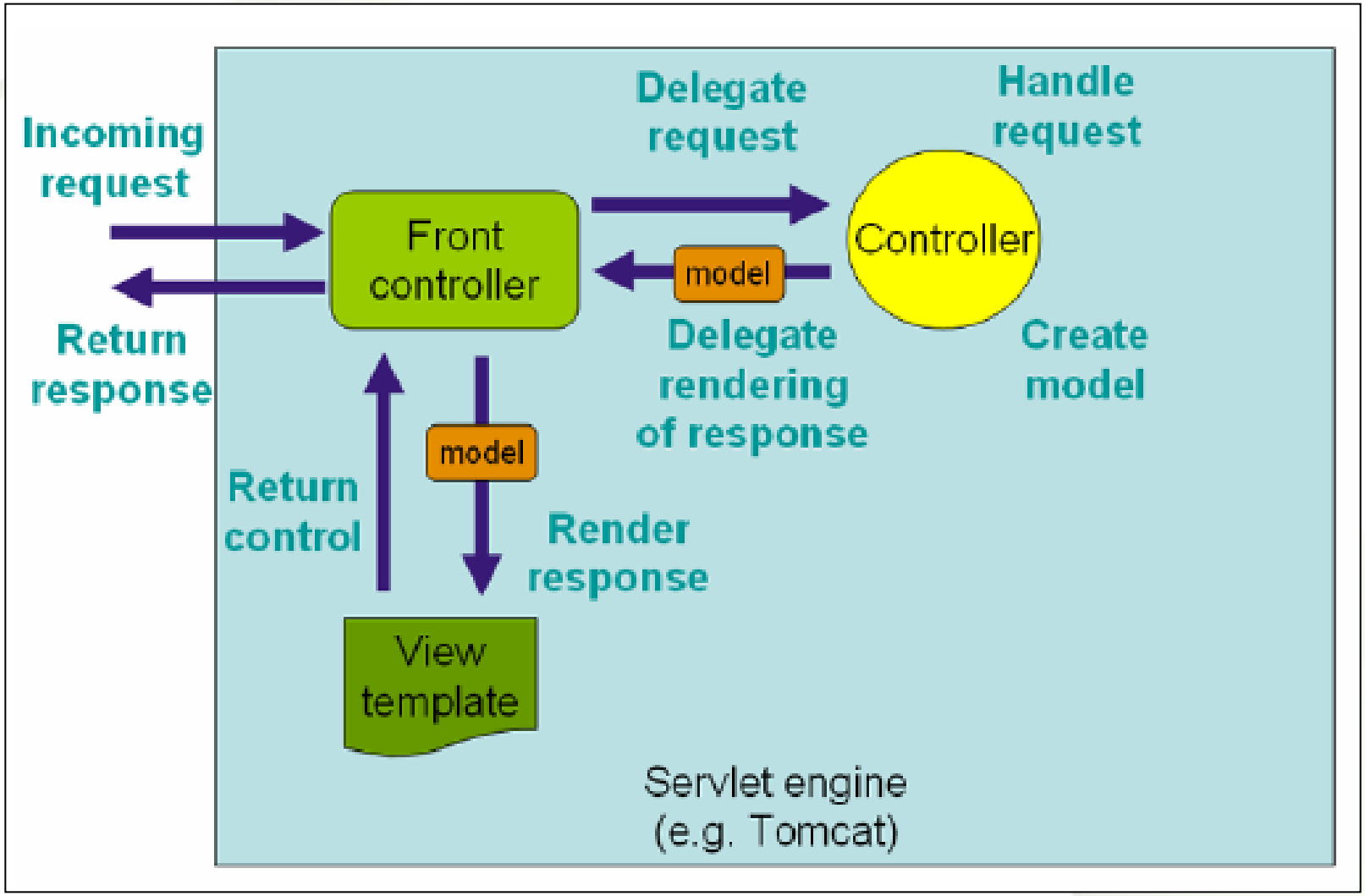
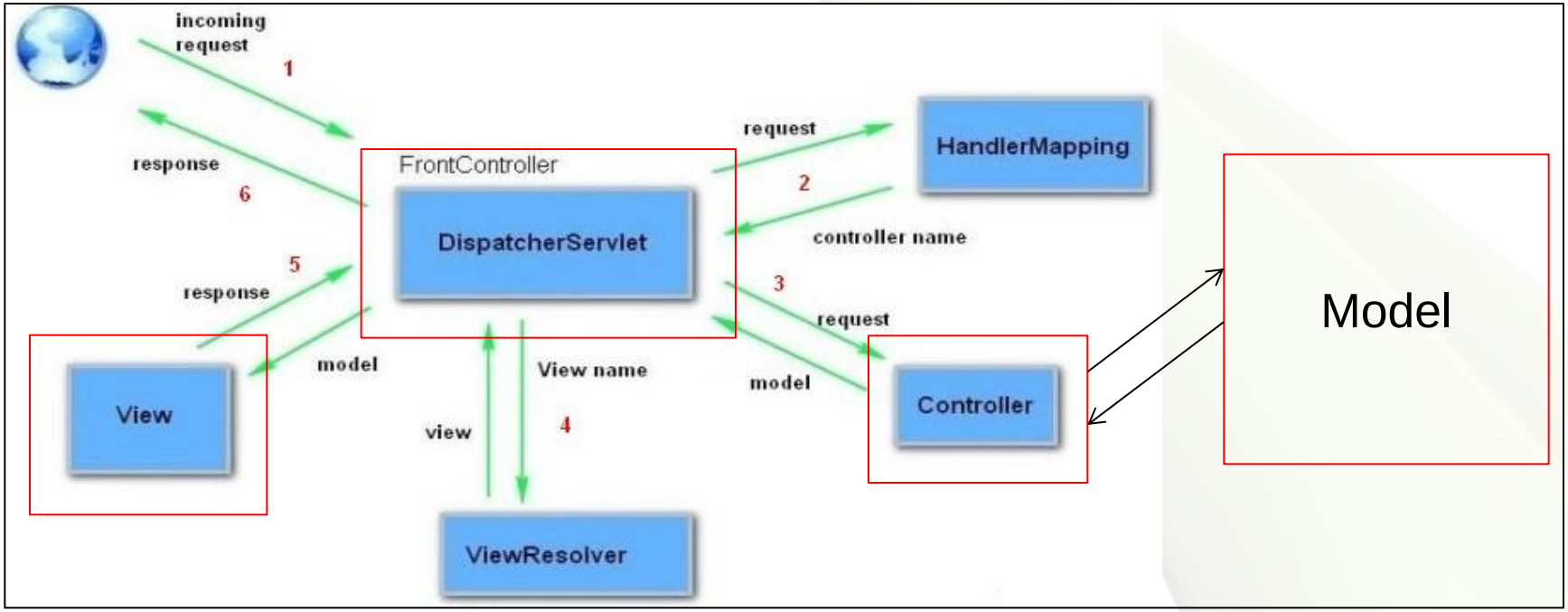
스프링 MVC

FrontController Based MVC Pattern



스프링 MVC

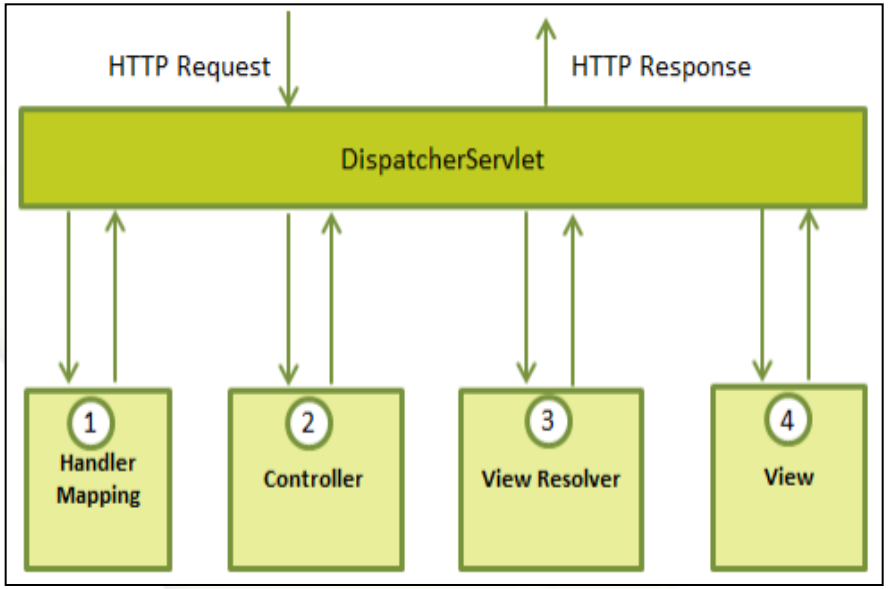
- Front Controller Pattern 기반 구현
- DispatcherServlet을 FrontController 구현체로 제공
- DispatcherServlet 생성과 함께 WebApplicationContext 객체 생성
- web.xml 및 스프링 bean 설정 파일의 내용에 따라 동작



WebApplicationContext

요청 처리 과정

- DispatcherServlet 요청 수신
- DispatcherServlet은 HandlerMapping을 통해 적절한 Controller를 선택하고 Controller에 요청 위임
- Controller는 Model 영역의 객체 생성 및 호출 후 Model 객체 생성 / 데이터 할당 / 뷰이름 지정한 후 FrontController에 반환
  - 일반적으로 ModelAndView 형식의 객체 사용해서 결과 반환

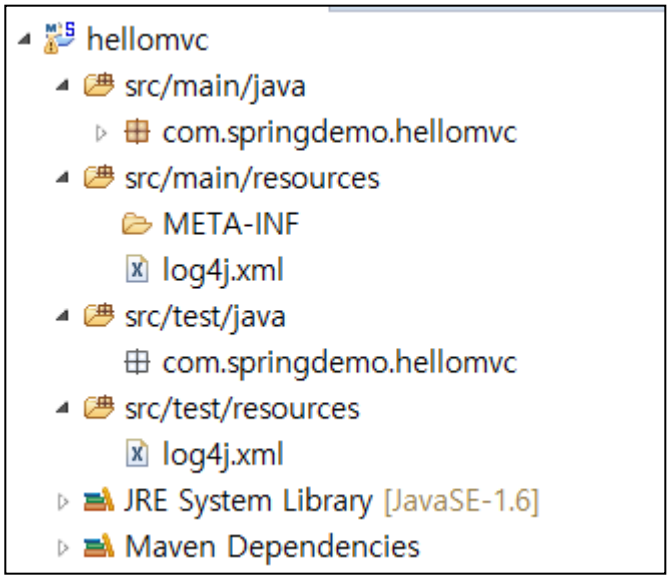


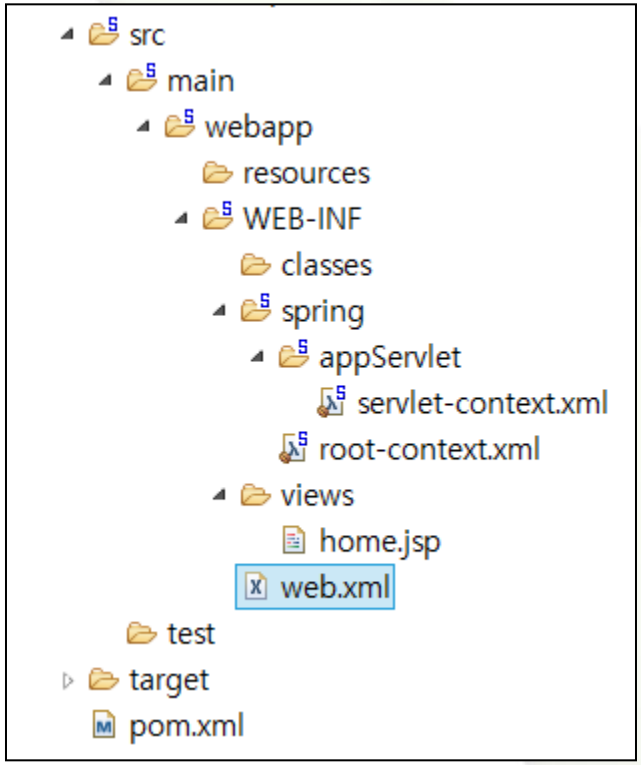
- DispatcherServlet은 Controller의 반환 값에 따라 ViewResolver를 이용해서 View를 결정하고 호출
- 호출된 View는 전달된 Model 객체를 이용해서 화면을 구성하고 반환
- DispatcherServlet은 View의 반환 결과를 요청 영역에 응답

프로젝트 생성, 구조 확인

- 프로젝트 만들기
  - File → New → Spring Project 메뉴 항목 선택
  - 프로젝트 이름 입력 / Templates에서 Spring MVC Project 선택 후 Next
  - 패키지 이름 입력 후 Finish

- 프로젝트 구조 확인





Spring 버전 설정

- pom.xml 파일의 〈properties〉 설정 항목에서 spring의 버전 설정

```
<modelVersion>4.0.0</modelVersion>
<groupId>com.ensoa</groupId>
<artifactId>order</artifactId>
<name>SpringMVCOrder</name>
<packaging>war</packaging>
<version>1.0.0-BUILD-SNAPSHOT</version>
<properties>
    <java-version>1.6</java-version>
    <org.springframework-version>4.1.0.RELEASE</org.springframework-version>
    <org.aspectj-version>1.6.10</org.aspectj-version>
    <org.slf4j-version>1.6.6</org.slf4j-version>
</properties>
<dependencies>
    <!-- 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>
```

루트 웹 애플리케이션 컨텍스트 설정

- 웹 애플리케이션이 시작될 때 최상위 전역 웹 애플리케이션 컨텍스트 생성

- 모든 서블릿에서 공통으로 사용되는 빈을 설정하는 용도로 사용

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

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

서블릿 웹 애플리케이션 컨텍스트 설정

- 서블릿이 처음 요청될 때 웹 애플리케이션 컨텍스트 생성

- 이후 해당 서블릿에 대한 요청이 발생할 때 동작

```
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

한글 지원을 위한 설정

- Spring 지원 Character Encoding Filter 적용

```
<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>
```

HandlerMapping 빈 등록

- 빈 설정 파일(servlet-context.xml)에 어노테이션 와이어링을 활성화하고 기본 HandlerMapping 빈을 등록하는 설정 적용

```
<annotation-driven />
```

  - 이 때 RequestMappingHandlerMapping 클래스가 HandlerMapping을 처리하는 빈으로 등록되며,
  - @Controller 어노테이션이 설정된 컨트롤러와 요청이 매핑되도록 동작

ViewResolver 빈 등록 및 설정

- 빈 설정 파일(servlet-context.xml)에 등록된 InternalResourceViewResolver의 설정을 통해서 뷰가 선택되고 호출됨

- 뷰 이름은 컨트롤러에서 반환된 이름 사용

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

ViewName이 hello-result인 경우

→ "/WEB-INF/view/" + hello-result + ".jsp" 를 View의 경로로 사용

정적 리소스 처리기 설정

- image, css, javascript 등의 정적 리소스는 일반적인 MVC 처리 프로세스를 따르지 않고 응답할 수 있도록 설정

- Spring 빈 설정 파일에 등록

```
<resources mapping="/resources/**" location="/resources/" />
```

  - /resources/ URL로 오는 모든 정적 요청은 /resources 디렉터리에서 처리

리뷰 : 요청 발생시 컨트롤러 호출 및 뷰 선택 과정

- 요청 처리 과정
  - DispatcherServlet 요청 수신
  - DispatcherServlet은 HandlerMapping을 통해 적절한 Controller를 선택하고 Controller에 요청 위임
  - DispatcherServlet은 Controller의 반환 값에 따라 ViewResolver를 이용해서 View를 결정하고 호출
  - Controller는 Model 영역의 객체 생성 및 호출 후 Model 객체 생성 / 데이터 할당 / 뷰이름 지정한 후 FrontController에 반환
  - 호출된 View는 전달된 Model 객체를 이용해서 화면을 구성하고 반환
  - DispatcherServlet은 View의 반환 결과를 요청 영역에 응답

- 이 때 컨트롤러, 뷰 등 관련 객체의 생성 및 관리는 요청과 관련된 서블릿 웹 애플리케이션 컨텍스트와 빈 설정 파일의 내용을 통해서 구현됨

스프링 컨트롤러

- DispatcherServlet에서 전달된 개별 요청을 처리하는 객체
- 스프링 IoC 컨테이너에서 관리
- 스프링 3.0 버전부터 Annotation 기반 컨트롤러 클래스 구현 권장

컨트롤러 구현

- 컨트롤러 클래스를 선언할 때 @Controller 어노테이션으로 빈 설정 등록

```
@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
```

- 빈 설정 파일(servlet-context.xml)에는 패키지를 등록해서 빈이 자동 인식되도록 설정

```
<context:component-scan base-package="com.ensoa.order" />
```

- @RequestMapping 어노테이션으로 요청과 컨트롤러 매핑

```
@Controller
public class HomeController {

    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale);
```

컨트롤러 구현

- 전송 방식 제어
  - Get방식의 요청과 Post 방식의 요청을 구분해서 요청 컨트롤러 매핑 설정 가능

GET 방식 요청 발생

```
@Controller
@RequestMapping("/article/newArticle.do")
public class NewArticleController {
    @Autowired
    private ArticleService articleService;

    @RequestMapping(method = RequestMethod.GET)
    public String form() {
        return "article/newArticleForm";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String submit(@ModelAttribute("command") NewArticleCommand command) {
        articleService.writeArticle(command);
        return "article/newArticleSubmitted";
    }

    public void setArticleService(ArticleService articleService) {
        this.articleService = articleService;
    }

}
```

POST 방식 요청 발생

요청 매핑 구성 요소

- web.xml 파일에 등록된 DispatcherServlet의 서블릿 매핑 url

```
<servlet-mapping>
  <servlet-name>customer</servlet-name>
  <url-pattern>/customer/*</url-pattern>
</servlet-mapping>
```

- @Controller 어노테이션이 지정된 컨트롤러 클래스의 @RequestMapping에 지정된 경로
  - 생략되면 @RequestMapping(value= “/”)

```
@Controller
@RequestMapping(value="/")
public class CustomerController {
```

- 컨트롤러 클래스에 포함된 메서드의 @RequestMapping에 지정된 경로

```
@RequestMapping(value="edit.do", method=RequestMethod.GET)
public String edit(Model model) {
```

- 최종 경로

http://…/customer/edit.do

요청 처리기 메서드 전달인자

- 요청 처리기 메서드가 클라이언트가 전달하는 요청 데이터, 헤더 등의 정보를 수신할 수 있도록 다양한 전달인자 형식을 통해 데이터 전달

- 종류

종류	설명
모델	• Model, ModelMap, Map • 컨트롤러에서 데이터를 저장하고 뷰로 전달되는 용도의 전달인자 (클라이언트가 전달하는 데이터 수신 기능은 없음)
@MoedelAttribute	• 사용자 정의 객체 모델 지정 (DTO를 이용한 데이터 수신) • 어노테이션 생략 가능
@PathVariable	• @RequestMapping에 지정된 URL 중 {}에 명시된 경로 변수 • @RequestMapping(“/customer/{name}”) handler(@RequestParam(“name”) String name) { …

요청 처리기 메서드 전달인자

- 종류 (계속)

종류	설명
@RequestParam	• 개별 Http 요청 패러미터 저장하는 전달인자 지정 • 어노테이션 생략 가능 • 모든 요청을 일괄 수신하기 위해 Map<String, String> 사용
@RequestBody	• Http 요청의 본문을 저장하는 전달인자 지정
HttpServletRequest, HttpServletResponse	• 일반 서블릿에 전달되는 요청, 응답 객체
HttpSession	• 일반 서블릿에 전달되는 세션 객체
Locale	• Locale Resolver가 결정한 Locale 정보
스트림	• InputStream, Reader, OutputStream, Writer • 요청 및 응답에 대응하는 저수준 스트림 객체
@RequestHeader	• Http 헤더 정보를 전달인자에 매핑
@Cookievalue	• Http 쿠키 값을 전달인자에 매핑

요청 데이터 매핑

```
<form method="post">
    <input type="hidden" name="parentId" value="0" />
    제목: <input type="text" name="title" /><br/>
    내용: <textarea name="content"></textarea><br/>
    <input type="submit" />
</form>
```

```
@Controller
@RequestMapping("/article/newArticle.do")
public class NewArticleController {
    @Autowired
    private ArticleService articleService;

    @RequestMapping(method = RequestMethod.GET)
    public String form() {
        return "article/newArticleForm";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String submit(String title, String content, int parentId) {
        NewArticleCommand command = new NewArticleCommand();
        command.setTitle(title);
        command.setContent(content);
        command.setParentId(parentId);
        articleService.writeArticle(command);
        return "article/newArticleSubmitted";
    }

    public void setArticleService(ArticleService articleService) {
        this.articleService = articleService;
    }

}
```

```
public class NewArticleCommand {

    private String title;
    private String content;
    private int parentId;

    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public int getParentId() {
        return parentId;
    }
    public void setParentId(int parentId) {
        this.parentId = parentId;
    }

}
```

요청 데이터 매핑

```
<form method="post">
    <input type="hidden" name="parentId" value="0" />
    제목: <input type="text" name="title" /><br/>
    내용: <textarea name="content"></textarea><br/>
    <input type="submit" />
</form>
```

NewArticleCommand 객체 생성 및 데이터 저장

```
@Controller
@RequestMapping("/article/newArticle.do")
public class NewArticleController {
    @Autowired
    private ArticleService articleService;

    @RequestMapping(method = RequestMethod.GET)
    public String form() {
        return "article/newArticleForm";
    }

    @RequestMapping(method = RequestMethod.POST)
    public String submit(NewArticleCommand command) {
        articleService.writeArticle(command);
        return "article/newArticleSubmitted";
    }

    public void setArticleService(ArticleService articleService) {
        this.articleService = articleService;
    }

}
```

```
public class NewArticleCommand {

    private String title;
    private String content;
    private int parentId;

    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getContent() {
        return content;
    }
    public void setContent(String content) {
        this.content = content;
    }
    public int getParentId() {
        return parentId;
    }
    public void setParentId(int parentId) {
        this.parentId = parentId;
    }

}
```

요청 처리기 메서드 반환 타입

- 요청 처리기 메서드는 논리적 뷰와 모델을 DispatcherServlet에게 반환
- 반환 형식 종류

종류	설명
자동 추가되는 객체 모델	• Model, ModelMap, Map 타입 전달인자 • @ModelAttribute 어노테이션이 명시적/암시적 지정된 전달인자
@ModelAttribute	• 메서드에 @ModelAttribute 어노테이션이 지정된 반환 타입 (이 때 뷰 이름은 메서드 이름에 일치)
ModelAndView	• 뷰와 반환 데이터를 저장할 수 있는 전용 타입
String	• 뷰 이름으로 사용될 문자열 (이 때 모델 데이터는 다른 방법으로 전달하도록 구현)
void	• 뷰 이름은 메서드 이름에 일치 • 모델 데이터는 다른 방법으로 전달하도록 구현
View	• 사용자 정의 구현 내용을 포함하는 뷰 객체 반환
@ResponseBody	• HTTP 응답 메시지 본문을 문자열로 반환

뷰 리졸버

- 컨트롤러가 반환한 논리적 뷰 정보를 이용해서 물리적 뷰를 결정하는 객체

- InternalResourceViewResolver가 기본으로 설정됨

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

- Spring은 다양한 뷰를 지원하는 뷰 리졸버 타입을 제공

뷰 리졸버 종류

종류	설명
BeanNameViewResolver	• Spring 설정 파일에서 뷰 이름과 일치하는 Spring 빈 이름을 가지는 View 객체 반환
XmlViewResolver	• 추가적인 Spring 빈 설정 파일에서 뷰 이름과 일치하는 Spring 빈 이름을 가지는 View 객체 반환 • 기본 설정 파일 : /WEB-INF/views.xml
ResourceBundleViewResolver	• 추가적인 .properties 설정 파일에서 뷰 이름과 일치하는 View 클래스와 URL을 반환 • 기본 설정 파일 : 클래스경로의 views.properties
UrlBasedViewResolver	• 설정된 prefix와 suffix의 패턴에 따라 뷰 이름을 URL에 매핑 • 사용되는 View 클래스 함께 지정
InternalResourceViewResolver	• 디폴트 뷰가 JstlView 클래스인 UrlBasedViewResolver
TilesViewResolver	• Apache Tiles 프레임워크와 연동하는 페이지 구성 지원

뷰 리졸버 종류

종류	설명
JasperReportsViewResolver	• JasperReports 보고서 엔진을 사용하는 다중 뷰 렌더링 • 종류 ▪ csv : JasperReportsCsvView ▪ html : JasperReportsHtmlView ▪ pdf : JasperReportsPdfView ▪ xls : JasperReportsXlsView
XsltViewResolver	• 뷰 이름을 xslt 스타일 시트로 매핑
ContentNegotiatingViewResolver	• 뷰 이름과 컨텐츠 유형에 따라 뷰 결정
FreeMarkerViewResolver	• FreeMarker 템플릿 엔진 연동
VelocityViewResolver	• Velocity 템플릿 엔진 연동

뷰 구현

```jsp
<%@ page language="java" contentType="text/html; charset=EUC-KR"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">
<title>게임 검색 결과</title>
</head>
<body>
인기 키워드: <c:forEach var="popularQuery" items="${popularQueryList}">${popularQuery} </c:forEach>
<form action="game.do">
<select name="type">
    <c:forEach var="searchType" items="${searchTypeList}">
    <option value="${searchType.code}" <c:if test="${command.type == searchType.code}">selected</c:if>>
    ${searchType.text}</option>
    </c:forEach>
</select>
<input type="text" name="query" value="${command.query}"/>
<input type="submit" value="검색" />
</form>
검색 결과: ${searchResult}
</body>
</html>
```