16. 스프링 웹 MVC

기존 MVC 패턴은 스프링에 적용하고자 설계된 것이 스프링 웹 MVC 모듈이다. 스프링 웹 MVC 모듈은 요청 기반의 웹 MVC 프레임워크로 설정 가능한 핸들러 매핑, 뷰 분석, 파일 업로드를 위한 테마 분석과 함께 핸들러에 요청을 할당하는 디스패처 서블릿 기반으로 설계되었고, 웹 애플리케이션 배치를 쉽게 해주는 많은 기능을 제공한다. 기본 핸들러는 @Controller와 @RequestMapping 어노테이션이며, @RequestMapping을 통하여 유연한 요청 처리가 가능하다.

스프링 웹 MVC 모듈은 모든 웹 클라이언트의 요청을 하나의 서블릿이 받아서 처리하는 프론트 컨트롤러를 사용하여 구현되었고, 프론트 컨트롤러라고 부르는 디스패처 서블릿이 핸들러를 호출하여 동작한다.

(1) 스프링 웹 MVC의 주요 구성 요소

스프링 웹 MVC의 주요 구성요서는 **디스패처 서블릿(DispatcherServlet,** Dispatcher:운행관리자**), 핸들러 매핑(HandlerMapping), 컨트롤러(Controller), 뷰리졸버(ViewResolver,** Resolve:해결하다**), 뷰(View)**이다.

· 디스패처 서블릿(DispatcherServlet)은 클라이언트의 요청을 전달받아 컨트롤러에게 요청을 전달한다. 컨트롤러가 반환한 결과값을 뷰에 전달하여 적절한 응답을 생성하는 역할을 하며, 이 정보를 "web.xml" 설정 파일에 정의한다.

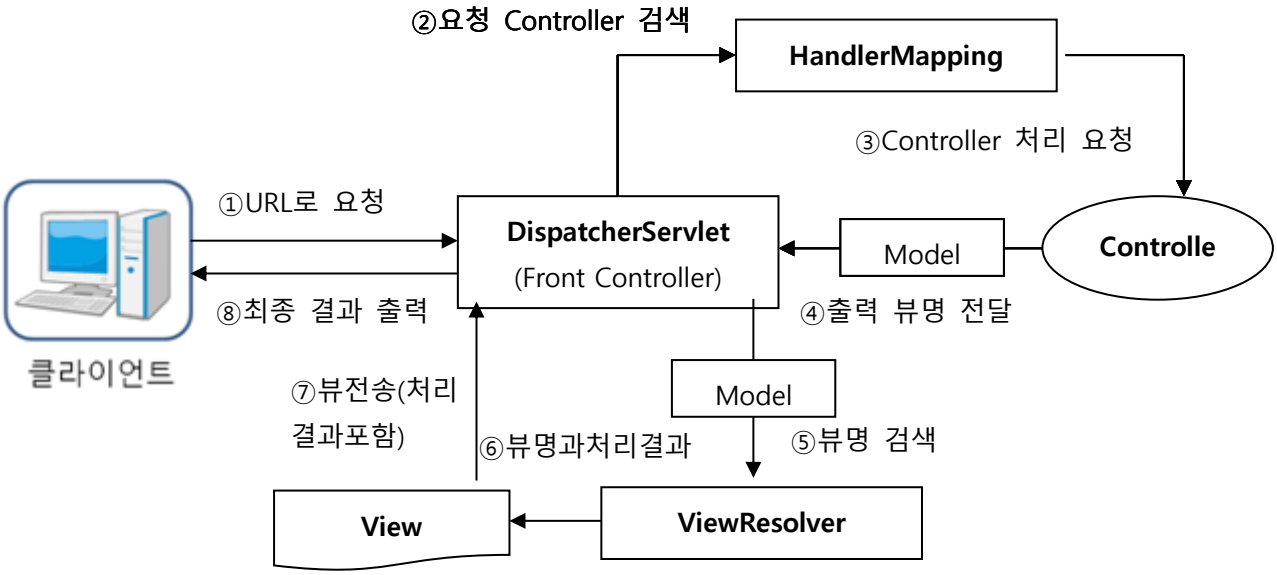
· 핸들러 매핑은 클라이언트의 요청을 처리할 컨트롤러를 찾는 처리를 한다.

· 컨트롤러는 클라이언트의 요청 처리를 수행하고, 로직을 담당하는 모델에 데이터를 전달하여 처리하게 한다

· 뷰리졸버는 응답할 뷰를 찾는 처리를 한다.

· 뷰는 컨트롤러는 모델로부터 처리 결과를 전달받아 뷰로 처리 결과 화면을 생성하는 역할을 한다.

(2) 스프링 웹 MVC의 요청 처리 절차



① 클라이언트에서 url 형태로 웹 서버에게 요청한다.

② 요청 정보는 프론트 컨트롤러인 디스패처 서블릿에 전달되고, 요청한 컨트롤러가 있는지 핸들러 매핑을 검색한다.

③ 핸들러 매핑은 디스패처 서블릿이 요청한 적절한 컨트롤러를 검색하여 컨트롤러에게 처리를 요청한다.

④ 컨트롤러는 요청에 대한 비즈니스 로직을 처리하고, 출력할 뷰명과 처리 결과를 디스패쳐 서블릿에 전달한다.

⑤ 디스패처 서블릿는 컨트롤러가 전송한 뷰명을 뷰리졸버를 통하여 뷰를 검색한다.
⑥ 뷰리졸버는 해당되는 뷰명에게 처리 결과를 보낸다.
⑦ 뷰는 처리 결과가 포함된 뷰를 디스패처 서블릿에게 전송한다.
⑧ 디스패처 서블릿은 요청한 최종 결과를 클라이언트에 출력한다.

클래스	기능
DispatcherServlet	유일한 서블릿 클래스로서 모든 클라이언트의 요청을 가방 먼저 처리하는 Front Controller
HandlerMapping	클라이언트의 요청을 처리할 Controller 매핑
Controller	실질적인 클라이언트 요청 처리
ViewResolver	Controller가 리턴한 View 이름으로 실행될 JSP 경로 완성
View	JSP 파일

(3) 스프링 웹 MVC의 구현 내용과 순서
스프링 웹 MVC에서 사용자가 구현하는 부분은 컨트롤러와 뷰이며, 웹 애플리케이션 설정파일, 스프링 설정파일, 핸들러 매핑, 뷰리졸버 등은 설정만 하면 된다. 구현 순서는 사용자의 취향이며 중요하지 않다. 웹 애플리케이션에 따라 다음과 같은 구현 내용은 달라질 수 있다.
· 뷰를 JSP 페이지로 작성한다.
· 컨트롤러를 작성한다.
· 웹 애플리케이션 설정 파일(web.xml)에 디스패처 서블릿을 설정한다.
· 스프링 설정 파일에 핸들러 매핑, 컨트롤러, 뷰리졸버를 설정한다.
· 웹 애플리케이션의 데이터를 다루는 자바빈을 생성한다.
· 데이터베이스 연동에 필요한 SQL 구문을 작성한다.
· 데이터베이스를 다루는 DAO 인터페이스와 구현 클래스를 작성한다.
· 컨트롤러와 데이터베이스를 다루는 서비스 로직을 작성한다.
· 데이터베이스 서버 연동에 필요한 프로퍼티 파일을 생성한다.

16.1 디스패처 서블릿
디스패처 서블릿(DispatcherServlet)이란 모델(Model)과 컨트롤러(Controller)와 뷰(View)를 조합하여 웹 브라우저로 출력해 주는 역할을 수행하는 클래스이며, 해당 애플리케이션으로 들어오는 모든 요청을 핸들링하는 역할을 담당한다. 디스패처 서블릿(DispatcherServlet)은 서블릿과 같이 웹 애플리케이션의 "WEB-INF" 폴더의 "web.xml" 파일에 사용자 요청을 url을 설정한다.

(1) 디스패처 서블릿(DispatcherServlet)과 스프링 설정
디스패처 서블릿(DispatcherServlet)의 설정은 웹 애플리케이션 설정 파일인 "WEB-INF" 폴더의 "web.xml" 파일에 다음과 같이 2가지를 설정한다.
· 클라이언트의 요청을 전달받을 디스패처 서블릿(DispatcherServlet)
· 공통으로 사용할 애플리케이션 컨텍스트(context)

웹 애플리케이션 설정 파일인 "web.xml"에서 <servlet>과 <servlet-mapping>으로 서블릿명과 url 패턴을 설정한다. 디스패처 서블릿(DispatcherServlet) 클래스는 <servlet-name> 요소에 "spring"과 같은 서블릿명

을 정의하고 <servlet-mapping> 요소에 사용자 요청 url 패턴을 정의한다. 컨테이너에 로딩외는 스프링 설정 파일명은 "spring-servlet.xml"과 같이 "서블릿명"에 "**-servlet.xml**"을 추가한다. 이 파일을 "WEB-INF" 폴더에 생성하면 자동으로 컨테이너에 로딩된다. 디스패처 서블릿의 기본 설정은 다음과 같다.

```
<!-- ================ 스프링 관련 설정 ================ -->
<servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

```
<!-- ================ 스프링 관련 설정 ================ -->
<servlet>
        <servlet-name>springMVC</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/framework/*.xml</param-value>
        </init-param>
</servlet>
<servlet-mapping>
        <servlet-name>springMVC</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

"web.xml"의 <url-pattern>이 *.do로 설정할 경우 클라이언트에서 "*.do"로 끝나는 url 요청 정보가 디스패처 서블릿에 전달된다. 디스패처 서블릿(DispatcherServlet)은 "WEB-INF" 폴더의 "spring-servlet.xml"의 스프링 설정 파일을 초기화한다. 이 설정 파일에 웹 애플리케이션에 필요한 추가적인 기능들을 선언 할 수도 있다. 디스패처 서블릿(DispatcherServlet)이 요청 처리에 필요한 스프링 MVC 구성요소인 핸들러 매핑, 컨트롤러, 뷰리졸버, 뷰 등을 빈으로 설정한다.

16.2 컨트롤러

컨트롤러는 스프링 MVC 설계 패턴의 하나이며, 사용자가 구현하는 부분이다. 컨트롤러는 사용자 요청을 해석하고 모델과 뷰를 반환하는 기능을 가진 메서드이다. 기본적이 요소로 컨트롤러의 메서드, 어노테이션, 그리고 사용자에게 요청한 정보를 제공할 모델과 뷰명이 포함되어야 한다. 컨트롤러는 디스패처 서블릿이 요청한 핸들러 매핑에 의해 호출되며 비즈니스 로직을 처리한다. 핸들러 매핑에 필요한 컨트롤러의 설정은 스프링 설정 파일에 빈으로 정의하며 표기 방법도 다양하다. 스프링 3.0 이전의 컨트롤러는 "org.springframework.web.servlet.mvc.Controller" 인터페이스를 구현하여 AbstractController, MultiAction Controller 등 다양하게 작성하였다. 컨트롤러를 처음 작성할 때 스프링의 다양한 종류의 컨트롤러, 파라

미터, 반환타입에 힘들어 한다. 그러나 스프링 3.0부터 좀 간결하게 작성할 수 있는 @어노테이션 기반의 컨트롤러 사용을 권장하고 있다.

종류	설명	용도
Controller, AbstractController	모든 처리 작업 직접 구현	단순 처리
AbstractCommandController	파라메타 값 검증기능 제공	파라미터 매핑
AbstractFormController	폼 전송 지원, 유효성 검사	자바빈 사용
SimpleFormController	폼 출력, 입력 데이터 처리	입력 폼 처리
MultiActionController	다수의 로직을 하나로 구현	다중 액션
UrlFilenameViewController	단순히 요청을 뷰로 전달	정적 뷰 매핑

(1) 컨트롤러의 기본 구조

컨트롤러는 일반적으로 다음과 같은 단계로 구현되는 메서드이다. 사용자의 요청을 해석한다. → 요청에 대한 처리를 서비스 로직 등에 위임한다 → 반환된 값으로 모델을 생성한다. → 뷰를 결정한다. → 뷰와 모델을 반환한다 하나의 컨트롤러는 클라이언트 요청 처리에 대해서 다수의 구현 메서드가 필요할 수도 있다. @Controller로 컨트롤러임을 선언하고 @RequestMapping으로 요청 url 패턴을 지정한다. 비즈니스 로직들은 서비스 객체 등에 위임하고 반환된값으로 모델을 생성하여 디스패처 서블릿에 반환한다.
컨트롤러의 기본적인 구조는 다음과 같다.

```
@Controller
public class 컨트롤러명 {
        @RequestMapping("요청url패턴")
        public 반환타입 메서드명(Model 파라메타명...) {
                비즈니스 로직;
                return "뷰명"
        }
}
```

사용 예

```
@Controller
public class HelloWorldController {
        @RequestMapping("hello.do")
        public String helloWorld(Model model) {
                model.addAttribute("message", "Hello World SpringFrameWork!!!");
                return "HelloWorld"
        }
}
```

(2) 컨트롤러 @어노테이션

어노테이션이란 메타 데이터를 기술하는 특별한 구문으로, 클래스, 메서드, 변수, 파라메타 선언과 패키지 등의 소스 코드에 삽입하여 사용할 수 있다. 이를 통하여 소스 코드의 가독성을 높일 수 있으며, **"@어노테이션"** 형태로 표기한다. 스프링 3.0 이전에 트랜잭션 관리, 빈 관리와 의존성 주입 등이 어노테이션이 사용되었다. 스프링 3.0에서는 @Controller의 컨트롤러의 구현을 권장한다.

종류	설명
@Controller	스프링 웹 MVC의 컨트롤러 선언. 클래스 타입에 적용. org.springframework.stereotype.Controller
@RequestMapping	컨트롤러와 매핑되는 url 패턴 정의. http 메서드 등 지원. org.springframework.web.bind.annotation.RequestMapping
@SessionAttribute	세션 설정과 세션 유지 org.springframework.web.bind.annotation.SessionAttribute

① @Controller

@Controller는 특정 클래스의 상단에 명시하여 컨트롤러를 선언한다. @Controller는 org.springframework.stereotype.Controller 패키지가 필요하며, @Controller를 선언한 컨트롤러는 컴포넌트 자동 검색 대상이며, 스프링 설정 파일에서 <context:component-scan base-package="패키지명" /> 태그에 컨트롤러의 **패키지명**을 선언해야 어노테이션이 선언된 컨트롤러를 자동으로 로딩할 수 있다.

② @RequestMapping

@RequestMapping은 클라이언트의 요청 url 패턴이나 http 요청 메서드에 대해서 컨트롤러의 클래스나 메서드에 선언한다. @RequestMapping은 문자열 url, http 메서드, params() 타입, header(), 메서드 레벨와 매핑한다.

입력타입	매팅 어노테이션	설명
url/POST	@RequestParam	GET/POST/DELETE/PUT으로 호출될 때 전달되는 파라미터 값 단일 http 요청 파라미터를 메서드 파라메터에 전달하는 어노테이션. login() 메서드에 @RequestParam의 "id"와 "pwd"를 전달한다. login(@RequestParam("id") String id, @RequestParam("pwd") String pwd){...}
Path value	@PathVariable	url에 포함된 특정 영역 문자열
Servlet	-	HttpServletRequest, HttpServletResponse를 직접 핸들링하는 기존 서블릿과 같은 코드 사용
Cookie	@CookieValue	쿠키값을 얻거나 설정
Session	@SessionAttribute	세션값을 얻거나 설정
Body	@RequestBody	Request의 Body를 String으로 얻어낼 때 (http 요청 본문 부분이 전달되는 어노테이션

- 문자열을 이용한 url 매핑

@RequestMapping의 괄호()속에 "url 패턴"을 기술한다. "url 패턴"은 ant 형식과 대체문자 또는 "url 패턴"의 "{}"값을 파라메타로 전달 받을 수도 있다.

"{}"의 변수를 "path variable"이라 부른다.

@RequestMapping("/hello") 또는 RequestMapping("/admin/**/user") @RequestMapping("/user/**{userid}**")

- value와 method를 이용한 HTTP 메서드 매핑

value에 "url 패턴", method에 "RequestMethod.메소드명"를 기술하고, 메서드명은 GET 또는 POST를 기술한다.

```
@RequestMapping(value="/add", method=RequestMethod.GET)
@RequestMapping(RequestMethod.POST)
```

- 타입 레벨과 메서드 레벨의 매핑

클래스와 인터페이스 타입 레벨에 붙은 @RequestMepping은 타입내의 모든 매핑용 메소드의 공통 조건을 지정할 때 사용한다. 메서드 레벨의 매핑은 클래스 레벨의 매핑을 상속한다. 컨트롤러가 "/user" 타입에 각각 "/add", "/edit" 메소드의 url에 매핑되는 경우는 다음과 같다.

```
@RequestMapping("/user")
public class UserController{
    @RequestMapping("/add")
    public String add() { ... }
}
```

③ @SessionAttributes("cmd")

@SessionAttributes("cmd")는 클래스 상단에 선언하여 세션으로 커맨드 객체를 저장하는 애너테이션이며 두 가지 기능이 있다.

- 첫째, 컨트롤러 메서드가 생성하는 모델정보 중에서 @SessionAttributes에 지정한 이름과 동일한 것이 있으면 이를 세션에 저장한다. 부가 이 모델을 참조하여 기존 정보를 폼(form)에 보여주는 기능이다.
- 둘째, @ModelAttribute가 지정된 파라미터가 있을 때 이 파라미터에 전달해줄 오브젝트를 세션으로 가져오는 것이다.

(3) 스프링 어노테이션

컨트롤러, 비즈니스 로직이나 DAO 설정 등에 사용하는 스프링 어노테이션이다.

종류	설명
@Service	컨트롤러와 데이터베이스를 처리하는 클래스에 비즈니스 로직이나 트랜잭션을 처리하는 클래스를 두게 되는데 이 역할을 담당하는 클래스를 서비스 클래스로 설정한다. @Service어노테이션을 선언한 클래스는 자동 검색을 통해 빈으로 자동 설정한다. org.springframework.stereotype.Service
@Reponsitory	DAO의 역할을 담당하여 데이터 베이스와 연동해서 데이터 검색이나 입력, 수정하는 클래스를 빈으로 설정하기 위해서 사용한다. org.springframework.stereotype.Reponsitory
@Component	<context:component-csan>태그로 클래스를 빈으로 자동 설정한다. org.springframework.stereotype.Component
@Autowired	의존 관계 자동 설정. 생성자, 필드, 메서드에 적용 가능. 즉 생성자, 메서드, 필드에 붙여 스프링에서 자동 주입을 명시한다. org.springframework.beans.Factory.annotation.Autowired
@Transactional	트랜잭션 처리시 자동으로 트랜잭션을 제어하는 기능을 제공한다. org.springframework.transaction.annotation.Transactional

@Scope	빈의 범위를 싱글톤이 아닌 request, session, prototype 등으로 설정한다. org.springframework.context.annotation.Scope

(4) 컨트롤러의 메서드 파라메타 종류

① @RequestParam

단일 HTTP 요청 파라메타를 메소드 파라메타에 전달하는 애너테이션이다.

```
public String login(@RequestParam("id") String id, @RequestParam("pw") String pw) { ... }
```

② @PathVariable

@RequestMapping의 url 중괄호 { } path 변수를 @PathVariable로 받는다.

```
@RequestMapping("/user/{userid}")
Public String view(@PathVariable("userid") String userid) { .... return "뷰명"; }
```

③ @ModelAttribute

클라이언트에서 컨트롤러에게 하나 이상의 값이 전달되는 경우가 있다. 하나 이상의 값을 가져오는 옵,젝트 형태로 만들 수 있는 구조적인 정보를 @ModelAttribute 모델이라 부르며, 컨트롤러가 전달 받은 오브젝트 형태의 정보가 @ModelAttribute이다.

컨트롤러에서 폼의 다수 데이터를 UserVO 자바빈에 저장하여 insert() 메서드에서 @ModelAttribute로 받는 경우다.

```
@RequestMapping("/add", mothed="RequestMethod.POST")
pubic User insert(@ModelAttribute UserVO userVO) {
        userService.insert(UserVO());
        ... }
```

(5) 컨트롤러의 반환 타입

컨트롤러의 메서드가 return으로 반환하는 값의 타입을 선언한다. 반환 타입은 ModelAndView, Model, Map, View, String, void 등이 있다.

객체 반환 타입	설명
ModelAndView	뷰와 모델 정보를 모두 포함하고 있는 반환 타입
Model,Map,ModelMap	뷰에 전달할 객체 정보만 포함하고 있는 반환 타입
String	뷰만 반환. JSP 또는 HTML View 등 호출시 사용
View	뷰 객체를 직접 반환. 해당 뷰 객체를 이용해 뷰 생성
void	반한 타입을 기술하고 않고, 기본으로 자동 생성

① ModelAndView

ModelAndView는 컨트롤러가 디스패처 서블릿에 반환해야 하는 뷰와 모델 정보를 모두 포함하고 있는 객체를 반환한다.

- ModelAndView에서 모델과 뷰를 설정하는 메소드는 3가지이다.

· setViewName(String 뷰명) : 응답 뷰명 설정

· addObject(String 뷰명, Object name) : 뷰명과 모델 설정

· addAllObject(Map map) : 뷰에 전달할 값을 Map에 설정

② String

String 반환 타입은 뷰명만 반환한다.

```
public String 메서드명 { ... return "뷰명" }
```

③ Model, Map, ModelMap

Model, Map, ModelMap 객체는 String 반환 타입으로 선언된 구현 메서드에서 뷰에 데이터만을 전달하는 객체들이다. Map에는 put() 메서드로 설정하고, Model과 ModelMap은 addAttribute() 메서드로 파라메터를 설정 할 수 있다.

```
@RequestMapping("/test")
public ModelMap test(TestVO testVO, ModelMap modelMap) {
        modelMap.addAttribute("name", "model값");
        return modelMap;
}
```

④ void

메서드에서 HTTP 응답을 처리할 때나 뷰명이 RequestToViewNameTranslator에 의해 내부적으로 자동 생성되는 타입이다.

```
@RequestMapping("요청url")
public void 메서드명(@RequestParam String name, ...){
        객체명.메서드명("name", name);
}
```

⑤ View

뷰 객체로 반환하는 반환 타입이다. 엑셀, 파일 다운로드 등에 사용된다.

16.3 핸들러 매핑

핸들러 매핑이란 사용자의 요청이 있을 때 디스패처 서블릿은 어떤 컨트롤러에게 위임할 것인가를 결정하게 되는데 그 요청을 처리하는 컨트롤러의 매핑을 담당하는 인터페이스이다. 핸들러 매핑은 클라이언트의 요청을 어떤 컨트롤러가 처리할 것인가에 대한 정보를 디스패처 서블릿에 반환단다.

(1) 핸들러 매핑의 종류와 프로퍼티

스프링 설정 파일에서 핸들러 매핑의 BeanNameUrlHandlerMapping과 DefaultAnnotationHandlerMapping은 기본으로 스프링 MVC에 탑재되어 있기 때문에 특별한 경우가 아니면 별도로 설정할 필요는 없다.

핸들러 매핑 클래스	설명
DefaultAnnotationHandlerMapping	@RequestMapping이 적용된 컨트롤러 메서드와 컨트롤러 url에 매핑. 가장 많이 사용.
SimpleUrlHandlerMapping	url과 컨트롤러의 매핑정보를 빈의 프로퍼티에 넣어준다. 기본 핸들러 매핑이 아니기 때문에 프로퍼티에 매핑 정보를 직접 명시적으로 기술해야 SimpleUrlHandlerMapping 빈을 사용할 수 있다.

BeanNameUrlHandlerMapping	http 요청을 웹 애플리케이션 컨텍스트 파일에 명시된 빈의 이름으로 매핑.
ControllerBeanNameHandlerMapping	빈의 id나 빈의 이름을 이용하여 매핑해주는 핸들러 매핑.
ControllerClassUrlHandlerMapping	클래스명을 url에 매핑해 주는 클래스이다.
AbsractUrlHandlerMapping	클라이언트의 요청 url로부터 서블릿 컨텍스트 경로를 기준으로 컨트롤러를 매핑

핸들러 매핑을 정의할 때 핸들러를 확장하기 위한 프로퍼티

프로퍼티명	설 명
intercepters	인터셉터 목록
defaultHander	해당 핸들러 매핑이 없을 때 사용하는 기본 핸들러
order	컨텍스트내 사용 가능한 모든 핸들러 매핑의 순서 정렬
alwaysUserFullPath	true 설정시 완전 경로 사용 유무. 기본값은 false
urlPathHelper	url 조사시 사용. 디폴트 값은 변경하지 말 것
urlDecode	디코드되지 않은 요청 url과 uri를 반환. 기본값은 false
lazyInitHanders	싱글톤 핸들러의 늦은 초기화 허락. 기본값은 false

① DefaultAnnotationHandlerMapping

@Controller를 사용한 컨트롤러는 컴포넌트 자동 검색 대상으로 스프링 설정 파일에서 <context:component-scan base-package="" /> 태그로 컨트롤러가 저장된 패키지명을 선언해야 컨트롤러를 자동으로 로딩할 수 있다.

DefaultAnnotationHandlerMapping 핸들러 매핑과 핸들러 어댑터를 함께 설정한다.

```
<bean class="org.springframework.~.DefaultAnnotationHandlerMapping">
        <property name="alwayUsePath" value="true" />
</bean>
<bean class="org.springframework.~.AnnotationMethodHandlerAdapater"
p:alwaysUseFullPath="true">
</bean>
```

```
@Controller
public class 컨트롤러명 { ... }
```

```
<!-- Controller -->
<context:component-scan base-package="패키지명" />
```

② SimpleUrlHandlerMapping

url과 컨트롤러의 매핑 정보를 빈의 프로퍼티에 넣어준다. 기본 핸들러 매핑이 아니기 때문에 프로퍼티에 매핑 정보를 직접 명시적으로 기술해야 한다.

③ BeanNameUrlHandlerMapping

HTTP 요청을 웹 애플리케이션 컨텍스트 파일에 명시된 빈의 이름으로 매핑한다. 여러 개의 핸들러 매핑

을 사용하는 경우 "order" 프로퍼티로 순서를 명시하는 것이 좋다.

④ ControllerBeanNameHandlerMapping
빈의 아이디나 빈의 이름을 이용하여 매핑해주는 핸들러 매핑이다.

⑤ ControllerClassUrlHandlerMapping
컨트롤러 클래스명을 url에 매핑해주는 클래스이다. 기본적으로 컨트롤러 클래스명을 모두 url로 사용하지만 "Controller" 문자열로 끝날 때는 "Controller"를 제외한 나머지 이름의 첫 글자를 소문자로 변경하여 url에 매핑해 준다.

⑥ AbsractUrlHandlerMapping
클라이언트의 요청 url로부터 서블릿 컨텍스트경로를 기준으로 컨트롤러 매핑하며 "alwaysUseFullPath" 프로퍼티의 값에 따라서 최종 경로가 결정된다.

16.4 뷰리졸버(ViewResolver)와 뷰

스프링 MVC 프레임워크는 뷰를 결정(해결)하는 방법이 제공된다. 스프링은 특정 뷰 기술을 위해 특별하게 선언할 필요없이 웹 브라우저에서 모델을 표현할 수 있도록 만들어주는 특별한 두개의 인터페이스 뷰리졸버(해결자)와 뷰를 제공한다 뷰리졸버는 뷰명과 실제 뷰간의 매핑을 제공하고, 뷰 인터페이스는 준비된 요청을 할당하고 요청을 뷰의 하나에게 처리하도록 넘겨버린다.

(1) 뷰리졸버
스프링 MVC 컨트롤러의 모든 핸들러 메서드는 명시적 또는 암시적으로 논리적 뷰명이 결정되어야만 한다. String, ModelAndView 반환 타입과 같은 명시적인 방법과 규칙에 기반 할 수도 있다. 뷰들은 논리적 뷰명에 의해 할당되고, 뷰리졸버에 의해서 결정된다.

뷰리졸버	설명
UrlBasedViewResolver	매핑없이 논리적 뷰명을 url로 처리하는 뷰리졸버
InternalResourceViewResolver	서블릿/JSP 뷰와 JstView 등과 같은 하위 클래스를 지원하는 UrlBasedViewResolver의 하위 클래스
XmlViewResolver	Xml로 작성한 설정 파일을 받는 뷰리졸버. 기본 설정 파일은 /WEB-INF/view.xml

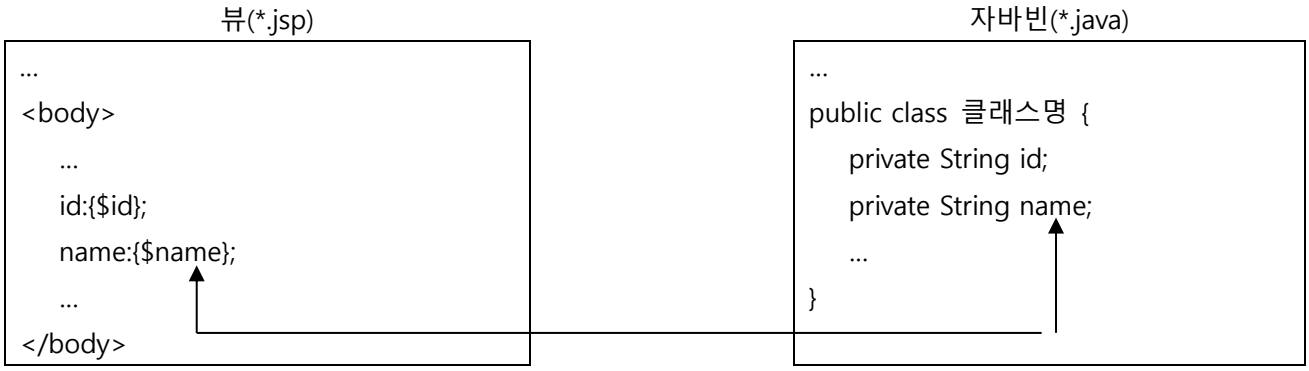
뷰리졸버에는 컨트롤러가 반환하는 뷰로 JSP를 사용할 때 UrlBasedViewResolver를 사용할 수 있다. 뷰리졸버는 뷰명을 url로 해석하고 뷰를 표현하기 위한 요청을 RequestDispacher로 보낸다. 빈의 접두어가 "/WEB-INF/jsp/"로 설정하고, 접미어가 ".jsp"로 설정하는 뷰리졸버이다.
그래서 만약 "test" 뷰명이 반환될 경우 뷰리졸버는 "/WEB-INF/jsp/test.jsp"로 요청 정보를 RequestDispacher로 보내게 된다.

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

(2) 뷰와 뷰 작성

뷰란 사용자가 요청 또는 요청 결과를 웹 브라우저에 출력하는 것으로, JSP와 JSTL 등을 이용하여 구현할 수 있다. 뷰 작성할 때 주의 할 점은 화면의 입력 데이터나 응답하기 위한 값들이 자바빈을 통하여 컨트롤러와 뷰에 전송되거나 전달된다. 그래서 뷰에 기술하는 필드명들은 반드시 자바빈의 프로퍼티명과 동일하게 코딩되어야 한다.

또한 스프링 MVC 프레임워크에서는 클라이언트에서 직접 접근할 수 없도록 뷰파일의 저장 위치를 "/WEB-INF" 폴더 내에 위치시키는 것을 강력하게 권장하고 있다.



(3) JSP와 JSTL로 작성한 뷰 파일은 "/WEB-INF" 하위 폴더에 저장하고 스프링 설정파일에 InternalResourceViewResolver로 폴더의 위치를 설정한다.

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

(4) 스프링 폼 태그

"spring-webmvc-*.jar" 파일의 스프링 폼 태그 라이브러리를 사용할 수 있다. 스프링 폼 태그를 사용할 경우에는 JSP 페이지에 taglib을 추가해야 한다.

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

스프링 폼 태그는 <form:>로 시작하며 내부 속성을 사용할 수 있다.

스프링 폼 태그	스프링 태그 내부 속성
<form: form>	id, method, action, modelAttribute(5.x), commandName(4.x)
<form: input>	id, size, path, value
<form: password>	id, path, value
<form: radiobutton>	path, value, label
<form: radiobuttons>	path, items, itemValue, itemLabel
<form: checkbox>	path, value, label
<form: checkboxes>	path, items, itemValue, itemLabel
<form: select>	id, path, items

<form: option>	value, label
<form: options>	items, itemValue, itemLabel
<form: textarea>	path, rows, cols
<form: hidden>	path
<form: label>	path, delimeter, cssErrorClass
<form: errors>	path, cssClass

16.5 스프링 웹 MVC 개발 환경 구축

스프링 프레임워크를 이용한 프로그래밍 개발 환경 방법은 매우 중요하다. 스프링 웹 MVC 프로그래밍을 위한 개발 환경 구축은 2가지 방법이 있다.

· 스프링에서 제공하는 이클립스 기반의 스프링 개발 환경에 최적화 되어 있는 개발 툴인 STS(Spring Tool Suite)를 "http://spring.io/tools/sts/all" 사이트에서 다운로드하여 설치한다(스프링 부트 개발).

· 이클립스 개발 도구에 스프링 프레임워크 개발을 위한 스프링 관련 플러그인 STS(Spring Tool Suite)를 추가로 설치한다.

18. 스프링 웹 MVC 예제

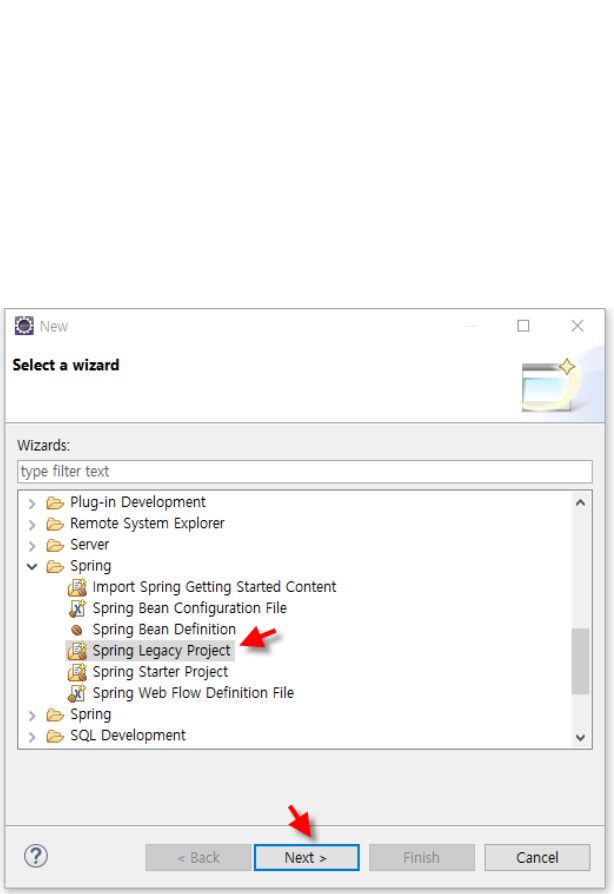
스프링 웹 MVC 프로그래밍을 위한 프로젝트와 패키지 생성, 컨트롤러 입력, 폴더 생성과 뷰 입력, 웹 애플리케이션 설정 파일과 스프링 설정 파일을 설정하고 실행하는 과정은 다음과 같다.

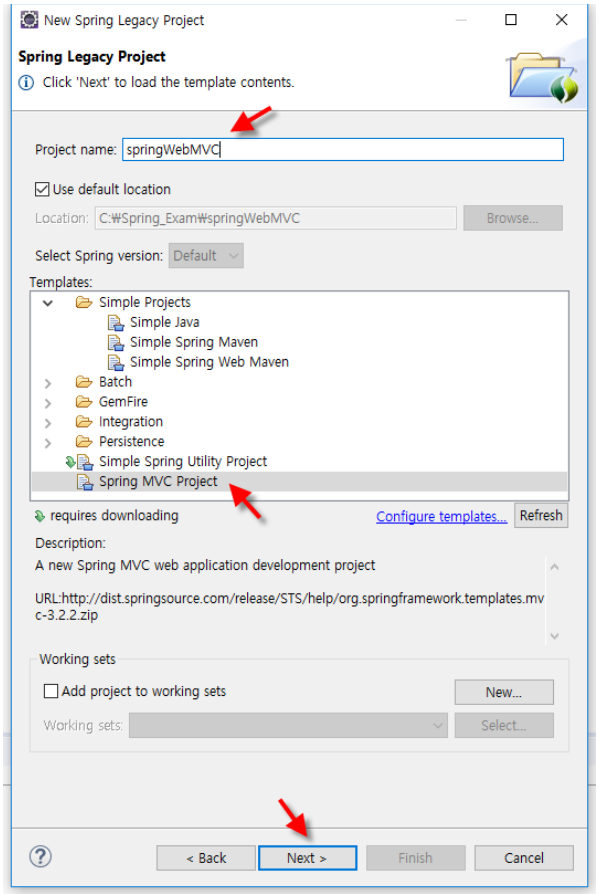
18.1 스프링 웹 MVC를 위한 프로젝트 생성

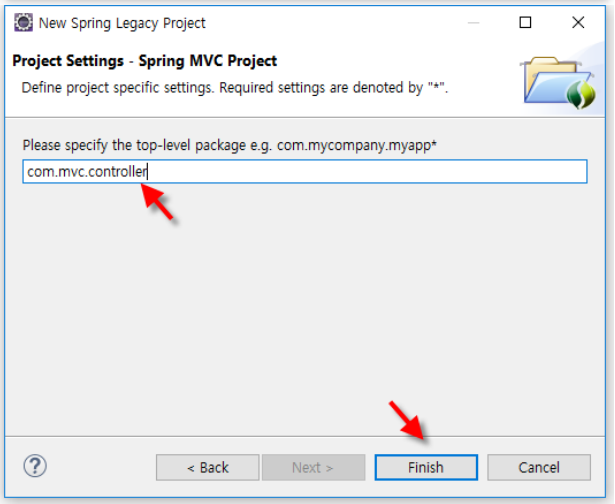
스프링 웹 MVC 프로그래밍은 Spring Legacy Project를 생성하면 된다.

① [File] - [New] - [Other] 메뉴를 선택한다.

② Spring 의 Spring Legacy Project 선택 **"springWebMVC"**로 입력하고, [Next] 버튼을 클릭한다.

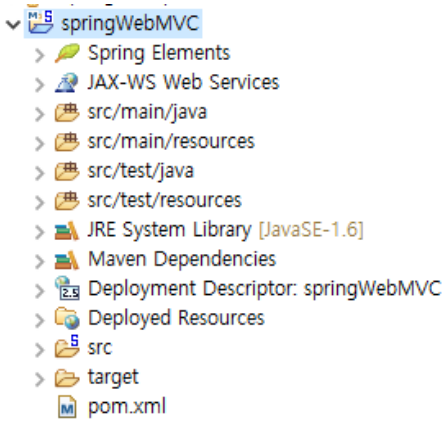






③ top-level package를 com.mvc.controller 입력하고 [Finish]버튼을 클릭한다.

④ 다음 그림과 같이 "springWebMVC" 프로젝트가 생성된다.



⑤ "springWebMVC" 프로젝트명에 "s" 표시가 나타나며 정상적으로 생성된 것이다.

⑥ pom.xml 파일을 열어 자바와 스프링 버전등을 변경한다.

```
<?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 https://maven.apache.org/maven-v4_0_0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.mvc</groupId>
	<artifactId>controller</artifactId>
	<name>springWebMVC</name>
	<packaging>war</packaging>
	<version>1.0.0-BUILD-SNAPSHOT</version>
	<properties>
		<java-version>1.8</java-version>
		<org.springframework-version>5.0.7.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>

		<!-- 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>javax.servlet-api</artifactId>
            <version>3.1.0</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>4.12</version>
                        <scope>test</scope>
                </dependency>
        </dependencies>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-eclipse-plugin</artifactId>
                <version>2.9</version>
                <configuration>
                    <additionalProjectnatures>

<projectnature>org.springframework.ide.eclipse.core.springnature</projectnature>
                    </additionalProjectnatures>
                    <additionalBuildcommands>

<buildcommand>org.springframework.ide.eclipse.core.springbuilder</buildcommand>
                    </additionalBuildcommands>
                    <downloadSources>true</downloadSources>
                    <downloadJavadocs>true</downloadJavadocs>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <compilerArgument>-Xlint:all</compilerArgument>
                    <showWarnings>true</showWarnings>
                    <showDeprecation>true</showDeprecation>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-plugin</artifactId>
                <version>1.2.1</version>
                <configuration>
```

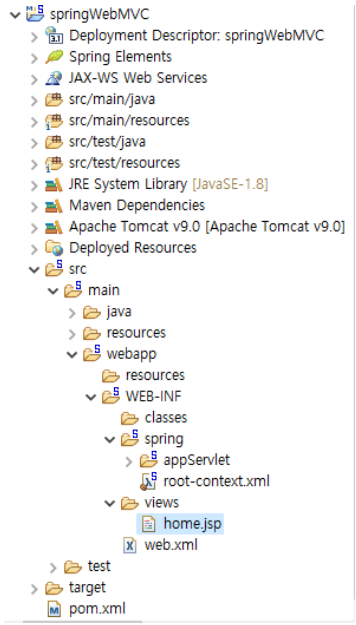
```
                        <mainClass>org.test.int1.Main</mainClass>
                    </configuration>
                </plugin>
            </plugins>
        </build>
</project>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" 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
https://java.sun.com/xml/ns/javaee/web-app_3_1.xsd">

        <!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/spring/root-context.xml</param-value>
        </context-param>
        <!-- Creates the Spring Container shared by all Servlets and Filters -->
        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
        </listener>
        <!-- Processes application requests -->
        <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 처리를 위한 UTF-8 필터 등록 -->
        <filter>
                <filter-name>encodingFilter</filter-name>
```

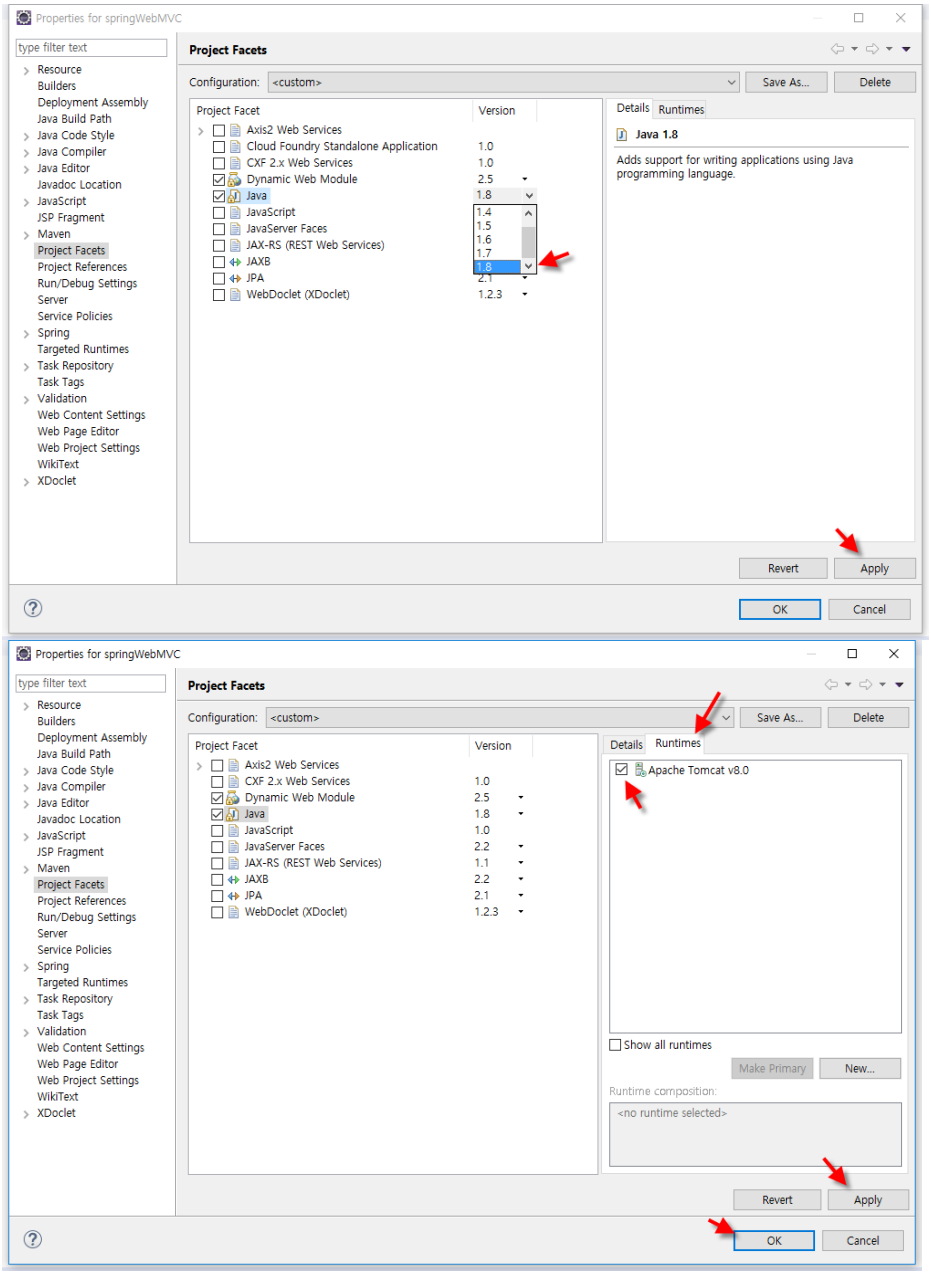
```
		<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
		<init-param>
			<param-name>encoding</param-name>
			<param-value>UTF-8</param-value>
		</init-param>
	</filter>
	<filter-mapping>
		<filter-name>encodingFilter</filter-name>
		<url-pattern>/*</url-pattern>
	</filter-mapping>
</web-app>
```



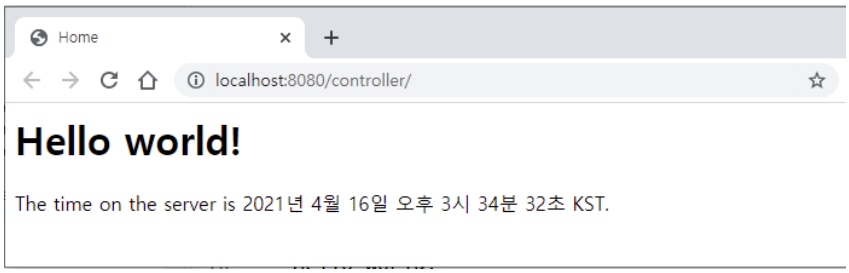
home.jsp

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
	<title>Home</title>
</head>
<body>
<h1>
	Hello world!
</h1>
<P>  The time on the server is ${serverTime}. </P>
</body>
</html>
```

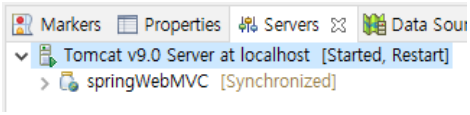
⑦ springWebMVC 프로젝트의 Propertices



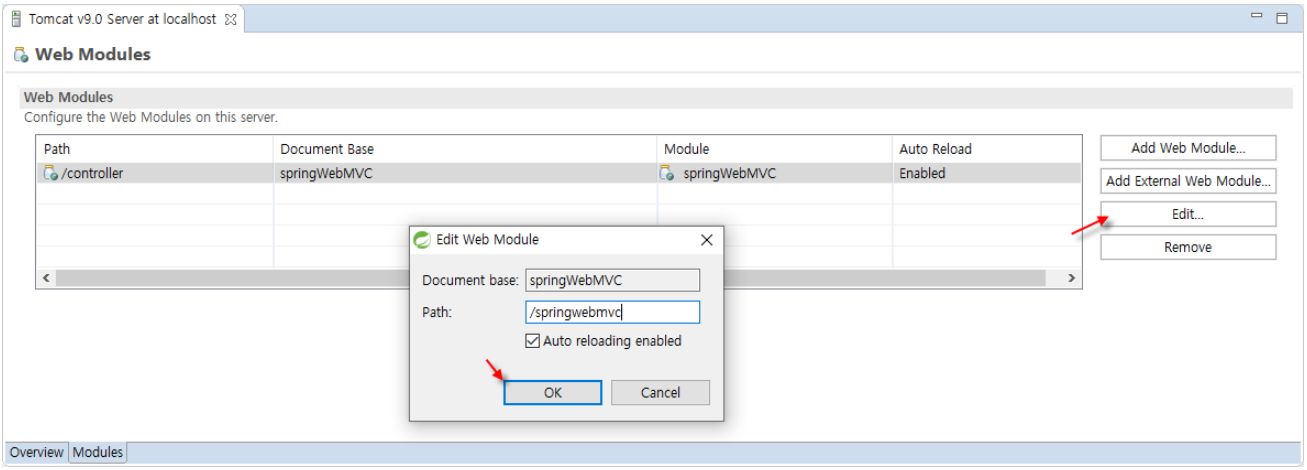
⑧ springWebMVC 오른쪽클릭 Run As -> Run on Server (다른 프로젝트는 제거) 다음과 같은 결과가 출력된다.



Tomcat v9.0 를 더블 클릭



다음과 같이 Path을 /springwebmvc 로 수정한다.

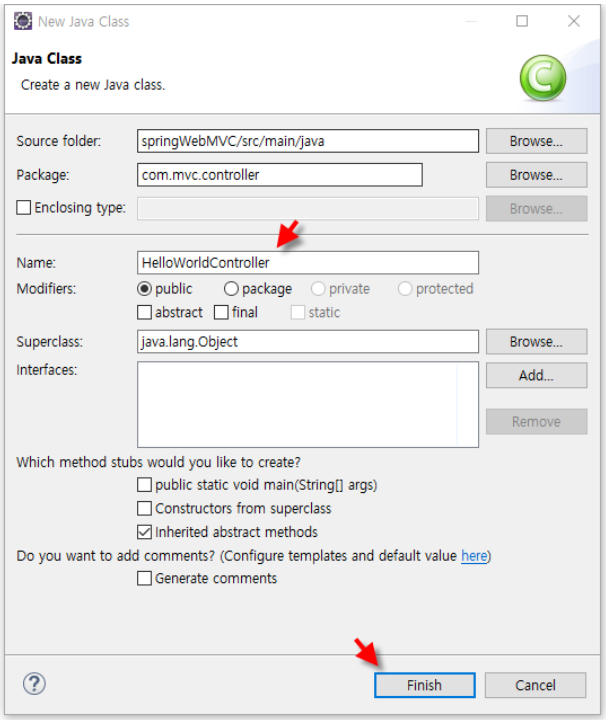


저장한다.

18.2 웹 예제 추가와 실행

(1) com.mvc.controller 패키지에 HelloWorldController.java 콘트롤 클래스 생성

① "com.mvc.controller" 패키지명을 선택하고, 마우스 오른쪽 버튼을 클릭하여 [New] - [Class] 메뉴를 선택하여 Name에 "HelloWorldController" 클래스를 생성한다.



② 클래스명 위에 "@Controller"를 입력한다. @Controller 어노테이션을 입력하면, "import org.springframework.stereotype.Controller;"가 추가된다.

※ 참고 : @Con을 입력한 후 "Ctrl + Space"키를 클릭하면 자동완성 키로 추가할 수 있다.

*HelloWorldController.java

```java
package com.mvc.controller;

import org.springframework.stereotype.Controller;

@Controller
public class HelloWorldController {

}
```

③ 클래스 내에 @RequestMapping 어노테이션 입력하면, "import org.springframework.web.bind.annotation.RequestMapping;"이 자동으로 추가된다. 그리고 컨트롤러의 본문을 입력하고, 저장한다.

*HelloWorldController.java

```java
package com.mvc.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloWorldController {
    @RequestMapping("helloWorld")
    public String helloWorld(Model model){
        model.addAttribute("message", "Hello World 스프링!!");
        return "helloWorld";
    }
}
```

(3) 웹 애플리케이션 환경 설정 파일에서 디스패처 서블릿 설정

웹 애플리케이션 설정 파일인 "web.xml" 파일은 WEB-INF 폴더에 생성되어 있다.

src/main/webapp/WEB-INF/web.xml 웹 애플리케이션 환경 설정 파일에 추가 및 수정

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" 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
https://java.sun.com/xml/ns/javaee/web-app_3_1.xsd">

        <welcome-file-list>
                <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>

        <!-- The definition of the Root Spring Container shared by all Servlets and Filters -->
        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/spring/root-context.xml</param-value>
        </context-param>

        <!-- Creates the Spring Container shared by all Servlets and Filters -->
        <listener>
                <listener-class>org.springframework.web.context.ContextLoaderListener</listener-
class>
```

```
	</listener>

	<!-- Processes application requests -->
	<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>*.do</url-pattern>
	</servlet-mapping>

	<!-- 한글 처리를 위한 UTF-8 필터 등록 -->
	<filter>
		<filter-name>encodingFilter</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>
	</filter>
	<filter-mapping>
		<filter-name>encodingFilter</filter-name>
		<url-pattern>/*</url-pattern>
	</filter-mapping>
</web-app>
```

(4) 스프링 설정 파일 내용 확인

/WEB-INF/spring/appServlet/servlet-context.xml

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

```
		xsi:schemaLocation="http://www.springframework.org/schema/mvc
https://www.springframework.org/schema/mvc/spring-mvc.xsd
			http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans.xsd
			http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context.xsd">

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

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

		<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in
the ${webappRoot}/resources directory -->
		<resources mapping="/resources/**" location="/resources/" />

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

		<context:component-scan base-package="com.mvc.controller" />

</beans:beans>
```

(5) 뷰 입력과 저장

① "WEB-INF" 폴더의 "views"폴더에 생성한다.

② 폴더에 [New] - [JSP file]을 선택하고, 파일명으로 "helloWorld" 입력한 후, [Finish] 버튼을 클릭한다.

③ 소스 코드와 같이 "${message}"를 입력하고, 저장한다.

WEB-INF/views/helloWorld.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>스프링 MVC</title>
</head>
<body>
```

```
        <h3>${message}</h3>
</body>
</html>
```

(6) 요청을 위한 JSP 페이지 작성과 실행

① "webapp" 폴더에서 [New] - [JSP file] 을 선택하고, 파일명으로 "index.jsp"를 입력 한 후 [Finish] 버튼을 클릭한다.

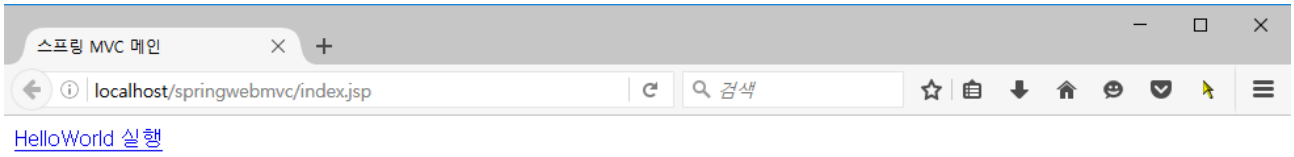
② 소스 코드를 입력하고 저장 버튼을 클릭한다.

src/webapp/index.jsp

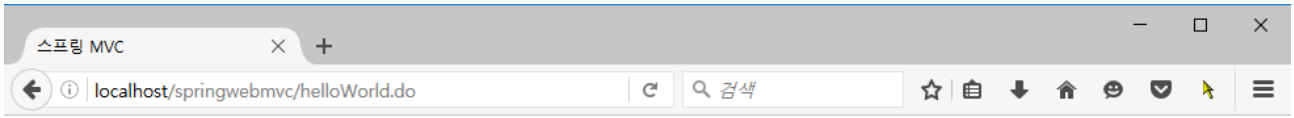
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>스프링 MVC 메인</title>
</head>
<body>
        <a href="helloWorld.do">HelloWorld 실행</a>
</body>
</html>
```

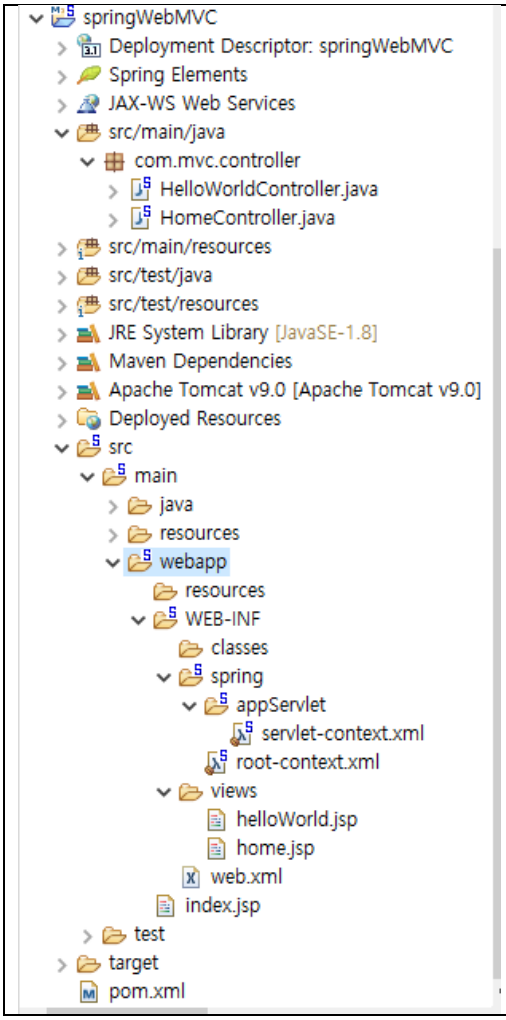
③ "index.jsp" 파일을 실행하여 "HelloWorld 실행"을 클릭하여 실행한다.

http://localhost:8080/springwebmvc/index.jsp



http://localhost:8080/springwebmvc/helloWorld.do





Eclipse내에서 서버 실행시 에러 발생

SEVERE: Error configuring application listener of class org.springframework.web.context.ContextLoaderListener

해결방법은 다음과 같다.

1. 프로젝트 목록에서 오른쪽 버튼 클릭하여 'Properties'를 선택한다.
2. 'Deployment Assembly'를 선택한다.
3. 'Add' 단추를 클릭한다.
4. 'Java Build Path Entries'가 목록 선택한다.
5. Next 단추를 클릭한다.
6. 'Maven Dependencies'가 목록 선택하고 'Finish'를 클릭한다.
7. Server를 실행하면 Console화면에 에러 없이 서버가 시작되는 것을 확인한다.