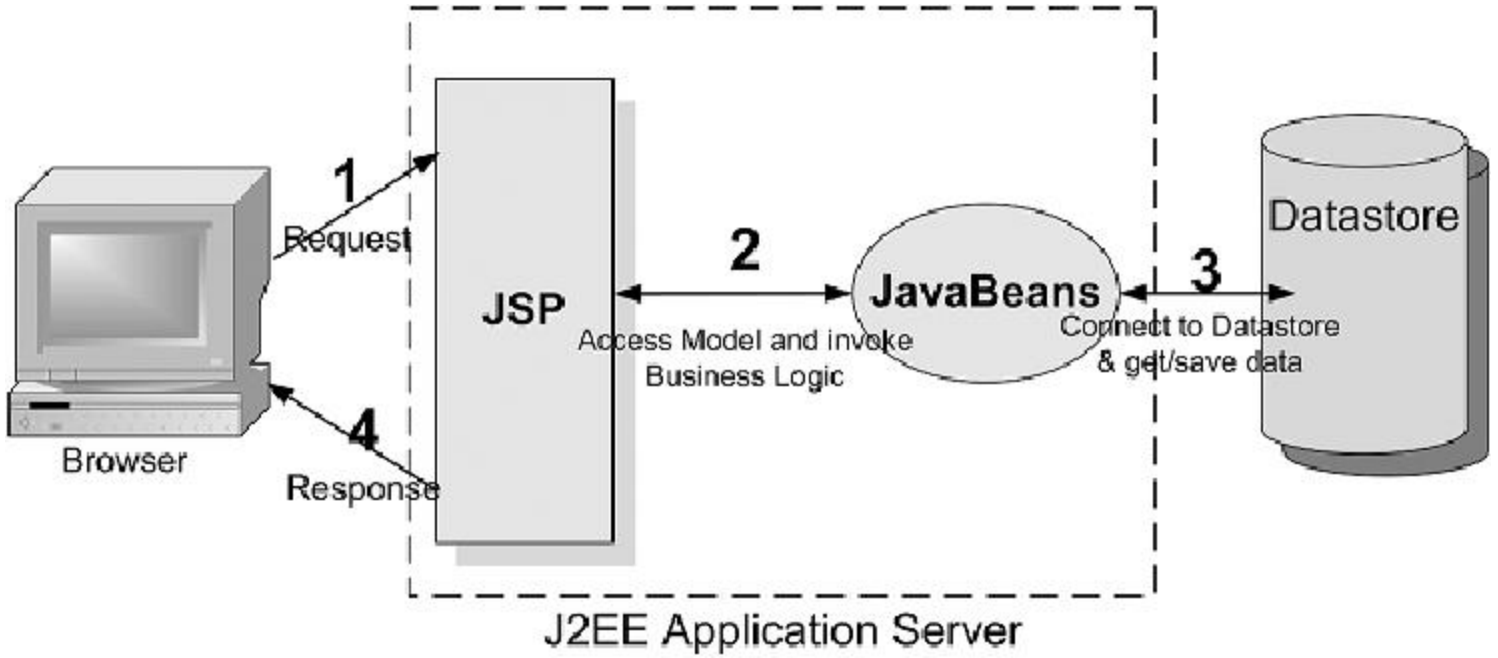
- Spring MVC -

Model 1 Architecture

- 모델1 개요
  - ✓ JSP 만 이용하여 개발하는 경우
  - ✓ JSP + Java Bean을 이용하여 개발하는 경우
  - ✓ Model2의 Controller 개념이 모호

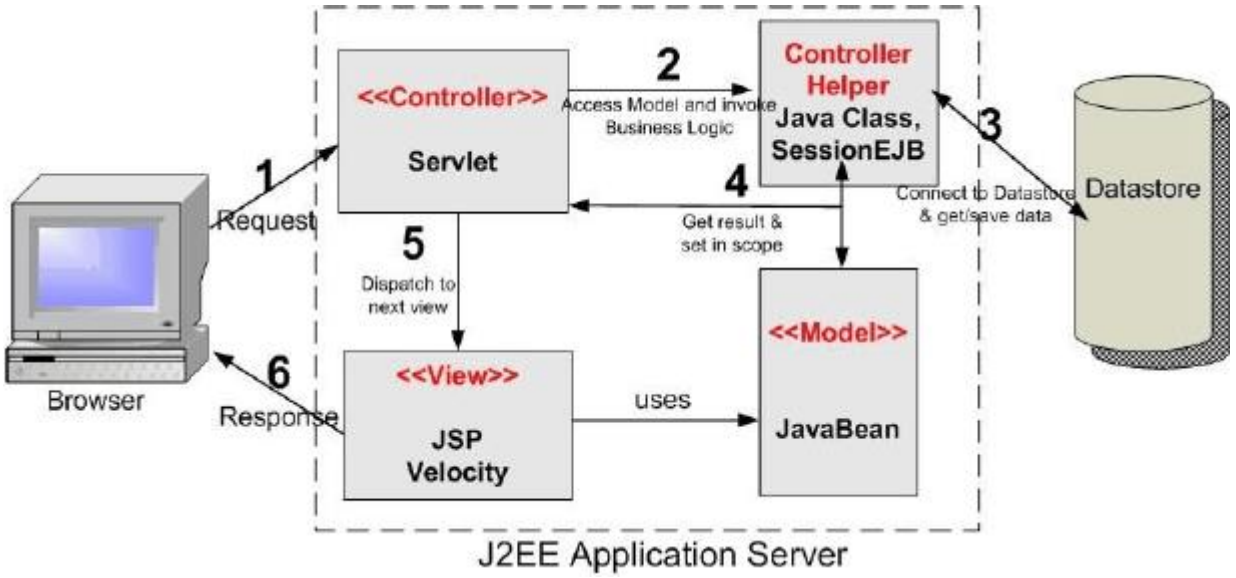


Model 1 Architecture 장단점

- 장점
  - ✓ 개발속도가 빠름
  - ✓ 개발자의 기술적인 숙련도가 낮아도 배우기 쉬워 빠르게 적용 가능
- 단점
  - ✓ JSP 페이지에서 프레젠테이션 로직과 비즈니스 로직이 혼재되어 복잡
  - ✓ 로직의 혼재로 인해 개발자와 디자이너의 작업 분리가 어려움
  - ✓ JSP 코드의 복잡도로 인해 유지보수가 어려워짐
  - ✓ 웹 애플리케이션이 복잡해지고 사용자 요구가 증가함에 따라 새로운 개발방식을 요구

Model 2 Architecture

- GUI 개발모델인 MVC를 웹 애플리케이션에 적용하여 구현한 방식
- Application의 역할을 Model – View – Controller로 분리
  - ✓ Model: Business Logic을 담당한다. – Java Bean으로 구현
    - – Business Service(Manager) – Business Logic의 workflow를 관리
    - – DAO (Data Access Object) – Database와 연동하는 Business Logic을 처리.
  - ✓ View: Client에게 응답을 처리한다. – JSP로 구현
  - ✓ Controller: 클라이언트의 요청을 받아 Model과 View사이에서 일의 흐름을 조정



Model 2 Architecture 장단점

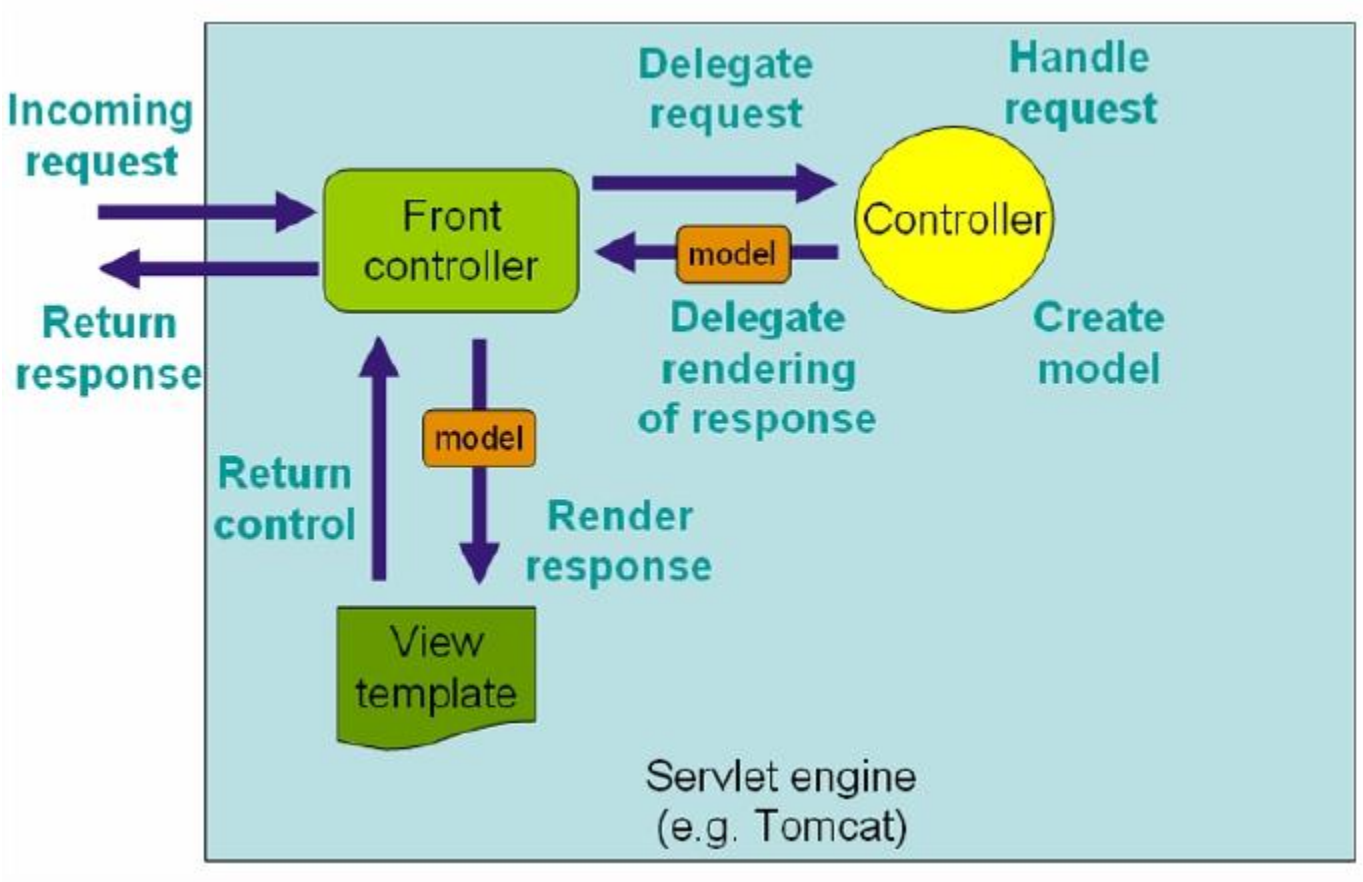
- 장점
  - ✓ 비즈니스 로직과 프리젠테이션의 분리로 인해 어플리케이션이 명료해지며 유지보수와 확장이 용이함
  - ✓ 디자이너와 개발자의 작업을 분리해 줌
- 단점
  - ✓ 개발 초기에 아키텍처 디자인을 위한 시간의 소요로 개발 기간이 늘어남
  - ✓ MVC 구조에 대한 개발자들의 이해가 필요함

Spring MVC 개요

- 뷰 선택, 로케일, 테마 결정, 핸들러 맵핑 설정, 파일 업로드 등의 클라이언트의 요청(request)을 처리할 수 있도록 고안된 MVC 프레임워크
- 기본 핸들러는 @Controller와 @RequestMapping 어노테이션을 통해 HTTP 메소드(Get, Put, Post, Delete 등)을 유연하게 처리
- Spring 3.0
  - ✓ @PathVariable 어노테이션과 다른 기능을 통해 @Controller에서 RESTful 웹 사이트와 애플리케이션을 생성 가능하게 해 줌
  - ✓ Controller 구현체는 응답 스트림(response stream)을 직접 처리할 수도 있지만, 일반적으로는 ModelAndView 인스턴스를 사용하는데, 이는 뷰 이름과 `Map`으로 구성
  - ✓ MVC의 M은 java.util.Map 인터페이스로 구성되며, 뷰 기술을 추상화함으로써 JSP, Velocity 등과 통합 가능하도록 함

핵심 - DispatcherServlet

- Spring MVC는다른 MVC 프레임워크처럼 요청 기반(Request Driven)이며, 요청을 컨트롤러로 보내는(Dispatch)
- 아래 그림의 FrontController가 DispatcherServlet의 역할



핵심 - DispatcherServlet

- DispatcherServlet은 HttpServlet 기본 클래스를 상속한 Servlet 클래스
- 웹 애플리케이션의 web.xml 에 서블릿으로 URL 맵핑을 사용하여 요청을 처리
- .form으로 끝나는 모든 요청은 example DispatcherServlet에 의해 처리

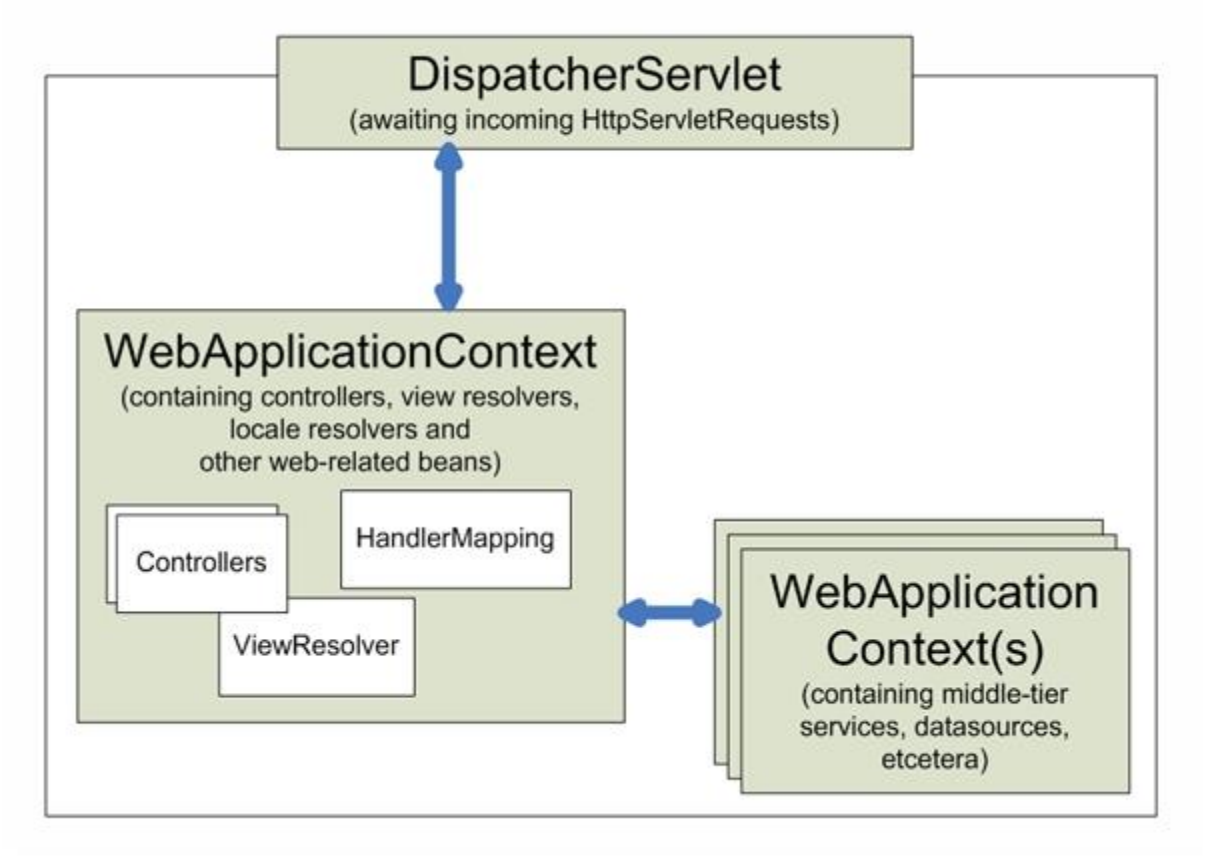
```
<web-app>
 <servlet>
    <servlet-name>example</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
 </servlet>

 <servlet-mapping>
    <servlet-name>example</servlet-name>
    <url-pattern>*.form</url-pattern>
 </servlet-mapping>

</web-app>
```

MVC의 ApplicationContext – WebApplicationContext

- MVC 프레임워크의 빈을 관리하기 위해 각 DispatcherServlet은 WebApplicationContext 루트 내에 사전에 정의된 모든 빈을 가진 자신만의 WebApplicationContext을 가짐



MVC의 ApplicationContext – WebApplicationContext

- DispatcherServlet이 초기화되면 MVC 프레임워크는 웹 애플리케이션의 WEB-INF 디렉토리에서 [servlet-name]-servlet.xml 파일을 검색

```
<web-app>
  <servlet>
    <servlet-name>golfing</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>golfing</servlet-name>
    <url-pattern>/golfing*</url-pattern>
  </servlet-mapping>
</web-app>
```

- 위 설정시 /WEB-INF/golfing-servlet.xml 파일을 검색
- WebApplicationContext는 웹 애플리케이션 처리에 필요한 부가 정보를 가진 ApplicationContext

사용자 위치 지정 Spring MVC 설정 파일 등록

- 사용자가 원하는 위치를 init-param을 통해 등록 가능
  - ✓ <servlet>의 하위태그인 <init-param>에 contextConfigLocation 이름으로 등록
  - ✓ 경로는 Application Root부터 절대경로로 표시
  - ✓ 여러 개의 경우 , 또는 공백으로 구분

```
<servlet>

   <servlet-name>dispatcher</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
      <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>/WEB-INF/server-service.xml, dao-service.xml</param-value>
      </init-param>
</servlet>
```

설정 파일을 등록하는 또 다른 방법 – ContextLoaderListener

- DispatcherServlet 여러 개 설정시 공통 Spring 설정파일 등록
  - ✓ 컨텍스트 설정 파일(스프링 설정파일)들을 로드하기 위해 리스너(ContextLoaderListener) 설정
  - ✓ 설정파일 <context-param>으로 등록

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

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/service-service.xml
                 /WEB-INF/dao-data.xml
    </param-value>
</context-param>
```

WebApplicationContext 내의 특수한 빈들

- DispatcherServlet은 요청을 처리하고 뷰를 표시하기 위해 특별한 빈들을 사용
- 특수한 빈들은 WebApplicationContext 내에 설정됨

빈유형	설명
controller	MVC의Controller에해당하는부분
handlermappings	전처리(pre),후처리를위한실행목록
viewresolvers	뷰이름을실제뷰에적용
localeresolvers	i18n형태뷰를제공하기위해사용되며,클라이언트의로케일을적용
themeresolvers	개인화된레이아웃에사용될수있는테마를적용
multipartfileresolver	HTML폼을통해파일업로드처리기능을제공
handlerexceptionresolvers	Exception을뷰에맵핑시키거나,보다복잡한예외처리코드구현

- i18n: Internaltionalization

요청시 DispatcherServlet 처리 순서

- 다음의 순서에 따라 DispatcherServlet은 요청을 처리
  1. 컨트롤러와 다른 엘리먼트에서 사용할 수 있는 요청을 바인딩한 WebApplicationContext
     검색된다.
  2. Locale resolver는 요청 처리시 사용하는 로케일을 찾기 위해 프로세스 내의 엘리먼트로 요
     청으로 바인드된다. 로케일을 사용하지 않으면 이 과정은 무시된다.
  3. Theme resolver는 테마로 사용될 뷰와 같은 엘리먼트를 요청에 바인딩한다. 테마를 사용하
     지 않으면 이 과정은 무시된다.
  4. Multipart resolver가 명시된 경우, 요청은 멀티파트를 위해 검사되고 멀티파트가 발견되면
     요청은 MultipartHttpServletRequest로 랩핑된다.
  5. 요청에 대해 적절한 핸들러가 검색되고 핸들러가 발견되면 핸들러와 연관된 실행 체인(pr
     processor, post-processor, controllers)이 모델 또는 렌더링을 위해 실행된다.
  6. 만약 모델이 반환되면 뷰가 표시된다. 모델이 반환되지 않으면, 뷰가 표시되지 않는다.

- WebApplicationContext에 선언된 handler exception resolver는 요청 처리 도중 발생한 예외를 얻어낸다.

Spring MVC 구현 순서

- Spring MVC를 이용한 어플리케이션 작성 순서
  1. web.xml에 DispacherServlet 등록 및 Spring 설정파일 등록
  2. 설정파일에 HandlerMapping 설정
  3. 컨트롤러 구현 및 Spring 설정파일에 등록
  4. 컨트롤러와 JSP의 연결 위해 View Resolver 설정
  5. JSP 코드 작성

Controller 구현

- Controller는 하나의 서비스 인터페이스를 통해 정의된 애플리케이션에 대한 접근제
- 표준 서블릿 API 혹은 컨트롤러 등을 상속받을 필요가 전혀 없음
- 뷰에 의해 표시되는 모델로 사용자의 입력을 파싱하고 변환
- Spring 2.5부터 @RequestMapping, @RequestParam, @ModelAttribute 등을 사용
- Spring MVC 구현의 모든 핵심은 ModelAndView, @Controller, @RequestMapping

```
@Controller
public class HelloWorldController {

    @RequestMapping("/helloWorld")
    public ModelAndView helloWorld() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("helloWorld");
        mav.addObject("message", "Hello World!");
        return mav;
    }
}
```

@Controller 스캔을 위한 설정

- @Controller를 사용하기 위해서 반드시 설정에 컴포넌트 스캐닝에 대한 설정 추가 필

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:p="http://www.springframework.org/schema/p"
   xmlns:context="http://www.springframework.org/schema/context"
   xsi:schemaLocation="
      http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
      http://www.springframework.org/schema/context
      http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package="com.hhi.web"/>
   // ...

</beans>}
```

URL을 위한 클래스 레벨 @RequestMapping

- /appointments 와 같은 URL을 맵핑하기 위해서 @RequestMapping 어노테이션 사용

```
@Controller
@RequestMapping("/appointments")
public class AppointmentsController {
    private @Autowired AppointmentBook appointmentBook;

   @RequestMapping(method = RequestMethod.GET)
    public Map<String, Appointment> get() {
        return appointmentBook.getAppointmentsForToday();
    }
   @RequestMapping(value="/new", method = RequestMethod.POST)    // ←부분은 /appointments/new 를 통해 접근
    public String add(@Valid AppointmentForm appointment, BindingResult result) {
        if (result.hasErrors()) {
            return "appointments/new";
        }
        appointmentBook.addAppointment(appointment);
        return "redirect:/appointments";
    }
}
```

URL을 위한 메소드 레벨 @RequestMapping

- @RequestMapping 은 클래스 레벨이 아닌 메소드 레벨에서 직접 사용 가능
- 모든 경로는 절대 경로이며, 상대 경로가 아님

```
@Controller
public class ClinicController {

    private final Clinic clinic;
    @Autowired
    public ClinicController(Clinic clinic) {
        this.clinic = clinic;
    }

    @RequestMapping("/")
    public void welcomeHandler() {
    }

    @RequestMapping("/vets")
    public ModelMap vetsHandler() {
        return new ModelMap(this.clinic.getVets());
    }
}
```

RESTful 서비스를 위한 @PathVariable

- URI 템플릿을 위한 @PathVariable을 통해 입력 값을 얻어낼 수 있음
- {변수명} 규칙을 통해 값을 메소드 파라미터를 통해 받음

```
@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable String ownerId, Model model) {  // ← 변수명과 맵핑 시킴
    // implementation omitted
}
@RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
public String findOwner(@PathVariable("ownerId") String theOwner, Model model) {   // ← Path명과 변수명이 다름
    // implementation omitted
}
```

```
@RequestMapping(value="/owners/{ownerId}/pets/{petId}", method=RequestMethod.GET)  // ← 2개 파라미터
public String findPet(@PathVariable String ownerId, @PathVariable String petId, Model model) {
  Pet pet = owner.getPet(petId);
  model.addAttribute("pet", pet);
  return "displayPet";
}
```

파라미터를 담아내기 위한 @ModelAttribute

- 요청 파라미터를 데이터바인딩을 통해 모델 객체에 담는 역할을 담당

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<html><body>
<form:form commandName="commandObject">
    이름: <form:input path="name"/><br>
    별명: <form:input path="nickName"/>
</form:form>
</body></html>
```

```
@Controller
public class HomeController {
    @RequestMapping("/command")
    public String command(CommandObject command) {
    }
    @RequestMapping("/modelCommand")
    public String modelCommand(@ModelAttribute CommandObject command) {
    }
}
```

두 개의 메소드는 동일한 결과임

세션을 담아내기 위한 @SessionAttribute

- 특정 핸들러에 의해 사용되는 세션 속성을 정의하기 위해 @SessionAttribute 사용
- 일반적으로 요청 간 세션에 저장된 모델 애트리뷰트의 이름이나 유형을 나열

```
@Controller
@RequestMapping("/editPet.do")
@SessionAttributes("pet")
public class EditPetForm {
    // ...
}
```

HTTP 쿠키값을 얻어내기 위한 @CookieValue

- HTTP Cookie 값을 얻어낼 수 있는 파라미터에 바인딩되도록 @CookieValue를 사용
- 쿠키값이 JSESSIONID=415A4AC178C59DACE0B2C9CA727CDD84와 같은 경우 아래와 같은 코드를 통해 접근 가능

```
@RequestMapping("/displayHeaderInfo.do")
public void displayHeaderInfo(@CookieValue("JSESSIONID") String cookie)  {

  //...

}
```

요청 헤더 값을 알아내기 위한 @RequestHeader

- 요청 헤더 파라미터를 얻을 수 있도록 사용되는 어노테이션
- 헤더 예

```
Host                    localhost:8080
Accept                  text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language         fr,en-gb;q=0.7,en;q=0.3
Accept-Encoding         gzip,deflate
Accept-Charset          ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive              300
```

- 코드 예

```
@RequestMapping("/displayHeaderInfo.do")
public void displayHeaderInfo(@RequestHeader("Accept-Encoding") String encoding,
                              @RequestHeader("Keep-Alive") long keepAlive) {

 //...
}
```

@RequestMapping의 메소드 시그니처에 적용가능한 객체

- ServletRequest/ServletResponse, HttpSession: 표준 서블릿 API
- org.springframework.web.context.request.WebRequest/NativeWebRequest
- java.util.Loale: 로케일 확인
- java.io.InputStream/java.io.Reader: 요청 콘텐츠에 대한 접근 가능(서블릿 API)
- java.io.OutputStream/Writer: 클라이언트용 응답 스트림(서블릿 API)
- java.security.Principal: 현재 인증된 사용자 정보를 가짐
- @PathVariable: URI 템플릿 변수에 접근
- @RequestParam: URI 요청 파라미터 접근(http://url?param=value)
- @RequestHeader: 특정 서블릿 요청 헤더 파라미터 접근 가능
- @RequestBody: HTTP 요청 바디에 대한 접근, 파라미터는 HttpMessageConverter를 통해 변
- @HttpEntity<?>: 서블릿 요청 HTTP 헤더와 콘텐츠에 대한 접근 가능
- java.util.Map, org.springframework.ui.Model/ModelMap: 뷰로 전달될 파라미터
- org.springframework.validation.Errors /BindingResult: 명령, 폼 객체에 대한 유효성 검증결과
- org.springframework.web.bind.support.SessionStatus: 세션 애트리뷰트에 대한 상태

핸들러에서 사용 가능한 반환 타입(return type)

- ModelAndView, Model, Map, View, String, void 유형
- @RequestBody

```
@RequestMapping(value= "/something", method = RequestMethod.PUT)
public void handle(@RequestBody String body, Writer writer) throws IOException {
  writer.write(body);
}
```

- @ResponseBody

```
@RequestMapping(value= "/something", method = RequestMethod.PUT)
@ResponseBody
public String helloWorld()  {
  return "Hello World";
}
```

- 뷰 처리 -

※ 본 단원의 주제:
뷰를 결정하기 위해 사용된 Resolver들을 살펴보고 각각의 설정 방법을 살펴본다.

View Resolver(뷰 처리기)

- Spring MVC는 여러 가지 뷰 형태를 지정하는 위한 방법을 제공하며, 이를 ViewResolver라 함
- 기본적으로 특별한 뷰 기술없이 브라우저 내에서 모델을 표현하기 위해 View Resolver 제공
- 두 가지 중요 인터페이스 : View, ViewResolver
  - ✓ View Resolver – 뷰 이름과 실제 뷰 간의 맵핑을 제공
  - ✓ View – 요청을 다른 뷰 기술중의 하나로 넘겨주는 기능을 제공

ViewResolver	설명
AbstractCachingViewResolver	뷰를캐싱하는추상뷰선택기.종종사용되기전에뷰가사전에필요할수도있다.
XmlViewResolver	SpringXML빈과동일한DTD를사용하여XML로작성된설정파일을얻는 ViewResolver.기본설정파일은/WEB-INF/views.xml파일이다.
ResourceBundleViewResourceResolver	ResourceBundle빈정의를사용하는ViewResolver구현체.클래스패스에위치한 프로퍼티파일내의번들을정의.기본파일이름은view.properties
UrlBasedViewResource	논리적인뷰이름을URL로맵핑시켜주는가장단순한형태의ViewResolver구현체
InternalResourceViewResolver	JstlView와TilesView와같은서브클래스와InternalResourceView를지원하는 UrlBasedViewResolver
VelocityViewResolver/ FreeMarkerViewResolver	VelocityView또는FreeMarkerView를지원하는UrlBasedViewResolver의하위클 래스
ContentNegotiationgViewResolver	요청파일이름이나Accept헤더에기반한뷰를선택하는ViewResolver구현체

JSP를 뷰 기술로 사용하고자 하는 경우

- JSP를 뷰 기술로 활용할 경우 일반적으로 UrlBasedViewResolver를 사용할 수 있음
- 뷰 처리기(View Resolver)는 뷰 이름을 URL로 변환하고 뷰를 보여주기 위해 RequestDispatcher로 요청을 넘김

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

※ 위의 예제로 test라는 논리적인 뷰이름을 반환했을 때, 처리기는 요청을 /WEB-INF/jsp/test.jsp로 요청을 포워딩한다.

뷰로 리다이렉팅하기 – RedirectView

- RequestDispatcher.forware(), RequestDispatcher.include()와 같이 뷰가 렌더링(표시)되기 전에 클라이언트로 리다이렉트하고자 할 경우 사용
- RedirectView : HttpServletResponse.sendRedirect()를 호출하여 클라이언트 브라우저로 HTTP 리다이렉트를 반환도 가능하며, 일반적인 뷰 처리 기능이 작동하지 않음
- 컨트롤러에서 뷰 처리기를 거치지 않고 컨트롤러로 이동할 때 사용

```
import org.springframework.web.servlet.view.RedirectView;
…
@RequestMapping(value="/owners/{ownerId}/pets/{petId}", method=RequestMethod.GET)
 public ModelAndView findPet(@PathVariable String ownerId, @PathVariable String petId, ModelAndView mav) {
  mav.setView(new RedirectView("overview.ads?method=overview"));
  mav.addObject("pet", pet);
  return mav;
}
```

뷰로 리다이렉팅하기 – redirect:

- 컨트롤러가 응답이 어떻게 처리되는 지 관여하지 않을 경우 사용
- 뷰 이름이 redirect: 으로 시작되면, UrlBasedViewResolver는 이후 문자열이 리다이렉트가 필요한 것으로 인식하고, 리다이렉트시킴.
- redirect:/my/response/action.do 와 같은 경우 상대 경로로, redirect:http://host/pat 로 호출될 경우 절대 경로로 리다이렉트

```
@RequestMapping(method = RequestMethod.POST)
  public String add(@Valid AppointmentForm appointment, BindingResult result) {
      if (result.hasErrors()) {
          return "appointments/new";
      }
      appointmentBook.addAppointment(appointment);
      return "redirect:/appointments";
  }
```

뷰로 리다이렉팅하기 – forward:

- UrlBasedViewResolver와 하위 클래스에 의해 처리되는 뷰 이름을 위한 지시어
- 사용자의 요청 a.do를 처리하는 과정에 가공된 request, response들을 b.do를 처리하는 컨트롤러에서 모두 사용가능

```
@RequestMapping(method = RequestMethod.POST)
  public String add(@Valid AppointmentForm appointment, BindingResult result) {
    if (result.hasErrors()) {
      return "appointments/new";
    }
    appointmentBook.addAppointment(appointment);
    return “forward:/appointments";
  }
```

가장 효과적인 뷰 처리기 - ContentNegotiatingViewResolver

- 뷰 처리가(View Resolver) 자신이 뷰를 처리하지 않고, 다른 뷰 처리기로 작업을 위임하며, 클라이언트에 의해 요청된 표현 형태로 구성
- 요청에 따라 JSP로 보여줄 수도 있으며, Ajax를 통한 JSON 등으로 보여주고자 하는 경우 등

http://www.example.com/users/sample.html ← 일반적인 JSP를 통해 렌더링하고자 하는 경우
http://www.example.com/users/sample.json ← JSON으로 결과값을 얻고자 하는 경우
http://www.example.com/users/sample.atom ← RSS로 결과값을 얻고자 하는 경우

가장 효과적인 뷰 처리기 - ContentNegotiatingViewResolver

```
<bean class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
  <property name="mediaTypes">
    <map>
      <entry key="atom" value="application/atom+xml"/>
      <entry key="html" value="text/html"/>
      <entry key="json" value="application/json"/>
    </map>
  </property>
  <property name="viewResolvers">
    <list>
      <bean class="org.springframework.web.servlet.view.BeanNameViewResolver"/>
      <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/jsp/"/>
        <property name="suffix" value=".jsp"/>
      </bean>
    </list>
  </property>
  <property name="defaultViews">
    <list>
      <bean class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />
    </list>
  </property>
</bean>
```

파일 업로드 처리 – MultipartResolver

- Spring의 내장(built-in) 파일 업로드 지원 기능이 존재
- org.springframework.web.multipart.MultipartResolver 클래스를 통해 지원하며, 아파치 Commons FileUpload를 사용
- commons-fileupload.jar 라이브러리가 클래스 패스에 있어야 함

```
<bean id="multipartResolver"
   class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

   <!-- one of the properties available; the maximum file size in bytes -->
   <property name="maxUploadSize" value="100000"/>
</bean>
```

- DispatcherServlet이 multi-part 요청임을 인식하면 컨텍스트에 선언된 처리기로 요청을 넘김
- 처리기는 HttpServletRequest를 파일업로드를 지원하는 MultipartServletRequest로 랩핑하여 처리

파일 업로드 처리 – 화면(HTML)

- HTML 코드

```
<html>
  <head>
    <title>Upload a file please</title>
  </head>
  <body>
    <h1>Please upload a file</h1>
    <form method="post" action="/form" enctype="multipart/form-data">
      <input type="text" name="name"/>
      <input type="file" name="file"/>
      <input type="submit"/>
    </form>
  </body>
</html>
```

파일 업로드 처리 –컨트롤러

- 컨트롤러 코드

```
@Controller
public class FileUpoadController {

   @RequestMapping(value= "/form", method = RequestMethod.POST)
   public String handleFormUpload(@RequestParam("name") String name,
      @RequestParam("file") MultipartFile file) {

      if (!file.isEmpty()) {
         byte[] bytes = file.getBytes();
         // store the bytes somewhere
         return "redirect:uploadSuccess";
      } else {
         return "redirect:uploadFailure";
      }
   }
}
```

파일 업로드 처리 – 컨트롤러와 처리기 설정

- applicationContext.xml

```
<beans>
   <bean id="multipartResolver"
      class="org.springframework.web.multipart.commons.CommonsMultipartResolver"/>
   <!-- Declare explicitly, or use <context:annotation-config/> -->
   <bean id="fileUploadController" class="examples.FileUploadController"/>

</beans>
```

- 이름 규칙 -

※ 본 단원의 주제:

모델-뷰-컨트롤러에 대해 사용되는 일반적인 이름 등의 규칙들이 어떻게 적용되는지 살펴본다.

요청별 URL을 분리하기 - ControllerClassNameHandlerMapping

- 요청 URL와 해당 요청을 처리하는 Controller 인스턴스 간의 맵핑을 사용하여 URL을 구분

```
public class ViewShoppingCartController implements Controller {
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {
        // the implementation is not hugely important for this example...
    }
}
```

```
<bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>
<bean id="viewShoppingCart" class="x.y.z.ViewShoppingCartController">
    <!-- inject dependencies as required... -->
</bean>
```

- DispatcherServlet이 multi-part 요청임을 인식하면 컨텍스트에 선언된 처리기로 요청을 넘김
- 규칙: *Controller 클래스들을 모두 찾아내고 Controller 글자를 뺀 앞의 이름을 URL로 인식
  - ✓ WelcomeController → /welcome* request URL
  - ✓ HomeController → /home* request URL
  - ✓ IndexController → /index* request URL
  - ✓ RegisterController → /register* request URL

뷰를 위한 객체 – ModelMap(ModelAndView)

- View에 나타날 추가적인 객체를 만들수 있는 기본적으로 중요한 Map

```
public class DisplayShoppingCartController implements Controller {
   public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {

      List cartItems = // CartItems 객체목록 얻기
      User user = // 쇼핑 중인 사용자 정보 얻기

      ModelAndView mav = new ModelAndView("displayShoppingCart"); <-- 논리 뷰 이름

      mav.addObject(cartItems); <-- 이름없는 그냥 cartItems 객체
      mav.addObject(user); <-- 이름 없는 그냥 user 객체

      return mav;
   }
}
```

뷰를 위한 객체 – ModelMap(ModelAndView)

- 객체 생성시 만들어지는 이름 규칙은 다음과 같음
- 객체 클래스의 이름은 패키지명을 뺀 클래스 이름의 소문자 형태로 생성
  - ✓ x.y.User 인스턴스는 user 라는 이름으로 생성
  - ✓ x.y.Foo 인스턴스는 foo 라는 이름으로 생성
  - ✓ java.util.HashMap 인스턴스는 hashmap 이라는 이름으로 생성
- Set, List, 배열의 경우 패키지명을 뺀 클래스 이름 + List 단어를 뒤에 붙임
  - ✓ x.y.User [] 인스턴스는 userList 라는 이름으로 생성
  - ✓ x.y.Foo [] 인스턴스는 fooList 라는 이름으로 생성
  - ✓ java.util.ArrayList<User> 인스턴스는 userList
  - ✓ java.util.HashSet<Foo> 인스턴스는 fooList

이름없는 뷰를 위한 처리 – RequestToViewNameTranslator

- 논리적인 뷰 이름을 명시적으로 제공하지 않은 경우 RequestToViewNameTranslator 인터페이스가 논리적인 뷰 이름을 결정

```
public class RegistrationController implements Controller {

   public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {
      // 요청처리
      ModelAndView mav = new ModelAndView();
       // 필요에 따라 데이터를 모델에 추가
      return mav;
      // 뷰 이름이나 논리 뷰 이름이 없는 상태로 반환
   }
}
```

※ ModelAndView에 어떠한 값도 세팅되어 반환되기 않기 때문에 RequestToViewNameTranslator는 요청URL로부터 이름을 추출하는데, 위의 경우는 클래스 이름이 RegistrationController 이므로 ControllerClassNameHandlerMapping을 활용하여 registration이라는 이름을 사용

※ 즉, http://localhost/registration.html의 요청은 /WEB-INF/jsp/registration.jsp 로 전달

이름없는 뷰를 위한 처리 – RequestToViewNameTranslator

- Spring XML 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN"
      "http://www.springframework.org/dtd/spring-beans-2.0.dtd">
<beans>
   <!-- this bean with the well known name generates view names for us -->
   <bean id="viewNameTranslator" class="org.springframework.web.servlet.view.DefaultRequestToViewNameTranslator"/>
   <bean class="x.y.RegistrationController">
      <!-- inject dependencies as necessary -->
   </bean>

   <!-- maps request URLs to Controller names -->
   <bean class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandlerMapping"/>
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/jsp/"/>
      <property name="suffix" value=".jsp"/>
   </bean>
</beans>
```