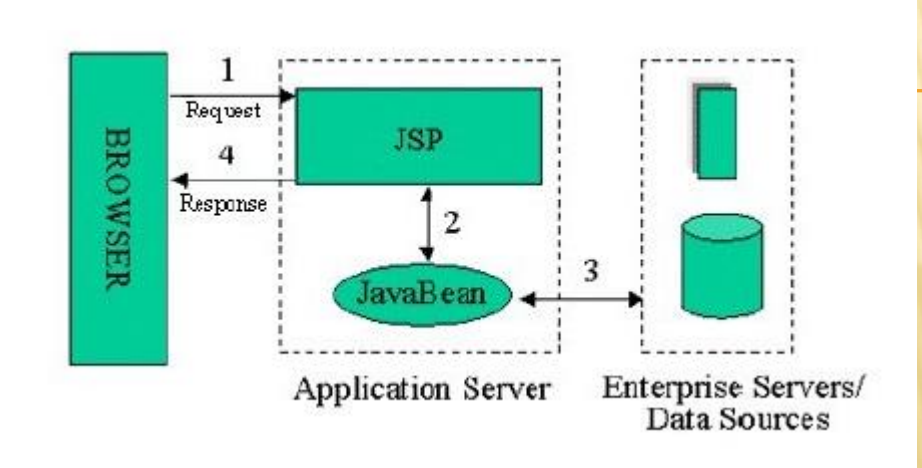
# SPRING MVC

백성애

# MVC란?

- Model -View- Controller의 약자
- 사용자 인터페이스(UI)로부터 비즈니스 로직을 분리하여 어플리케이션의 시각적 요소나 실행되는 비즈니스 로직을 서로 영향 없이 쉽게 고칠 수 있도록 하는 응용프레임워크의 기반이되는 구조적 패턴
- Model: 어플리케이션의 정보(데이터)를 나타냄
  - \+ (DTO/VO, DAO)
- View : User Interface (JSP)
- Controller: 데이터와 비즈니스 로직 사이의 상호동작을 관리 (Servlet)

# MODEL1방식



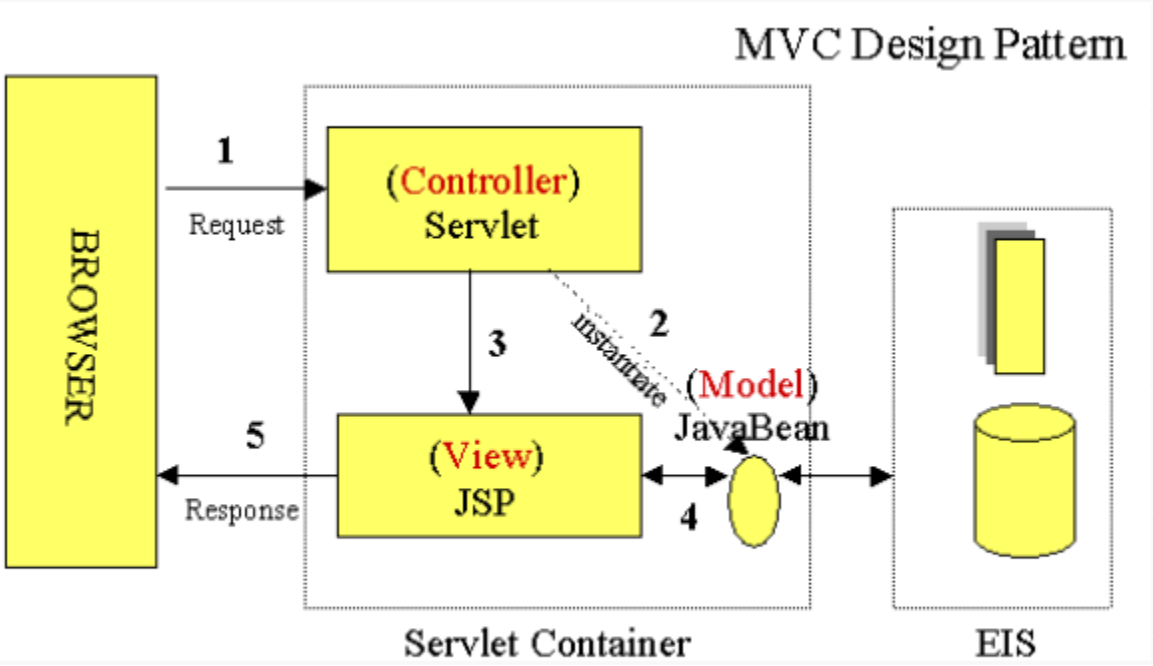

장점

1. 개발 속도가 빠르다.
2. 개발자의 높은 스킬을 요구하지 않음

단점

1. JSP페이지에서 프리젠테이션 로직과 비즈니스 로직을 모두 포함하기 때문에 JSP페이지가 복잡해 짐
2. 프리젠테이션 로직과 비즈니스 로직이 혼재되어 있기 때문에 개발자와 디자이너의 분리된 작업이 어려워진다.
3. JSP페이지의 코드가 복잡해 짐으로 인해 유지보수 하기 어려워진다

# MODEL2 방식



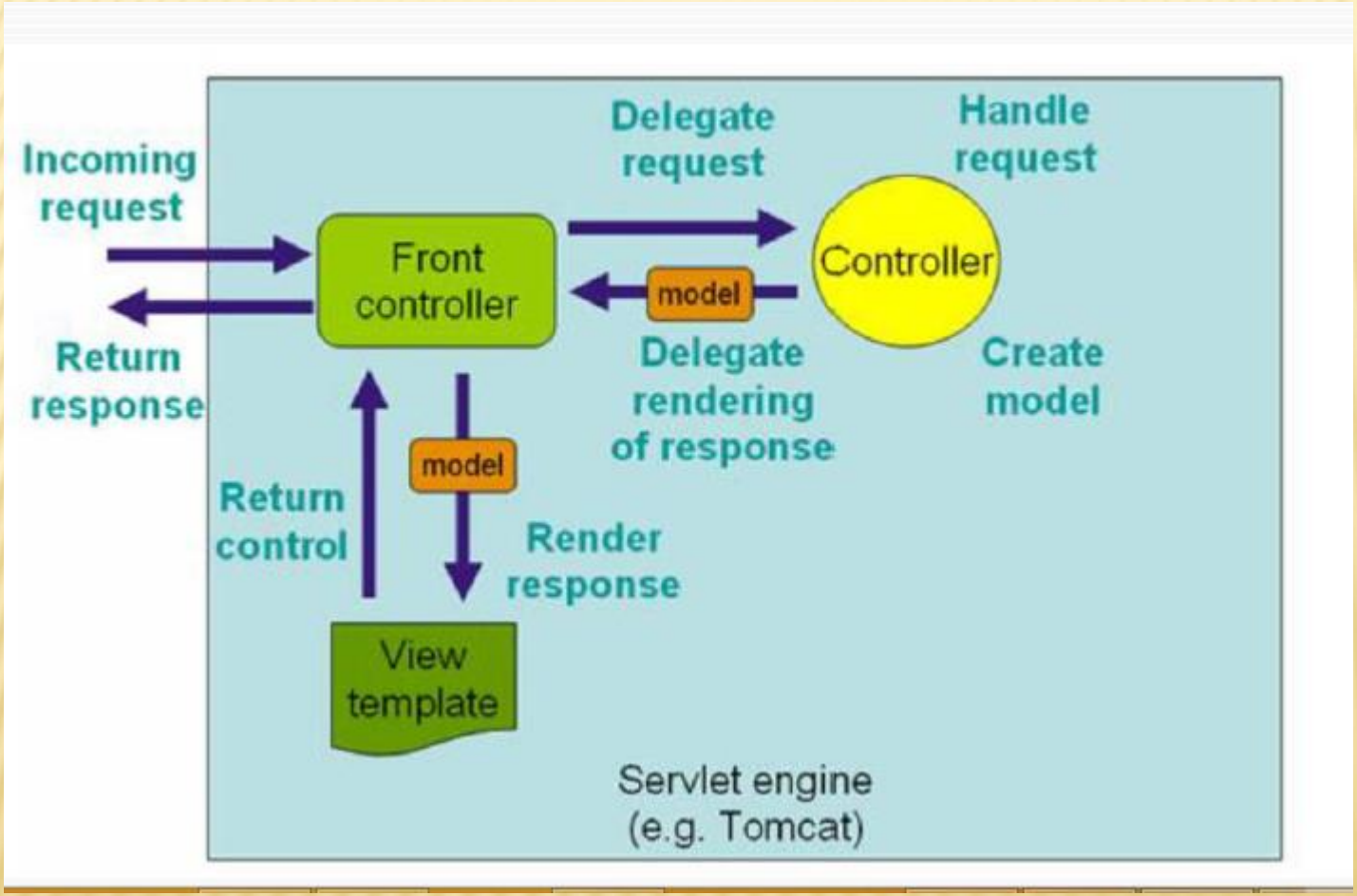

Controller – 데이터의 처리와 화면의 분기를 담당

View – 화면상의 처리

Model – View에서 필요한 데이터

모든 request는 Controller 에서 받는다.

# SPRING MVC


Incoming request
Return response
Front controller
Delegate request
Handle request
Controller
model
Delegate rendering of response
Create model
model
Return control
Render response
View template
Servlet engine (e.g. Tomcat)

# Spring MVC에서 개발자가 하는 일

| 스프링 MVC 가 처리해 주는 작업 | 개발자가 직접 해야 하는 작업 |
| --- | --- |
| URI 를 분석해서 적절한 컨트롤러를 찾는 작업<br>컨트롤러에 필요한 메소드를 호출하는 작업<br>컨트롤러의 결과 데이터를 뷰로 전달하는 작업<br>적절한 뷰를 찾는 작업 | 특정 URI 에 동작하는 컨트롤러를 설계하는 작업<br>서비스 객체의 생성<br>DAO 객체의 생성<br>컨트롤러 내에 원하는 결과를 메소드로 설계<br>뷰에서 전달받은 데이터의 출력 |

# Spring MVC로 인해 편리해 지는 것들

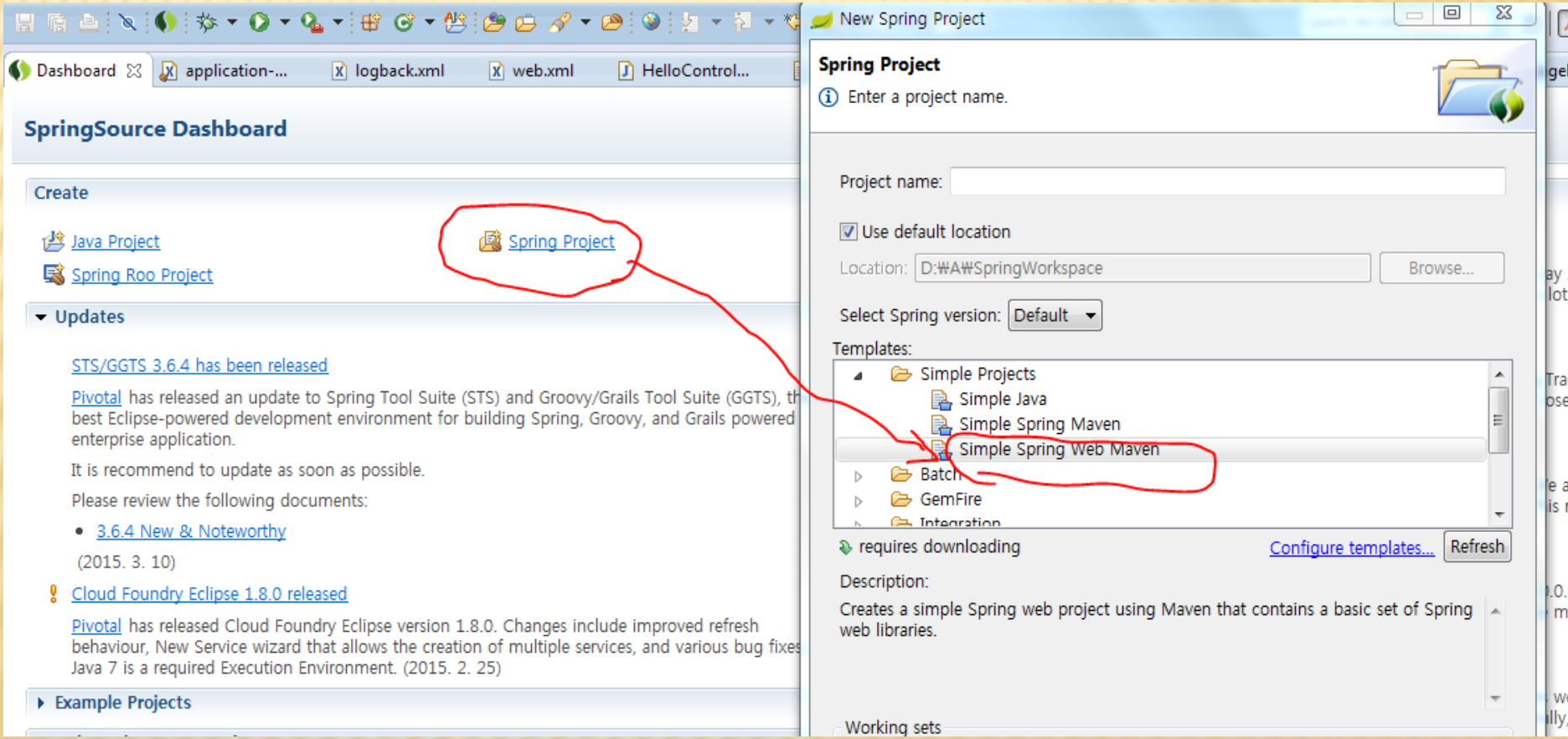
- 파라미터의 수집을 간편히
- 애노테이션 설정을 통한 **URI**설정
- 로직의 집중(모듈화)
- 테스트의 제공

# 기존 MVC와 다른 점들

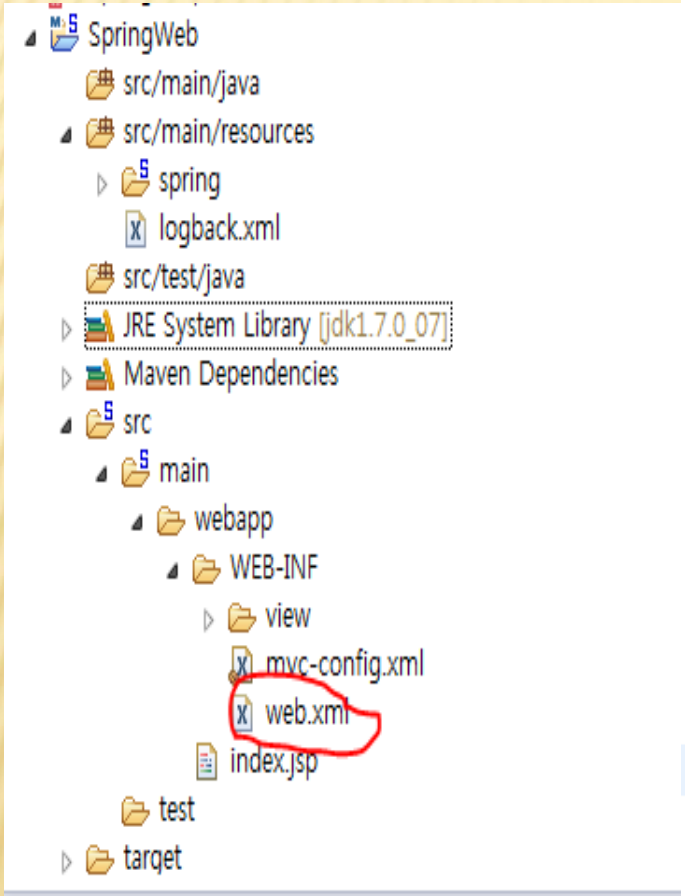
- 상속이나 인터페이스의 제약에 자유롭다.
- 파라미터와 리턴타입의 자유도

# SPRING PROJECT생성

- Spring Dashboard에서 Spring Project를 생성하고 프로젝트명을 SpringWeb으로 주자.



# WEB.XML에 SPRING MVC 설정 수정




```
<!--
    - Servlet that dispatches request to registered handlers (Controller implementatio
-->
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/mvc-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
    <servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

# WEB.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">

<display-name>SpringWeb</display-name>

<!-- - Location of the XML file that defines the root application context.
- Applied by ContextLoaderListener. -->
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value>classpath:spring/application-config.xml</param-value>
</context-param>
```

# WEB.XML

```
<listener>
<listener-class>org.springframework.web.context.ContextLoaderListener</listener-
    class></listener>
<!– - Servlet that dispatches request to registered handlers (Controller
implementations). DispatcherServlet은 내부적으로 스프링 컨테이너를
생성하는데 contextConfigLocation초기화  파라미터를 이용해서 컨테이너를 생성할 때
    사용할 설정 파일을 지정한다. 이 예제에서는 mvc-config.xml 파일을 스프링 설정 파
    일로 사용한다. –>
<servlet>
<servlet-name>dispatcherServlet</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
<init-param>
<param-name>contextConfigLocation</param-name>
<param-value>/WEB-INF/mvc-config.xml</param-value>
</init-param>
<load-on-startup>1</load-on-startup>
</servlet>
```

# WEB.XML

```
<servlet-mapping>
<servlet-name>dispatcherServlet</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>

<!-- ===POST방식의 한글 처리를 위한 인코딩 필터=== -->
<filter>
<description></description>
<display-name>SpringEncodeFilter</display-name>
<filter-name>SpringEncodeFilter</filter-name>
<filter-
    class>org.springframework.web.filter.CharacterEncodingFilter</filter-
    class>
```

# WEB.XML

```
<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>SpringEncodeFilter</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping>
<!– ========================================== –>

</web-app>
```

# WEB.XML

-DispatcherServlet은 내부적으로 스프링 컨테이너를 생성하는데 contextConfigLocation초기화 파라미터를 이용해서 컨테이너를 생성할 때 사용할 설정 파일을 지정한다.

이 예제에서는 mvc-config.xml 파일을 스프링 설정 파일로 사용한다.

dispatcherServlet에 대한 매핑을 *.do 로 지정한다.

이는 웹브라우저 요청 중  확장자가 do로 끝나는 모든 요청을

dispatcherServlet이 처리하게 된다.

또한 POST방식일 때 한 글 처리를 위해 인코딩 필터를 설정하였다. 요청 파라미터를 UTF-8로 처리하게 된다.

➔수업 중 수정된 web.xml 파일 참조

# MVC–CONFIG.XML

```
<?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:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
xsi:schemaLocation="http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc.xsd
http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context.xsd">
  <!– Uncomment and your base-package here:
      <context:component-scan    base-package="org.springframework.samples.web"/>  –->
  <mvc:annotation-driven />
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <!– Example: a logical view name of 'showMessage' is mapped to '/WEB-
    INF/jsp/showMessage.jsp' –>
      <property name="prefix" value="/WEB-INF/view/"/>
      <property name="suffix" value=".jsp"/>
</bean>
</beans>
```

# 컨트롤러 구현

```
package my.day1;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloController {
//hello.do로 요청하면 hello()메소드가 호출됨.
@RequestMapping("/hello.do")
public String hello(Model model){
   model.addAttribute("greeting","안녕~~");
   model.addAttribute("name","Mr. Hong~~");
return "helloView";//view이름으로 사용
}
}
```

# 스프링 MVC설정 [MVC-CONFIG.XML]

- 스프링 mvc를 설정하려면 최소한 다음 구성 요소를 빈 객체로 등록해주어야 한다.
- HandlerMapping 구현 객체
- HandlerAdapter구현 객체
- ViewResolver구현 객체
- 이 중 HandlerMapping/ HandlerAdapter 는
- <mvc:annotation-driven /> 태그를 이용하면 설정이 끝난다.

# 스프링 MVC설정 [MVC-CONFIG.XML]

- mvc:annotaion driven 태그는 RequestMapingHandlerMapping/RequestMappingHandlerAdapter 클래스를 빈으로 등록해준다.

- 이 두 클래스는 Controller어노테이션이 적용된 클래스를 컨트롤러로 사용할 수 있도록 해준다.

# 스프링 MVC설정 [MVC-CONFIG.XML]

- 또한 <context:component-scan base-package="*my.day1*"/> *설정을 추가한다.*
- <context:componet-scan>은 어노테이션을 통해 빈을 생성할 수 있게 해준다.
- 빈을 생성할 기본 패키지를 my.day1 패키지로 설정한다.

# MVC-CONFIG.XML에 설정 추가

SpringWeb/p... | web.xml | web.xml | HelloControl... | showMessage.jsp | helloView.jsp | *mvc-config.xml | SpringWeb/p... | Welcome

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">
    <!-- Uncomment and your base-package here: -->
    <context:component-scan base-package="my.day1"/>
    <!-- componet scan : 어노테이션을 통해 빈을 생성할 수 있게 해준다. -->

    <mvc:annotation-driven />
    <!-- mvc:annotaion driven 태그는 RequestMapingHandlerMapping/RequestMappingHandlerAdapter 클래스
    를 빈으로 등록해준다. 이 두 클래스는 Controller어노케이션이 적용된 클래스를 컨트롤러로 사용할 수 있도록
    해준다.  -->
    <bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <!-- Example: a logical view name of 'showMessage' is mapped to '/WEB-INF/jsp/showMessage.jsp' -->
        <property name="prefix" value="/WEB-INF/view/" />
        <property name="suffix" value=".jsp" />
    </bean></beans>
```

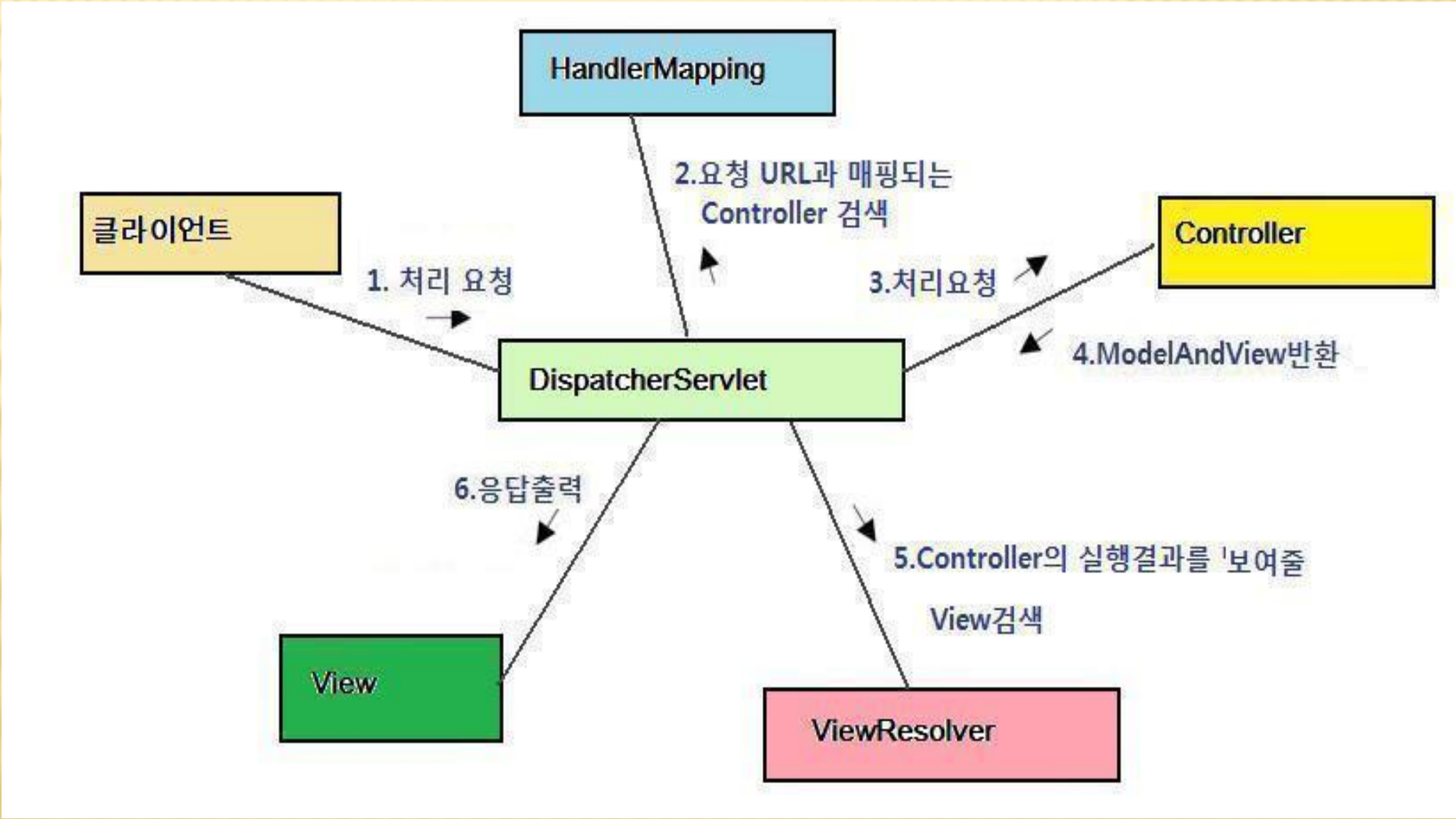
# 스프링 MVC설정 [MVC-CONFIG.XML]

- 위 설정에서 *InternalResourceViewResolver* 는 JSP를 이용해서 뷰를 생성할 때 사용되는 ViewResolver 구현체다.
- 컨트롤러에서 반환한 뷰 이름(“helloView”)을 prefix 설정된 곳에서 찾아 .jsp 접미사가 붙은 파일을 찾아 보여준다.
- *즉 [컨텍스트]/WEB-INF/view/“ 에서 helloView.jsp를 찾아 보여줌*

# SPRINGWEB을 톰캣에 DEPLOY

- SpringWeb프로젝트 를 우클릭하여 Maven 선택->Update Project 한 뒤
- Server에서 Tomcat을 선택하여 우 클릭하여 Add And Remove 클릭한 뒤 SpringWeb을 선택하여 오른쪽 리스트로 보낸다.
- 서버가 시작되면 웹브라우저를 열어
- http://localhost:8080/SpringWeb/hello.do
- 하면 실행 결과를 볼 수 있다.

# SPRING MVC 요청 처리 과정



# CONTROLLER구현 과정

- 1. @Controller 어노테이션을 클래스에 적용
- 2. @RequestMapping 을 이용해 요청 경로 지정
- 3. 웹브라우저 요청을 처리할 메소드를 구현하고 뷰 이름을 반환

# DISPATCHERSERVLET설정

- DispatcherServlet은 기본적으로 웹어플리케이션의 /WEB-INF/ 디렉토리에 위치한 [서블릿이름]-servlet.xml파일로 부터 설정 정보를 읽어온다.

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>
```

- 위 같은 설정일 경우 action-servlet.xml이 기본 설정파일이 됨.
- 그러나 한 개 이상의 설정 파일을 사용하고자 할 경우 contextConfigLocation 초기화 파라미터에 설정할 파일 목록을 지정토록 한다.

# DISPATCHERSERVLET설정

```
<servlet>
    <servlet-name>dispatcherServlet2</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
        /WEB-INF/distacherServlet2-servlet.xml
        /WEB-INF/mvc-config.xml
        </param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>
```

- 설정 파일 목록은 콤마(,), 공백문자( ), 세미콜론(;), 줄바꿈, 탭 등을 이용하여 구분할 수 있다.
- 각 설정 파일 경로는 웹 어플리케이션 루트 디렉토리를 기준으로 함

# DISPATCHERSERVLET설정

- DispatcherServlet은 한 개 이상 설정하는 것이 가능.

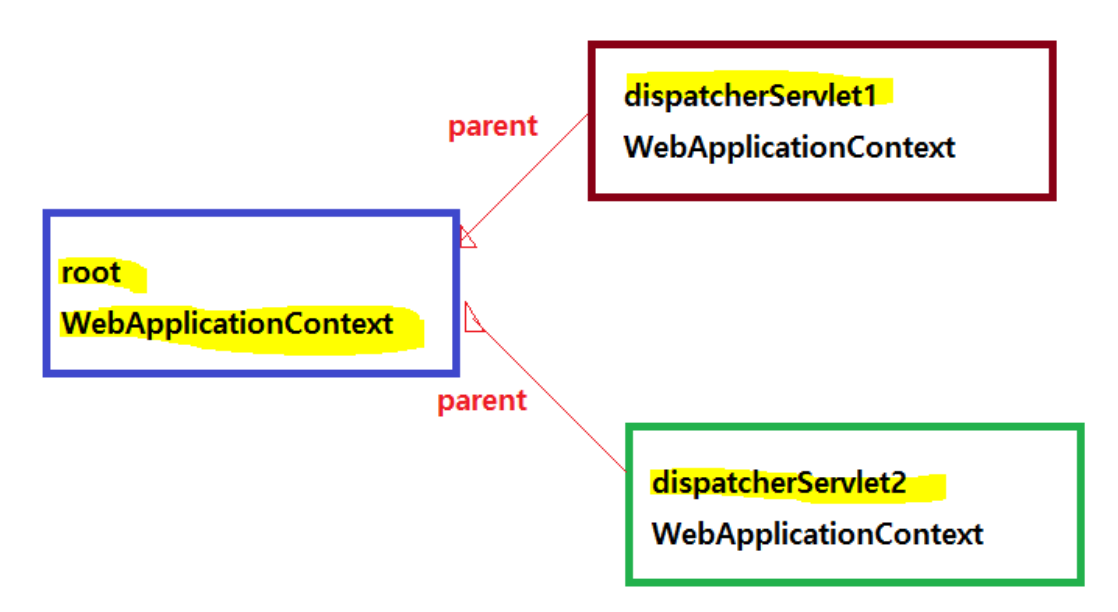
```
-->
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <!-- <param-value>/WEB-INF/dispatcherServlet-servlet.xml</param-value> -->
        <param-value>/WEB-INF/mvc-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

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

<servlet>
    <servlet-name>dispatcherServlet2</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
        /WEB-INF/distacherServlet2-servlet.xml
        /WEB-INF/mvc-config.xml
        </param-value>
    </init-param>
```

# WEBAPPLICATIONCONTEXT설정

- 앞 그림과 같이 여러 개의 DispatcherServlet을 등록하고, 이 두 서블릿이 공통으로 사용될 빈이 필요하다면
- ContextLoaderListener를 ServletListener로 등록하고 contextConfigLocation 컨텍스트 파라미터(context-param)를 이용하여 공통으로 사용될 빈 정보를 설정할 파일 목록을 지정하면 됨

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring/application-config.xml</param-value>
</context-param>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
<servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <!-- <param-value>/WEB-INF/dispatcherServlet-servlet.xml</param-value> -->
        <param-value>/WEB-INF/mvc-config.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping><servlet-name>dispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
<servlet>
    <servlet-name>dispatcherServlet2</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
        /WEB-INF/distacherServlet2-servlet.xml
        /WEB-INF/mvc-config.xml
```

- ContextLoaderListener와 DispatcherServlet은 각각 WebApplicationContext객체를 생성한다.

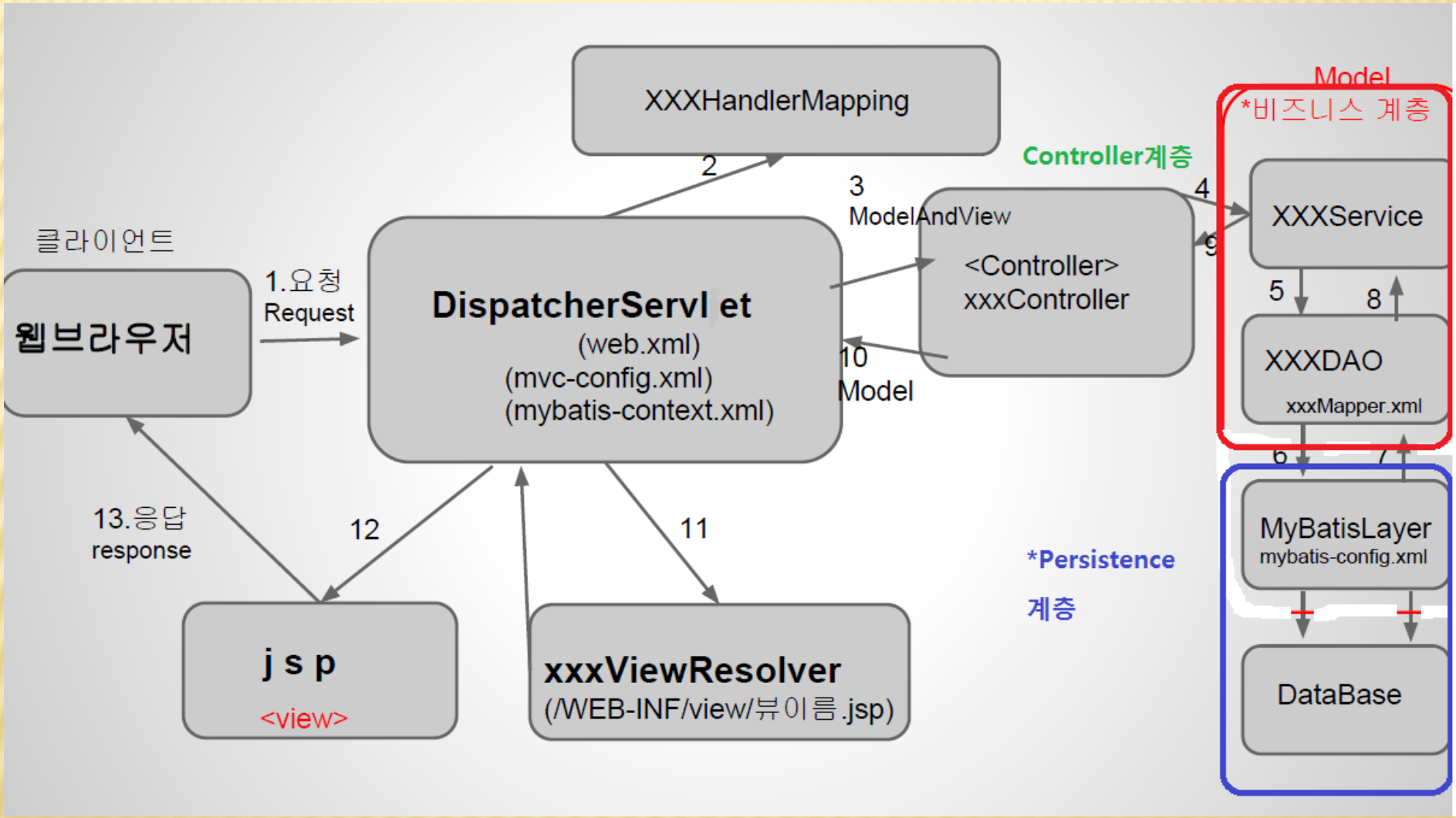



- 이때 ContextLoaderListener가 생성하는 WebApplicationContexst가 루트 컨텍스트가 됨

# WEBAPPLICATIONCONTEXT 설정

- DispatcherServlet이 생성하는 WebApplicationContext는 root context를 부모로 사용하는 자식 context가 된다.
- 이 때 자식은 root가 제공하는 빈을 사용할 수 있다.
- 각각의 DispatcherServlet이 공통으로 사용하는 빈(가령 데이터 소스 설정 관련) 설정은 ContextLoaderListener를 이용하여 설정한다.
- ContextLoaderListener 는 contextConfigLocation 컨텍스트 파라미터를 명시하지 않으면 /WEB-INF/applicationContext.xml을 설정파일로 사용한다.

# SPRING MVC WEB의 전체 흐름도



# 웹 어플리케이션에서 사용되는 주 어노테이션

- @Controller
- @Service
- @Repository
- @Resource
- @RequestMapping
- @RequestParam
- @ModelAttribute

# @RESOURCE

개요 : 어플리케이션에서 필요로 하는 자원을 자동 연결할 때 사용. 스프링 2.5 부터 지원.
스프링에서는 의존하는 빈 객체 전달할 때 사용.

@Autowired 와 같은 기능을 하며

@Autowired와 차이점은

@Autowired는 타입으로(by type),

@Resource는 이름으로(by name)으로 연결.

# @RESOURCE

- 설정위치 : 프로퍼티, setter 메소드
- 추가설정 : <bean class="org.springframework.beans.factory.annotation.CommonAnnotationBeanPostProcessor"/> 클래스를 빈으로 등록해야 함.
- 해당 설정 대신에 <context:annotation-config> 태그를 사용해도 된다.
- 옵션 : name
- name속성에 자동으로 연결될 빈객체의 이름을 입력한다.
- @Resource(name="testDao“)

# @SERVICE

- 개요 : @Service를 적용한 클래스는 비지니스 로직이 들어가는 Service로 등록됨.
- Controller에 있는 @Autowired는 @Service("xxxService")에 등록된 xxxService와 변수명이 같아야 하며
- Service에 있는 @Autowired는 @Repository("xxxDao")에 등록된 xxDao와 변수명이 같아야 한다.

# @SERVICE

```
@Service("helloService")
 public class HelloServiceImpl implements HelloService {
    @Autowired
    private HelloDao helloDao;
     public void hello() {
            System.out.println("HelloServiceImpl :: hello()");
            helloDao.selectHello();
        }
    }
```

# @REPOSITORY

- 패키지 : org.springframework.stereotype
- 버 전 : spring 2.0부터 사용
- 개요 : Repository 어노테이션은 일반적으로 DAO에 사용되며 DB관련 Exception을 DataAccessException으로 변환한다.

# @CONTROLLER

- 패키지 : org.springframework.stereotype
- 개　요 : Spring MVC의 Controller 선언을 단순화시켜준다.
- @Controller로 등록된 클래스 파일에 대한 bean을 자동으로 생성해준다.
- Controller로 사용하고자 하는 클래스에 @Controller 지정해 주면 component-scan으로 자동 등록된다.

# @CONTROLLER

- - xml 설정

```
<context:component-scan base-package="my.*"/>
```

- - java

```
package my;
import org.springframework.stereotype.Controller;
 @Controller
  public class SpringC {

   }
```

# @REQUESTMAPPING

- 개요 : url을 class 또는 method와 mapping 시켜 주는 역할을 한다.
- class에 하나의 url mapping을 할 경우, class위에 @RequestMapping("/url")을 지정하며,

  GET 또는 POST 방식 등의 옵션을 줄 수 있다.
- 해당되는 method가 실행된 후, return 페이지가 따로 정의되어 있지 않으면

  @RequestMapping("/url")에서 설정된 url로 다시 돌아간다.

# @REQUESTMAPPING

[예제]

```
@Controller
    @RequestMapping("/my/hello/*")
    public class helloController{
        @RequestMapping(method=RequestMethod.GET, value="go")
        public returntype getMethodName(){
            :
        }
        @RequestMapping(method=RequestMethod.POST, value="go2")
        public returntype getMethodName2(){
            :
        }
    }
```

# @MODELATTRIBUTE

- 개요 : ModelAttribute annotation은 화면의 form 속성으로 넘어온 model을 맵핑된 method의 파라미터로 지정해주는 역할을 한다.
- 주로 POST 타입으로 넘어오는 form 속성의 model 값을 받아 올 때 사용된다.
- get/post 모두 통용된다.

# @MODELATTRIBUTE

[예제]

```
@Controller
    public class BlogController {
 /* 중략 */
  @RequestMapping("/updateBlog")
 public String updateBlogHandler(
                    @ModelAttribute("blog") Blog blog) {
            blogService.updateBlog(blog);
  return "redirect:findBlogs.do";
 }

}
```

# @SESSIONATTRIBUTES

개요 : 세션상에서 model의 정보를 유지하고 싶을 때 사용

```
@Controller
@SessionAttributes("blog")
public class BlogController {
  /* 중략 */
  @RequestMapping("/createBlog")
   public ModelMap createBlogHandler() {
       blog = new Blog();
       blog.setRegDate(new Date());
       return new ModelMap(blog);
 }
}
```

# @INITBINDER

- 개요 : WebDataBinder를 초기화하는 method를 지정 할 수 있는 설정을 제공한다.
- 일반적으로 WebDataBinder는 annotation handler 메소드의 command 와 form 객체 인자를 조작하는데 사용된다.
- InitBinder 메소드가 필수적으로 반환 값을 가질 필요는 없으며, 일반적으로 이런 경우에 void를 선언한다.
- 특별한 인자는 WebdataBinder와 WebRequest또는 Locale의 조합으로 이루어지며, 이러한 조건이 만족되면 context-specific editors를 등록하는 것이 허용된다.

# ANNOTATION 기반 CONTROLLER 에서 SERVLETCONTEXT 구하기

```
@Controller
@RequestMapping("/common/download")
public class DownloadController {
    @Autowired
    private ServletContext sc;
    @RequestMapping
    public ModelAndView download(@RequestParam("filePath")
  String filePath) throws Exception {
        String path = sc.getRealPath(filePath);
        return new ModelAndView("common.download",
                                "downloadFile", new File(path));
    }
}
```

- 참고서적:
- 스프링3.0-가메출판사
- 코드로 배우는 스프링웹프로젝트-남가람북스