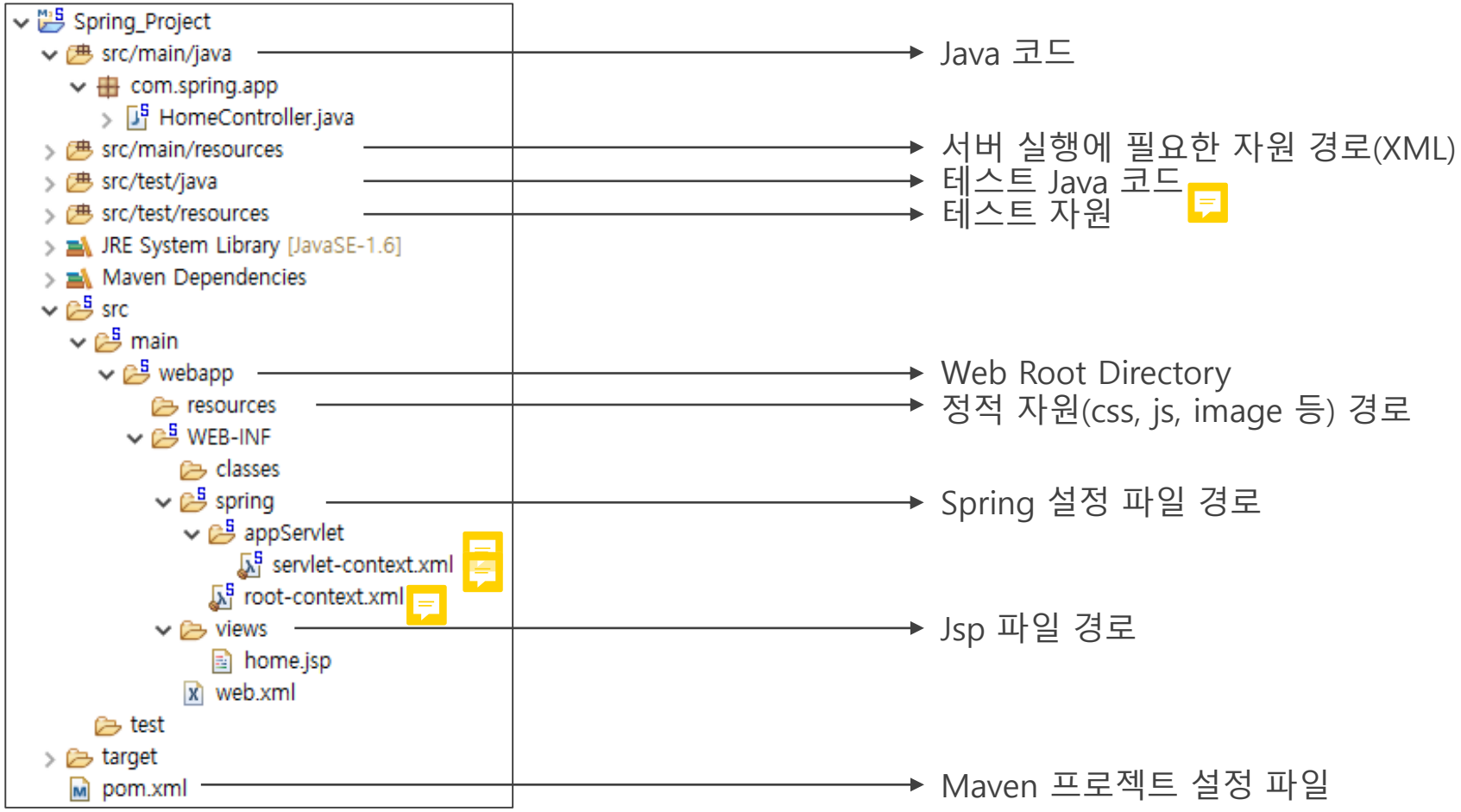
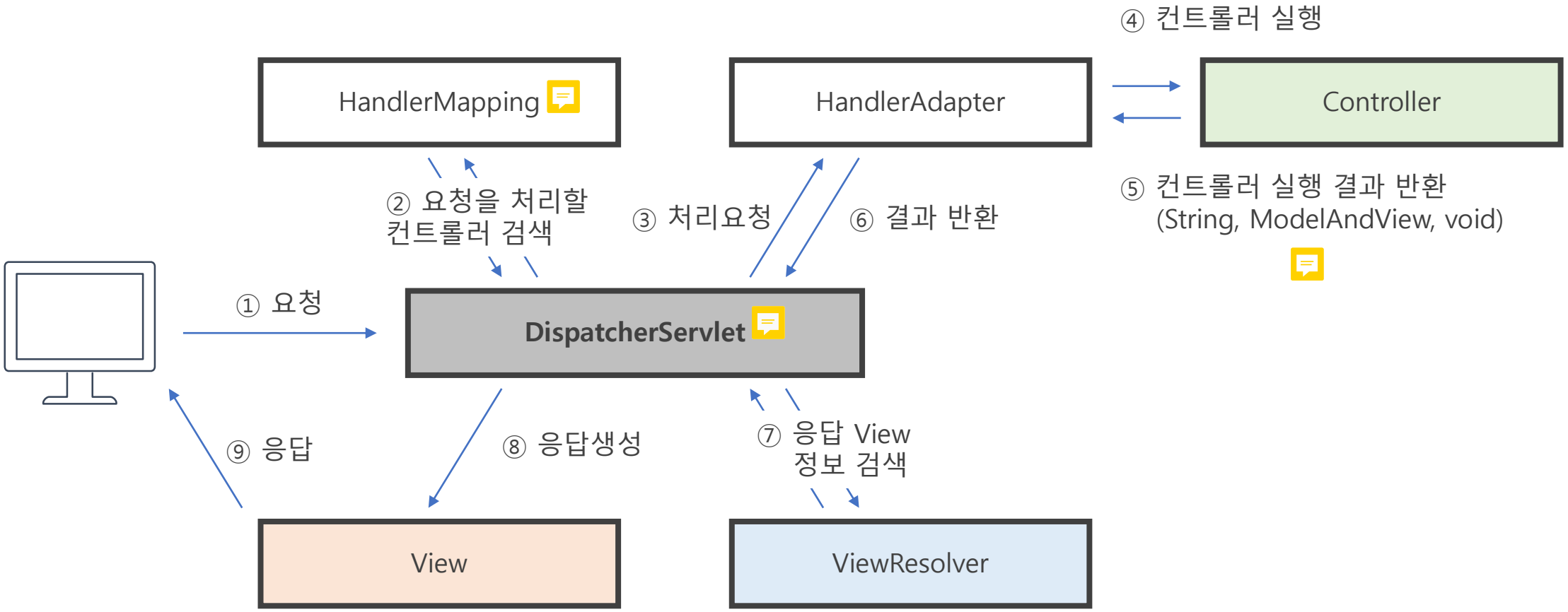
Spring MVC

Spring MVC

- Spring MVC Template



Spring MVC 동작 원리



DispatcherServlet

- DispatcherServlet
  - ✓ 스프링 프레임워크의 핵심 서블릿
  - ✓ web.xml에 관련 정보 작성
  - ✓ 기본적으로 servlet-context.xml 설정 파일을 읽어서 스프링 프레임워크를 구동함

```
<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>
```

DispatcherServlet

DispatcherServlet은 servlet-context.xml의 내용을 이용해서 동작함

DispatcherServlet이 동작하는 경로는 "/"이므로 동일한 ContextPath를 가진 모든 경로에서 동작함

spring

servlet-context.xml

- <annotation-driven />
  - ✓ @Controller Annotation을 활성화
  - ✓ Spring MVC에서 Controller에게 요청하기 위해 필요한 HandlerMapping과 HandlerAdapter를 자동으로 bean으로 등록

- HandlerMapping
  - ✓ @Controller가 적용된 객체를 컨트롤러하고 함
  - ✓ HandlerMapping은 요청을 처리할 컨트롤러를 @RequestMapping값을 이용해서 검색함

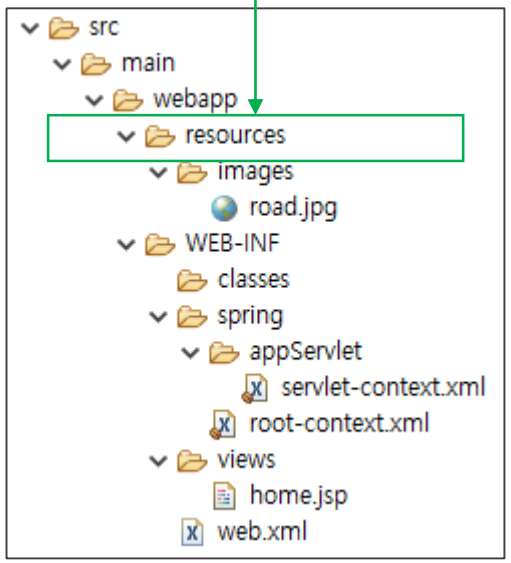
- HandlerAdapter
  - ✓ 요청을 처리할 컨트롤러의 메소드를 실행함
  - ✓ 메소드 실행 결과를 ModelAndView 객체로 변환한 뒤 DispatcherServlet에게 반환함

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

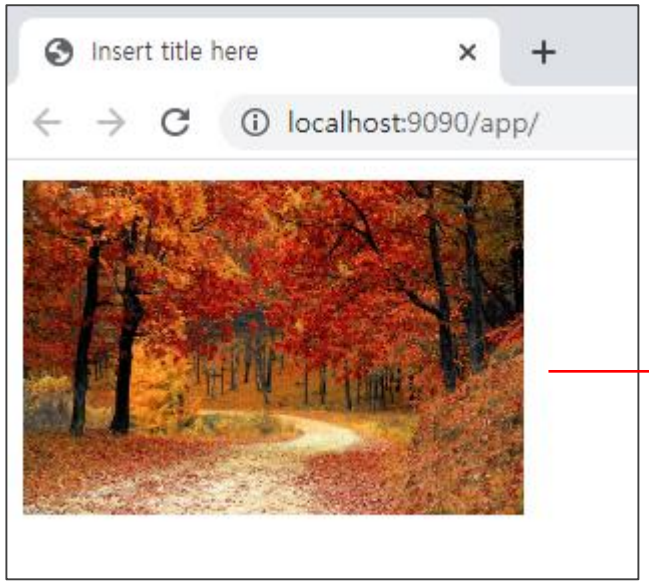
servlet-context.xml

- <resources mapping="/resources/**" location="/resources/">
  - ✓ 웹 구성 요소 중에서 정적 자원들의 경로와 호출 방법을 기술
  - ✓ 정적 자원 : 멀티미디어 데이터(이미지, 오디오, 비디오 등), CSS, JS 등

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




```
<img src="<%=request.getContextPath()%>/resources/images/road.jpg" width="200px">
```



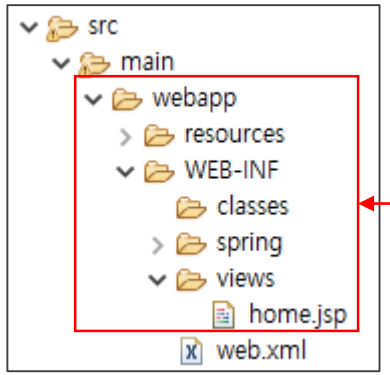
이미지 경로
http://localhost:9090/app/resources/images/road.jpg

app는 <%=request.getContextPath()%> 이므로 ContextPath를 의미함

servlet-context.xml

- <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  - ✓ 뷰 리졸버
  - ✓ HandlerAdapter에 의해서 반환된 ModelAndView 객체에 저장된 뷰 정보를 처리함

```
<!-- 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>
```



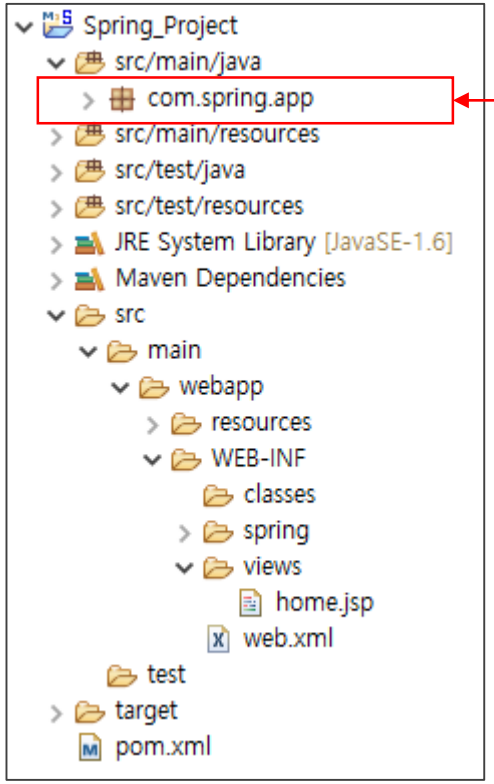

```
@RequestMapping(value="/", method=RequestMethod.GET)
public String home() {
    return "home";
}
```

실제 응답할 뷰는 뷰 리졸버에 의해서 다음과 같이 처리됨
"home" 앞에 "/WEB-INF/views/" 추가
"home" 뒤에 ".jsp" 추가

return "/WEB-INF/views/home.jsp"

servlet-context.xml

- <context:component-scan base-package="com.spring.app" />
  - ✓ base-package에 지정된 패키지에 저장된 클래스들을 스캔하고 자동으로 bean을 생성함
  - ✓ @Component, @Controller, @Service, @Repository 등의 Annotation이 추가된 클래스를 bean으로 등록함
  - ✓ Spring MVC Project의 top-level package와 동일해야 함




```
<context:component-scan base-package="com.spring.app" />
```

동일해야 함

Spring MVC 주요 Annotation

- 주요 Annotation

Annotation	의미	사용
@Controller	스프링 MVC의 컨트롤러 객체임을 명시	클래스
@RequestMapping	특정 RequestURI에 매핑되는 컨트롤러(클래스)나 메소드임을 명시	클래스, 메소드
@RequestParam	요청에서 특정 파라미터의 값을 가져올 때 사용	파라미터
@RequestHeader	요청에서 특정 HTTP헤더 정보를 가져올 때 사용	파라미터
@ModelAttribute	파라미터를 처리한 객체를 뷰까지 전달	메소드, 파라미터
@Component	Bean으로 만들어 둬야 할 객체임을 명시	클래스
@Service	서비스 객체에 추가하는 @Component	클래스
@Repository	DAO 객체에 추가하는 @Component	클래스
@PathVariable	RequestURI에 포함된 값을 가져올 때 사용	파라미터
@RequestBody	요청 본문에 포함된 데이터가 파라미터로 전달	파라미터
@ResponseBody	반환 값이 HTTP 응답 메시지로 전송	메소드, 리턴타입
@CookieValue	쿠키가 존재하는 경우 쿠키 이름을 이용해서 쿠키 값을 가져올 때 사용	파라미터
@SessionAttribute	Model의 정보를 세션에서 유지할 때 사용	클래스

Controller

- Controller
  - ✓ @Controller가 적용된 클래스
  - ✓ @RequestMapping을 이용해 요청 URL 및 요청 메소드 파악
  - ✓ 메소드 단위로 요청을 처리
  - ✓ 메소드는 결과를 출력할 뷰 이름을 반환

클래스를 컨트롤러로 인식

```
@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);

        Date date = new Date();
        DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);

        String formattedDate = dateFormat.format(date);

        model.addAttribute("serverTime", formattedDate );

        return "home";
    }

}
```

RequestURI가 ContextPath("/")인 GET 방식의 요청을 처리

model을 이용해 뷰로 전달(forward)할 속성(Attribute)을 저장

응답 결과가 나타날 뷰이름은 "home.jsp"

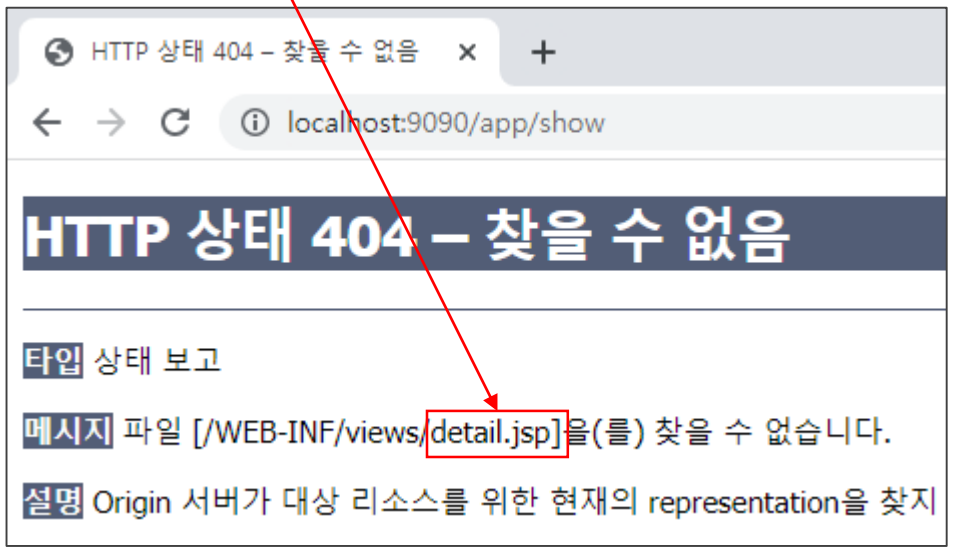
spring

Controller Method 반환 타입

- Controller Method
  - ✓ 컨트롤러는 하나의 요청을 하나의 메소드로 처리함
  - ✓ Spring MVC Pattern에서는 메소드의 반환 타입을 String 또는 void로 설정할 수 있음

- 반환 타입이 String인 경우
  반환값을 뷰(Jsp)의 이름으로 인식

```
@RequestMapping(value="show")
public String method1() {
    return "detail";
}
```



- 반환 타입이 void인 경우
  매핑값을 뷰(Jsp)의 이름으로 인식

```
@RequestMapping(value="show")
public void method2() {

}
```

spring

@RequestMapping

- @RequestMapping
  - ✓ 요청 URL과 요청 메소드를 인식할 수 있는 Annotation
  - ✓ 요청 메소드에 따라서 @GetMapping, @PostMapping 등으로 변경할 수 있음
- @RequestMapping 주요 기능

구분	예시	의미
value	@RequestMapping(value="/")	"/" 요청
	@RequestMapping(value={"/", "index"})	"/"와 "index" 요청
	@RequestMapping(value="/member/*.do")	"/member" 로 시작하고 ".do"로 끝나는 요청
method	@RequestMapping(method=RequestMethod.GET)	GET 방식(조회)
	@RequestMapping(method=RequestMethod.POST)	POST 방식(삽입)
	@RequestMapping(method=RequestMethod.PUT)	PUT 방식(수정)
	@RequestMapping(method=RequestMethod.DELETE)	DELETE 방식(삭제)
content type	@RequestMapping(consumes="application/json")	요청 컨텐트가 JSON임
	@RequestMapping(produces="application/json")	응답 컨텐트가 JSON임

기본 URL Pattern

- web.xml의 기본 URL Pattern
  - ✓ 기본 Pattern은 컨텍스트 패스(Context Path)로 되어 있음
  - ✓ 다른 Pattern으로 수정하면 전체 URL이 다르게 설정됨

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

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

- @RequestMapping(value="members") 처리 방식

컨텍스트 패스	<url-pattern>	전체 URL
app	/	http://localhost:8080/app/members
	/home/*	http://localhost:8080/app/home/members

Encoding Filter

- Encoding Filter
  - ✓ web.xml에 CharacterEncodingFilter를 추가
  - ✓ request.setCharacterEncoding("UTF-8")을 대체할 수 있는 필터

```
<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>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

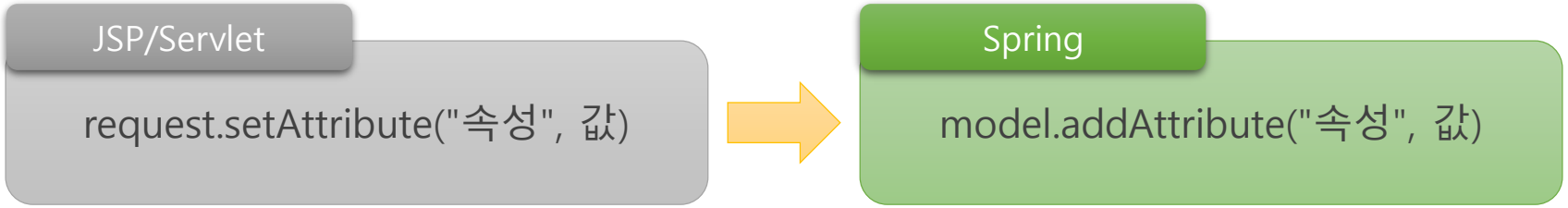
문자셋 인코딩

UTF-8

동일한 ContextPath를 가지는 모든 경로에 적용

Model

- Model
  - ✓ 뷰가 응답 화면을 구성할 때 필요로하는 데이터를 전달하는 인터페이스
  - ✓ JSP/Servlet에서는 request를 이용해서 데이터를 전달하였으나, 스프링에서는 model을 이용하여 데이터를 전달




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

    Date date = new Date();
    DateFormat dateFormat = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.LONG, locale);

    String formattedDate = dateFormat.format(date);

    model.addAttribute("serverTime", formattedDate );

    return "home";
}
```

컨트롤러의 메소드 매개변수로 Model model 선언

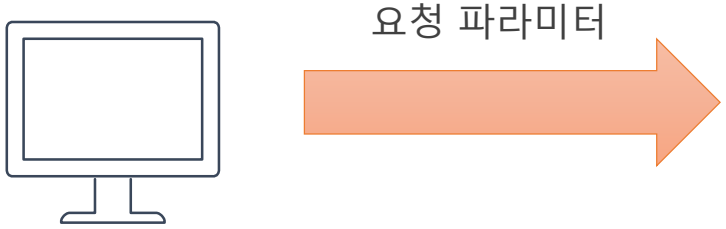
model을 이용해 뷰로 전달(forward)할 속성(Attribute)을 저장

home.jsp에서 ${serverTime}으로 전달된 값을 확인

spring

요청 파라미터

- 스프링의 요청 파라미터 처리 방식
  - ① HttpServletRequest의 getParameter() 메소드
  - ② @RequestParam 애너테이션
  - ③ 커맨드 객체



Controller

JSP/Servlet

한 가지 방법만 지원

HttpServletRequest

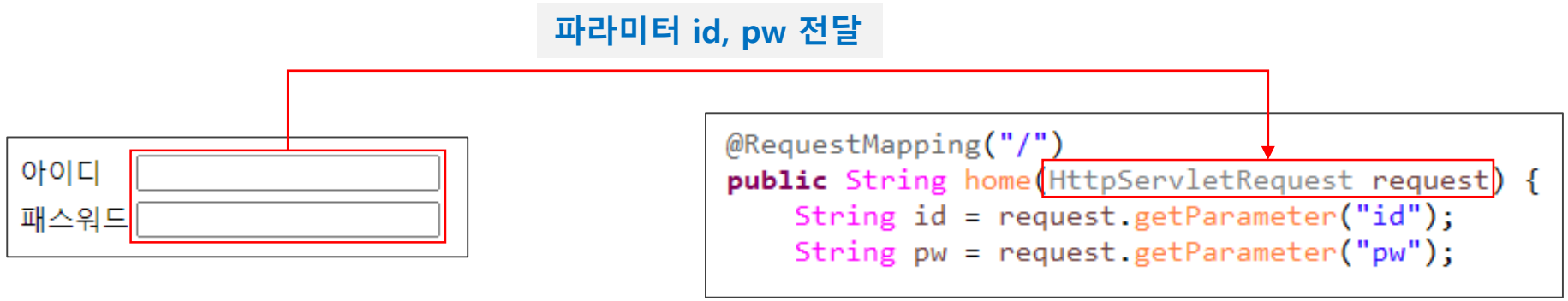
Spring

다양한 방법을 지원

- HttpServletRequest
- @RequestParam
- MemberVO member

HttpServletRequest

- HttpServletRequest
  - ✓ 요청을 처리하는 HttpServletRequest 인터페이스
  - ✓ 파라미터로 전달된 값을 getParameter() 메소드를 이용해서 처리



@RequestParam

- @RequestParam
  - ✓ 파라미터를 인식하고 변수에 저장하는 애너테이션
  - ✓ @RequestParam 애너테이션을 생략하고 변수명만 작성할 수 있음
  - ✓ 속성
    - ➢ value : 파라미터 이름 작성
    - ➢ required : 필수 여부 지정(디폴트 true)
    - ➢ defaultValue : 파라미터가 없는 경우 사용할 문자열 기본값

```
<a href="${contextPath}/read?pid=admin&week=sat">
```

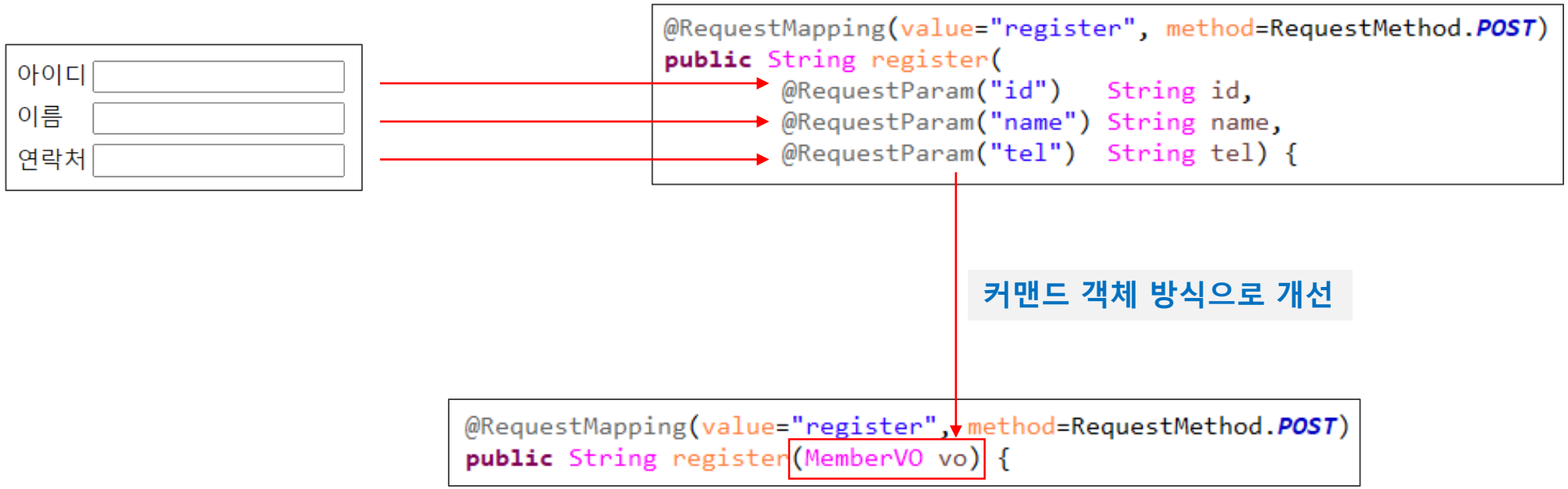
파라미터 pid, week 전달

```
@RequestMapping(value="read", method=RequestMethod.GET)
public String read(
        @RequestParam(value="pid") String pid,
        @RequestParam(value="week", required=false, defaultValue="mon") String week) {
```

파라미터 pid는 필수이므로 전달되지 않는다면 Exception 발생
파라미터 week는 필수가 아니며 만약 전달되지 않는다면 week=mon으로 처리

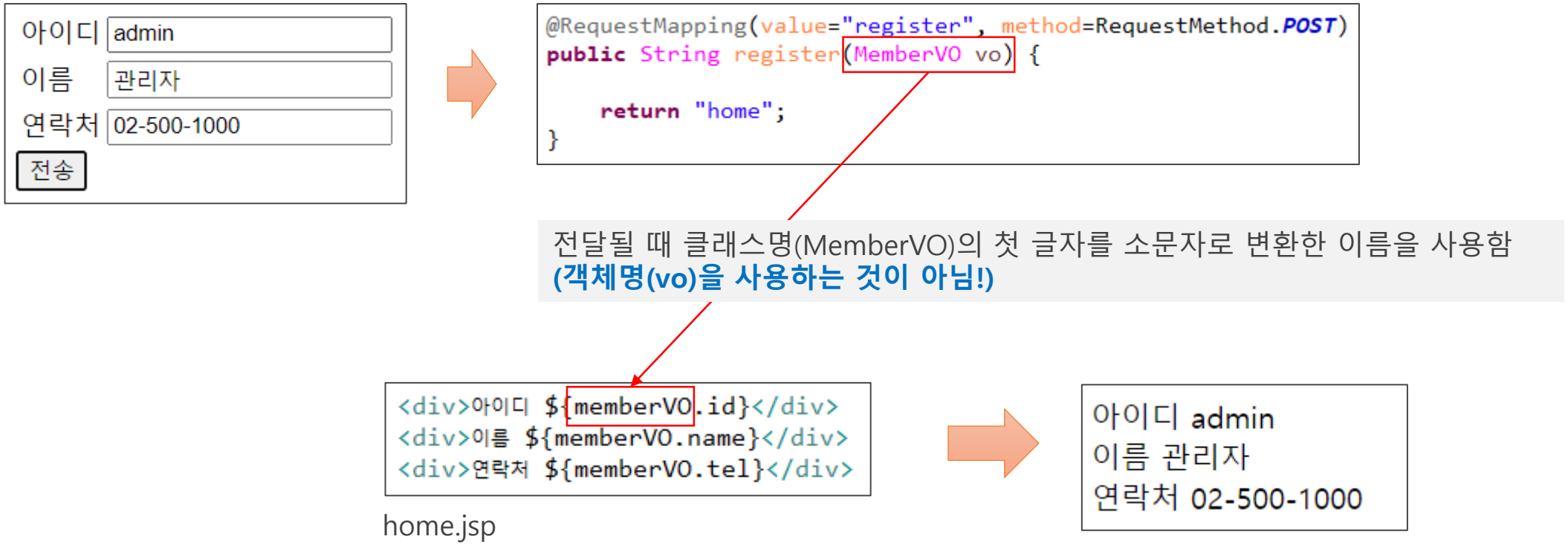
커맨드 객체

- 커맨드 객체
  - ✓ 여러 개의 요청 파라미터를 객체로 전달 받는 방식
  - ✓ 여러 개의 요청 파라미터를 일일이 처리하는 것보다 편리하게 개선된 방식
  - ✓ 커맨드 객체의 setter를 이용해서 파라미터를 전달 받음



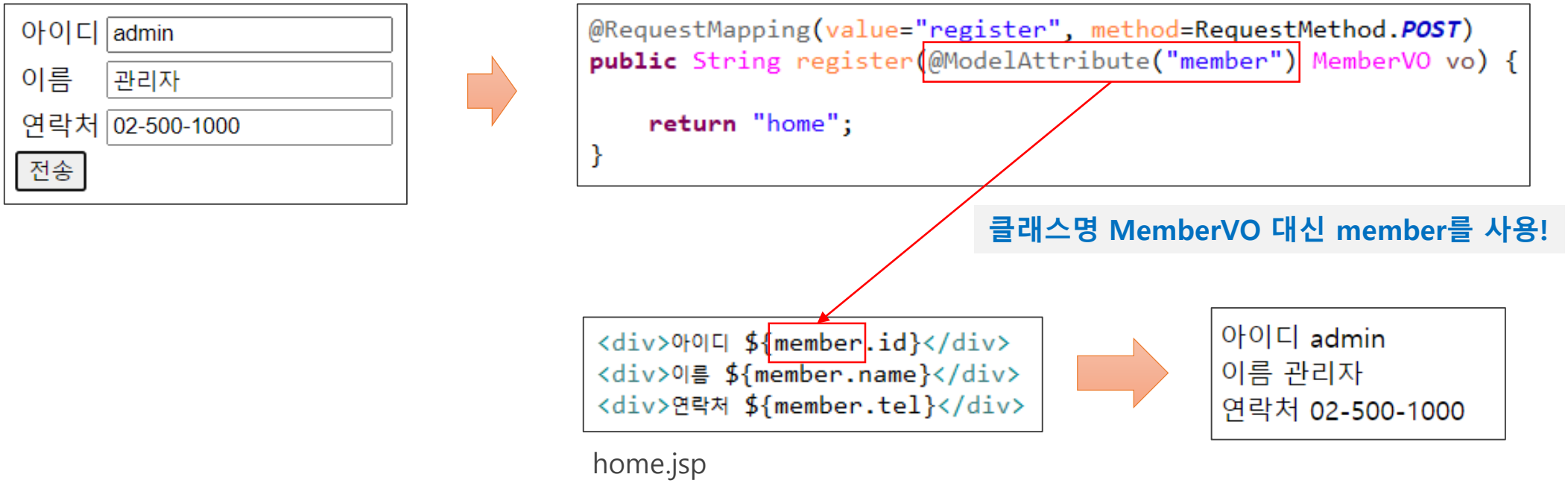
커맨드 객체

- 커맨드 객체 특징
  - ✓ 커맨드 객체를 자동으로 뷰까지 전달함
  - ✓ 별도로 model.addAttribute()를 처리할 필요가 없음



@ModelAttribute

- @ModelAttribute
  - ✓ 커맨드 객체를 model로 전달할 때 이름을 변경하고자 하는 경우에 사용
  - ✓ @ModelAttribute에서 지정한 이름으로 뷰까지 전달함



Redirect

- Redirect
  - ✓ 컨트롤러의 반환값이 "redirect:"으로 시작하면 리다이렉트로 이동함
  - ✓ response.sendRedirect()를 대체하는 스프링의 방식
  - ✓ "redirect:" 뒤에는 새로운 요청 URL이 오기 때문에 특정 URLMapping값을 작성해야 함 (뷰이름을 작성하는 것이 아님!)

```
@RequestMapping(value="register", method=RequestMethod.POST)
public String register(@ModelAttribute("member") MemberVO vo) {

    return "redirect:/view";
}

@RequestMapping(value="view", method=RequestMethod.GET)
public String view() {

    return "detail";
}
```

URLMapping "view"로 리다이렉트

리다이렉트는 기존 request를 유지하지 않음

detail.jsp

```
<div>아이디 ${member.id}</div>
<div>이름 ${member.name}</div>
<div>연락처 ${member.tel}</div>
```

아이디
이름
연락처