

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

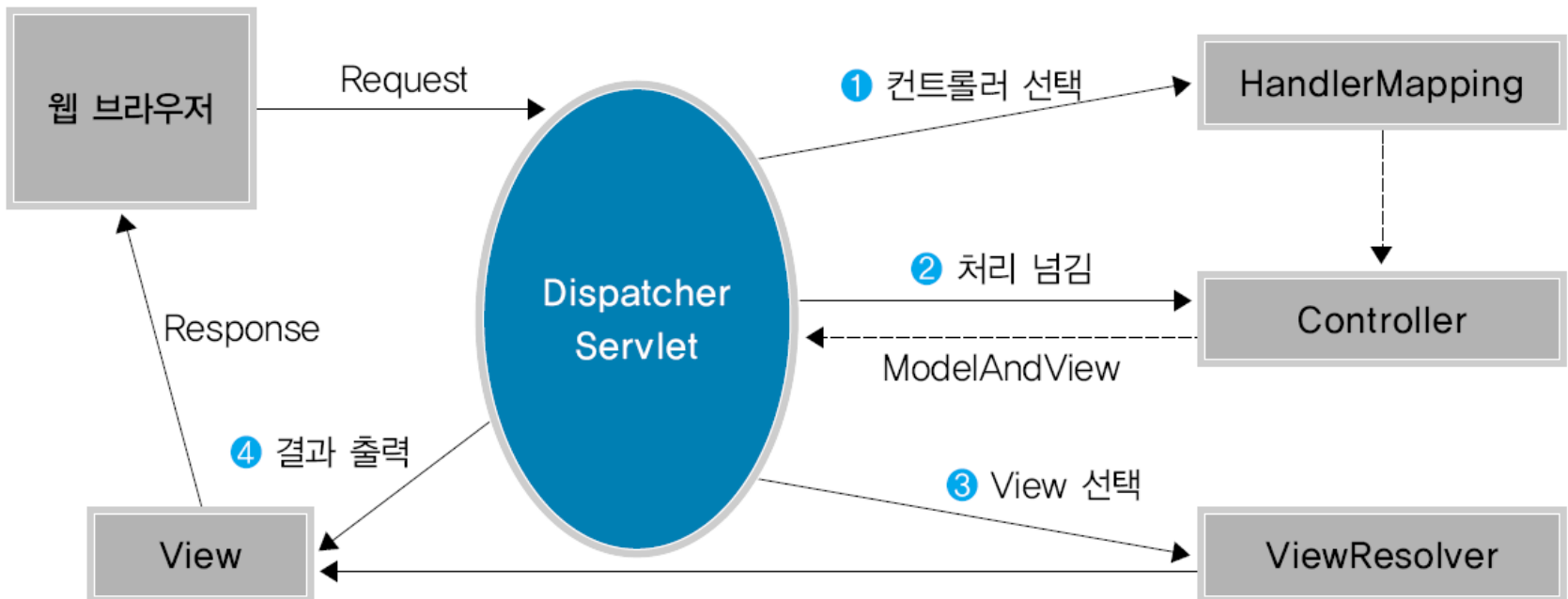
Service :

여러 **DAO**를 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

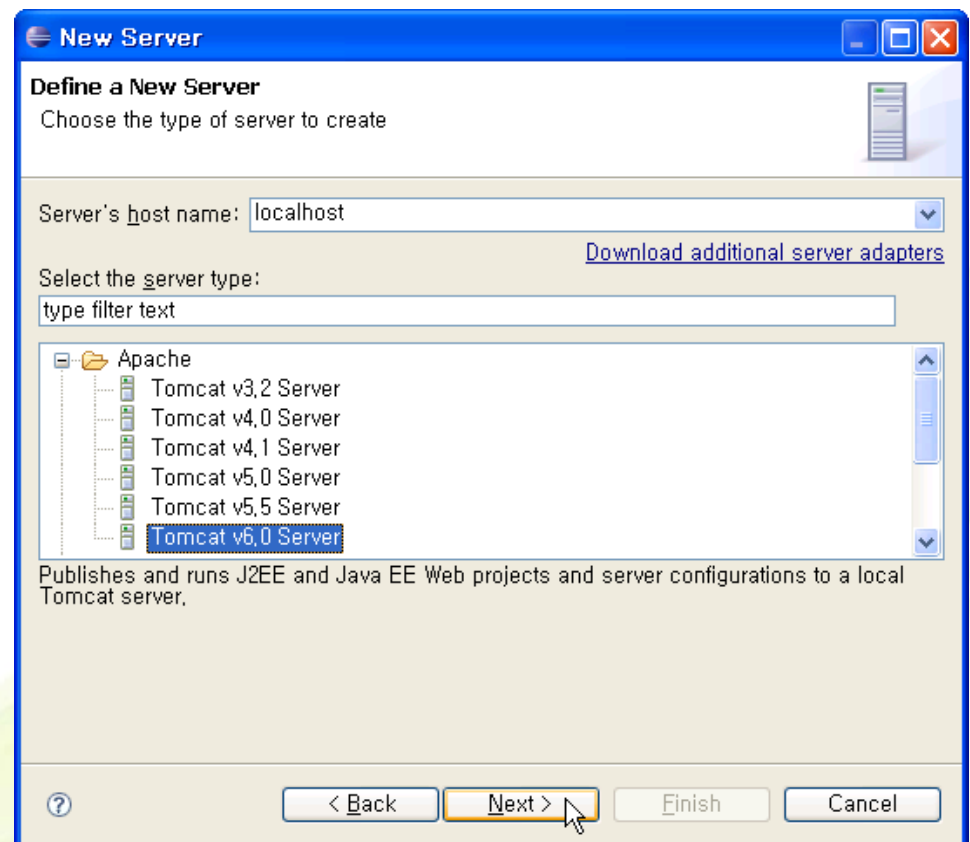
이클립스에 동적 웹 프로젝트 설정

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');
```

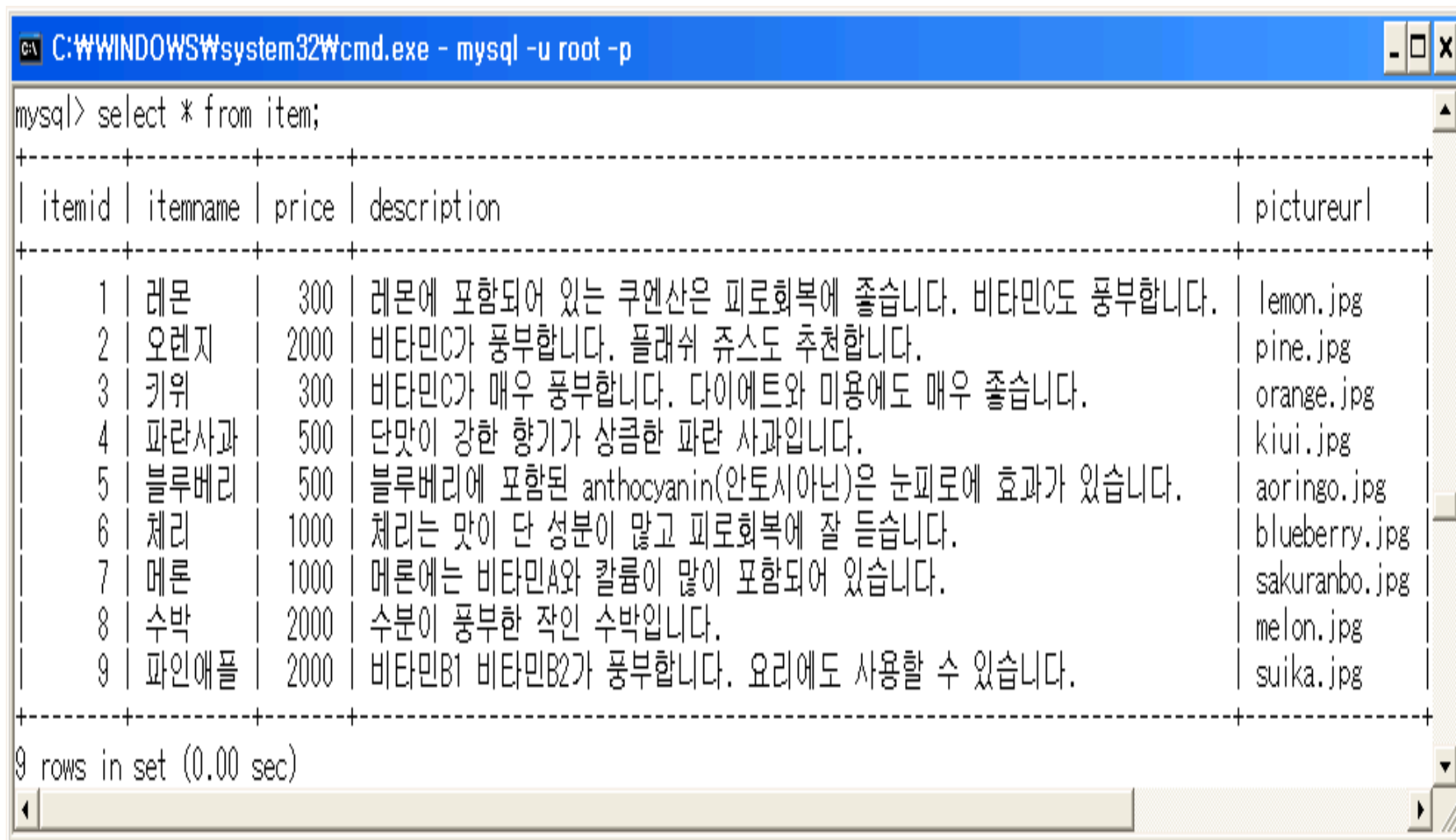
```
insert into item values(item_seq.nextval,'딸기',300,'폴리페놀을 다량 함유하고  
있어 항산화 작용','ichigo.jpg');
```

```
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



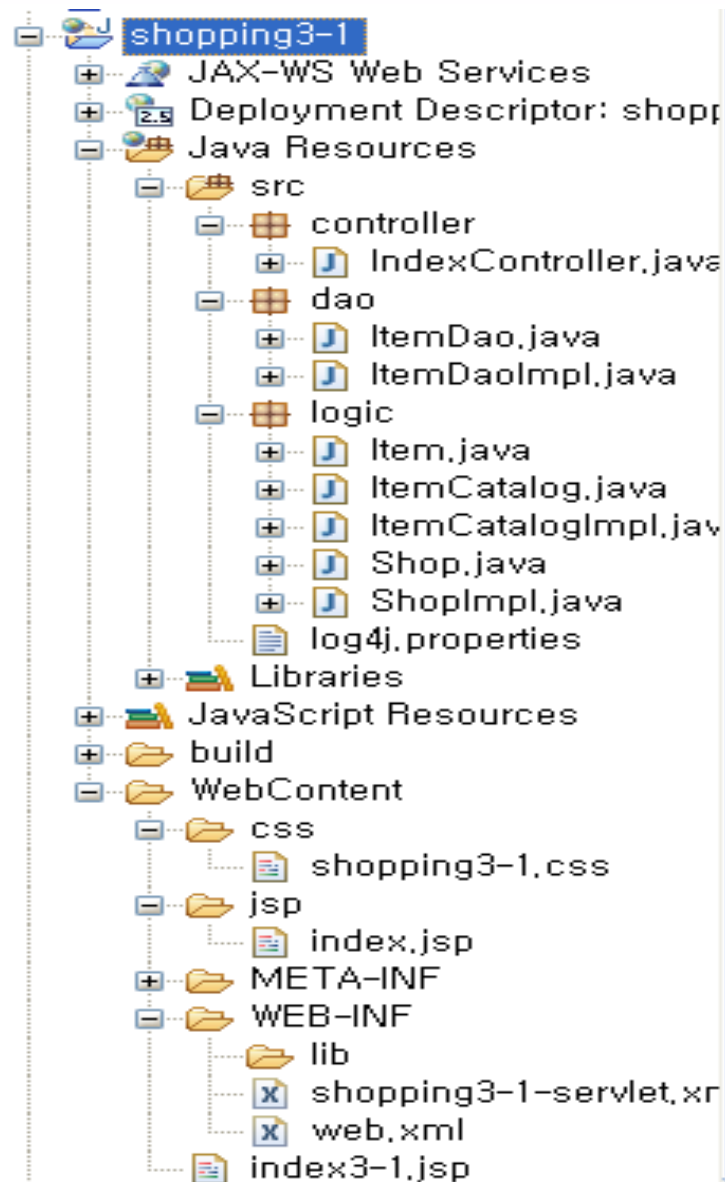
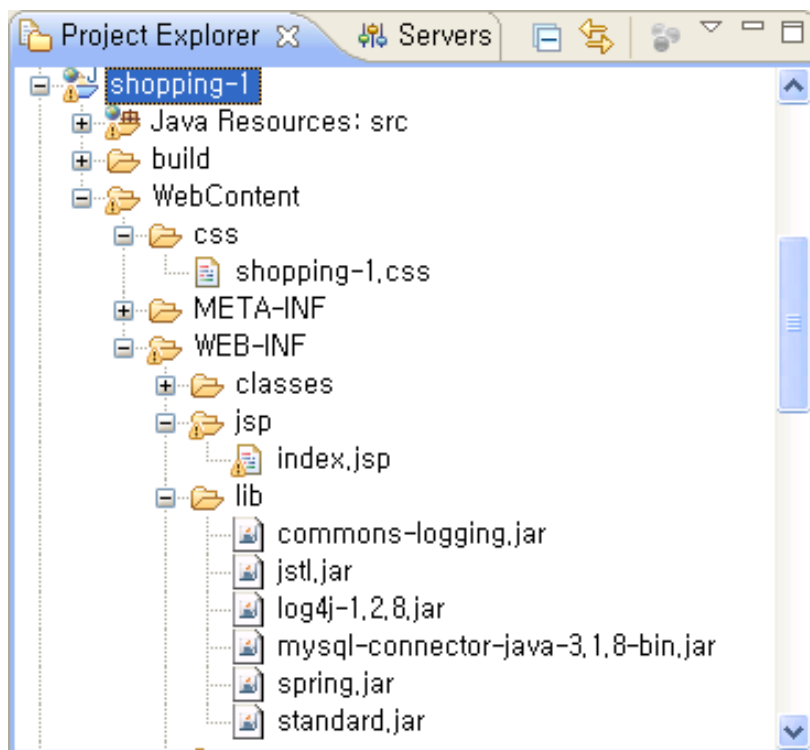
The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe - mysql -u root -p". The user has entered the command "mysql> select * from item;". The output is a table with 5 columns: itemid, itemname, price, description, and pictureurl. There are 9 rows of data, each representing a fruit item with its details 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.ch</groupId>
  <artifactId>shopping1</artifactId>
  <name>shopping1</name>
  <packaging>war</packaging>
  <version>1.0.0-BUILD-SNAPSHOT</version>
  <properties>
    <java-version>1.8</java-version>
    <org.springframework-
version>4.2.4.RELEASE</org.springframework-version>
    <org.aspectj-version>1.6.10</org.aspectj-version>
    <org.slf4j-version>1.6.6</org.slf4j-version>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.projectlombok</groupId>
      <artifactId>lombok</artifactId>
      <version>1.18.12</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>
<!-- validator -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.1.3.Final</version>
</dependency>
<!-- oracle -->
<dependency>
    <groupId>com.oracle.ojdbc</groupId>
    <artifactId>ojdbc8</artifactId><version>19.3.0.0</version>
</dependency>
<!-- dbcp jdbc connection pool -->
<dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.2.2</version>
</dependency>
```

```
<dependency>
    <groupId>com.zaxxer</groupId>
    <artifactId>HikariCP</artifactId>
    <version>2.7.4</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<!-- mybatis -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.2.3</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.2.1</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<!-- cglib ibatis -->
<dependency>
    <groupId>cglib</groupId>
    <artifactId>cglib</artifactId>
    <version>2.2</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-expression</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${org.springframework-version}</version>
    <exclusions>
        <exclusion>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjrt</artifactId>
    <version>${org.aspectj-version}</version>
</dependency>
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>${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>2.2</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>


```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId> <version>1.2</version>
</dependency>
<!-- Spring Security -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-core</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId><version>1.3.2</version>
</dependency>
```

```
<dependency>
    <groupId>org.scala-lang</groupId>
    <artifactId>scala-library</artifactId><version>2.10.4</version>
</dependency>
<!-- javax.mail -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
<dependency>
    <groupId>javax.mail</groupId>
    <artifactId>mail</artifactId><version>1.4.7</version>
</dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>${java-version}</source>
                <target>${java-version}</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

<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

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(❷).
- DispatcherServlet 클래스에는 <servlet-name> 태그에 의해 'shopping3-1' 이라고 서블릿 이름을 지정하여 정의하고 있다(❸).
- 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>
```



db.properties

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



shopping1-servlet.xml

```
<!-- Handler -->
<!-- classpath: - resources에 파일이 있음 -->
<context:property-placeholder location="classpath:db.properties" />
<context:component-scan base-
package="com.ch.shopping1"></context:component-scan>
<bean id="ic" name="/fruitList.html"
      class="com.ch.shopping1.controller.ItemController"></bean>
<bean id="dc" name="/deptList.html"
      class="com.ch.shopping1.controller.DeptController"></bean>
<bean id="jdbcTemplate"
      class="org.springframework.jdbc.core.JdbcTemplate">
  <constructor-arg ref="dataSource" />
</bean>
<!-- DB연결 풀 -->
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
  <property name="driverClassName" value="${driver}" />
  <property name="jdbcUrl" value="${url}" />
  <property name="username" value="${userId}" />
  <property name="password" value="${password}" />
</bean>
<bean id="dataSource" class="com.zaxxer.hikari.HikariDataSource"
      destroy-method="close">
  <constructor-arg ref="hikariConfig"></constructor-arg>
</bean>
```


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.

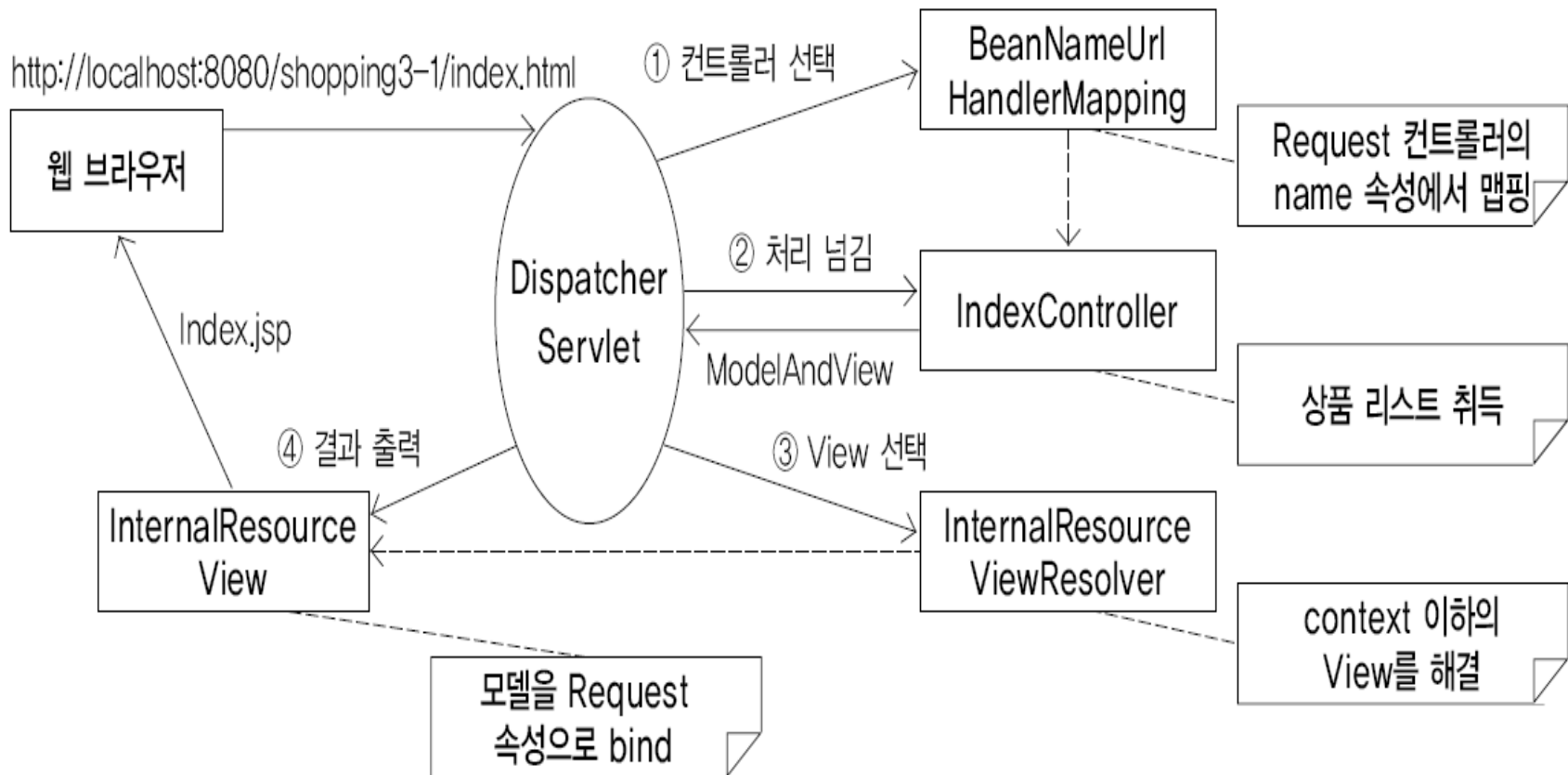
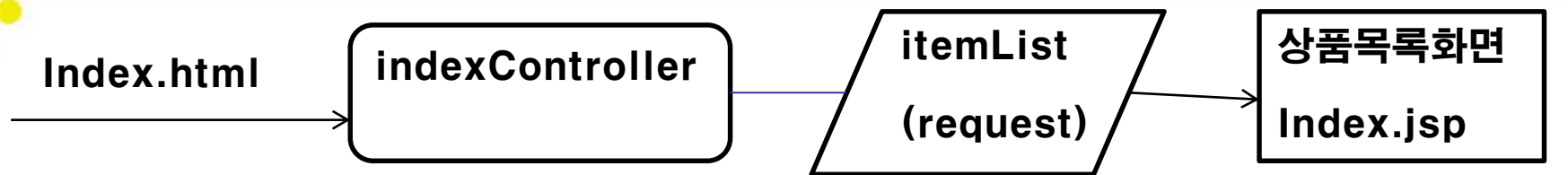
가지고 있는 값에 따라 초기화 순서가 결정되며, 값이 같을 경우 **WAS** 임의로 순서를 정해서 초기화한다.

<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

DeptController.java, Dept.java, deptList.jsp

2. 변경

**ItemDao.java, ItemDaoImpl.java,
Shop.java, ShopImpl.java
shopping1-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

ViewResolver 구현하는 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>
```

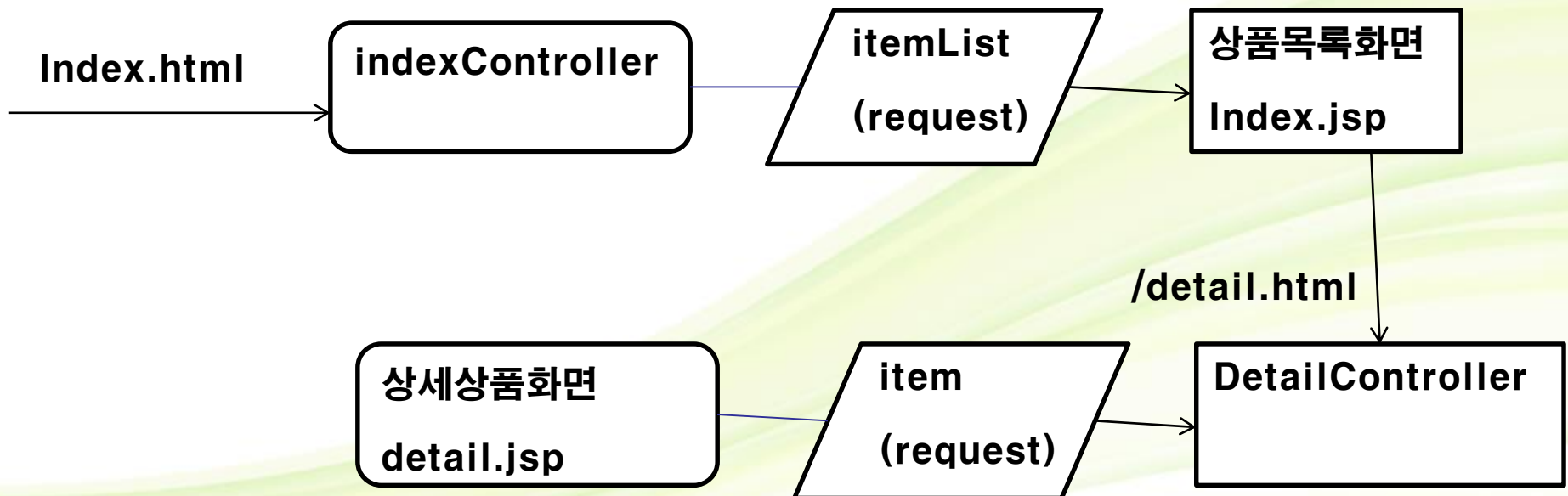
JstlView의 핵심

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

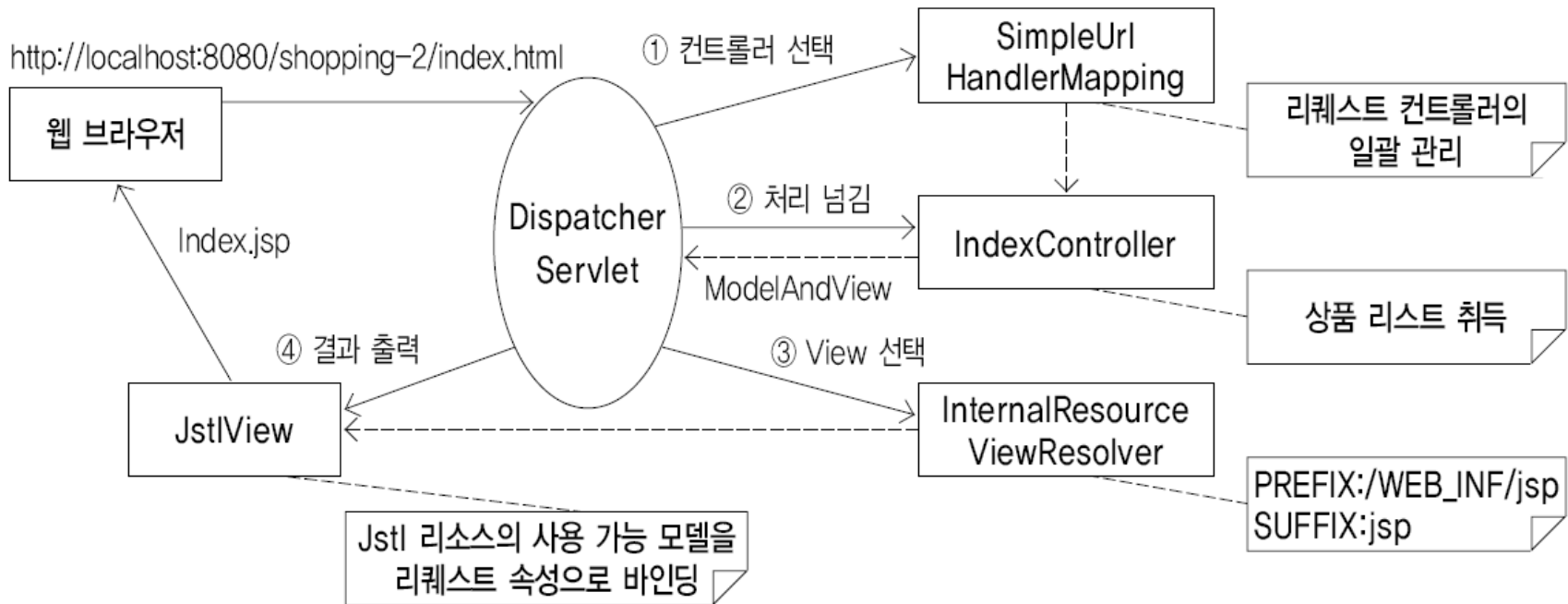
설정파일(web.xml)의 핵심

- 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

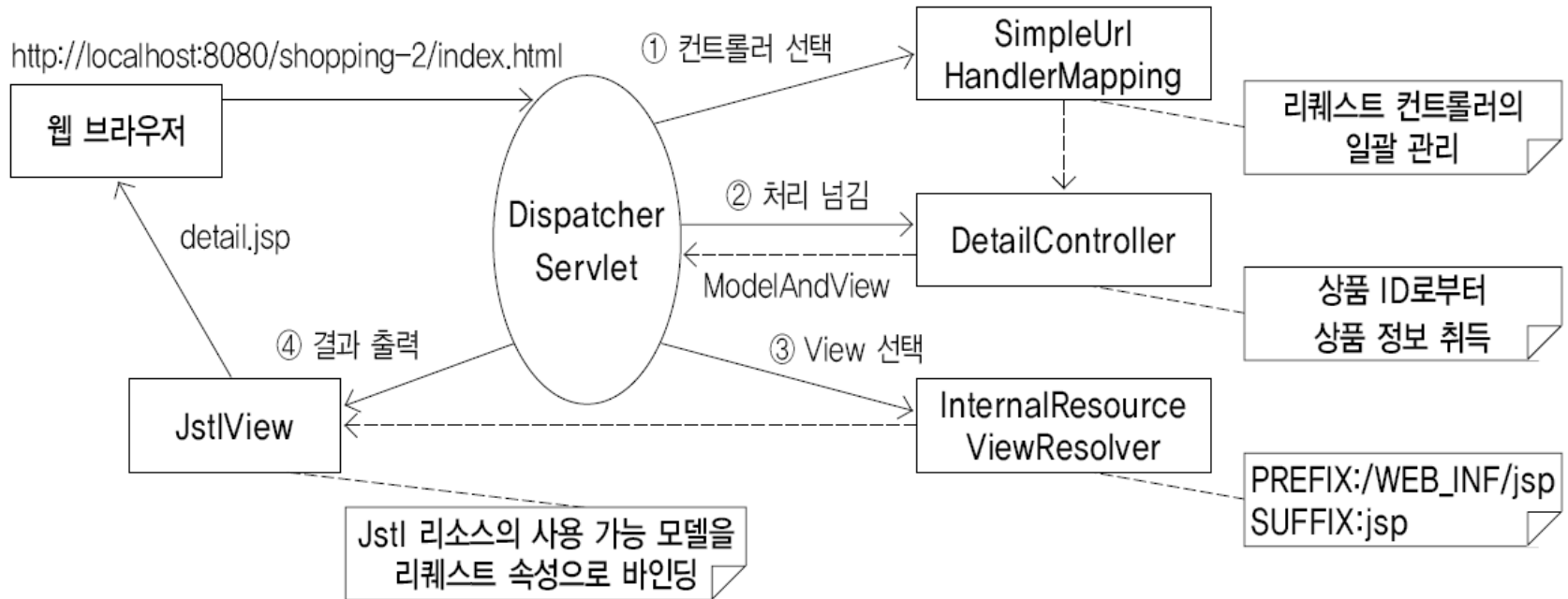


예제의 화면 이동과 화면 정보 입출력



▲ 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

/list.html=ic

/detail.html=ic

/deptList.html=dc

/detailDept.html=dc

@RequestMapping(value="detailDept")

이클립스 Maven 연동 시 plug in 에러 날 경우

POM.xml에

<dependency>

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

<artifactId>maven-resources-plugin</artifactId>

<version>2.4.3</version>

</dependency>

이렇게 작성해준 뒤에,

1. 프로젝트 우클릭 > **Run As > Maven Install**
2. 이클립스 프로젝트 탐색기에서 해당 프로젝트 클릭 후 **F5(새로고침)**
3. 프로젝트 우클릭 > **Maven > Update Project**