***스프링 MVC의 주요 구성 요소 및 처리 흐름***

**1. index.jsp 파일에서 location.href="/SpringWeb/hello.do“ 요청 하면 web.xml 파일에서 클라이언트에서 \*.do로 요청이 들어오면 DispatcherServlet 을 설정해준다.**

|  |
| --- |
| <servlet>  <servlet-name>springapp</servlet-name>  <servlet-class>  org.springframework.web.servlet.DispatcherServlet  </servlet-class>  </servlet>    <servlet-mapping>  <servlet-name>springapp</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping> |

\* DispatcherServlet

- 클라이언트의 요청을 전달 받는다. 컨트롤러에게 클라이언트의 요청을 전달하고, 컨트롤러가 리턴한 결과 값을 View에 전달하여 알맞은 응답을 생성한다.

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

Ex) 여기서는 <servlet-name>springapp</servlet-name> 지정 되면 springapp-servlet.xml이렇게 지정해주면된다.

- 이파일에서는 HandlerMapping, Controller, ViewResolver, View 등의 빈을 설정한다.

**2-3. 컨트롤러를 요청 & / “hello.do” 빈 리턴**

|  |
| --- |
| <bean id="handlerMapping" class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">  </bean> |

\* HandlerMapping

- 클라이언트의 요청을 어떤 컨트롤러가 처리할 컨트롤러를 구하기 위해 즉 HandlerMapping을 통하여 해당하는 Controller와 연결 시켜준다.

- 여기서 사용하는 BeanNameUrlHandlerMapping은 클라이언트의 요청 URL과 동일한 이름을 갖는 빈을 컨트롤러로 사용하므로 “**/hello.do”빈** 를 리턴한다.

**4. HandlerMapping으로 “/hello.do”를 리턴하면 “/hello.do”인 컨트롤러를 이용하여 클라이언트를 요청 처리한다.**

|  |
| --- |
| <bean id="handlerMapping" class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping">  </bean>  <bean name="/hello.do" class="springapp.web.HelloController"></bean> |

여기서는 “/hello.do” 컨트롤러는 springapp.web.HelloController 컨트롤러를 처리한다.

|  |
| --- |
| **package** springapp.web;  **import** java.util.Calendar;  **import** javax.servlet.http.HttpServletRequest;  **import** javax.servlet.http.HttpServletResponse;  **import** org.springframework.web.servlet.ModelAndView;  **import** org.springframework.web.servlet.mvc.AbstractController;  **public** **class** HelloController **extends** AbstractController {  @Override  // Controller가 어떤 컨트롤러인지에 따라서 handleRequestInternal 이 메소드 이름이 바뀐다.  **protected** ModelAndView handleRequestInternal(HttpServletRequest arg0,  HttpServletResponse arg1) **throws** Exception {  ModelAndView mav = **new** ModelAndView();  //뷰정보와 가져갈 정보를 ModelAndView객체에 넣어준다.  mav.setViewName("hello");  // 여기서는 값만 넘겨주지 어디로 가라는지는 알려주지 않는다.  //viewResolver에서 어디경로로 가라는지 지정해준다.  mav.addObject("greeting",getGreeting());  **return** mav;  }    **private** String getGreeting()  {  **int** hour = Calendar.*getInstance*().get(Calendar.*HOUR\_OF\_DAY*);  **if**(hour>=6 && hour<=10){  **return** "좋은 아침입니다";  }**else** **if**(hour>=12 && hour<=16){  **return** "점심 식사 했나요?";  }**else** **if**(hour>=18 && hour<=20){  **return** "좋은 밤 되세요";  }**else**{  **return** "안녕하세요";  }  }  } |

\*컨트롤러(Controller)

- 클라이언트의 요청을 알맞은 작업을 처리한 뒤 그 결과를 DispatcherServlet에 알려준다.(스트럿쳐의 Action과 동일)

- 컨트롤러의 처리 결과를 ModelAndView에 담아서 DispatcherServlet에 전달한다.

- AbstractController : 별도의 기능을 제공하지 않는 컨트롤러이다. 단순히 ModelAndView를 리턴하고 싶을 때 상속받아 컨트롤러를 구현한다.

- ModelAndView : Controller의 처리 결과를 보여줄 뷰이름, 뷰에서 사용될 값들을 저장

setViewName() 메서드 : viewResolver에서 사용하는 이름으로 여기서는 /view/hello.jsp파일을 가리킨다.

addObject() 메서드 : 처리된 결과를 담는 메서드. struts에서 setAttribute()로 해서 값을 전달한 것과 같다.

**5. Controller를 처리하고 나서 ModelAndView(뷰이름 : “hello”)를 리턴한다.**

|  |
| --- |
| ModelAndView mav = **new** ModelAndView();  mav.setViewName("hello"); |

**6-7. “hello” 매핑되는 View 객체 요청 하고 만들어진 “/view/hello.jsp” View 객체를 리턴**

|  |
| --- |
| <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="prefix" value="/view/"/>  <property name="suffix" value=".jsp"/>  </bean> |

\* ViewResolver

- 컨트롤러의 처리 결과를 생성할 뷰를 결정한다.

- 여기서 뷰이름을 “hello”를 지정했는데 ViewResolver는 이 뷰 이름을 이용하여 알맞은 View 객체를 선택한다. 여기서 결과는 /view/hello.jsp를 뷰를 사용하는 View객체를 생성하였다.

- InternalResourceViewResolver : JSP를 이용하여 뷰를 구현할 때 사용되는 ViewResolver이다.

: 컨트롤러가 지정한 뷰 이름 앞뒤로 (mav.setViewName("hello");) prefix 프로퍼티와 suffix프로퍼티를 붙인 값이 실제로 사용될 자원의 경로가 된다.

**8-9. View를 리턴 받으면 DispatcherServlet에서 응답 생성 요청 하면 InternalResourceView를 사용하므로 JSP를 이용하여 응답 처리한다.**

|  |
| --- |
| <body>  인사말 : <strong>${greeting }</strong>  </body> |

\* View

- 컨트롤러의 처리 결과 화면을 생성한다.

***Dependency Injection패턴***

-스프링 컨테이너가 지원하는 핵심 개념 중의 하나로 객체 사이의 의존 관계를 객체 자신이 아닌 외부의 조립기가 수행 하는 개념

- DI 사용 이유

1. 코드에 직접 의존 클래스 명시

-> 단위 테스트(의존하는 클래스가 있으면 의존하는 클래스는 반드시 올바르게 동작을 해야한다.)가 여렵고 의존하는 클래스가 변경되는 경우 코드를 변경해야 하는 문제가 있다.

2. Factory 패턴 사용

-> Factory 구현 클래스에 대한 의존 관계를 줄일 수 있지만, 대신 Factory 클래스에 대한 의존 관계가 추가된다.

즉!!!! 1, 2와 같은 방법의 단점을 극복하기 위해서 나온게 DI패턴이다.

- 스프링에서의 DI

|  |
| --- |
| 자바파일 |
| **public** **class** WriteArticleServiceImpi **implements** WriteArticleService {  ArticleDao articleDao;  **public** WriteArticleServiceImpi(ArticleDao articleDao)  {  **this**.articleDao = articleDao;  }  **public** **void** write(Article article) {  System.*out*.println("WriteArticleServiceImpi.write() 메서드 출력됬거덩!");  articleDao.insert(article);  }  }  **public** **interface** WriteArticleService {  **public** **void** write(Article article);  }  **public** **class** MysqlArticleDao **implements** ArticleDao {  **public** **void** insert(Article article) {  System.*out*.println("MysqlArticleDao.insert() 메서드 호출 됬거덩");  }  **public** **interface** ArticleDao {  **void** insert(Article article);  }  **public** **class** Article {} |

|  |
| --- |
| 스프링 서식 파일을 이용한 의존 관계 설정(조립기에 해당된다.) |
| <?xml version="1.0" encoding="UTF-8"?>  <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">  <bean name = "writeArticleService" class="springapp.WriteArticleServiceImpi">  <constructor-arg>  <ref bean="articleDao"/>  </constructor-arg>  </bean>  <bean name="articleDao" class="springapp.MysqlArticleDao">  </bean>  </beans> |
| 여기가 어셈블러 역할(조립기)을 한다. 관계를 주고 받는 역할을 여기서 한다. new를 하지 않고 been을 통해서 객체를 생성하고 넘겨주는 역할을 한다.  <beans> 태그 : 스프링 설정 파일의 루트 트개ㅡ  <bean> 태그 : 스프링이 관리할 하나의 객체를 설정  Name : 빈의 이름 class : 생성될 객체의 클래스 타입  < constructor-arg > 태그 : 생성자에 전달할 파라미터를 명시하기 위해 사용된다. |

|  |
| --- |
| 스프링 컨테이터로부터 빈 객체 가져와 사용 |
| **import** org.springframework.beans.factory.BeanFactory;  **import** org.springframework.beans.factory.xml.XmlBeanFactory;  **import** org.springframework.core.io.Resource;  **import** org.springframework.core.io.ClassPathResource;  **public** **class** Main {  **public** **static** **void** main(String[] args) {  //applicationContext.xml 파일의 리소스를 가져온다.  Resource resource = **new** ClassPathResource("applicationContext.xml");  BeanFactory beanFactory = **new** XmlBeanFactory(resource);    //<been> 태그 name으로 해당하는 객체를 구할 수 있다.  WriteArticleService articleService = (WriteArticleService) beanFactory.getBean("wrteArticleService");    articleService.write(**new** Article());  }  } |
| XmlBeanFactory 클래스 : Resource가 나타내는 XML 파일로부터 스프링 설정 내용 로딩  BeanFactory 클래스 : 빈 객체를 생성  <been> 태그로 만들걸 하나씩 가져오는걸 getBean()메서드이다. Object타입이므로 형변환 해준다. |

*AOP (Aspect Oriented Programming)*

- 공통 관심 사항(ex : 로깅, 트렌젝션처리, 보안 등)을 구현한 모듈에 의존 관계를 갖지 않아 의존 관계의 복잡성과 코드 중복을 해소해주는 프로그래밍 기법