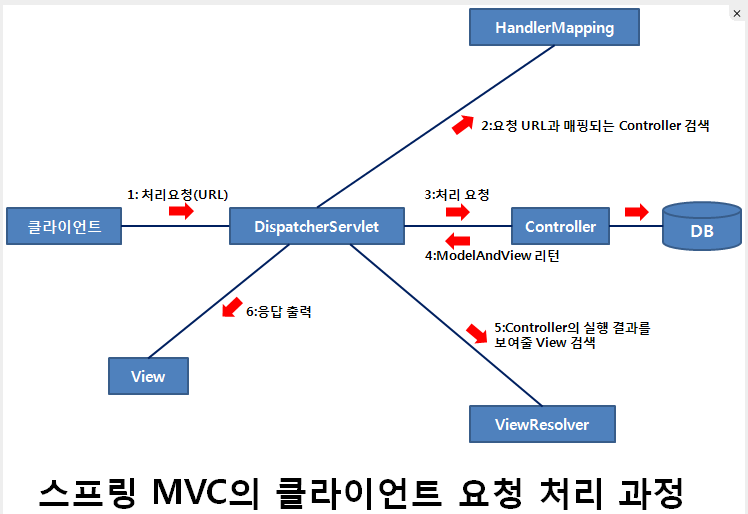
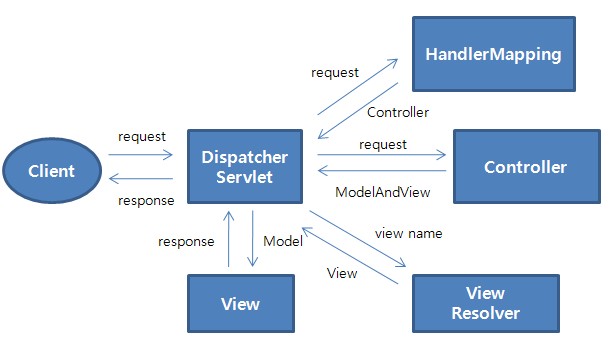
**스프링 MVC의 주요 구성 요소 및 처리 흐름**

[](https://camo.githubusercontent.com/c4427e73615ce42b251fe13319ff29047c830d22/687474703a2f2f6366696c6531302e75662e746973746f72792e636f6d2f696d6167652f31383736383633353530394341393239324339443938)[](http://www.egovframe.org/wiki/lib/exe/detail.php?id=egovframework:rte:ptl:spring_mvc_architecture&media=egovframework:rte:ptl:springmvcstructure.jpg)

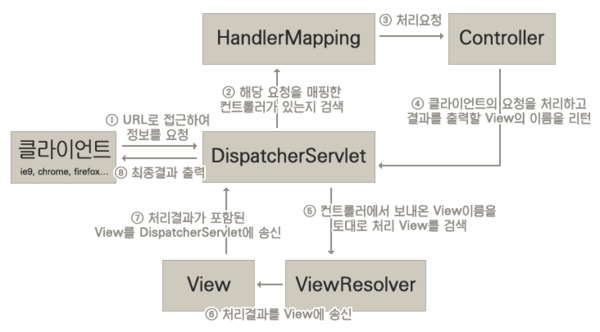
|  |  |
| --- | --- |
| **Component** | **개요** |
| **DispatcherServlet** | Spring MVC Framework의 Front Controller, 웹요청과 응답의 Life Cycle을 주관한다.  클라이언트의 요청을 전달받는다. Controller에게 클라이언트의 요청을 전달하고, Controller가 리턴 한 결과 값을 View에 전달하여 알맞은 응답을 생성하도록 한다. |
| **HandlerMapping** | 웹요청시 해당 URL을 어떤 Controller가 처리할지 결정한다. |
| **Controller** | 비지니스 로직을 수행하고 그 결과를 DispatcherServlet에 알려준다. 스트럿츠의 Action과 동일한 역할을 수행한다. |
| **ModelAndView** | Controller가 수행 결과를 반영하는 Model 데이터 객체와 이동할 페이지 정보(View객체)로 이루어져 있다. |
| **ViewResolver** | 어떤 View를 선택할지 결정한다. |
| **View** | 결과 데이터인 Model 객체를 display한다. |

이들 컴포넌트간의 관계와 흐름을 그림으로 나타내면 아래와 같다.

1. Client의 요청이 들어오면 DispatchServlet이 가장 먼저 요청을 받는다.
2. DispatcherServlet은 HandlerMapping이 요청에 해당하는 Controller객체를 구한 후 return한다.
3. Controller는 비지니스 로직을 수행(호출)하고 결과 데이터를 ModelAndView에 반영하여 return한다.
4. ViewResolver는 view name을 받아 해당하는 View 객체를 return한다.
5. View는 Model 객체를 받아 rendering한다.

==심화

@MVC에서 DispatcherServlet가 담당하는 역할이 무엇인지 알아봅시다. 먼저 DispatcherServlet에 대해 간단히 정의해보자면 우리가 각각 분리하여 만든 Model 파트와 Controller파트 View파트를 조합하여 브라우저로 출력해주는 역할을 수행하는 클래스라 할 수 있겠네요.

[](http://pds21.egloos.com/pds/201202/24/49/d0144949_4f47a0b19ed33.png)

간단하게 DispatcherServlet이 어떤 식으로 클라이언트의 요청을 처리하고 응답하는지 UML과 비슷한 방식으로 나타내 보았습니다. 아마 DispatcherServlet을 처음 접해본 분이시라면 모델2보다 복잡한 처리과정에 당황하실 수도 있겠네요. 하지만 위의 그림이 아무리 복잡해도 당황하실 필요는 없습니다. 어디까지나 저 처리과정의 대부분은 컨테이너가 대신 작업해주며 사용자가 직접 구현해야 될 분량은 얼마 되지 않으니까요. 먼저 위의 작업흐름을 풀어 자세히 설명하자면 다음과 같습니다.

① 클라이언트가 해당 어플리케이션에 접근하면 접근한 URL 요청을 DispatcherServlet이 가로챕니다. 이렇게 요청을 가로챌 수 있는 이유는 web.xml에 등록된 DispatcherServlet의 <url-pattern>이 '/'와 같이 해당 어플리케이션의 모든 URL로 등록되있기 때문입니다. 만약 특정 URL만 적용하고 싶다면 <url-pattern>의 내용을 바꿔주어 범위를 변경시키주면 됩니다.

② 가로챈 정보를 HandlerMapping에게 보내 해당 요청을 처리할 수 있는 Controller를 찾아냅니다. (스프링은 기본적으로 5가지의 핸들러 매핑을 제공합니다.) 이 부분은 스프링의 디폴트 전략에 의해 BeanNameUrlHandlerMapping과 DefaultAnnotationHandlerMapping이 기본으로 스프링MVC에 탑재되있기 때문에 특별한 경우가 아니라면 따로 설정할 필요가 없습니다.

③ 핸들러매핑이 해당 요청을 처리할 컨트롤러를 찾아냈다면 요청을 컨트롤러에 보내줍니다. 컨트롤러는 사용자가 직접 구현해주는 부분입니다. @MVC는 매우 다양한 코딩방식과 직관적이고 편리한 컨트롤러 작성방법을 제공하므로 이 부분에 대해서는 차후 심층적인 분석하여 자신에게 알맞는 전략을 선정해야 합니다.

④ 컨트롤러를 해당 요청을 처리한 후에 보통 컨트롤러는 요청을 응답받을 View의 이름을 리턴하게 됩니다. (물론 다른 핸들러 매핑 전략을 이용한다면 응답 과정이 다를 수도 있습니다.) 그 때 이 이름을 ViwResolver가 먼저 받아 해당하는 View가 존재하는지 검색합니다.

⑥ 해당 View가 있다면 처리결과를 View에 보낸 후 ⑦ 이 결과를 다시 DispatcherServier에 보낸 후 ⑧ DispatcherServlet은 최종 결과를 클라이언트에 전송합니다.

// - Controller > Service > ServiceImpl > DAO > SQL(XML) > JSP

각각 주석 참조

**1. 단계 : DispatcherServlet 설정 및 스프링 컨텍스트 설정**

- 자바 웹 어플리케이션의 설정 파일인 web.xml에 다음의 정보를 추가  
- a : 클라이언트의 요청을 전달받을 DispatcherServlet 설정  
- b : 공통으로 사용할 어플리케이션 컨텍스트 설정

DispatcherServlet의 설정은 웹 어플리케이션의 /WEB-INF/web.xml파일에 추가하면 됨.

<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>

DispatcherServlet은 WEB-INF/ 디렉토리에 위치한 [서블릿명]-servlet.xml 파일을 스프링 설정 파일로 사용한다.

**2 단계 : Controller 구현 및 설정 추가**

HandlerMapping은 클라이언트의 요청을 어떤 Controller가 처리할지에 대한 정보를 제공  
http://host:port[/컨텍스트경로]/hello.do

@Controller

public class HelloController{

@RequestMapping("/hello.do")

public ModelAndView hello(){

return ...;

}

}

[서블릿명]-servlet.xml 설정  
dispatcher-servlet.xml

<bean id="helloController" class="madvirus.spring.chap06.controller.HelloController"></bean>

위의 내용은 아래의 내용으로 대체

net 하위 모든 패키지에 있는 파일의 어노테이션을 스캔해서 빈으로 등록하겠다는 말

즉 사용하는 bean을 일일이 xml에 선언하지 않고도 필요한 것을 어노테이션(Annotation)을 자동으로 인식하게 하는 역할

<context:component-scan base-package="net"></context:component-scan>

**3단계 : 설정 파일에 ViewResolver 설정 추가**

@Controller

public class HelloController{

@RequestMapping("/hello")

public ModelAndView hello(){

ModelAndView mav = new ModelAndView();

mav.setViewName("hello");

mav.addObject("greeting",getGreeting());

return mav;

}

}

private String getGreeting(){

...

return "안녕하세요";

}

<bean

class="org.springframework.web.servlet.view.UrlBasedViewResolver" p:order="1"

p:viewClass="org.springframework.web.servlet.view.JstlView"

p:prefix="/" p:suffix=".jsp"> <!-- prefix는 페이지의 기본 루트 // 페이지의 기본 확장자 -->

</bean>

**4단계 : 뷰 코드 구현**  
hello.jsp

<html>

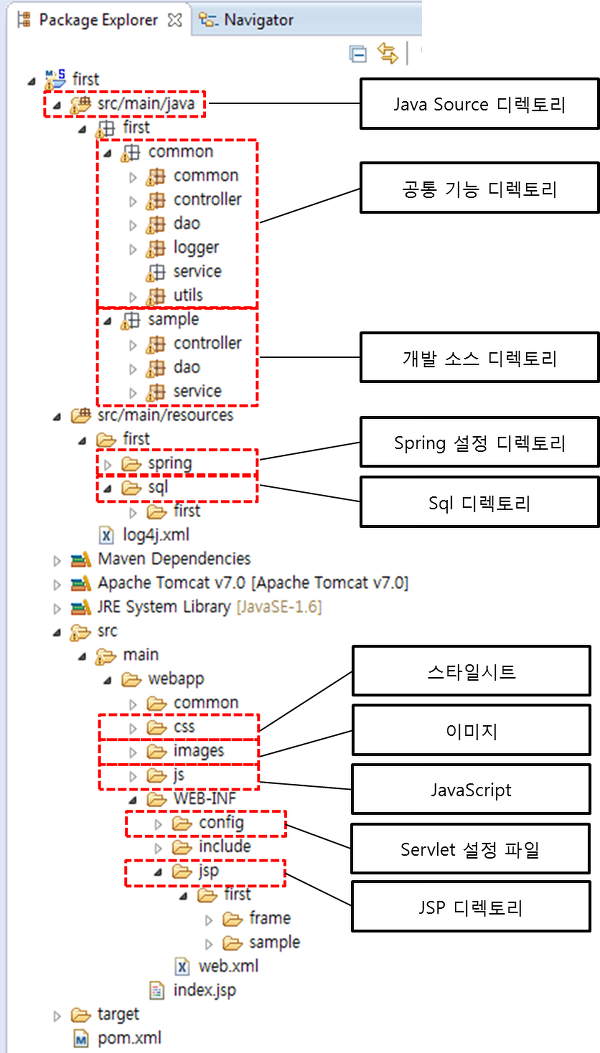
<body>

인사말 : ${greeting}

</body>

<html>

**5단계 실행**  
<http://localhost:8080/chap06/hello.do>



1) src/main/java는 java 파일이 모여있는 디렉토리이다. 우리가 앞으로 만들 java 파일은 전부 이 디렉토리에 구성된다.

2) src/main/resources는 여기서는 사용하지 않지만, 추후 스프링 설정 파일이나 쿼리가 저장될 디렉토리이다.

3) src/test/ 관련 폴더는 test 관련 폴더인데, 우리는 TDD(Test Driven Development) 방법론이나 테스트코드를 따로 작성하는 방식은 아직까지는 사용하지 않기 때문에, 현재로써는 필요없는 폴더이다. 과감히 삭제.

4) 우리는 메이븐을 사용하는데, 메이븐의 기본 폴더는 src/main/webapp 폴더가 기본 폴더이다. webapp 폴더 밑에 모든 jsp 및 js 등의 파일이 포함된다.

\*view 위치만 변동

5) servlet-context.xml, root-context.xml은 서블릿(Servlet)관련 설정파일이다.