title here</title></head>
<body>
<div align="center" class="body">
    <h2>로그인</h2>
<c:if test="${log!=null}">${log }<p></c:if>
<form method="post" action="login2.html">
<table>
    <tr height="40px"><td>유저ID</td>
        <td><input type="text" name="userId" /></td>
    </tr>
    <tr height="40px"><td>패스워드</td>
        <td><input type="text" name="passWord" /></td>
    </tr>
</table>
<table>
    <tr><td align="center"><input type="submit" value="로그인"></td>
        <td align="center"><input type="reset" value="리셋"></td></tr>
</table>
</form></div>
</body></html>
```


문제

member.jsp : ID와 **Password**를 입력하는 화면

- **html5 required**를 이용하여 데이터 입력 여부 체크

Member클래스 (**table member**와 매핑)

MemberFormController.java작성하여 처리

- **MemberDao.java, MemberDaoImpl.java**작성
- **Shop.java, ShopImpl.java** 멤버부분 추가

ID 또는 **Password**가 틀릴 경우
member.jsp화면에 에러 표시

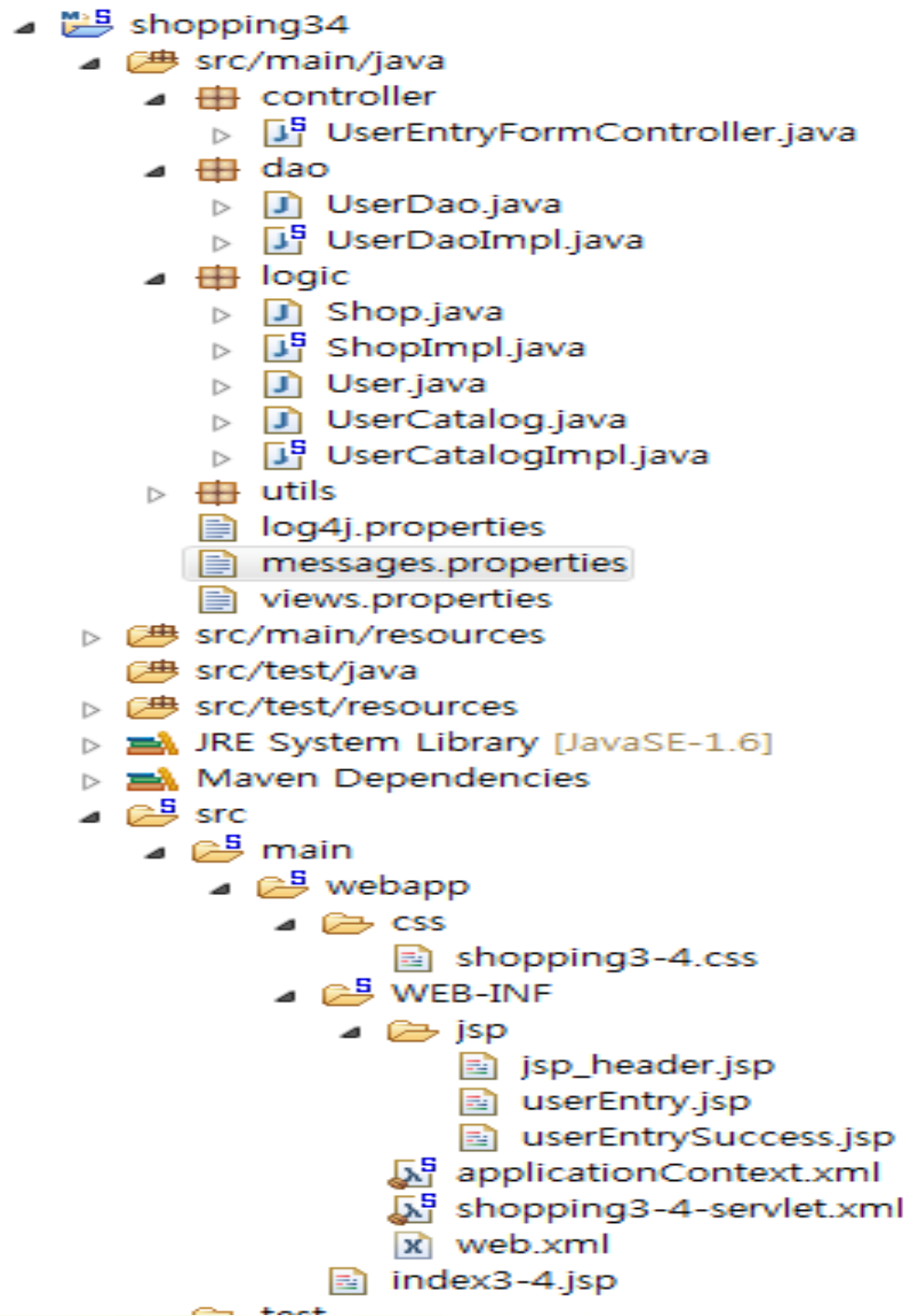
성공일 경우 **memberSuccess.jsp**에
xxx님 환영합니다

shopping33-servlet.xml 변경

회원 가입 페이지 만들기

이번 장의 목차

1. 개요
2. 화면에 표시하기



회원 가입 화면

http://localhost:8080/shopping-4/userEntry.html

회원 가입 화면

회원 ID

패스워드

이름

우편번호

주소

E-MAIL

직업

생년월일

회원 가입 완료 화면

http://localhost:8080/shopping-4/userEntry.html

회원 가입 완료 화면

회원 가입이 완료되었다.

회원 ID pinksubean

패스워드 pppp1111

이름 전수빈

우편번호 138-140

주소 서울시 송파구 석촌동

E-MAIL pinksubean@nate.com

직업 의사

생년월일 Tue Apr 14 00:00:00 KST 1998

개요

회원 가입 화면

회원 ID

패스워드

이름

우편번호

주소

E-MAIL

직업

생년월일

회원 가입 화면

입력된 회원 ID는 이미 사용되고 있습니다.

회원 ID

패스워드

이름

우편번호

주소

E-MAIL

직업

생년월일

샘플에서 사용하는 스프링 **MVC** 클래스

인터페이스	구현 클래스
HandlerMapping	ResourceBundleViewResolver
ViewResolver	ResourceBundleViewResolver
View	JstlView

ResourceBundleViewResolver

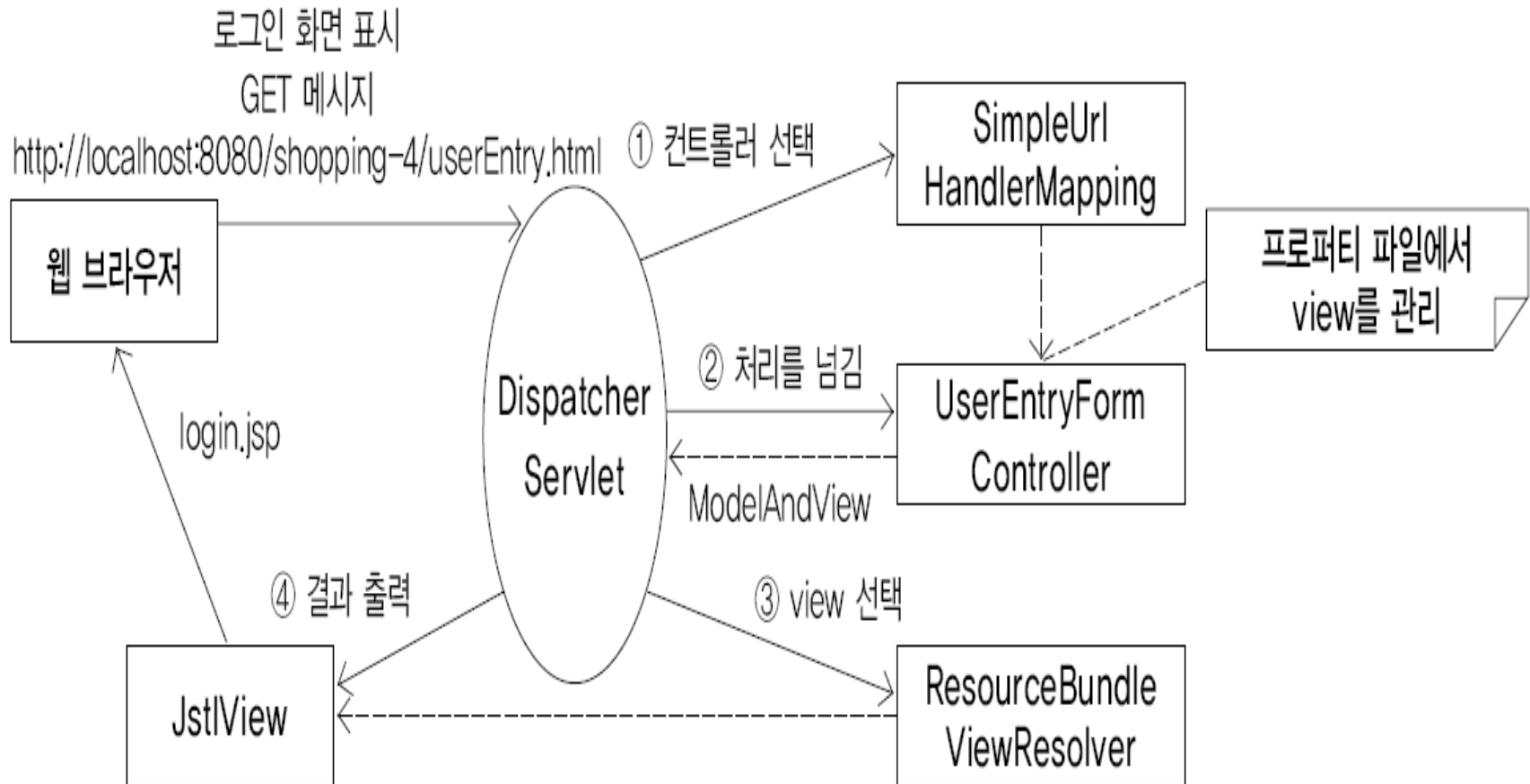
View 인터페이스를 구현하는 클래스와 이동할 뷰의 지정과 뷰 이름의 매핑을 프로퍼티 파일로 관리

뷰 이름은 실제 뷰 파일이 아닌 임의의 이름을 붙일 수 있기 때문에 뷰 파일 이름이 변경 되더라도 뷰 이름을 변경할 필요가 없고, 프로퍼티 파일 매핑 내용을 수정함, 컨트롤러를 수정할 필요 없음

프로퍼티의 파일 이름은 **basename** 프로퍼티에 지정

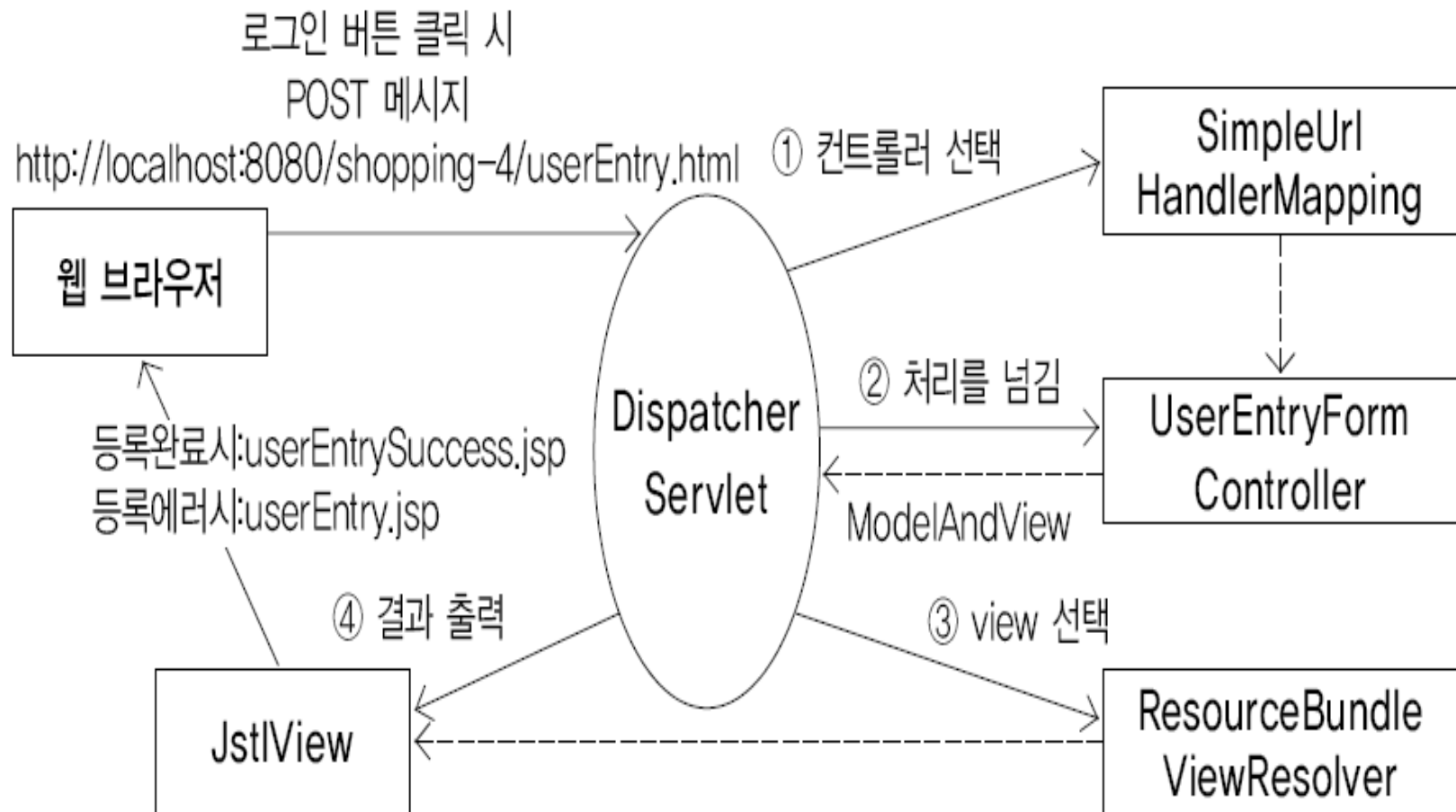
```
<!-- ViewResolver -->
<bean id="resourceBundleViewResolver" class
="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="basename">
        <value>views</value>
    </property>
</bean>
```

예제의 화면이동과 화면정보 입출력



▲ 회원 가입 화면 표시의 처리 흐름

예제의 화면이동과 화면정보 입출력



▲ [등록] 버튼 클릭 시 처리 흐름

UserDao.java

```
package dao;  
import logic.User;  
public interface UserDao { void create(User user); }
```

UserDaoImpl.java

```
package dao;  
@Repository  
public class UserDaoImpl implements UserDao{  
    @Autowired  
    private NamedParameterJdbcTemplate template;  
    public void create(User user) {  
        String sql = "insert into user_account (user_id,"  
            + "user_name, password, postcode, address, email,"  
            + "job, birthday) values(:userId, :userName,"  
            + ":password, :postCode, :address, :email,"  
            + ":job, :birthday)";  
        SqlParameterSource src =  
            new BeanPropertySqlParameterSource(user);  
        template.update(sql,src);  
    }  
}
```

Shop.java

```
package logic;  
public interface Shop {  
    void entryUser(User user);  
}
```

ShopImpl.java

```
package logic;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import dao.UserDao;  
@Service  
public class ShopImpl implements Shop {  
    @Autowired  
    private UserDao userDao;  
    public void create(User user) {  
        userDao.create(user);  
    }  
}
```

폼 데이터 검증

1. 웹 프로젝트에서 요청 파라미터 값을 확인하지 않고 그대로 사용하게 되면 데이터베이스에 잘못된 데이터가 저장되는 일이 발생할 수 있습니다.
2. 유효성 검사 방법
 - 1) 자바스크립트를 이용해서 데이터를 웹 서버에게 전송하기 전에 검사
 - 2) 전송받은 파라미터를 서버에서 검사한 후 파라미터가 잘못된 경우 재입력을 위한 화면을 출력하는 방법
3. 스프링은 이 중에서 서버 측에서 검사하는 방법을 제공
4. 자바 스크립트를 이용해서 유효성 검사를 하는 방법은 사용자에게 빠르게 검사 결과를 보여줄 수 있다는 장점이 있지만 악의적인 코드를 전송하는 등의 일이 발생할 수 있으므로 서버에서도 유효성 검사를 해주는 것이 바람직합니다.

폼 데이터 검증

1. Spring에서는 유효성 검사와 관련된 기능을 제공하는데 데이터 검증에 사용되는 인터페이스인 `org.springframework.validation.Validator`와 유효성 검사 결과를 저장할 `Errors/BindingResult` 인터페이스를 제공
2. `Validator` 인터페이스를 이용해서 유효성 검사를 하고 유효성 검사에 실패한 경우 `Errors` 인터페이스를 이용해서 에러 메시지를 저장한 후 뷰에 전달해서 출력하고 `BindingResult` 인터페이스에 에러 발생여부를 저장
3. `Validator` 인터페이스에는 2개의 메서드가 존재

- ❖ `public boolean supports(Class<?> clazz):` `Validator`가 해당 클래스에 대한 값 검증을 지원하는지의 여부를 설정하는 메서드

Ex) `return Info.class.isAssignableFrom(clazz);`

- ❖ `public void validate(Object target, Errors errors):` `target`에 대한 실제 검증을 수행합니다.

검증 결과에 문제가 발생하면 `errors` 객체에 문제에 대한 정보를 저장합니다.

Ex) `Info info = (Info) target;`

```
if (info.getId() == null || info.getId().trim().isEmpty()) {  
    errors.rejectValue("id", "required");  
}
```

`id`는 유효성 검사에 실패한 객체 이름이 되고 `required`는 뷰에서 에러메시지를 출력하도록 설정하면 출력되는 에러 메시지가 됩니다.

폼 데이터 검증

1. Errors 인터페이스: 유효성 검증 결과를 저장하기 위한 인터페이스 메서드

- 1) reject(String errorCode): 전체 객체에 대한 에러 코드 설정
- 2) reject(String errorCode, String defaultMessage): 에러 코드와 기본 메시지 설정
- 3) rejectValue(String field, String errorCode): field 대한 에러 코드 설정
- 4) rejectValue(String field, String errorCode, String defaultMessage)

2. BindingResult 인터페이스: Errors의 하위 인터페이스로 폼의 값들을 command 객체에 저장하고 에러 코드로부터 에러 메시지를 가져오는 인터페이스

- 1) boolean hasErrors()
- 2) int getErrorCount()
- 3) boolean hasGlobalErrors()
- 4) int getGlobalErrorCount()
- 5) boolean hasFieldErrors()
- 6) int getFieldErrorCount()
- 7) boolean hasFieldErrors(String field)
- 8) int getFieldErrorCount(String field)

UserEntryValidator.java

```
package utils;          import logic.User;
import org.springframework.util.StringUtils;
import org.springframework.validation.Errors;
import org.springframework.validation.Validator;
@Component
public class UserEntryValidator implements Validator {
    public boolean supports(Class<?> clazz) {
        return false;    }
    public void validate(Object command, Errors errors) {
        User user = (User) command;
        if (!StringUtils.hasLength(user.getUserId())) {
            errors.rejectValue("userId", "error.required");    }
        if (!StringUtils.hasLength(user.getPassword())) {
            errors.rejectValue("password", "error.required");    }
        if (!StringUtils.hasLength(user.getUserName())) {
            errors.rejectValue("userName", "error.required");    }
        if (!StringUtils.hasText(user.getPostCode())) {
            errors.rejectValue("postCode", "error.required");    }
        if (!StringUtils.hasText(user.getAddress())) {
            errors.rejectValue("address", "error.required");    }
        if (!StringUtils.hasText(user.getEmail())) {
            errors.rejectValue("email", "error.required");    }
        if (errors.hasErrors()) { errors.reject("error.input.user");    }
    }
}
```

@InitBinder

@InitBinder 를 이용한 타입 변환 처리스프링은 기본적으로 문자열을 지정한 타입의 값으로 변환할 때 자비빈의 **PropertyEditor** 를 사용.

스프링 **MVC** 가 **HTTP** 요청 파라미터를 객체의 프로퍼티로 저장할 때 사용하는 **WebDataBinder** 도 **PropertyEditor** 를 사용.

예:

'2012-12-02' 와 같은 **DATE** 형 문자열이 **HTTP** 요청 패러미터로 넘어왔다. 이것을 **Date** 타입의 프로퍼티에 저장하려면 **WebDataBinder** 에 커스텀 **PropertyEditor** 를 등록한다.

@InitBinder

```
protected void initBinder(WebDataBinder binder){  
    DateFormat dateFormat = new SimpleDateFormat('yyyy-MM-dd');  
    binder.registerCustomEditor(Date.class, new  
        CustomDateEditor(dateFormat, true));  
}
```

registerCustomEditor 의 패러미터는

1. 바인딩할 타입 클래스.
2. **CustomDateEditor** 객체

CustomDateEditor 의 패러미터는

1. 데이터포맷
2. null값 허용 여부(**boolean**) ;
false 설정 후 공백이나 **null** 을 넘기면 **typeMismatch** 코드 에러가 발생한다.
이 에러는 **BindingResult** 를 통해 확인할 수 있다

@ModelAttribute

- 1) **@ModelAttribute** 는 이름 그대로 모델로 사용되는 오브젝트이다.
컨트롤러가 뷰에 출력할 정보를 전달하기 위해 ModelAndView 에 담아서 전달하는 모델과는 조금 의미가 다르다.
- 2) 컨트롤러가 뷰에 전달하는 모델 오브젝트는 하나가 아니다.
맵 형태의 컬렉션을 이용해 여러 개의 모델 오브젝트를 담아서 전달 하는 것이다.
- 3) **@ModelAttribute** 는 모델 맵에 담겨서 뷰에 전달되는 모델 오브젝트의 한 가지라고도 볼 수 있다. 기본적으로 **@ModelAttribute** 는 별도의 설정 없이도 자동으로 뷰에 전달된다.
- 4) **@ModelAttribute** 와 **@RequestParam** 의 차이점 단지 요청 파라미터를 메소드 파라미터에서 1:1 로 받으면 **@RequestParam** 이고,
도메인 오브젝트나 DTO 의 프로퍼티에 바인딩해서 한 번에 받으면 **@ModelAttribute** 라고 볼 수 있다.
- 5) 하나의 오브젝트에 클라이언트의 요청정보를 담아서 한 번에 전달되는 것이기 때문에 이를 커맨드 패턴에서 말하는 커맨드 오브젝트라고 부르기도 한다

UserEntryFormController.java

```
package controller;
```

```
import java.text.DateFormat;  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import logic.Shop;  
import logic.User;  
import org.springframework.beans.propertyeditors.CustomDateEditor;  
import org.springframework.context.MessageSource;  
import org.springframework.context.support.MessageSourceAccessor;  
import org.springframework.dao.DataIntegrityViolationException;  
import org.springframework.stereotype.Controller;  
import org.springframework.validation.BindingResult;  
import org.springframework.web.bind.WebDataBinder;  
import org.springframework.web.bind.annotation.InitBinder;  
import org.springframework.web.bind.annotation.ModelAttribute;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestMethod;  
import org.springframework.web.servlet.ModelAndView;  
import utils.UserEntryValidator;
```

@Controller

public class UserEntryFormController {

@Autowired

private Shop shop;

@Autowired

private Validator validator;

@Autowired

private MessageSource messageSource;

@RequestMapping(method=RequestMethod.*GET*)

public String userEntry() { return "userEntry"; }

@ModelAttribute

public User setForm() {

User user = new User();

MessageSourceAccessor msa =

new MessageSourceAccessor(messageSource);

user.setUserId(msa.getMessage("user.userId.default"));

user.setUserName(msa.getMessage("user.userName.default"));

return user;

}

@InitBinder

public void InitBinder(WebDataBinder wdb) {

DateFormat df = new SimpleDateFormat("yyyy-MM-dd");

df.setLenient(false);

wdb.registerCustomEditor(Date.class, "birthday",

new CustomDateEditor(df, false));

}

```
@RequestMapping(method=RequestMethod.POST)
public ModelAndView onsubmit(User user, BindingResult br) {
    ModelAndView mav = new ModelAndView();
    validator.validate(user, br);
    if (br.hasErrors()) {
        mav.getModel().putAll(br.getModel());
        return mav;
    }
    try {
        shop.create(user);
        mav.addObject("user",user);
        mav.setViewName("userEntrySuccess");
    } catch (DataIntegrityViolationException e) {
        br.reject("error.duplicate.user");
        mav.getModel().putAll(br.getModel());
        return mav;
    }
    return mav;
}
```

폼 데이터 검증

❖에러코드에 해당하는 프로퍼티에 에러 메시지 작성은 단순히 메시지만 작성할 수 있고 특정 객체에 대한 메시지를 작성할 수 있습니다.

required=\ud544\uc218 \ud56d\ubaa9\uc785\ub2c8\ub2e4.

required.password=\ud544\uc218 \ud56d\ubaa9\uc785\ub2c8\ub2e4

messages.properties

error.input.user= 입력정보에 문제가 있습니다.

error.required.user.userId=유저 ID를 입력하세요

error.required.user.password=패스워드를 입력하세요

error.required.user.userName=이름을 입력하세요

error.required.user.postCode=우편번호를 입력하세요

error.required.user.address=주소를 입력해 주세요

error.required.user.phoneNo=전화번호를 입력하십시오

error.required.user.email=E-MAIL을 입력하세요

error.duplicate.user=이미 사용중임 **user ID**입니다

user.userId.default=로그인할 때 필요합니다

user.userName.default=배송할 때 필요합니다

typeMismatch.user.birthDay=생년월일은 **1900-01-01** 형식으로 입력하십시오

view.properties

#userEntry View

userEntry.(class)=org.springframework.web.servlet.view.JstlView

userEntry.url=WEB-INF/jsp/userEntry.jsp

#userEntrySuccess View

userEntrySuccess.(class)=org.springframework.web.servlet.view.JstlView

userEntrySuccess.url=WEB-INF/jsp/userEntrySuccess.jsp

shopping3-4.css

```
body { font-family: 굴림체; }
div.body { overflow-y: auto; scrollbar-face-color: #C9BFED;
  scrollbar-shadow-color: #EDEDED;
  margin-top: 50px; margin-bottom: 50px;
}
tr.header { background: #C9BFED; }
tr.record { background: #EDEDED; }
input.userId { height: 20px; width: 150px;
  border-width: 1px; border-style: solid; }
input.password { height: 20px; width: 150px;
  border-width: 1px; border-style: solid; }
input.userName { height: 20px; width: 150px;
  border-width: 1px; border-style: solid; }
input.postCode { height: 20px; width: 80px;
  border-width: 1px; border-style: solid; }
input.address { height: 20px; width: 240px;
  border-width: 1px; border-style: solid; }
input.email { height: 20px; width: 240px;
  border-width: 1px; border-style: solid; }
input.birthDay { height: 20px; width: 100px;
  border-width: 1px; border-style: solid; }
select.jobs { width: 100px; }
```

jsp_header.jsp 빨강부분 추가

```
<%@ page session= "false"%>
```

```
<%@ taglib prefix= "c" uri="http://java.sun.com/jsp/jstl/core"%>
```

```
<%@ taglib prefix= "spring" uri="http://www.springframework.org/tags"%>
```

```
<%@ taglib prefix= "form" uri="http://www.springframework.org/tags/form"%>
```

```
<%@ taglib prefix= "f" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

```
<link rel= "stylesheet" type="text/css" href="css/shopping3-4.css">
```

jsp에서 에러 메시지 출력

<spring:hasBindErrors name= “커맨드 객체 이름” > 커스텀 태그를
이용해서 에러 정보를 설정하고 <form:errors path= “커맨드 객체” />를
이용하여 **global** 에러 정보를 설정하고 각각의 에러는 <form:errors
path= “커맨드 객체.프로퍼티” />를 이용


```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ include file=" ../jsp/jsp_header.jsp"%>
<html><head><title>유저 등록 화면</title></head>
<body>
<div align="center" class="body"><h2>유저 등록 화면</h2>
<form:form modelAttribute="user" method="post" action="userEntry.html">
  <spring:hasBindErrors name="user"><font color="red">
    <c:forEach items="${errors.globalErrors}" var="error">
      <spring:message code="${error.code}" />
    </c:forEach> </font>
  </spring:hasBindErrors>
<table>
  <tr height="40px"><td>유저ID</td>
    <td><form:input path="userId" maxLength="20" cssClass="userId" />
      <font color="red"><form:errors path="userId" /></font></td> </tr>
  <tr height="40px"><td>패스워드</td>
    <td><form:password path="password" maxLength="20"
      cssClass="password" /> <font color="red">
      <form:errors path="password" /></font></td> </tr>
  <tr height="40px"><td>이름</td>
    <td><form:input path="userName" maxLength="20" cssClass="userName"
      /><font color="red"><form:errors path="userName" /></font></td></tr>
  <tr height="40px"><td>우편번호</td>
    <td><form:input path="postCode" maxLength="8" cssClass="postCode" />
      <font color="red"><form:errors path="postCode" /></font></td> </tr>
  <tr height="40px"><td>주소</td>
    <td><form:input path="address" maxLength="50" cssClass="address" />
```



```

<font color="red"><form:errors path="address" /></font></td> </tr>
<tr height="40px"><td>E-MAIL</td>
    <td><form:input path="email" maxlength="50" cssClass="email" />
        <font color="red"><form:errors path="email" /></font></td> </tr>
<tr height="40px"><td>직업</td>
    <td><form:select path="job" cssClass="jobs">
        <form:option value="사회인" label="사회인" />
        <form:option value="주부" label="주부" />
        <form:option value="학생" label="학생" />
        <form:option value="그외" label="그외" />
    </form:select></td></tr>
<tr height="40px"><td>생년월일</td>
    <td><form:input path="birthDay" maxlength="10" cssClass="birthDay" />
        <font color="red"><form:errors path="birthDay" /></font></td></tr>
</table>
<table>
    <tr>
        <td height="40px" align="center"><input type="submit"
            name="btnSubmit" value="등록"></td>
        <td height="40px" align="center"><input type="reset"
            name="btnReset" value="리셋"></td>
    </tr>
</table>
</form:form></div>
</body>
</html>

```

userEntrySuccess.jsp

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ include file=" ../jsp/jsp_header.jsp"%>
<html><head><title>유저 등록 완료 화면</title></head>
<body>
  <div align="center" class="body"><h2>유저 등록 완료 화면</h2>
  <b><font color="red">유저 등록이 완료되었습니다.</font></b><br>
  <table>
    <tr height="40px"><td>유저ID</td><td>${user.userId}</td></tr>
    <tr height="40px"><td>패스워드</td><td>${user.password}</td></tr>
    <tr height="40px"><td>이름</td><td>${user.userName}</td></tr>
    <tr height="40px"><td>우편번호</td><td>${user.postCode}</td></tr>
    <tr height="40px"><td>주소</td><td>${user.address}</td></tr>
    <tr height="40px"><td>E-MAIL</td><td>${user.email}</td></tr>
    <tr height="40px"><td>직업</td><td>${user.job}</td></tr>
    <tr height="40px"><td>생년월일</td>
      <td><f:formatDate value="${user.birthDay}"
        pattern="yyyy년MM월dd일" /></td>
    </tr>
  </table>
</div>
</body>
</html>
```

shopping3-4.xml

```
<!-- HandlerMapping -->
<bean id= "handlerMapping"
  class= "org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name= "mappings">
    <value>
      /userEntry.html=userEntryFormController
    </value>
  </property>
</bean>
<!-- Controller -->
<bean id= "userEntryFormController" class= "logic.UserEntryFormController" />
<!-- ViewResolver -->
<bean id= "resourceBundleViewResolver"
  class = "org.springframework.web.servlet.view.ResourceBundleViewResolver">
  <property name= "basename" value= "views" />
</bean>
<context:component-scan base-package= "dao,logic" />
</beans>
```

폼 데이터 검증

에러 코드에 대한 에러 메시지 설정을 하고자 할 때는 프로퍼티 파일을 **MessageSource**로 등록해야 합니다.

```
<bean id= "아이디"  
class="org.springframework.context.support.ResourceBundleMessageSource">  
    <property name="basenames">  
        <list>  
            <value>properties 파일 경로</value>  
        </list>  
    </property>  
</bean>
```

applicationContext.xml

```
<!-- Data Source -->
<bean id= "template"
class="org.springframework.jdbc.core.namedparam.NamedParameterJdbcTem
plate">
<constructor-arg ref= "dataSource" />
</bean>
<bean id= "dataSource"
class= "org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name= "driverClassName"
        value= "oracle.jdbc.driver.OracleDriver"/>
    <property name= "url"
        value= "jdbc:oracle:thin:@127.0.0.1:1521:xe"/>
    <property name= "username" value="scott"/>
    <property name= "password" value="tiger"/>
</bean>
<!-- MessageSource -->
<bean id= "messageSource"
class= "org.springframework.context.support.ResourceBundleMessageSource"
>
<property name= "basenames">
    <list> <value>messages</value> </list>
</property>
</bean>
</beans>
```

문제

Emp 테이블에 다음을 입력
사번, 이름, 업무, 관리자, 입사일, 급여, 보너스, 부서코드

힌트

Emp

```
private int empno; private String ename;  
private String job;private int mgr;  
private String hiredate;private int sal;  
private int comm;private int deptno;
```

```
@RequestMapping(value="empEntry",method=RequestMethod.POST)  
public ModelAndView onsubmit(@ModelAttribute Emp emp,  
    HttpServletRequest request)
```

EmpDao.java, EmpDaoImpl.java, Emp.java. EmpEntryFormController.java 작성
Shop.java, ShopImpl.java , shopping34-servlet.xml, views.properties수정

Web

empEntry.jsp, empEntrySuccess.jsp 작성