



Spring Framework

✓ 원리를 알면 IT가 맛있다

Spring Framework for Beginners



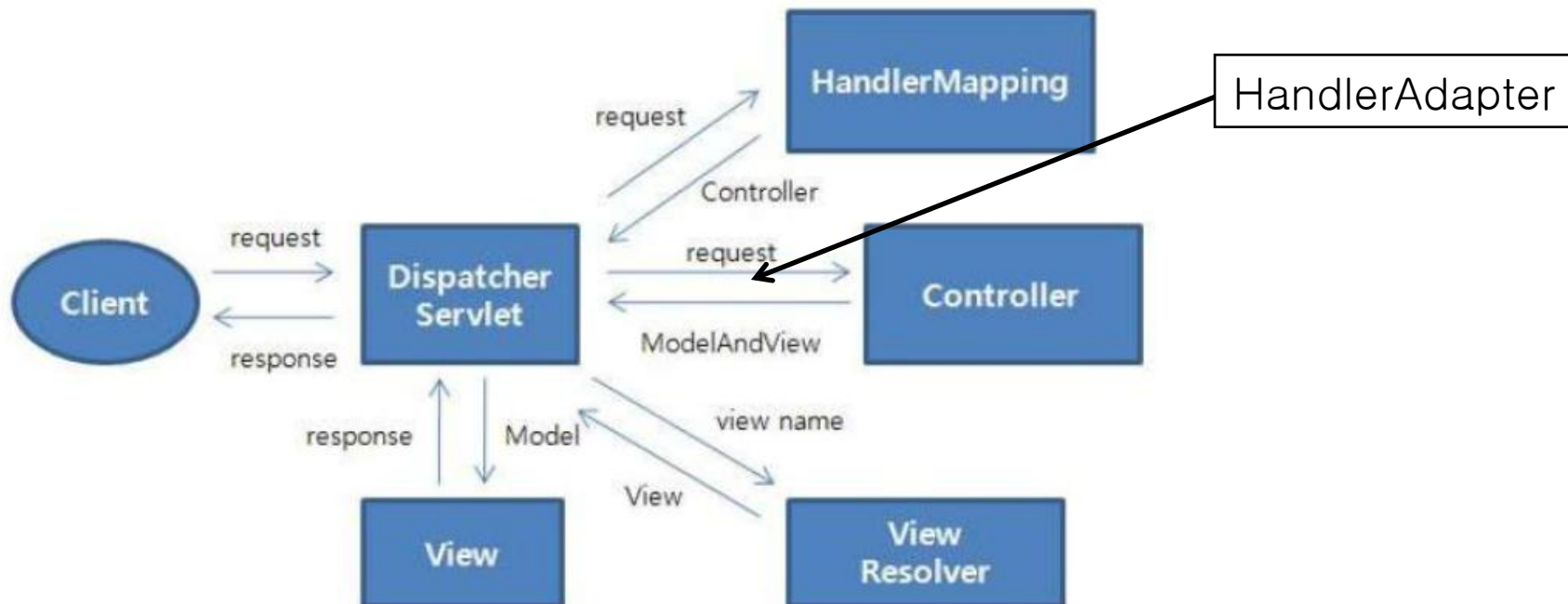
chapter **05.**

Spring Web MVC

- 스프링의 MVC 아키텍처
- 스프링 MVC 핵심 컴포넌트
- DispatcherServlet 기능
- HandlerMapping 기능
- ViewResolver 기능.
- 어노테이션 기반의 Controller 구현 방법
 - @Controller , @RequestMapping
 - @RequestParam , @ModelAttribute 등
- 스프링 파일 업로드 및 다운로드 방법
- HandlerInterceptor 인터페이스
- 스프링 예외처리
- flash 스코프
- MVC 네임스페이스

- MVC (Model-View-Controller)패턴은 코드를 기능에 따라 Model,View,Controller 3가지 요소로 분리하여 구현하는 개발 방법이다.
 - Model : 어플리케이션의 데이터와 비즈니스 로직을 담는 객체이다.
 - View : Model의 정보를 사용자에게 표시한다. 하나의 Model을 다양한 View에서 사용할 수 있다.
 - Controller : Model과 View의 중계역할을 담당한다. 사용자의 요청을 받아 Model에 변경된 상태를 반영하고 응답을 위한 View를 선택한다.
- MVC 패턴은 UI코드와 비즈니스 코드를 분리함으로써 종속성을 줄이고 재사용성을 높여서 보다 쉬운 유지보수성을 확보할 수 있다.
- 오픈소스 Web MVC Framework
예> Spring MVC, Struts, Struts2 등

- 클라이언트 요청이 들어오면 DispatcherServlet이 가장 먼저 요청을 받는다.
- HandlerMapping이 요청에 해당하는 Controller(.java)를 return한다.
- Controller는 비즈니스 로직을 수행(호출)하고 결과 데이터를 ModelAndView에 반영하여 return 한다.
- ViewResolver는 논리적인 view 이름을 받아 해당하는 View객체를 return 한다.
- 마지막으로 Model을 가지고 최종적으로 보여질 JSP로 포워드 된다.



- DispatcherServlet
 - 클라이언트의 요청을 전달받아서 컨트롤러에게 클라이언트의 요청을 전달하고 컨트롤러가 리턴한 결과값을 View에게 전달하여 알맞은 응답을 생성하는 역할.
- HandlerMapping
 - 클라이언트의 요청 URL을 어떤 컨트롤러가 처리할 지를 결정.
- Controller
 - 실제 비즈니스 로직을 처리하는 컴포넌트이다.
 - Spring API 또는 POJO 형식으로 작성한다.
- ModelAndView
 - 컨트롤러가 처리한 결과 정보 및 뷰 선택에 필요한 정보를 저장.
- ViewResolver
 - 컨트롤러의 처리결과를 생성할 뷰를 결정.
- View
 - 컨트롤러의 처리 결과 화면을 생성

- 웹 어플리케이션에서 동작하는 IoC컨테이너는 2가지 방법으로 만들어진다. (일반적으로 같이 사용: 웹기술에 의존적인 부분과 아닌부분을 분리할 목적)

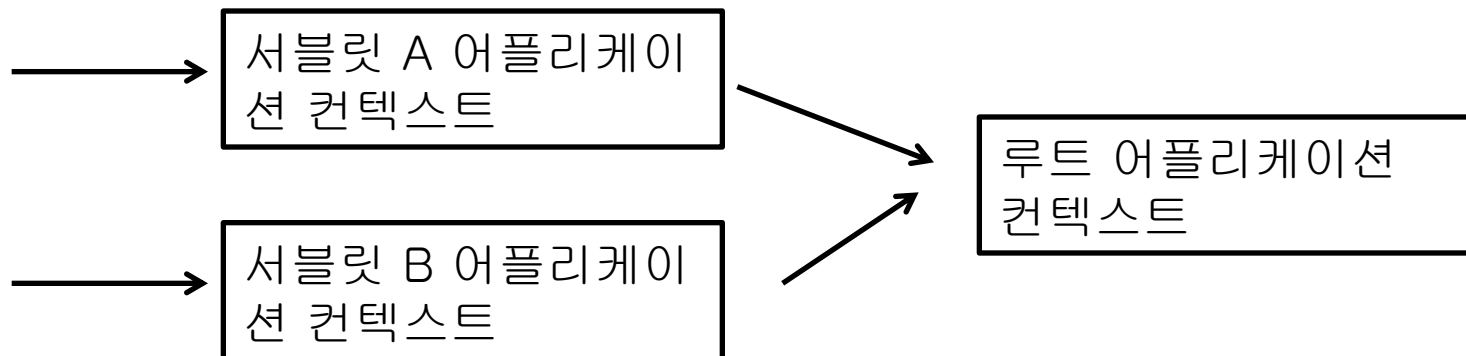
가. 서블릿 어플리케이션 컨텍스트

: 스프링 어플리케이션의 요청을 처리하는 서블릿안에서 생성.

나. 루트 어플리케이션 컨텍스트

: 웹 어플리케이션에서 생성.

: 서블릿 레벨에 등록되는 컨테이너들의 부모 컨테이너가 된다.



- 클라이언트의 요청을 처리할 DispatcherServlet을 web.xml 파일에 등록한다.
- 웹 어플리케이션에서 사용할 빈들의 설정 파일명은 기본적으로 등록된 서블릿명을 사용한다. 예> **서블릿명-servlet.xml** ➔ **서블릿 어플리케이션 컨텍스트**

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
...
<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>
...
</web-app>
```

➔ **dispatcher-servlet.xml**

***url 패턴**

가. 디렉토리 패턴
예> /xyz

나. 확장자 패턴
예> *.do

- 웹 어플리케이션에서 사용할 빈들의 설정 파일명은 기본적으로 등록된 서블릿명을 사용한다. 하지만 다음처럼 명시적으로 설정도 가능하다. 복수설정도 가능.

web.xml

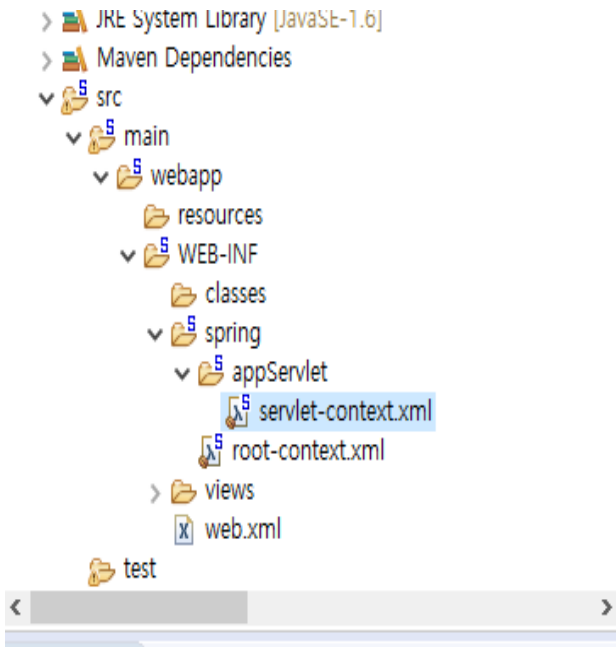
```
<servlet>
  <servlet-name>dept</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
      <!-- /WEB-INF/test-context.xml
        classpath:service-context.xml
      -->
      /WEB-INF/*-context.xml
      /WEB-INF/test2.xml
    </param-value>
  </init-param>
```

1-project의 생성 및 pom.xml 의 수정

The screenshot shows the 'New' project wizard in the Spring IDE. The 'Spring Legacy Project' option is selected. The 'Properties' tab is active, showing the 'org.springframework-version' set to '4.3.22.RELEASE' and the 'org.aspectj-version' set to '1.6.10'. The 'dependencies' tab is also visible, showing the 'maven-compiler-plugin' configuration with the 'target' set to '1.8'.

위 두 부분 수정 후 프로젝트 마우스 오른쪽 -
maven-update
실행

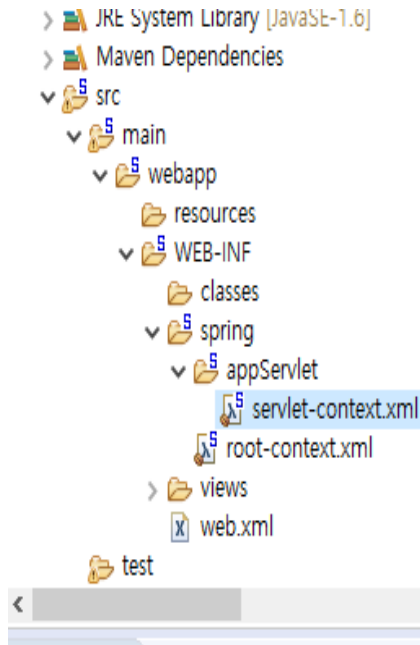
1-project의 전체 구성



1.web.xml : context환경설정(한글지원 , dispatcher 설정)

2.root-context.xml : mvc 모델의 Model, tx, dataSource설정

3.servlet-context.xml : mvc 모델의 controller, view 설정
Root-context의 네임스페이스를 복사하여 servlet-context에
복사하여 붙여넣기 후 servlet-context.xml 사용



1. 프로젝트명 - context명 aaa1
2. <artifactId>aaa1</artifactId>
3. Pom.xml
4. <org.springframework-
version>4.3.22.RELEASE</org.springframework-
version>
<configuration>

<source>1.8</source>
 <target>1.8</target> 로 수정
4. 자동생성된 HomeController 삭제할 것
5. src/main/webapp/의 web.xml 에 dispatcherservlet의 유
무 여부확인
6. root-context 의 네임스페이스를 복사 후
servlet-context. xml에 붙여넣기하여 사용하기 그냥 사용하는
경우 mvc 태그를 사용
<beans:beans>를 사용한다.

7. src/main/webapp/web-inf/spring/appservlet/servlet-context.xml에 controller 등록 방법
sevlet-context.xml에

```
<bean id="xyz" class="com.controller.TestController" />
```

또는 beans를 사용하여

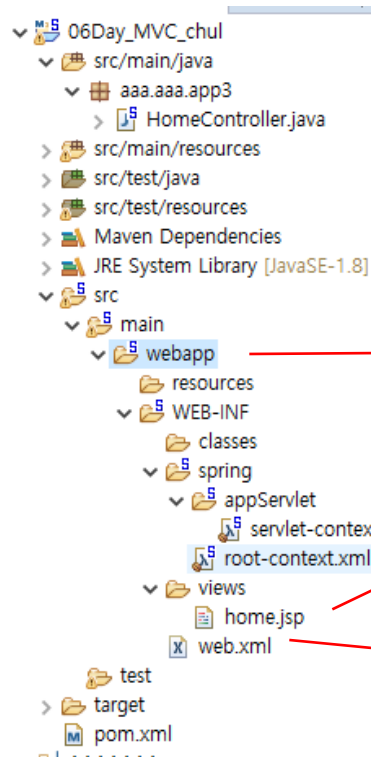
```
<context:component-scan base-package="com.controller"> </context:component-scan>
```

8. 프로젝트 마우스 오른쪽 run on server로 실행

<artifactId>app1</artifactId>가

http://localhost:포트번호/app1이 주소가 됨.

서버 모듈에서 context 주소 확인 후 controller의 매핑 부분을 참조하여 request를 요청함



controller

.js, .css, .html

.js, .css, .html .jsp

Controller 설정

```

<annotation-driven />
<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.InternalResourceViewResolve
r">
<beans:property name="prefix" value="/WEB-INF/views/" />
<beans:property name="suffix" value=".jsp" />
</beans:bean>

<context:component-scan base-package="aaa.aaa.app3" />
  
```

DispatcherServlet의 등록(servlet-context.xml)

```

<!-- 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>
  
```

06Day_MVC2_basic01_chul 프로젝트 생성하기

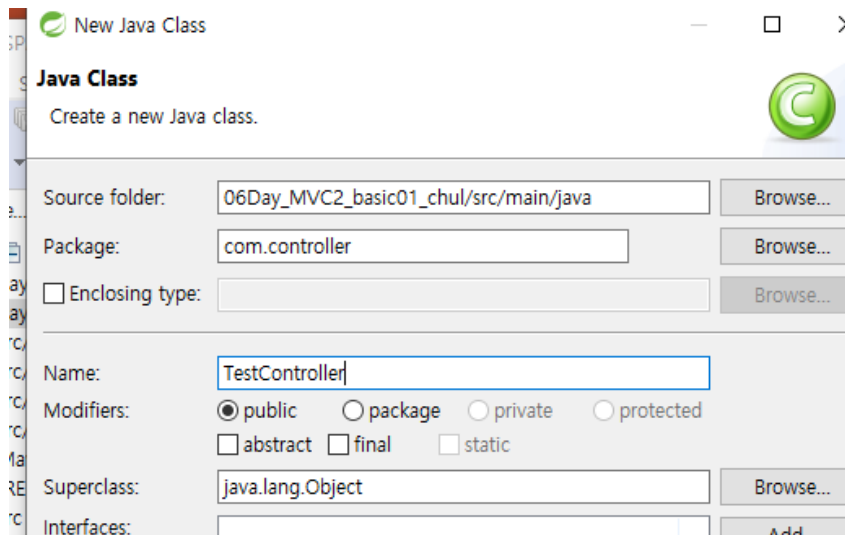
```
<java-version>1.8</java-version>
<org.springframework-version>4.3.2.RELEASE</org.springframework-ver:
<org.aspectj-version>1.6.10</org.aspectj-version>
<org.slf4j-version>1.6.6</org.slf4j-version>
</parent>
```

```
<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>
</configuration>
```

Web.xml 의 설정 - 수정불필요

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" 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">
<!-- 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>
</web-app>
```

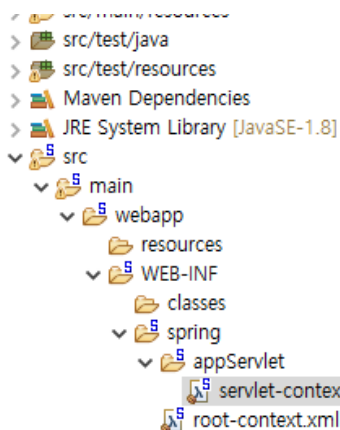

- Controller class의 작성



```

1  ; @Controller
2  public class TestController {
3      // http://localhost:8090/app3/kkk
4      @RequestMapping("/kkk")
5      public String xxx() {
6          System.out.println("xxx메소드");
7          return "/WEB-INF/views/hello.jsp";
8          //return "hello.jsp";
9          //return "hello"; //InternalResourceViewRe
10     }
11
12     @RequestMapping("/kkk2")
13     public String xxx2() {
14         System.out.println("xxx2메소드");
15         return "/WEB-INF/views/hello.jsp";
16     }
17 }

```



Root-context.xml의 네임스페이스 복사
servlet-context.xml에 붙이기

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans https://www
5
6       <!-- Root Context: defines shared resources visible to all other web compon
7
8 </beans>

```

Servlet-Context.xml 에 controller class의 등록

Select XSD namespaces to use in the configuration file

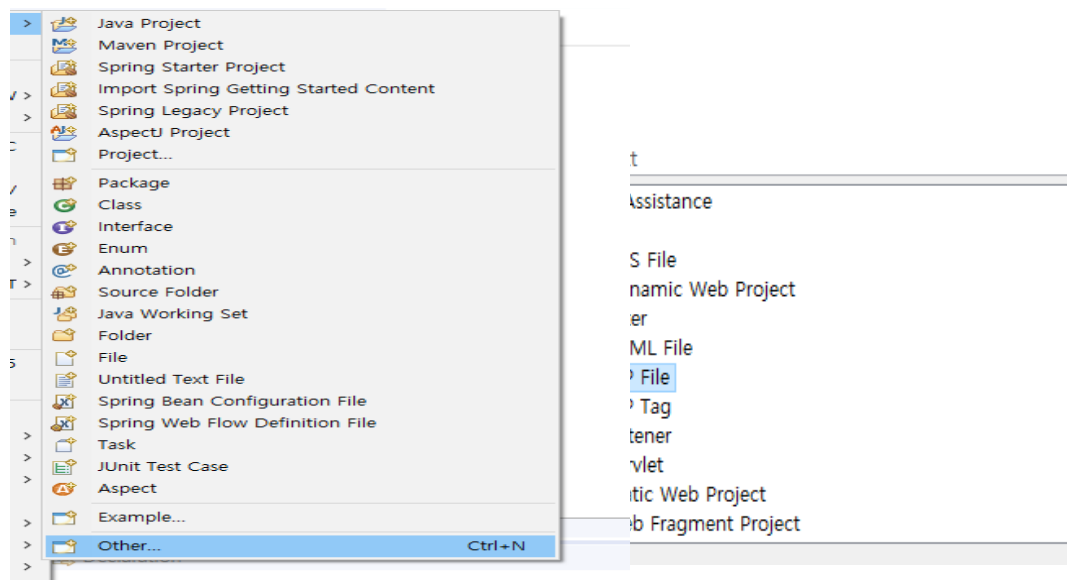
- ☐ aop - http://www.springframework.org/schema/aop
- ☒ beans - http://www.springframework.org/schema/beans
- ☐ c - http://www.springframework.org/schema/c
- ☐ cache - http://www.springframework.org/schema/cache
- ☐ context - http://www.springframework.org/schema/context
- ☐ jee - http://www.springframework.org/schema/jee
- ☐ lang - http://www.springframework.org/schema/lang
- ☐ mvc - http://www.springframework.org/schema/mvc
- ☐ p - http://www.springframework.org/schema/p
- ☐ task - http://www.springframework.org/schema/task
- ☐ util - http://www.springframework.org/schema/util

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans https://
5
6       <bean id="xyz" class="com.controller.TestController" />
7
8 </beans>

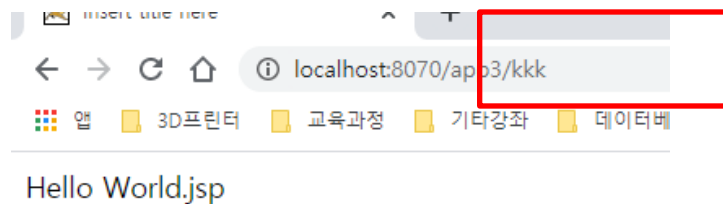
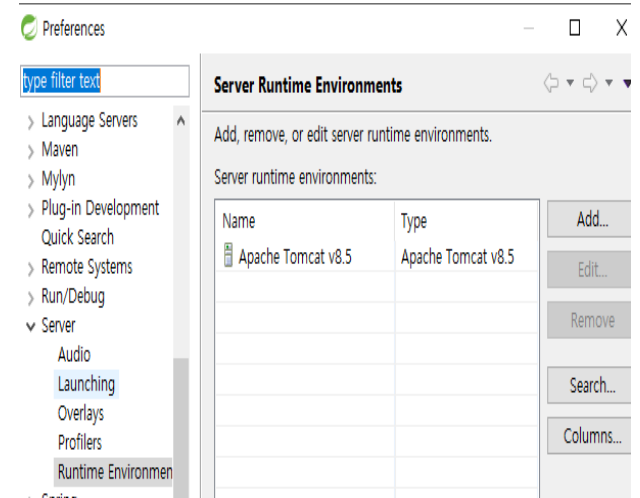
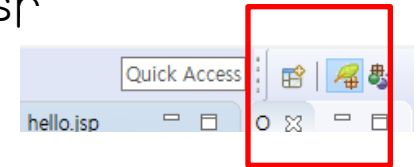
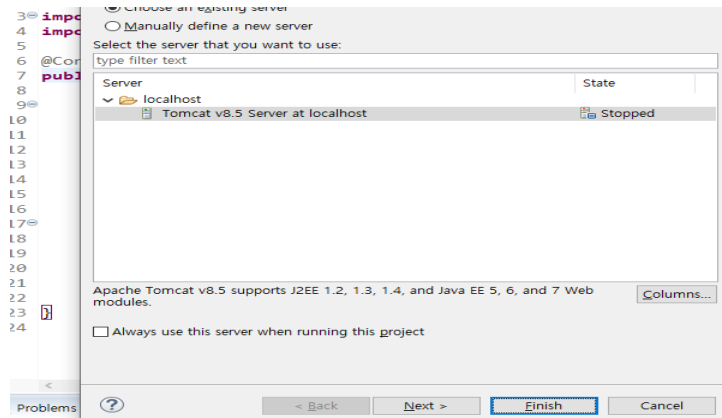
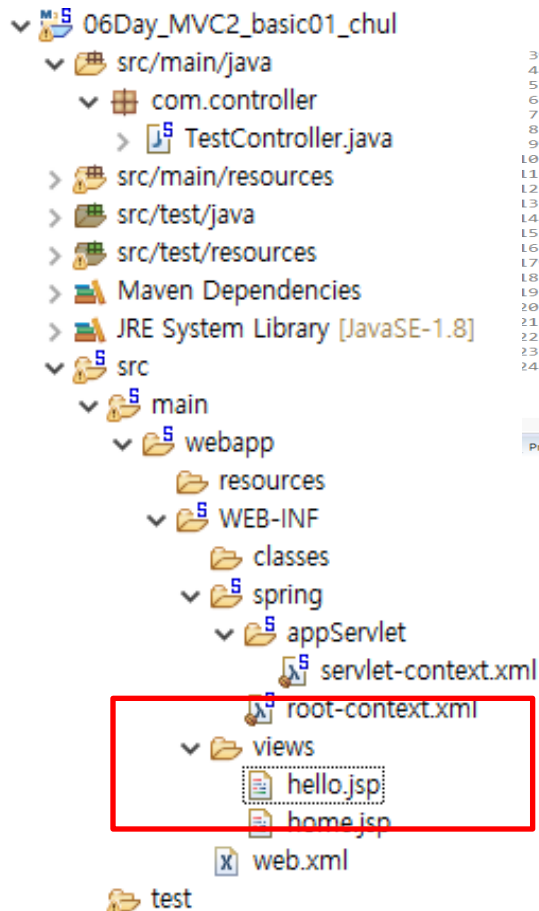
```

View부분 .jsp 만들기



View부분 .jsp 만들기 : src/main/webapp/WEB-INF/views/hello.jsp

실행 ctrl + F11 사용



06Day_MVC3_no_initParam_프로젝트 생성 및 Controller작성

– web.xml의 수정

```

1 package com.controller;
2
3 import org.springframework.stereotype.Controller;
4
5 @Controller
6 public class TestController {
7
8     @RequestMapping("/kkk")
9     public String tt() {
10         System.out.println("tt 메서드");
11         return "/WEB-INF/views/hello.jsp";
12     }
13 }

```

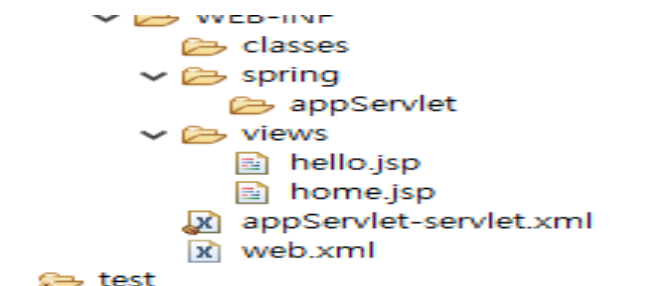
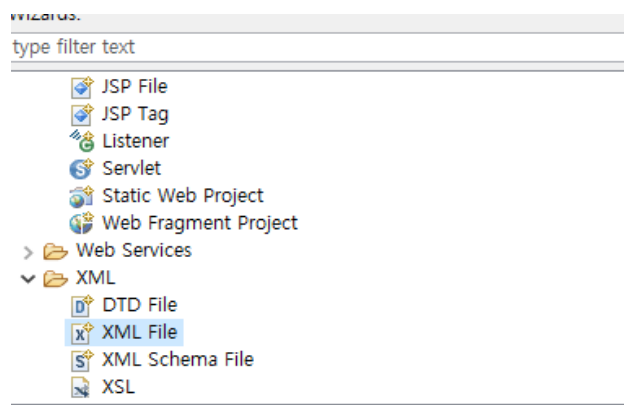
```

<!-- 삭제 -->
<!-- Processes application requests -->
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<!-- 삭제 -->
</servlet>

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

```

– appServlet-servlet.xml의 생성

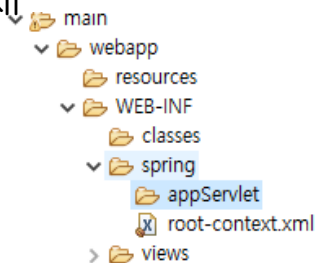


- appServlet-servlet.xml의 작성(web.xml의 네임스페이스 복사 붙여 넣기 사용)

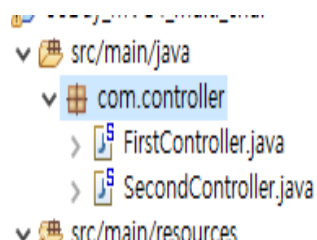
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4       xsi:schemaLocation="http://www.springframework.org/schema/beans https://www
5
6       <bean id="xxx" class="com.controller.TestController"></bean>
7
8 </beans>
9 |
```

- main/webapp/WEB-INF/spring/appServlet의 servlet-context.xml

삭제



-06Day_MVC4_multi_chul



```

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

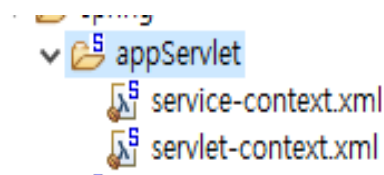
<servlet-mapping>
  <servlet-name>appServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

```

localhost:8900/aap1/kkk

localhost:8070/aap1/kkk2

3D프린터 교육과정 기타강좌 데이터베이스



```

4      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans-3.0.xsd"
5
6      <bean id="xxx" class="com.controller.FirstController"></bean>
7
8    </beans>

```

```

1      xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans-3.0.xsd"
2
3      <bean id="yyy" class="com.controller.SecondController"></bean>
4
5    </beans>

```

- 스프링에서는 인코딩 처리를 위하여 다음과 같은 Filter를 사용한다.(web.xml)

```
<!-- POST 한글처리 -->
<filter>
  <filter-name>xx</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>xx</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

○ 용도

: DispatcherServlet이 요청을 전송할 Controller를 알아내기 위해 하나 이상의 HandlerMapping에게 도움을 청한다.

○ 종류

가. BeanNameUrlHandlerMapping (기본)

: Controller의 빈 이름에 근거해 Controller를 URL에 맵핑한다.

나. ControllerClassNameHandlerMapping

: Controller의 클래스 이름을 URL의 기반으로 이용해 Controller를 URL에 맵핑한다.

다. DefaultAnnotationHandlerMapping (기본)

: **@RequestMapping 어노테이션이 적용된 Controller 메소드와 맵핑한다.**

라. SimpleUrlHandlerMapping

: 스프링 애플리케이션 컨텍스트에 정의되어 있는 프로퍼티 컬렉션을 이용해 Controller를 URL에 맵핑한다.

가. BeanNameUrlHandlerMapping

- 기본 핸들러 맵핑이다.
- 빈의 이름에 들어있는 URL을 HTTP요청의 URL과 비교해서 일치하는 빈을 찾아준다. 가장 직관적이고 사용하기 쉬운 핸들러 맵핑 전략이다.
- ANT 패턴이라고 불리는 , *나 **, ?와 같은 와일드카드 사용 가능.
**는 하나이상의 경로를 지정할 때 사용한다.

➔ /s, /s1, /sabce 같은 URL에 맵핑됨.

```
<bean name="/s*" class="com.test.XXXController" />
```

```
<bean name="/root/**/sub" class="com.test.XXXController" />
```

```
<bean name="/hello.do" class="com.test.XXXController" />
```

나. ControllerClassNameHandlerMapping

- 컨트롤러 클래스 이름을 URL에 매핑해주는 핸들러 매핑 클래스이다.
- 기본적으로 클래스 이름을 모두 URL로 사용하지만 Controller로 끝날 때는 Controller를 뺀 나머지 이름을 URL에 매핑해준다.

CoC 스타일 (Convention Over Configuration)

예> WelcomeController → /welcome*

HelloGuestController → /helloguest*

```
<bean id="controllerClassMapping"  
class="org.springframework.web.servlet.mvc.support.ControllerClassNameHandler  
Mapping">  
</bean>
```

```
<!-- http://localhost:8090/06SpringMVC_template_basic/one.do -->  
<bean name="aaa" class="com.controller.OneController" />
```

```
<!-- http://localhost:8090/06SpringMVC_template_basic/fourcontext.do -->  
<bean name="ddd" class="com.controller.FourContext" />
```

다. DefaultAnnotationHandlerMapping

- **@RequestMapping 어노테이션을 컨트롤러 클래스나 메소드에 직접 부여하고 이를 이용하여 맵핑하는 전략이다.**
- URL뿐만 아니라 GET/POST 같은 HTTP메소드, 파라미터와 HTTP 헤더정보까지 맵핑에 활용할 수 있다.

```
@Controller
public class AnnoTestController {

    @RequestMapping("/xxx.do")
    public ModelAndView sayEcho(){

        System.out.println("AnnoTestController >>>>> xxx.do");
        return new ModelAndView("sayEcho");

    }//end sayEcho
```

라. SimpleUrlHandlerMapping

- URL과 컨트롤러의 맵핑 정보를 한곳에 모아놓을 수 있는 핸들러 맵핑 전략이다.

```
<bean id="simpleMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="mappings">
    <props>
      <prop key="a.do">aaa</prop>
      <prop key="b.do">bbb</prop>
      <prop key="/c.do">ccc</prop>
    </props>
  </property>
</bean>
```

```
<bean name="aaa" class="com.controller.OneController" />
<bean name="bbb" class="com.controller.TwoController" />
<bean name="ccc" class="com.controller.ThreeContextController" />
```

○ 기타 공통 설정

- 핸들러 맵핑 전략은 맵핑방식을 매우 세밀하게 제어할 수 있는 다양한 프로퍼티를 제공한다.

가. order

- 핸들러 맵핑은 한 개 이상을 동시에 사용할 수 있다. (권장은 하나로 통일)
- URL 맵핑 정보가 중복되는 경우에 우선순위를 지정할 수 있다.
값이 낮을수록 순위가 높다.

나. defaultHandler

- URL을 맵핑할 대상을 찾지 못했을 경우에 자동으로 default 핸들러를 선택해준다.
(404 방지)

```
<bean id="beanNameMapping"
class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">
    <property name="order" value="4"></property>
</bean>
<bean id="simpleMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
    <property name="mappings">
        <value>
            a.do=aaa
            b.do=bbb
        </value>
    </property>
    <!-- http://localhost:8080/06SpringMVC_XML_basic4_ControllerClassNameHandlerMapping/a.do 요청서
    OneController 실행됨. -->
    <property name="order" value="3"></property>
    <!-- http://localhost:8080/06SpringMVC_XML_basic4_ControllerClassNameHandlerMapping/kk.do 요청서
    TwoController 실행됨.-->
    <property name="defaultHandler" ref="bbb" />
</bean>

<bean name="aaa" class="com.controller.OneController" />
<bean name="bbb" class="com.controller.TwoController" />
<bean name="/a.do" class="com.controller.ThreeContextController" />
```

- 뷰(view)는 MVC 아키텍처에서 모델이 가진 정보를 어떻게 표현해야 하는지에 대한 로직을 갖고 있는 컴포넌트이다.
- 웹 환경에서 뷰가 생성하는 결과물은 일반적으로 브라우저에서 보여지는 HTML이다. 최근에는 클라이언트의 종류가 브라우저 외에도 다양해지고, 클라이언트가 요청하는 결과 포맷도 HTML이 아닌 경우가 점차 증가하고 있어서, 뷰의 기술도 그에 따라 다양하게 발전하고 있다.
- 뷰 정보를 ModelAndView 타입 객체에 담아서 돌려주는 2가지 방법이 제공된다.

가. View 타입의 객체

: InternalResourceView, JstlView, RedirectView, VelocityView, FreeMarkerView, MarshallingView 등.

나. 논리적인 뷰 이름 (일반적인 방법)

: 뷰 이름으로부터 실제 사용할 뷰를 결정해주는 뷰 리졸버(ViewResolver)가 필요하다.

```
@Override
public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response) throws Exception {

    Map<String, String> map = new HashMap<String, String>();
    map.put("aaa", "홍길동");

    View view = new InternalResourceView("/WEB-INF/jsp/spring/one.jsp");

    ModelAndView mav = new ModelAndView(view, map);
    return mav;
}
```

```
@Override
public ModelAndView handleRequest(HttpServletRequest request,
    HttpServletResponse response) throws Exception {

    View view = new RedirectView("one.do");

    ModelAndView mav = new ModelAndView(view);

    return mav;
}
```


○ 용도

: DispatcherServlet이 주어진 요청을 처리할 뷰를 선택하기 위해서 Controller에 의해 반환된 '논리적 뷰 이름'에 기반하여 뷰 선택.

○ 종류

가. InternalResourceViewResolver (기본)

: 자바의 JSP을 사용한 환경에서 사용되는 컴포넌트.

나. ResourceBundleViewResolver

: 리소스 번들에서 뷰 구현체를 찾는다.

다. BeanNameViewResolver

: 논리적 뷰 이름과 동일한 id값을 갖는 <bean>으로 등록된 View 구현체를 찾음.

라. TilesViewResolver

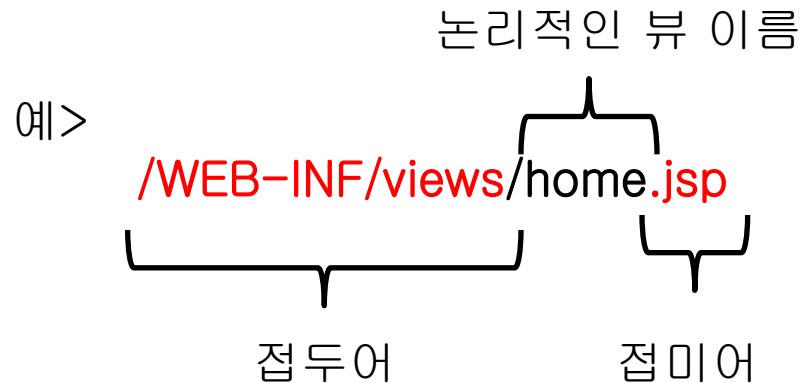
: tiles 템플릿으로 정의된 뷰를 찾는다. 템플릿 이름은 논리적 뷰 이름과 동일하다.

마 . FreeMarkerViewResolver,XmlViewResolver,VelocityViewResolver

...

○ InternalResourceViewResolver

: 접두어와 접미어로 구성된 논리적인 뷰 이름을 이용하여 웹 어플리케이션 내에 위치한 뷰(jsp파일)을 찾아낸다.



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

06Day_MVC5_InternalResourceViewResolver_chul2의 작성

```

3 import org.springframework.stereotype.Controller;
4 import org.springframework.web.bind.annotation.RequestMapping;
5
6 @Controller
7 public class TestController {
8     @RequestMapping("/kkk")
9     public String xx() {
10         System.out.println("xxx() 실행 =====");
11         return "hello"; // /WEB-INF/views/hello.jsp 가

```

```

6 <!-- 삭제 |-->

```

```

8 <!-- Processes application requests -->
9 <servlet>
10     <servlet-name>appServlet</servlet-name>
11     <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-cla
12 <init-param>
13     <param-name>contextConfigLocation</param-name>
14     <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
15 </init-param>
16 <load-on-startup>1</load-on-startup>
17 </servlet>
18
19 <servlet-mapping>
20     <servlet-name>appServlet</servlet-name>
21     <url-pattern>/</url-pattern>
22 </servlet-mapping>
23
24 </web-app>

```

root-context.xml 네임스페이스 복사
servlet-context.xml 수정

```

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>
<!-- Controller 등록 -->
<bean id="xyz" class="com.controller.TestController"></bean>

```

- 뷰와 관련된 DispatcherServlet 전략 중에는 **RequestToViewNameTranslator** 라는 것도 있다. 이 전략은 뷰 이름을 컨트롤러가 넘겨주지 않았을 때 URL을 이용해 자동으로 뷰 이름을 만들어 준다.

이름 규칙을 잘 정해두면 컨트롤러에서 비슷한 패턴의 뷰 이름을 매번 리턴 하지 않아도 된다.

예> /hello 라는 URL을 사용했고 컨트롤러가 뷰 이름을 리턴하지 않았다면 hello로 뷰 이름을 자동으로 생성한다.

/admin/member.do 라면 admin/member가 논리적인 뷰이름이 된다.

* CoC (Convention Over Configuration)

: 코드보다는 관례가 우선한다는 요즘 유행하는 개발 스타일이다.

○ ResourceBundleViewResolver

: 외부 리소스 번들에 각 뷰 이름에 해당하는 뷰클래스와 설정을 저장하고 참조.

: 기본적으로 클래스 패스의 **views.properties** 파일을 사용한다.

```
aaa.(class)=org.springframework.web.servlet.view.InternalResourceView  
aaa.url=/WEB-INF/jsp/spring/one.jsp
```

```
bbb.(class)=org.springframework.web.servlet.view.RedirectView  
bbb.url=one.do
```

컨트롤러가 aaa 또는 bbb이라는 뷰 이름을 돌려주면 ResourceBundleViewResolver는 views.properties 파일에서 뷰 이름으로 시작하는 키를 찾아서 뷰 클래스와 URL정보를 가져와 뷰를 생성한다.

```
ModelAndView mav = new ModelAndView("aaa" , map);
```

```
ModelAndView mav = new ModelAndView("bbb");
```

○ 용도

: 실제 비즈니스 로직을 처리하는 Controller 컴포넌트를 의미한다.

○ 구현 방법 2가지

가. Spring API 이용

: Controller 인터페이스 이용(`handleRequest` 메소드 구현)

나. POJO 클래스 및 어노테이션 이용

: `@Controller`

: `@RequestMapping`

: `@PathVariable`

: `@ModelAttribute`

: `@SessionAttribute`

○ 어노테이션 활용 (@MVC 라고 부른다.)

: 스프링 3.X부터는 @Controller 어노테이션을 이용해서 컨트롤러 클래스를 구현하도록 권장한다.

: @MVC의 가장 큰 특징은 핸들러 매핑과 핸들러 어댑터의 대상이 객체가 아니라 메소드라는 점이다. (@RequestMapping 사용)

: @MVC의 핸들러 매핑을 위해서는 DefaultAnnotationHandlerMapping이 필요하다. (디폴트 매핑임)

만약 다른 핸들러 매핑을 등록했을 경우에는 디폴트 핸들러 매핑이 적용되지 않기 때문에 같이 등록해야 된다.

: @RequestMapping은 클래스 레벨 및 메소드 레벨 모두 사용 가능하며, 2가지 정보를 취합하여 최종 매핑 정보를 생성한다.

- 가. @Controller 어노테이션을 컨트롤러 클래스(POJO)에 지정한다.
- 나. 클라이언트 요청을 처리할 메소드에 @RequestMapping 어노테이션을 설정한다.
- 다. 설정 파일에 컨트롤러 클래스를 빈으로 등록한다.

- 40 -

○ @RequestMapping 어노테이션

: 사용자가 요청한 URL 정보를 이용하여, 실제 처리되는 메소드를 선택하는 방법.
가. URL 패턴 이용.

- 기본 속성인 value는 스트링 배열타입으로 URL 패턴을 지정할 수 있다.
ANT 스타일 지정 가능하다.
- {}을 사용하는 URI 템플릿 형식 사용 가능하다

예>

```
@RequestMapping ( value = "/hello" )  
@RequestMapping ( "/hello" )  
@RequestMapping ( "/main*" )  
@RequestMapping ( "/view.*" )  
@RequestMapping ( "/admin/**/user" )  
@RequestMapping ( "/user/{userid}" )  
@RequestMapping ( { "/hello", "/hi" } )
```

- 디폴트 접미어 패턴이 적용된다. (default suffix pattern)

@RequestMapping ("/hello") 는 다음과 동일하다.

@RequestMapping ("/hello", "/hello/", "/hello.*")

나. HTTP 요청 메소드 이용.

- GET, POST, HEAD, PUT, DELETE 같은 요청 메소드 이용.
- 따라서 같은 URL이라고 하더라도 요청 메소드에 따라서 다른 메소드에 매핑이 가능하다. 배열 타입이므로 여러 개 지정 가능하다.

예>

```
@RequestMapping ( value = “/login.do” , method=RequestMethod.GET)
```

```
@RequestMapping ( value = “/login.do” , method=RequestMethod.POST)
```

- 용도:

폼을 처리할 때 동일한 URL에 대해서 GET과 POST로 요청을 구분해서 각각 폼을 띄우는 용도와 **폼을 서브밋(submit)하는 용도로 구분해서 사용한다.**

다. 요청 파라미터 이용.

- 요청 파라미터 값을 비교해서 맵핑.
- 파라미터는 key=value형식으로 지정한다.
배열형식으로 지정 가능하다.

예>

```
@RequestMapping ( value = “/login.do” , params=“type=admin”)  
  
@RequestMapping ( value = “/login.do” , params=“type=member”)  
  
@RequestMapping ( value = “/login.do” , params=“!type”)  
  
@RequestMapping ( value = “/login.do” )
```

○ 클래스 레벨과 메소드 레벨 결합

: 메소드에 설정된 URL은 클래스에 설정한 URL의 서브 URL이 된다.

예> /user/edit , /user/add , /user/delete 요청이 있는 경우

```
@Controller
@RequestMapping("/user")
public class AnnoTestController {
    @RequestMapping( value="/add" ) public void add(){}
    @RequestMapping( value="/edit" ) public void edit(){}
    @RequestMapping( value="/delete" ) public void delete(){}
}
```

```
@Controller
@RequestMapping("/user/**")
public class AnnoTestController {

    @RequestMapping( value="/add" ) public void add(){}
}
```

```
@Controller
@RequestMapping("/user/add")
public class AnnoTestController {
    @RequestMapping( methods=RequestMethod.GET ) public void form(){}
    @RequestMapping( methods=RequestMethod.POST ) public void submit(){}
}
```

```
@Controller
@RequestMapping("/user/*")
public class AnnoTestController {
    @RequestMapping public void add(){} => /user/add 로 맵핑
    @RequestMapping public void edit(){} => /user/edit 로 맵핑
}
```

- 클래스 레벨에서는 공통 맵핑 조건을 정의하고 각 메소드에서 세분화된 맵핑조건을 추가한다는 개념만 지키면 어떤 식의 조합도 가능하다.
- 클래스 레벨에서는 URL뿐만 아니라 methods 또는 params 까지도 정의 가능.
- @RequestMapping 정보는 상속되며, 재정의 가능하다. (인터페이스 이용)

○ 컨트롤러 메소드의 파라미터 타입

: 다음은 @RequestMapping 어노테이션이 적용된 메소드의 인자로 설정 가능한 타입이다.

파라미터 타입	설명
HttpServletRequest, HttpServletResponse, HttpSession	서블릿 API
InputStream,Reader	요청 콘텐츠 읽기
OutputStream,Writer	응답 콘텐츠 생성
@PathVariable	URI 템플릿 사용시
@RequestParam	HTTP 요청 파라미터 맵핑
@RequestHeader	HTTP 요청 헤더 맵핑
@CookieValue	HTTP 쿠키 맵핑
Map,Model	뷰에 전달할 모델 데이터 저장
컨맨드 객체	HTTP 요청 파라미터를 저장할 객체. 기본적으로 클래스명을 모델명으로 사용. @ModelAttribute 사용하여 모델명 설정 가능.
SessionStatus	@SessionAttribute 를 명시한 session속성을 제거할 때 사용.

가. @RequestParam

: HTTP의 요청 파라미터를 맵핑하는 어노테이션.

: 요청 파라미터는 필수사항이다. (required=false 지정 가능)

```
<form action="anno.do" method="get">
  name<input type="text" name="username" >
  age<input type="text" name="userage">
  <input type="submit" value="전송">
</form>
```

```
@RequestMapping(value="/anno.do")
public String get( @RequestParam("username") String name ,
                  int userage ){} 
```

정확하게 일치하면 생략가능, 자동형변환

```
public String get( String username , int userage ){}
public String get( @RequestParam Map<String,String> params ){}
public String get(@RequestParam( required=false , defaultValue="홍길동")
                  String username , int userage ){} 
```

나. @ModelAttribute

- @ModelAttribute는 모델 맵에 담겨서 뷰에 전달되는 모델 객체의 한 가지라고 볼 수 있다.

기본적으로 모든 @ModelAttribute는 별도의 설정 없이도 자동으로 뷰에 전달된다.

모델의 이름은 기본적으로 **첫글자 소문자인 클래스명**.

- 메소드 파라미터 및 메소드 레벨에 적용할 수 있다. 또한 생략이 가능하다.

* @ModelAttribute가 붙은 파라미터 처리

```
<form action="anno.do" method="post">
  name<input type="text" name="username" >
  age<input type="text" name="userage">
  <input type="submit" value="전송">
</form>
```

```
public class AnnoCommand {
```

```
String username;
```

```
int userage;
```

```
//setter 메소드
```

```
// 기본 생성자
```

```
@RequestMapping( value="/anno.do", method=RequestMethod.POST)
public String post( @ModelAttribute AnnoCommand command){ }
```

`${annoCommand.username}` 형식으로 jsp에서 출력가능.

- 자동으로 저장되는 모델의 이름은 @ModelAttribute 어노테이션을 이용하여 변경 가능하다.

```
@RequestMapping( value="/anno.do", method=RequestMethod.POST)
public String post( @ModelAttribute("command") AnnoCommand
command){

    System.out.println("POST " + command.getUsername()+"Wt" +
        command.getUserage());
    return "hello";
}
```

`${command.username}` } 형식으로 jsp에서 출력가능.

- @ModelAttribute가 붙은 파라미터를 처리할 때는 @RequestParam과 달리 검증작업을 할 수 있다. (타입 변환 실패등)

만약 타입변환 실패로 인한 예외가 발생되도 HTTP 400 응답코드가 바로 클라이언트에게 전달되지 않는다. 작업은 계속 진행되며, 단지 타입 변환중에 발생한 예외가 **BindException** 타입의 객체에 담겨서 컨트롤러에 전달된다.

```
@RequestMapping( value="/anno.do", method=RequestMethod.POST)
public String post( @ModelAttribute("command") AnnoCommand
command , BindingResult result){

    if(result.hasErrors()){
        return "redirect:index.do";
    }
    return "get";
}
```

* @ModelAttribute가 붙은 메소드 처리

- 뷰에서 참고정보로 사용되는 모델 객체를 생성하는 용도로 메소드에 사용함.

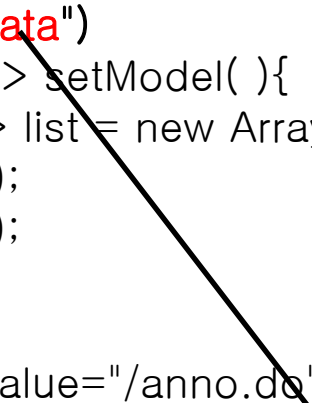
이를 이용하면 모델 객체 생성을 전담하는 메소드를 작성할 수 있다.

- @ModelAttribute가 붙은 메소드는 컨트롤러 클래스안에 정의하지만 컨트롤러 기능을 담당하지 않는다. 따라서 @RequestMapping 같이 사용 안함.

- @ModelAttribute 메소드가 생성하는 객체는 클래스내의 다른 컨트롤러 메소드의 모델에 자동으로 추가된다.

```
@ModelAttribute("myData")
public ArrayList<String> setModel( ){
    ArrayList<String> list = new ArrayList<String>();
    list.add("홍길동");
    list.add("이순신");
    return list;
}

@RequestMapping( value="/anno.do", method=RequestMethod.GET)
public String get(@ModelAttribute("myData") ArrayList<String> list ){}
```



```
<%
    ArrayList<String> value = (ArrayList<String>)request.getAttribute("myData");
%>
```

* @ModelAttribute가 붙은 메소드 특징

가. 모든 @RequestMapping 메소드 보다 먼저 실행된다.

나. 컨트롤러안에서 여러 메소드에 지정 가능하다.

다. @PathVariable같은 파라미터를 인자로 가질수 있다.

다. Model , Map, ModelMap 객체

: 뷰에 데이터 전달시 사용 가능한 객체로서 **request** 스코프에 자동 저장된다.

```
@RequestMapping(value="/anno.do")
public String get(Model model ){

    model.addAttribute("key", "홍길동");
    return "hello";
}
```

```
<body>
<%
    String value = (String)request.getAttribute("key");

%>
Model 데이터값: <%= value %><br>
${key }
</body>
</html>
```

```
@RequestMapping( value="/aaa.do")
public String aaa (Model model ){    //자동으로 Model attribute에 추가된다.

    Student stu = new Student("홍길동", 20 );
    model.addAttribute(stu); // Add as "student" //소문자 클래스명

    return "aaa";
}
```

```
@RequestMapping( value="/bbb.do")
public String bbb(Model model ){

    List<Student> xx = new ArrayList<>();
    xx.add(new Student("홍길동", 20 ));
    xx.add(new Student("이순신", 18 ));
    model.addAttribute(xx); // Add as "studentList" //소문자클래스List명

    return "bbb";
}
```

라. @RequestHeader

: HTTP의 요청 헤더값을 메소드의 파라미터로 전달 받을 수 있다.

```
@RequestMapping( method=RequestMethod.GET)
public String get(@RequestHeader("Accept-Language") String xxx ,
                  HttpServletRequest req){

    System.out.println("Accept-Language > " + xxx);
    Enumeration<String> enu =req.getHeaderNames();
    while(enu.hasMoreElements()){
        String key = enu.nextElement();
        System.out.println(key+" = " + req.getHeader(key));
    }
    return "hello";
}
```

마. @CookieValue

: HTTP의 요청과 함께 전달된 쿠키값을 메소드 파라미터에 저장. 필수 속성이다.

```
@RequestMapping( method=RequestMethod.GET)
public String get( @CookieValue(value="auth",required=false,defaultValue="NONE")
                  String xxx , HttpServletRequest req){

    System.out.println("쿠키값: " + xxx);
    return "hello";
}
```


바. HttpSession

: 뷰에 데이터 전달시 사용 가능한 객체로서 session스코프에 저장된다.

```
@RequestMapping( method=RequestMethod.GET)
public String get(HttpSession sess){

    sess.setAttribute("login", "1234");
    return "hello";

}
```

```
<body>
<%
    String value = (String)session.getAttribute("login");
%>
<%= value %><br>
${sessionScope.login }
</body>
```

사. @SessionAttributes , SessionStatus

: 모델 객체를 세션에 저장했다가 다음 페이지에서 사용 가능.

: SessionStatus의 setComplete() 메소드로 세션 삭제. 일반 세션은 삭제 안됨.

```
@Controller
@SessionAttributes( value="myCommand" ) // session 에 저장할 key값 설정.
public class AnnoTestController {

    public String second( @ModelAttribute( value="myCommand") AnnoCommand
command){} // AnnoCommand는 세션에 myCommand 이름으로 저장됨.

    public String invalidate(AnnoCommand command , SessionStatus status ){

        status.setComplete(); //세션 제거
    }
}
```

아. @Value, @PathVariable , @RequestBody

○ 컨트롤러 메소드의 리턴 타입

: @MVC 컨트롤러 메소드에는 파라미터뿐만 아니라 리턴 타입도 다양하게 사용할 수 있다.

: 다음은 @RequestMapping 어노테이션이 적용된 메소드의 사용 가능한 리턴 타입이다.

파라미터 타입	설명
ModelAndView	뷰 정보 및 모델 정보를 담고 있는 객체.
Model , Map	뷰에 전달할 객체 정보를 담고 있는 객체. 이때 뷰 이름은 요청 URL로부터 결정된다.
String	뷰 이름을 리턴한다.
View	View 객체를 직접 리턴한다.
void	메소드가 직접 응답을 처리하거나 또는 뷰 이름은 요청 URL로부터 결정된다.
@ResponseBody	리턴되는 결과값은 응답 메시지 본문으로서, JSON,XML 응답처리시 주로 사용된다.

```
@RequestMapping("/zzz.do")
public String callEcho(Model model){
    model.addAttribute("xyz", "홍길동");
    return "sayEcho"; // sayEcho.jsp
}
```

@RequestMapping("/aaa.do") //aaa.jsp 가 받는다.

@ModelAttribute("key") // aaa.jsp에 모델을 전달한다. addAttribute("key", kkk)와 동일하다.

```
public ArrayList<String> say(){

    ArrayList<String> kkk = new ArrayList<>();
    kkk.add("1");
    kkk.add("2");

    return kkk;
}
```

@RequestMapping("/sess.do") // sess.jsp

```
public void xxx(){

}
```

가. @ResponseBody

- @ResponseBody가 메소드 레벨에 부여되면 메소드가 리턴하는 객체는 뷰를 통해 결과를 만들어내는 **모델로 사용되는 대신, 메시지 컨버터를 통해 바로 HTTP 응답의 메시지 본문으로** 전환된다.
- 근본적으로 @RequestBody, @ResponseBody는 XML 이나 JSON과 같은 메시지 기반의 커뮤니케이션을 위해 사용된다.

```
@RequestMapping("/hello")
@ResponseBody
public String hello(){

    return "<html><body>Hello</body></html>";
}
```

상위의 코드에서 @ResponseBody가 없다면, String타입의 리턴값은 뷰이름으로 인식될 것이다. 하지만 @ResponseBody가 있기 때문에 HttpServletResponse의 출력 스트림으로 동작시킨다. **<mvc:annotation-driven/> 설정 필요.**

```
○<!-- JSON/Ajax start -->
○<dependency>
○  <groupId>com.fasterxml.jackson.core</groupId>
○  <artifactId>jackson-databind</artifactId>
○  <version>2.8.8</version>
○</dependency>
○<!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
○<dependency>
○  <groupId>com.fasterxml.jackson.core</groupId>
○  <artifactId>jackson-core</artifactId>
○  <version>2.8.8</version>
○</dependency>
```

- 스프링의 HandlerMapping은 HandlerInterceptor를 이용해서 Controller가 요청을 처리하기전과 후에 알맞은 기능을 수행하도록 지원한다.
조건에 따라 Controller에 요청을 전달하지 않고 싶거나 Controller가 요청을 처리한 후에 ModelAndView객체를 조작하고 싶은 경우에 HandlerInterceptor를 사용하면 된다.

○ 3가지 메소드

가. boolean preHandle(request, response, object)

:클라이언트 요청을 Controller에 전달하기 전에 호출.

: 체인인 경우에 false리턴하면 남아있는 인터셉터 skip하고 컨트롤러 호출함.

나. void postHandle(request, response, object, mav)

: 클라이언트 요청을 처리한 뒤에 호출.

: Controller 실행중에 예외발생시 postHandler 메소드 실행 안됨.

다. void afterCompletion(request, response, object, ex)

: 뷰를 통해서 클라이언트에 응답을 전송한 뒤에 호출.

- HandlerInterceptor 인터페이스 및 HandlerInterceptorAdapter 클래스를 사용하여 구현한다.

```
public class MyHandlerInterceptor extends HandlerInterceptorAdapter {

    //Controller가 실행되기 전에.
    public boolean preHandle(javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response, java.lang.Object handler) throws java.lang.Exception{
        System.out.println("MyHandlerInterceptor.preHandle >>>>>>>>>" + handler);
        return true;
    }

    //Controller가 실행된 후에.
    public void postHandle(javax.servlet.http.HttpServletRequest request, javax.servlet.http.HttpServletResponse
        response, java.lang.Object handler, org.springframework.web.servlet.ModelAndView modelAndView) throws
        java.lang.Exception{
        System.out.println("MyHandlerInterceptor.postHandle >>>>>>>>>");
    }

    //View가 실행된 후에.
    public void afterCompletion(javax.servlet.http.HttpServletRequest request,
        javax.servlet.http.HttpServletResponse response, java.lang.Object handler, java.lang.Exception ex) throws
        java.lang.Exception{
        System.out.println("MyHandlerInterceptor.afterCompletion >>>>>>>>>");
    }

}
```


- HandlerInterceptor 를 구현한 뒤에는 반드시 xml에 등록해야 된다.

```
<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.basic.app" />

<interceptors>
  <interceptor>
    <!-- <mapping path="/hello.do" /> →
    <mapping path="/**" />
    <beans:bean class="com.basic.app.MyHandlerInterceptor" />
  </interceptor>
</interceptors>
```

*Ant 경로 패턴

- ? : 1개의 문자와 매칭
- * : 0개 이상의 문자와 매칭
- ** : 0개 이상의 디렉토리와 매칭

- Controller클래스의 @RequestMapping메소드는 모든 타입의 예외를 발생 시킬 수 있다. 예외가 발생되면 500 에러코드와 함께 서블릿 컨테이너가 제공한 에러 페이지가 출력된다. (사용자가 이해하기 어려움)
- 발생한 예외 타입에 따라서 스프링MVC와 연동된 뷰를 이용해서 에러 페이지를 출력하고자 한다면, 스프링이 제공하는 HandlerExceptionResolver 인터페이스를 사용하면 된다.

○ HandlerExceptionResolver 인터페이스 구현 클래스

가. AnnotationMethodHandlerExceptionResolver

- : @ExceptionHandler 어노테이션이 적용된 메소드를 이용해서 예외처리 한다.
- : 디폴트 핸들러 예외 리졸버이다.

나. DefaultHandlerExceptionResolver

다. SimpleMappingExceptionResolver

- : 예외타입 별로 뷰 이름을 지정할 때 사용한다.

○ 가. @ExceptionHandler 어노테이션을 이용한 예외처리 방법

```
@RequestMapping(value = "/aaa", method = RequestMethod.GET)
public String home(Locale locale, Model model) {

    if(true)
        throw new NullPointerException("예외발생");
    return "home";
}

@ExceptionHandler( {NullPointerException.class, ArrayIndexOutOfBoundsException.class})
public String handleException(){
    System.out.println("NullPointerException 발생");
    return "handle";
}

@RequestMapping("/bbb")
public String bbb() {

    int n = 4/0;
    return "index";
}

@ExceptionHandler
public String handleException2(ArithmeticException e){
    System.out.println(e);
    return "handle2";
}
```

○ 나. SimpleMappingExceptionHandler 이용한 예외처리 방법

```
@RequestMapping(value = "/aaa", method = RequestMethod.GET)
public String home(Locale locale, Model model) {
    if(true)
        throw new NullPointerException("예외발생");
    return "home";
}
@RequestMapping("/bbb")
public String bbb() {

    int n = 4/0;
    return "index";
}
```

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

<beans:bean class="org.springframework.web.servlet.handler.SimpleMappingExceptionHandler">
    <beans:property name="exceptionMappings">
        <beans:props>
            <beans:prop key="java.lang.NullPointerException">handle</beans:prop>
            <beans:prop key="java.lang.ArithmeticException">handle2</beans:prop>
        </beans:props>
    </beans:property>
</beans:bean>
```

○ 웹 어플리케이션 스코프 종류

가. request 스코프 (HttpServletRequest)

나. session 스코프 (HttpSession)

다. application 스코프 (ServletContext)

라. flash 스코프

- 서블릿 기반 웹 어플리케이션에서 특정 서블릿이 다른 서블릿을 실행하려면 일반적으로 포워드나 리다이렉트를 한다. request 스코프는 리다이렉트하면 초기화되지만 flash 스코프는 리다이렉트해도 상태가 유지된다. 그러므로 리다이렉트하는 곳에서 flash스코프에 저장한 객체를 리다이렉트되는 곳에서 가져올 수 있다.
- <mvc:annotation-driven /> 설정 필요.

○ RedirectAttributes 이용

- addFlashAttribute(key,value) 메소드 이용.

```
@RequestMapping("/flash.do")
public String ccc(RedirectAttributes xxx , Model model) {
    xxx.addFlashAttribute("flash", "flash");
    model.addAttribute("xxx", "rrr");
    return "redirect:ccc.do";
}
@RequestMapping("/ccc.do")
public String flash() {
    return "flash";
}
```

```
<body>
<%
    String data = (String)request.getAttribute("flash");
    String data2 = (String)request.getAttribute("xxx");
%>
flash<%= data%><br>
<%= data2%>
</body>
</html>
```

- 스프링 3.0에서 대거 도입된 @MVC 관련 기능을 제대로 활용하려면 디폴트로 등록되는 AnnotationMethodHandlerAdapter의 설정으로는 부족한다.(추가작업필요)
- 스프링은 최신 @MVC기능을 손쉽게 등록할 수 있게 해주는 mvc 스키마의 전용태그를 제공한다.

가. <mvc:annotation-driven />

- @MVC방식의 컨트롤러를 사용할 때 필요한 DispatcherServlet 전략빈을 자동으로 등록해준다.

: DefaultAnnotationHandlerMapping

: AnnotationMethodHandlerAdapter

: ConfigurableWebBindingInitializer

: 메시지 컨버터

: <spring:eval>을 위한 컨버전 서비스 노출용 인터셉터

: validator

: conversion-service 등

나. <mvc:interceptors >

- 기존 Interceptors 프로퍼티를 이용한 방법보다 좋은 장점 2가지;

- a. 모든 핸들러 매핑에 일괄 적용되는 인터셉터를 한 번에 설정할 수 있다.
- b. URL 패턴을 지정할 수 있다.

```
<mvc:interceptors>
<mvc:interceptor>
  <!-- <mvc:mapping path="/**" /> -->
  <mvc:mapping path="/xxx.do" />
<beans:bean class="com.basic.app.MyHandlerInterceptor" />
</mvc:interceptor>
</imvc:nterceptors>
```

다. <mvc:view-controller>

- 모델도 없고 컨트롤러 로직도 없이 요청에 대해 특정 뷰를 지정만 하는 경우.

```
<mvc:view-controller path="/hello" view-name="index" />
```


- 스프링의 MVC 아키텍처
- 스프링 MVC 핵심 컴포넌트
- DispatcherServlet 기능
- HandlerMapping 기능
- ViewResolver 기능.
- 어노테이션 기반의 Controller 구현 방법
 - @Controller , @RequestMapping
 - @RequestParam , @ModelAttribute 등
- 스프링 파일 업로드 및 다운로드 방법



Thank you
